Евгений Бердышев

Технология MMX

новые возможности процессоров P5 и P6

Москва • "ДИАЛОГ-МИФИ" • 1997

УДК 681.3
ББК 32.97
Б48

Бердышев Е.

Б48 Технология. новые возможности процессоров P5 и P6. – М.: ДИАЛОГ-МИФИ, 1997.

ISBN 5-86404-105-x

Книга знакомит читателя с новым изобретением фирмы Intel – технологией MMX. В книге приводится описание архитектуры MMX, а также в качестве примеровтана на программистов, занимающихся разработкой программ на ассемблере.

Книга знакомит читателя с новым изобретением фирмы Intel – технологией MMX. В книге приводится описание архитектуры MMX, а также в качестве примеров приведены исходные тексты MMX-алгоритмов.

Кроме технологии MMX в книге рассмотрена работа конвейера процессоров P5 и P6, рассмотрены методы оптимизации программ,приведено описание некоторых новых возможностей процессоров P5 и P6 по сравнению с Intel486 (например, системный режим, расширенные возможности страничного преобразования и др.)

Весь материал книги является новым, в том смысле, что ранее на русском языке не издавался.

Книга рассчитана на программистов, занимающихся разработкой программ на ассемблере.

ООО "Издательство ДИАЛОГ-МИФИ"
115409, Москва, ул. Москворечье, 31, корп. 2. Т.: (495) 320-30-77, 320-43-77
Http://www.dialog-mifi.ru. E-mail:zakaz@dialog-mifi.ru
142100, г. Подольск, Московская обл., ул. Кирова, 25

ISBN 5-86404-105-x

ПРЕДИСЛОВИЕ

Объем и сложность данных, обрабатываемых современными компьютерами, стремительно увеличиваются. Новые средства связи, видео- и аудиоприложения выдвигают повышенные требования к производительности микропроцессора.

MMX-технология разработана для ускорения мультимедиа и коммуникационных программ. Она включает в себя новые команды и типы данных, что позволяет создавать приложения нового уровня. Технология основана на параллельной обработке данных. При этом сохраняется полная совместимость с существующими операционными системами и программным обеспечением.

MMX-технология – это самое значительное усовершенствование со времени создания процессора Intel386 (т. е. создание 32-разрядной архитектуры).

Наибольший эффект от использования MMX-технологии может быть достигнут в алгоритмах со следующими характеристиками:

⇒ малый размер данных (например, 8-битные графические пикселы, 16-битные звуковые данные);

⇒ короткие, часто повторяющиеся циклы;

⇒ частые умножения и накопления.

В основе MMX лежит принцип SIMD (Single Instruction Multiple Data), т. е. одной командой можно обработать сразу несколько единиц информации.

Кроме MMX-технологии в книге также рассматриваются некоторые новые (по сравнению с Intel486) возможности процессоров P5 и P6. В книге не приводится базовая архитектура Intel486. Если читатель незнаком с тем, что из себя представ-ляют процессоры Intel (архитектуры x86), то можно обратиться к следующей литературе:

⇒ Шагурин И. И., Бродин В. Б. Микропроцессор i486. Архитектура, программирование, интерфейс. М.: Диалог-МИФИ, 1993.

⇒ Григорьев В. Л. Микропроцессор i486. Архитектура и программирование: В 4 т. М.: Гранал, 1993.

1. ПРОГРАММНАЯ МОДЕЛЬ MMX

1.1. FPU и MMX регистры

MMX-регистры отображены на регистры FPU (рис. 1.1). Главным образом это сделано для сохранения полной совместимости с существующим программным обеспечением.

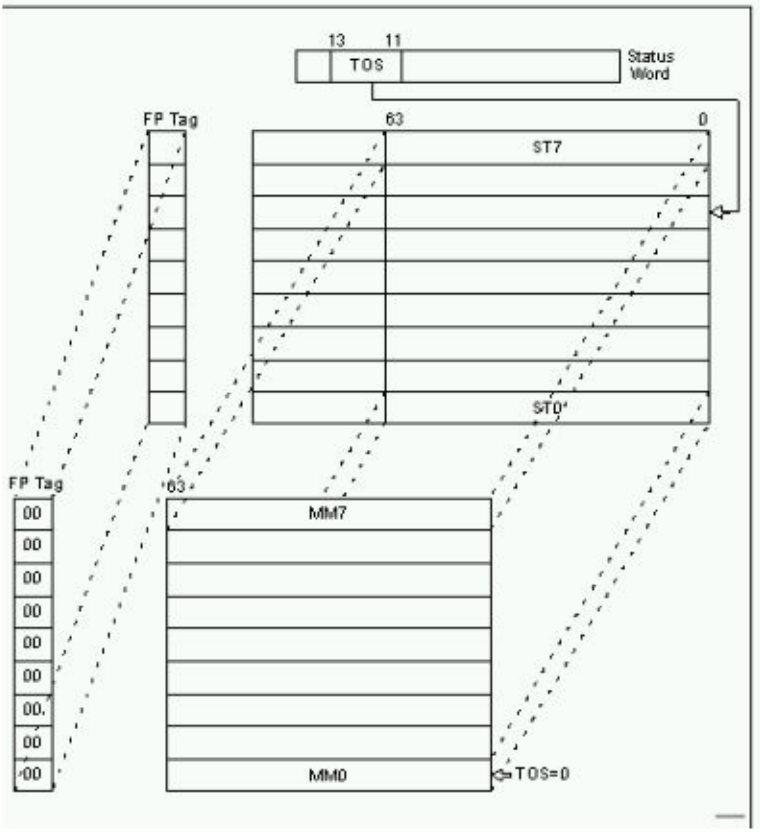


Рис. 1.1. Отображение MMX-регистров на FPU-регистры

Из рис. 1.1 видно, что MMX-регистры отображены на поля мантиссы в FPU-регистрах. Значение, записываемое в MMX-регистр, автоматически появляется в младших битах (биты 63-0) соответствующих FPU-регистров. При этом в поле порядка (биты 78-64) и в знаковый бит (бит 79) заносятся единицы. Значение поля TOS (Top Of Stack) устанавливается в 0 после выпол-

нения каждой MMX-команды. Значение мантиссы, записываемое в FPU-регистр с помощью FPU-команды, автоматически появляется в соответствующем MMX-регистре.

Отображение MMX-регистров фиксировано и не зависит от значения поля TOS (биты 11-13 в регистре состояния FPU). В обозначении MMn n указывает на физический номер регистра. Напомним, что в обозначении Stn n указывает на относительный номер регистра (относительно поля TOS).

При TOS=0: MM0 отображается на ST0, MM1 – на ST1 и т.д. При TOS=2: MM0 отображается на ST6, MM1 – на ST7, MM2 – на ST0 и т.д.

После выполнения любой MMX-команды (кроме EMMS) значения всех полей регистра тегов устанавливаются в 00. Команда EMMS устанавливает значения всех полей регистра тегов (табл. 11). Значение регистра тегов не оказывает никакого влияния на MMX-регистры или выполнение MMX-команд.

Таблица 1.1. Влияние MMX-команд на контекст FPU

Тип команды	*Регистр тегов*	*Поле TOS*	*Другие регистры*	*Поле порядка и знаковый бит MMn (79...64)*	*Поле мантиссы MMn (63...00)*
Чтение из MMX-регистра	Все поля 00	000	Не изменяется	Не изменяется	Не изменяется
Запись в MMX-регистр	Все поля 00	000	Не изменяется	Заполняется единицами	Перезаписывается
EMMS	Все поля 11	000	Не изменяется	Не изменяется	Не изменяется

Так как MMX и FPU используют физически одни и те же регистры, для сохранения и восстановления контекста MMX используются команды FSAVE (Store FP state) и FRSTOR (Restore FP state). Если при попытке выполнить MMX-команду бит TS в регистре CR0 установлен в единицу, то генерируется исключение Int 7. Благодаря этому факту обеспечивается прозрачность управления контекстом MMX для операционной системы.

Краткие рекомендации по работе с MMX и FPU регистрами:

⇒ не смешивайте MMX и FPU код на уровне команд. Делайте так, чтобы отдельные подпрограммы содержали команды только одного типа;

⇒ когда MMX-контекст больше не нужен, очистите его с помощью EMMS-команды;

⇒ выходите из секции FPU-кода с пустым регистровым стеком.

1.2. Новый формат представления данных

ММХ-технология вводит следующие 4 новых 64-разрядных типа данных (рис. 1.2):

⇒ упакованные байты (Packed bytes);

⇒ упакованные слова (Packed words);

⇒ упакованные двойные слова (Packed doublewords);

⇒ учетверенное слово (Quadword).

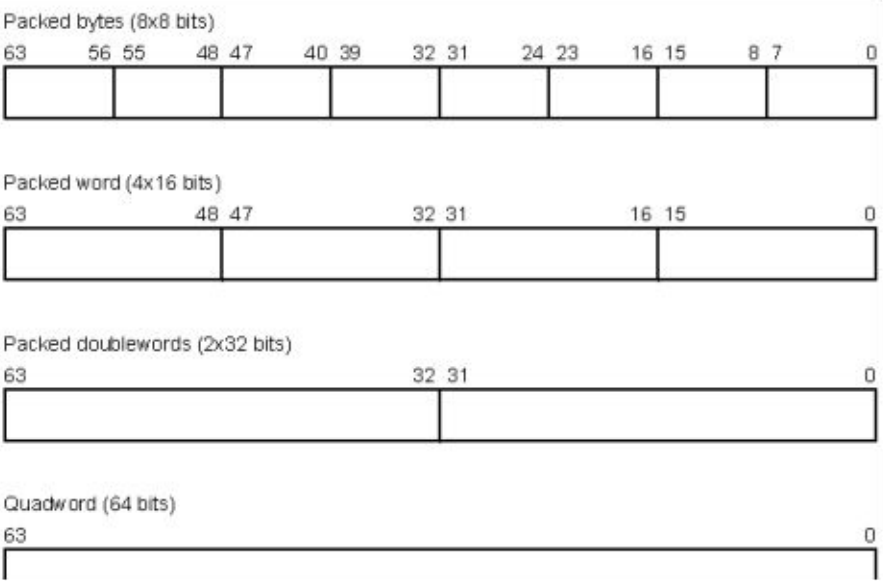


Рис. 1.2. Упакованные типы данных

Например, графические данные обычно представляются в виде байтов. ММХ-технология дает возможность поместить 8 байт вместе в один 64-разрядный ММХ-регистр. Когда исполняется ММХ-команда, берутся сразу все 8 байт из ММХ-регистра, затем производится операция над всеми 8 байтами параллельно и затем результат опять записывается в ММХ-регистр.

В памяти новые типы данных располагаются так, как это принято в Intel-архитектуре, т. е. по принципу младший байт первым (рис. 1.3).

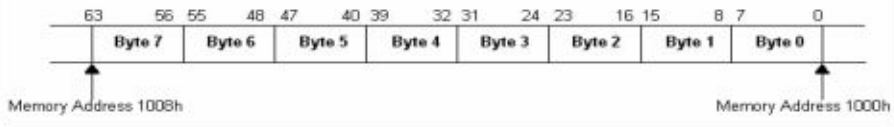


Рис. 1.3. Восемь упакованных байт в памяти по адресу 1000h

1.3. Арифметика с насыщением

MMX-технология поддерживает новую арифметику, называемую арифметикой с насыщением (Saturation arithmetic). Противопоставим ее арифметике с циклическим переносом (Wraparound arithmetic).

В режиме циклического переноса в случае переполнения результат обрезается и возвращаются только младшие биты результата (рис. 1.4). Данный метод вычисления хорошо известен, он используется при операциях над целочисленными регистрами.

В режиме насыщения результат операции, который выходит за границу размера данных, насыщается до предельно возможного значения для используемого типа данных (рис. 1.5).

Например, если результат превышает значение 0x7F (для операций над байтами со знаком) или значение 0xFF (для операций над байтами без знака), то результатом такой операции будет соответственно 0x7F или 0xFF. Если результат получается меньше 0x80 (для знаковых байтов) или меньше 0x00 (для беззнаковых байтов), то результатом данной операции будет соответственно 0x80 или 0x00 (табл. 1.2).

Таблица 1.2. Пределы насыщения

	Нижний предел		***Верхний предел***	
Со знаком	*Шестнадцатеричное*	*Десятичное*	*Шестнадцатеричное*	*Десятичное*
Байт Слово	80h 8000h	-128 -32768	7Fh 7FFFh	127 32767
Без знака				
Байт Слово	00h 0000h	0 0	FFh FFFFh	255 65535

Пример

a3	a2	a1	FFFFh
+	+	+	+
b3	b2	b1	8000h
=	=	=	=
a3+b3	a2+b2	a1+b1	7FFFh

Рис. 1.4. Арифметика с циклическим переносом

A3	a2	a1	FFFFh
+	+	+	+
b3	b2	b1	8000h
=	=	=	=
a3+b3	a2+b2	a1+b1	FFFFh

Рис. 1.5. Арифметика с насыщением

ММХ-команды не сообщают о переполнении с помощью генерации исключений или установки каких-либо флагов.

1.4. Особые случаи

ММХ-команды генерируют те же исключения при обращение к памяти, что и все остальные команды. Например, страничное нарушение (Int 14), нарушение общей защиты (Int 13) и т. д.

Попытка выполнить ММХ-команду, когда CR0.EM=1, приводит к генерации прерывания Int 6 – недействительный код операции. Эмуляция ММХ-команд невозможна.

При попытке выполнить ММХ-команду, когда CR0.TS=1, генерируется прерывание Int 7 – устройство недоступно.

Если при обработке FPU исключения встретится ММХ-команда, то произойдет исключение Int 16 и/или FERR#.

1.5. Влияние префиксов на выполнение ММХ-команд

В табл. 1.3 показано, какое влияние оказывают префиксы на выполнение ММХ-команд.

Таблица 1.3. Влияние префиксов на выполнение ММХ-команд

Тип префикса	*Влияние*
Размер адреса (67h)	Влияет на ММХ-команды с операндом в памяти. Игнорируется ММХ-командами без операндов в памяти
Размер операнда (66h)	Игнорируется

Тип префикса	*Влияние*
Замена сегмента	Влияет на MMX-команды с операндом в памяти. Игнорируется MMX-командами без операндов в памяти
Повторение	Игнорируется
Блокировка шины (Lock F0h)	Генерирует исключение недействительной операции

1.6. Определение MMX-технологии с помощью команды CPUID

Для того чтобы определить, поддерживает ли конкретная модель процессора MMX-технологию, можно воспользоваться командой CPUID.

```
mov     eax,1               ; Запрашиваем регистр с флагами
                            ; возможностей процессора
cpuid
test    edx,00800000h       ; Бит 23 регистра EDX установлен в
                            ; единицу, если процессор
                            ; поддерживает MMX-технологию
jnz     MMX_Found
```

Замечание. Команда CPUID будет продолжать выдавать присутствие технологии MMX, если CR0.EM=1.

2. СИСТЕМА КОМАНД ММХ

2.1. Арифметические команды

Команды PADD (Packed Add) и PSUB (Packed Subtract) производят операции сложения или вычитания над упакованными данными с использованием арифметики с циклическим переносом. Эти команды поддерживают упакованные байты, слова и двойные слова.

Команды PADDS (Packed Add with Saturation) и PSUBS (Packed Subtract with Saturation) производят операции сложения и вычитания над упакованными данными со знаком, используя арифметику с насыщением. Эти команды поддерживают упакованные байты и слова.

Команды PADDUS (Packed Add Unsigned with Saturation) и PSUBUS (Packed Subtract Unsigned with Saturation) выполняют действия, аналогичные командам PADDS и PSUBS, только над упакованными данными без знака.

Команды PMULHW (Packed Multiply High) и PMULLW (Packed Multiply Low) производят умножение упакованных слов со знаком. При этом команда PMULHW возвращает старшие 16 разрядов получившегося 32-разрядного результата, а команда PMULLW – младшие 16 разрядов.

Команда PMADDWD (Packed Multiply and Add) вычисляет произведение упакованных слов со знаком. Затем 4 промежуточных 32-х разрядных результата суммируются парами. В результате получается два 32-разрядных упакованных двойных слова (рис. 2.1).

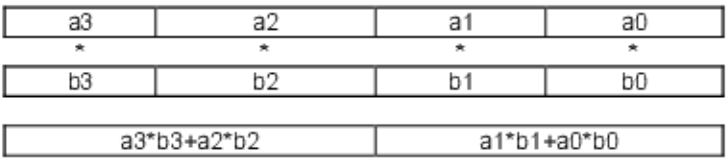


Рис. 2.1. Выполнение команды PADDWD

2.2. Команды сравнения

Команды PCMPEQ (Packed Compare for Equal) и PCMPGT (Packed Compare for Greater Then) производят сравнение соответствующих упакованных элементов операнда-приемника и операнда-источника. Команда PCMPEQ сравнивает, равны ли соответствующие элементы операндов между собой. Команда

PCMPGT сравнивает, больше ли элементы операнда-источника соответствующих элементов операнда-приемника. Результатом операции является маска из нулей (ложное сравнение) и единиц (истинное сравнение), записываемая в операнд-приемник (рис. 2.2). Эти команды поддерживают упакованные байты, слова и двойные слова.

23	45	16	34
gt?	**gt?**	**gt?**	**gt?**
31	7	16	67
0000h	FFFFh	0000h	0000h

Рис. 2.2. Выполнение команды PCMPGT

2.3. Команды преобразования

Команда PACKSS (Packed with Signed Saturation) преобразовывает слова со знаком в байты со знаком или двойные слова со знаком в слова со знаком, используя принцип насыщения.

Команда PACKUS (Packed with UNSIGNED Saturation) преобразовывает слова со знаком в байты без знака, используя принцип насыщения.

Команды PUNPCKH (Unpack High Packed Data) и PUNPCKL (Unpack Low Packed Data) преобразовывает байты в слова, слова в двойные слова и двойные слова в учетверенные слова.

2.4. Команды логических операций

Команды PAND (Bitwise Logical And), PANDN (Bitwise Logical And Not), POR (Bitwise Logical Or) и PXOR (Bitwise Logical Exclusive Or) производят соответствующие логические операции над 64-разрядными операндами.

2.5. Команды сдвига

Команды PSLL (Packed Shift Left Logical) и PSRL (Packed Shift Right Logical) производят логический сдвиг влево или вправо с заполнением освобождающихся позиций нулями. Эти команды поддерживают упакованные слова, двойные слова и учетверенные слова.

Команда PSRA (Packed Shift Right Arithmetic) выполняет арифметический сдвиг вправо, заполняя старшие биты элемента знаковым битом. Эта команда поддерживает упакованные слова и двойные слова.

2.6. Команды переноса данных

Команда MOVD (Move 32 Bits) переносит 32-разрядные упакованные данные из памяти в MMX-регистр и наоборот, а также из целочисленного регистра в MMX-регистр и наоборот.

Команда MOVEQ (Move 64 Bits) переносит 64-разрядные упакованные данные из памяти в MMX-регистр и наоборот, а также осуществляет обмен между MMX-регистрами.

2.7. Команда EMMS

Команда EMMS очищает контекст MMX, занося значение 11 во все поля регистра тегов. Эта команда должна выполняться в конце MMX-процедуры перед вызовом процедуры, в которой могут содержаться FPU-команды.

3. ПРИМЕРЫ ММХ-АЛГОРИТМОВ

Файл макроопределений iammx.inc для задания ММХ-команд

```
; file: iammx.inc

opc_Rdpmc    = 033H
opc_Emms     = 077H
opc_Movd_ld  = 06EH
opc_Movd_st  = 07EH
opc_Movq_ld  = 06FH
opc_Movq_st  = 07FH
opc_Packssdw = 06BH
opc_Packsswb = 063H
opc_Packuswb = 067H
opc_Paddb    = 0FCH
opc_Paddd    = 0FEH
opc_Paddsb   = 0ECH
opc_Paddsw   = 0EDH
opc_Paddusb  = 0DCH
opc_Paddusw  = 0DDH
opc_Paddw    = 0FDH
opc_Pand     = 0DBH
opc_Pandn    = 0DFH
opc_Pcmpeqb  = 074H
opc_Pcmpeqd  = 076H
opc_Pcmpeqw  = 075H
opc_Pcmpgtb  = 064H
opc_Pcmpgtd  = 066H
opc_Pcmpgtw  = 065H
opc_Pmaddwd  = 0F5H
opc_Pmulhw   = 0E5H
opc_Pmullw   = 0D5H
opc_Por      = 0EBH
opc_PSHimd   = 072H
opc_PSHimq   = 073H
opc_PSHimw   = 071H
opc_Pslld    = 0F2H
opc_Psllq    = 0F3H
opc_Psllw    = 0F1H
opc_Psrad    = 0E2H
opc_Psraw    = 0E1H
opc_Psrld    = 0D2H
opc_Psrlq    = 0D3H
opc_Psrlw    = 0D1H
opc_Psubb    = 0F8H
opc_Psubd    = 0FAH
opc_Psubsb   = 0E8H
```

```
opc_Psubsw    = 0E9H
opc_Psubusb   = 0D8H
opc_Psubusw   = 0D9H
opc_Psubw     = 0F9H
opc_Punpcklbw = 060H
opc_Punpckldq = 062H
opc_Punpcklwd = 061H
opc_Punpckhbw = 068H
opc_Punpckhdq = 06AH
opc_Punpckhwd = 069H
opc_Pxor      = 0EFH

.486P

rdpmc    macro
         db        0fh, opc_Rdpmc
endm

emms     macro
         db        0fh, opc_Emms
endm

movdt    macro   dst:req, src:req
     local   x, y
x:
     cmpxchg   src, dst
y:
     org     x+1
     byte    opc_Movd_ld
     org     y
     endm

movdf    macro   dst:req, src:req
     local   x, y
x:
     cmpxchg   dst, src
y:
     org     x+1
     byte    opc_Movd_st
     org     y
     endm

movq   macro   dst:req, src:req
     local   x, y
IF (OPATTR(dst)) AND 00010000y ; register
x:
     cmpxchg   src, dst
y:
     org     x+1
```

```
        byte    opc_Movq_ld
        org     y
ELSE
x:
        cmpxchg   dst, src
y:
        org     x+1
        byte    opc_Movq_st
        org     y
ENDIF
        endm

packssdw    macro   dst:req, src:req
        local   x, y
x:
        cmpxchg   src, dst
y:
        org     x+1
        byte    opc_Packssdw
        org     y
        endm

packsswb    macro   dst:req, src:req
        local   x, y
x:
        cmpxchg   src, dst
y:
        org     x+1
        byte    opc_Packsswb
        org     y
        endm

packuswb    macro   dst:req, src:req
        local   x, y
x:
        cmpxchg   src, dst
y:
        org     x+1
        byte    opc_Packuswb
        org     y
        endm

paddd    macro   dst:req, src:req
        local   x, y
x:
        cmpxchg   src, dst
y:
        org     x+1
        byte    opc_Paddd
```

```
        org     y
        endm

paddsb  macro   dst:req, src:req
        local   x, y
x:
        cmpxchg   src, dst
y:
        org     x+1
        byte    opc_Paddsb
        org     y
        endm

paddsw  macro   dst:req, src:req
        local   x, y
x:
        cmpxchg   src, dst
y:
        org     x+1
        byte    opc_Paddsw
        org     y
        endm

paddusb  macro   dst:req, src:req
        local   x, y
x:
        cmpxchg   src, dst
y:
        org     x+1
        byte    opc_Paddusb
        org     y
        endm

paddusw  macro   dst:req, src:req
        local   x, y
x:
        cmpxchg   src, dst
y:
        org     x+1
        byte    opc_Paddusw
        org     y
        endm

paddb   macro   dst:req, src:req
        local   x, y
x:
        cmpxchg   src, dst
y:
        org     x+1
```

```
        byte    opc_Paddb
        org     y
        endm

paddw   macro   dst:req, src:req
        local   x, y
x:
        cmpxchg  src, dst
y:
        org     x+1
        byte    opc_Paddw
        org     y
        endm

pand    macro   dst:req, src:req
        local   x, y
x:
        cmpxchg  src, dst
y:
        org     x+1
        byte    opc_Pand
        org     y
        endm

pandn   macro   dst:req, src:req
        local   x, y
x:
        cmpxchg  src, dst
y:
        org     x+1
        byte    opc_Pandn
        org     y
        endm

pcmpeqb   macro   dst:req, src:req
        local   x, y
x:
        cmpxchg  src, dst
y:
        org     x+1
        byte    opc_Pcmpeqb
        org     y
        endm

pcmpeqd   macro   dst:req, src:req
        local   x, y
x:
        cmpxchg  src, dst
y:
```

```
        org     x+1
        byte    opc_Pcmpeqd
        org     y
        endm

pcmpeqw   macro   dst:req, src:req
        local   x, y
x:
        cmpxchg   src, dst
y:
        org     x+1
        byte    opc_Pcmpeqw
        org     y
        endm

pcmpgtb   macro   dst:req, src:req
        local   x, y
x:
        cmpxchg   src, dst
y:
        org     x+1
        byte    opc_Pcmpgtb
        org     y
        endm

pcmpgtd   macro   dst:req, src:req
        local   x, y
x:
        cmpxchg   src, dst
y:
        org     x+1
        byte    opc_Pcmpgtd
        org     y
        endm

pcmpgtw   macro   dst:req, src:req
        local   x, y
x:
        cmpxchg   src, dst
y:
        org     x+1
        byte    opc_Pcmpgtw
        org     y
        endm

pmaddwd   macro   dst:req, src:req
        local   x, y
x:
        cmpxchg   src, dst
```

```
y:
      org    x+1
      byte    opc_Pmaddwd
      org    y
      endm

pmulhw    macro   dst:req, src:req
      local   x, y
x:
      cmpxchg   src, dst
y:
      org    x+1
      byte    opc_Pmulhw
      org    y
      endm

pmullw    macro   dst:req, src:req
      local   x, y
x:
      cmpxchg   src, dst
y:
      org    x+1
      byte    opc_Pmullw
      org    y
      endm

por    macro   dst:req, src:req
      local   x, y
x:
      cmpxchg   src, dst
y:
      org    x+1
      byte    opc_Por
      org    y
      endm


pslld    macro   dst:req, src:req
      local   x, y
IF (OPATTR(src)) AND 00000100y ; constant
x:
      btr   dst, src
y:
      org    x+1
      byte    opc_PSHimd
      org    y
ELSE
x:
      cmpxchg   src, dst
```

```
y:
        org     x+1
        byte    opc_Pslld
        org     y
ENDIF
        endm

psllw   macro   dst:req, src:req
        local   x, y
IF (OPATTR(src)) AND 00000100y ; constant
x:
        btr     dst, src
y:
        org     x+1
        byte    opc_PSHimw
        org     y
ELSE
x:
        cmpxchg   src, dst
y:
        org     x+1
        byte    opc_Psllw
        org     y
ENDIF
        endm

psrad   macro   dst:req, src:req
        local   x, y
IF (OPATTR(src)) AND 00000100y ; constant
x:
        bt      dst, src
y:
        org     x+1
        byte    opc_PSHimd
        org     y
ELSE
x:
        cmpxchg   src, dst
y:
        org     x+1
        byte    opc_Psrad
        org     y
ENDIF
        endm

psraw   macro   dst:req, src:req
        local   x, y
IF (OPATTR(src)) AND 00000100y ; constant
x:
```

```
        bt   dst, src
y:
        org    x+1
        byte    opc_PSHimw
        org    y
ELSE
x:
        cmpxchg   src, dst
y:
        org    x+1
        byte    opc_Psraw
        org    y
ENDIF
        endm

psrld   macro   dst:req, src:req
        local   x, y
IF (OPATTR(src)) AND 00000100y ; constant
x:
        cmpxchg dst,MM2
            byte       src
y:
        org    x+1
        byte    opc_PSHimd
        org    y
ELSE
x:
        cmpxchg   src, dst
y:
        org    x+1
        byte    opc_Psrld
        org    y
ENDIF
        endm

psrlq   macro   dst:req, src:req
        local   x, y
IF (OPATTR(src)) AND 00000100y ; constant
x:
        cmpxchg dst,MM2
            byte       src
y:
        org    x+1
        byte    opc_PSHimq
        org    y
ELSE
x:
        cmpxchg   src, dst
y:
```

```
        org     x+1
        byte    opc_Psrlq
        org     y
ENDIF
        endm


psllq   macro   dst:req, src:req
        local   x, y
IF (OPATTR(src)) AND 00000100y ; constant
x:
        btr     dst, src
y:
        org     x+1
        byte    opc_PSHimq
        org     y
ELSE
x:
        cmpxchg   src, dst
y:
        org     x+1
        byte    opc_Psllq
        org     y
ENDIF
        endm


psrlw   macro   dst:req, src:req
        local   x, y
IF (OPATTR(src)) AND 00000100y ; constant
x:
        cmpxchg dst,MM2
            byte        src
y:
        org     x+1
        byte    opc_PSHimw
        org     y
ELSE
x:
        cmpxchg   src, dst
y:
        org     x+1
        byte    opc_Psrlw
        org     y
ENDIF
        endm

psubsb  macro   dst:req, src:req
        local   x, y
```

```
x:
    cmpxchg  src, dst
y:
    org   x+1
    byte   opc_Psubsb
    org   y
    endm

psubsw   macro  dst:req, src:req
    local  x, y
x:
    cmpxchg  src, dst
y:
    org   x+1
    byte   opc_Psubsw
    org   y
    endm

psubusb   macro  dst:req, src:req
    local  x, y
x:
    cmpxchg  src, dst
y:
    org   x+1
    byte   opc_Psubusb
    org   y
    endm

psubusw   macro  dst:req, src:req
    local  x, y
x:
    cmpxchg  src, dst
y:
    org   x+1
    byte   opc_Psubusw
    org   y
    endm

psubb   macro  dst:req, src:req
    local  x, y
x:
    cmpxchg  src, dst
y:
    org   x+1
    byte   opc_Psubb
    org   y
    endm

psubw   macro  dst:req, src:req
```

```
        local   x, y
x:
        cmpxchg   src, dst
y:
        org     x+1
        byte    opc_Psubw
        org     y
        endm

punpcklbw    macro   dst:req, src:req
        local   x, y
x:
        cmpxchg   src, dst
y:
        org     x+1
        byte    opc_Punpcklbw
        org     y
        endm

punpckhdq    macro   dst:req, src:req
        local   x, y
x:
        cmpxchg   src, dst
y:
        org     x+1
        byte    opc_Punpckhdq
        org     y
        endm

punpcklwd    macro   dst:req, src:req
        local   x, y
x:
        cmpxchg   src, dst
y:
        org     x+1
        byte    opc_Punpcklwd
        org     y
        endm

punpckhbw    macro   dst:req, src:req
        local   x, y
x:
        cmpxchg   src, dst
y:
        org     x+1
        byte    opc_Punpckhbw
        org     y
        endm
```

```
punpckldq   macro   dst:req, src:req
      local   x, y
x:
      cmpxchg   src, dst
y:
      org     x+1
      byte    opc_Punpckldq
      org     y
      endm

punpckhwd   macro   dst:req, src:req
      local   x, y
x:
      cmpxchg   src, dst
y:
      org     x+1
      byte    opc_Punpckhwd
      org     y
      endm

pxor    macro   dst:req, src:req
      local   x, y
x:
      cmpxchg   src, dst
y:
      org     x+1
      byte    opc_Pxor
      org     y
      endm

psubd   macro   dst:req, src:req
      local   x, y
x:
      cmpxchg   src, dst
y:
      org     x+1
      byte    opc_Psubd
      org     y
      endm
;-----------------------------------------------------------------
```

3.1. Билинейная интерполяция

Во многих алгоритмах построение 3D-сцен сводится к следующему: поверхность объектов разбивается на множество треугольников, затем эти треугольники проектируются на 2D-поверхность экрана. Для повышения степени реалистичности образы треугольников копируются из текстур. Для этого каждому пикселу на экране ставится в соответствие точка на тек-

стуре. При таком отображении координаты точки (u, v) на поверхности текстуры могут быть не обязательно целыми. Далее для определения цвета точки можно просто округлить координаты (u, v), а можно провести интерполяцию цвета. В последнем случае изображение получиться более качественным.

Билинейная интерполяция использует простую формулу для вычисления цвета точки:

```
R_result =  R1 * (1 – dU) * (1 – dV) +
            R2 * (dU) * (1 – dV) +
            R3 * (1 – dU) * (dV) +
            R4 * (dU) * (dV)

G_result =  G1 * (1 – dU) * (1 – dV) +
            G2 * (dU) * (1 – dV) +
            G3 * (1 – dU) * (dV) +
            G4 * (dU) * (dV)

B_result =  B1 * (1 – dU) * (1 – dV) +
            B2 * (dU) * (1 – dV) +
            B3 * (1 – dU) * (dV) +
            B4 * (dU) * (dV)
```

В нашем примере процедуре MMx_BiLinearInterpolate передаются следующие параметры: указатель на текстуру (TextureMap); указатель на палитру цветов (ColorLookupTable); координаты точки (dwUVal, dwVVal); указатель на возвращаемое значение (lpRGBOut).

Текстура представляет собой двухмерный массив байтов, которые являются индексами палитры цветов. В нашем примере для упрощения вычислений текстура должна иметь длину строки 128 байт. Значения RGB в палитре цветов должны быть представлены в виде учетверенного слова, где значения RGB содержаться в старших восьми разрядах 16-разрядных полей. Координаты (u, v) представляются в виде двойных слов, где старшие 10 бит представляют собой целую часть, а младшие 22 бита – дробную.

Оптимизированный алгоритм билинейной интерполяции

```
;
; 'C' function prototype:
;
; void
; MMx_BiLinearInterpolate (BYTE    TextureMap[][128],
;                          RGBQWORD ColorLookupTable[],
;                          DWORD    dwUVal,
;                          DWORD    dwVVal,
```

```
;                   LPRGBQWORD  lpRGBOut)
;
; Abstract: this routine determines the bilinear interpolation of a RGB
;           value at (U, V) in a transformed space, given a pointer to the
;           orginal texture map, and a colorlookup table for that texture
;           map.
;
;         the texture map was converted for effeciency, from a
;         [58][72] byte array, to a [58][128] byte array with the
;         elements from [j][72] to [j][127] unused.
;
;         the color lookup table is a 256 element array, where each
;         element is a RGBQWORD (see below).
;
;         the U and V elements are formatted as 32-bit numbers where
;         the upper 10 bits are the integer portion, and the lower
;         22 bits are the fractional portion of the (U, V) point whose
;         color is being interpolated.
;
;         NOTE: the fractional delta values are scaled
;                 from 0 – 0xffff.
;         scale them down
;                 to 0 – 0x7fff
;         to avoid signed/unsigned multiply issues
;
;         the RGBQWORD is defined in 'C' as:
;
;         typedef struct {
;            WORD wB;
;            WORD wG;
;            WORD wR;
;            WORD wUnused;
;         } RGBQWORD, FAR * LPRGBQWORD;
;
;         the values in the ColorLookupTable have been pre-formatted so that the
RGB
;         values are in bits 7-15.
The intepolation can be depicted as:
;
;                       |             |           |
;                 RGB 1 |     RGB 2   |           |
;                  -----+-------------+-----------+----
;                       |           ^ |           |
;                       |           | |  delta    |
;                       |  ( U,V)   | |    V      |
;                       |           v |           |
;                       |<------>*    |           |
;                       |  delta      |           |
;                       |  U          |           |
```

```
;                  ----+-----------+----------+----
;                RGB 3|    RGB 4  |          |
;                     |           |          |
;                     |           |          |
```

R_result = R1 * (1 – delta U) * (1 – delta V) +
R2 * (delta U) * (1 – delta V) +
R3 * (1 – delta U) * (delta V) +
R4 * (delta U) * (delta V)

G_result = G1 * (1 – delta U) * (1 – delta V) +
G2 * (delta U) * (1 – delta V) +
G3 * (1 – delta U) * (delta V) +
G4 * (delta U) * (delta V)

B_result = B1 * (1 – delta U) * (1 – delta V) +
B2 * (delta U) * (1 – delta V) +
B3 * (1 – delta U) * (delta V) +
B4 * (delta U) * (delta V)

the computations are performed in 15 bits, with the multiplies resulting in 31/30-bits, with 16-bit precision being recovered at the final result. Note: the shifts are 'performed' by pmulh.

```
30 bits     15 bits          15 bits
result  =  (1 – delta U)  *  (1 – delta V)
```

then,

```
30 bits    15 bits
 result = result >> 15
```

then,

```
31 bits   16 bits   15 bits
 value  =  R1    *   result
```

then,

```
31 bits     16 bits
 value = value >> 15
```

NOTE: we leave the results in the lower 8 bits to simplify the pixel display by calling procedure.

```
;           Environment:
;           MASM 6.11D
;
;*****************************************************
;FUNCTION LIST:
;           (functions contained in this source module)
;
; MMX_BiLinearInterpolate() ;
;*****************************************************

.586
.MODEL FLAT, C

include iammx.inc

TAG STRUCT 2t
        wB                  WORD  ?
        wG                  WORD  ?
        wR                  WORD  ?
        wUnused             WORD  ?
TAG ENDS

RGBQWORD      TYPEDEF       TAG
LPRGBQWORD    TYPEDEF       FAR PTR TAG

.CODE
;-------------------------------
; location of local vars on the stack
SAVEESP             EQU DWORD PTR [esp+0]
RGB1                EQU DWORD PTR [esp+8]
RGB2                EQU DWORD PTR [esp+16]
RGB3                EQU DWORD PTR [esp+24]
RGB4                EQU DWORD PTR [esp+32]
QTEMP               EQU DWORD PTR [esp+40]
QTEMP1              EQU DWORD PTR [esp+48]
ENDOFLOCALS         EQU DWORD PTR [esp+56]

LocalFrameSize = 56
;*****************************************************
;
MMx_BiLinearInterpolate PROC PUBLIC,
          TextureMap          : PTR DWORD,
          ColorLookupTable    : PTR DWORD,
          dwUVal              : DWORD,
          dwVVal              : DWORD,
          lpRGBOut            : PTR DWORD

; push      ebp                   ; THE ASSEMBLER actually generates this
```

```
; mov     ebp, esp          ; THE ASSEMBLER actually generates this
sub     esp, LocalFrameSize

mov     SAVEESP, ebp      ; save original esp to restore in epilgue
push    edx

and     esp, 0fffffff8h   ; 8-byte align start of local stack frame
mov     eax, dwUVal       ; get dwUVal from parameter list

mov     edx, dwVVal       ; get dwVVal from parameter list
mov     esi, eax          ; save dwUVal

push    ebx
mov     ecx, edx          ; save dwVVal

shr     eax, 22           ; get integer portion of dwUVal
push    esi

push    ecx
and     esi, 3fffffH

shr     edx, 22           ; get integer portion of dwVVal
and     eax, 0000ffffH    ; eax = 0, U

shr     esi, 7            ; shift by 7 instead of 6 to provide the 0-0x7fff
scaling
and     ecx, 3fffffH

shr     ecx, 7            ; shift by 7 instead of 6 to provide the 0-0x7fff
scaling
mov     ebx, esi          ; ebx = 0, dU

shl     edx, 7            ; edx=V*128; 128 is hardcoded pitch value
push    edi

shl     ebx, 16           ; ebx = dU, 0
and     edx, 0000ffffH    ; edx = 0, V

or      ebx, esi          ; ebx = dU, dU
mov     edi, ecx          ; edi = 0, dV

shl     edi, 16           ; edi = dV, 0
mov     QTEMP, ebx        ;

and     esi, 0000ffffH    ; esi = 0, dU
or      edi, ecx          ; edi = dV, dV

and     ecx, 0000ffffH
mov     QTEMP+4, ebx
```

```
mov     QTEMP1, edi
mov     QTEMP1+4, edi

movq    mm4, QTEMP      ; mm4 = dU | dU || dU | dU

movq    mm5, QTEMP1     ; mm5 = dV | dV || dV | dV
movq    mm1, mm4        ; mm1 = dU | dU || dU | dU

mov     edi, 7fffH
mov     ebx, 7fffH

sub     edi, ecx        ; edi = 0, 1-dV
sub     ebx, esi        ; ebx = 0, 1-dU

mov     ecx, edi        ; ecx = 0, 1-dV
mov     esi, ebx        ; esi = 0, 1-dU

shl     ecx, 16         ; ecx = 1-dV, 0
add     edx, eax        ; edx = U+V*128

shl     esi, 16         ; esi = 1-dU, 0
add     edx, TextureMap ; edx = & TextureMap[U+V*128]

or      edi, ecx        ; edx = 1-dV, 1-dV
or      ebx, esi        ; ebx = 1-dU, 1-dU

mov     QTEMP, edi      ;
mov     cl, BYTE PTR [ edx]    ; bl = TextureMap[U+V*128]

mov     eax, ColorLookupTable          ; & ColorLookupTable[0]
and     ecx, 000000ffH

mov     QTEMP1, ebx     ;
mov     QTEMP1+4, ebx   ;

movq    mm6, QTEMP1     ; mm6 = 1-dU | 1-dU || 1-dU | 1-dU
movq    mm3, mm4        ; mm3 = dU | dU || dU | dU

mov     QTEMP+4, edi    ;
lea     ecx, [eax+ecx*8] ; ebx = & ColorLookupTable[ bl]

movq    mm7, QTEMP      ; mm7 = 1-dV | 1-dV || 1-dV | 1-dV
movq    mm0, mm6        ; mm0 = 1-dU | 1-dU || 1-dU | 1-dU

pmulhw  mm1, mm7        ; compute ( dU) * (1-dV)
movq    mm2, mm6        ; mm2 = 1-dU | 1-dU || 1-dU | 1-dU

pmulhw  mm0, mm7        ; compute (1 – dU) * (1 – dV)
```

```
mov     bl, BYTE PTR [edx+1]  ; bl = TextureMap[U+V*128+1]

pmulhw  mm2, mm5        ; compute (1-dU) * ( dV)
and     ebx, 000000ffh

pmulhw  mm3, mm5        ; compute ( dU) * ( dV)
pop     edi             ;

movq    mm4, mm0        ; mm4 = (1-dU)(1-dV)
movq    mm5, mm1        ; mm5 = ( dU)(1-dV)

movq    mm0, [ ecx]     ; mm0 = xxxx | R1 || G1 | B1
movq    mm6, mm2        ; mm6 = (1-dU)( dV)

lea     ebx, [eax+ebx*8]  ; ebx = & ColorLookupTable[ bl]
mov     cl, BYTE PTR [edx+128]          ; bl = TextureMap[U+V * 128+128]
;
; NOTE:  we leave the results in the lower 8 bits to simplify
;        the pixel dis play by calling procedure.
;
pmulhw  mm0, mm4        ; mm0 = xxxx | R1(1-dU)(1-dV) || G1(1-dU)(1-dV) |
                        ; B1(1-dU)(1-dV)
and     ecx, 000000ffH

movq    mm1, [ebx]      ; mm1 = xxxx | R2 ||  G2 | B2
movq    mm7, mm3        ; mm7 = ( dU)( dV)

pmulhw  mm1, mm5        ; mm1 = xxxx | R2( dU)(1-dV) || G2( dU)(1-dV) |
        ; B2( dU)(1-dV)
lea     ecx, [eax+ecx*8]  ; ebx = & ColorLookupTable[ bl]

pop     esi
mov     bl, BYTE PTR [edx+128+1]        ; bl = TextureMap[U+V*128+128+1]

movq    mm2, [ecx]      ; mm2 = xxxx | R3 ||  G3 | B3
psrlw   mm0, 5          ;

pmulhw  mm2, mm6        ; mm2 = xxxx | R3(1-dU)( dV) ||
                        ; G3(1-dU)( dV) | B3(1-dU)( dV)
and     ebx, 000000ffH

lea     ebx, [eax+ebx*8]  ; ebx = & ColorLookupTable[ bl]
psrlw   mm1, 5          ;

movq    mm3, [ebx]      ; mm3 = xxxx | R4 || G4 | B4
paddw   mm0, mm1

pmulhw  mm3, mm7    ; mm3 = xxxx | R4( dU)( dV) || G4( dU)( dV) | B4( dU)( dV)
psrlw   mm2, 5          ;
```

```
pop         edx                 ;
pop         ecx                 ;

paddw       mm2, mm0
psrlw       mm3, 5              ;

mov         eax, lpRGBOut
paddw       mm2, mm3

movq        [eax], mm2

emms        ; clear the FP stack

pop         ebx                 ;
; leave     ; THE ASSEMBLER actually generates this

ret         0                   ;

MMX_BiLinearInterpolate ENDP
```

3.2. 3D-геометрия

Преобразования в 3D-пространстве связаны с перемножением матриц. Сначала строится матрица желаемого преобразования [M]. Для построения матрицы [M] желаемое преобразование представляется в виде суперпозиции простых преобразований, матрицы которых известны:

M = T x S x H x Rx x Ry x Rz

Сложные объекты в 3D-пространстве моделируются из многоугольников (обычно треугольников). Для осуществления желаемого преобразования нужно применить матрицу [M] ко всем вершинам многоугольников, из которых состоит объект. Точка в 3D-пространстве представляется в виде вектора $[V] = [x, y, z, 1]^T$.

Translation (T) Перенос

$$[T] = \begin{pmatrix} 1 & 0 & 0 & Tx \\ 0 & 1 & 0 & Ty \\ 0 & 0 & 1 & Tz \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Ti – перенос по оси i

Scale (S) Масштабирование

$$[S] = \begin{pmatrix} Sx & 0 & 0 & 0 \\ 0 & Sy & 0 & 0 \\ 0 & 0 & Sz & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Si – коэффициент растяжения (сжатия) вдоль оси i

Shear (H)
Линейно зависимый перенос

$$[H] = \begin{pmatrix} 1 & Hxy & Hxz & 0 \\ Hyx & 1 & Hyz & 0 \\ Hzx & Hzy & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Hij – коэффициент зависимости переноса вдоль оси i от координаты j

Rotation about x (Rx)
Вращение вокруг оси x

$$[Rx] = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & Cx & Ix & 0 \\ 0 & -Ix & Cx & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Cx = cos(угол)
Ix = sin(угол)

Rotation about y (Ry)
Вращение вокруг оси y

$$[Ry] = \begin{pmatrix} Cy & 0 & Iy & 0 \\ 0 & 1 & 0 & 0 \\ -Iy & 0 & Cy & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Cy = cos(угол)
Iy = sin(угол)

Rotation about z (Rz)
Вращение вокруг оси z

$$[Rz] = \begin{pmatrix} Cz & Iz & 0 & 0 \\ -Iz & Cz & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Cz = cos(угол)
Iz = sin(угол)

Требуемое преобразование будет иметь вид:

$Vt_i = M \times Vo_i$

где Voi – исходный вектор (original); Vti – трансформированный вектор (transformed),

$0 < i < N$.

Так как число вершин многоугольников (N) у сложного объекта может быть достаточно большим (чем больше вершин, тем более реалистично выглядит объект), а перемножение матриц требует много времени, большое значение будет иметь то, сколько времени тратится на преобразование одной точки. Далее предлагается оптимизированный алгоритм перемножения матрицы [M] на вектор [Vo].

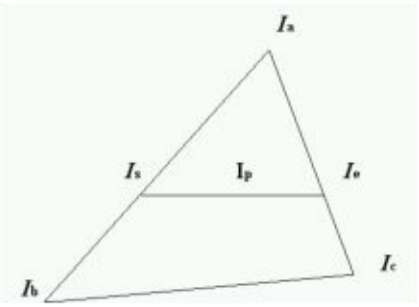
Процедуре M x V через стек передаются следующие параметры: указатель на матрицу [M], указатель на массив векторов [Vo], количество векторов, указатель на массив векторов [Vt]. Размер элементов – 16 бит.

Оптимизированный алгоритм умножения матрицы 4 x 4 на матрицу 4 x 1 (вектор)

```
INCLUDE iammx.inc          ; to parse MMX instructions
.486P                      ; enable all 486 instructions
.model FLAT                ; single 32-bit flat memory segment
.CODE

_MxV_mmx PROC PUBLIC ; MSVC linker mangles MxV_mmx
                     ; to _MxV_mmx

;C requires ebp,esi,edi preserved. This routine doesn't use them.
;get pointers to parameters
mov     eax,[esp]+ 4       ; eax = ptr to matrix
mov     ebx,[esp]+ 8       ; ebx = ptr to vector
mov     ecx,[esp]+12       ; ecx = numvec
mov     edx,[esp]+16       ; edx = ptr to result

;Load entire 3x4 matrix
movq    mm0, 0[eax]        ; Matrix row 0 (4 16-bit elements)
movq    mm1, 8[eax]        ; Matrix row 1 (4 16-bit elements)
movq    mm2,16[eax]        ; Matrix row 2 (4 16-bit elements)

NextVect:
movq        mm3,[ebx]      ; Load vector (4 16-bit elements)

movq        mm4,mm3        ; copy to other regs for use by 3 pmadds
pmaddwd     mm3,mm0        ; multiply row0 X vector

movq        mm5,mm4
pmaddwd     mm4,mm1        ; multiply row1 X vector

pmaddwd     mm5,mm2        ; multiply row2 X vector
add         ebx,8          ; increment to next input vector

movq        mm6,mm3        ; add row0 high and low order
                           ; 32-bit results
psrlq       mm3,32         ;

paddd       mm3,mm6        ;
movq        mm6,mm4        ; add row1 high and low order
                           ; 32-bit results

psrlq       mm4,32         ;
add         edx,8          ; increment (in advance) to next result

psrad       mm3,15-2       ; Shift 32 to 16
                           ; also app. specific <<2
paddd       mm4,mm6        ;
```

```
psrad       mm4,15-2        ; Shift 32 to 16
                            ; also app. specific <<2
movq        mm6,mm5         ; add row2 high and low order
                            ; 32-bit results
psrlq       mm5,32          ;

punpcklwd   mm3,mm4         ; Copy word0 of mm4 into
                            ; word0 of mm3
paddd   mm5,mm6             ;

psrad   mm5,15-2            ; Shift 32 to 16
                            ; also app. specific <<2

movd    [edx]-8+0,mm3       ; Store 1st and 2nd elements,
                            ; one 32-bit write

movd    [edx]-8+4,mm5       ; Store 3rd (and unused 4th)
                            ; 16-bit element

dec     ecx                 ; if not counted through all the vectors
                            ; jnz NextVect then loop back to do the
                            ; next one.

emms                        ; end MMX stae (remove fp reg valids)

ret 0

_MxV_mmx ENDP
END
```

3.3. Закраска поверхности



В задачах 3D-моделирования одним из этапов построения изображения является закраска поверхности, исходя из освещенности объекта. Проведение полного расчета интенсивности каждого пиксела может занять очень много времени. Рассмотрим упрощенный метод закраски поверхности. Определим интенсивность в вершинах грани (треугольника), затем используем билинейную интерполяцию для определения интенсивности каждого пиксела внутри грани.

Первый этап:

$Is = v * Ia + (1 - v) * Ib \qquad 0 <= v <= 1$
$Ie = t * Ia + (1 - t) * Ic \qquad 0 <= t <= 1$

Второй этап:

$Ip = u * Is + (1 - u) * Ie \qquad 0 <= u <= 1$

Далее предлагается реализация второго этапа. Так как операция сложения выполняется быстрее, чем умножение, перейдем к другой формуле:

$Ip(i + 1) = Ip(i) + dI \qquad i = 1, 2, 3, ..., n - 1$
$dI = (Ie - Is)/(n - 1)$

```
;*************************************************************
;Procedure Scan_In_RGB_MMX ;
; Description:
; Procedure Scan_In_RGB_MMX implements a RGB
; intensities ;interpolation on a scan line, which
; is a major part of ;Gouraud shading algorithm.
; The function of Scan_In_RGB_MMx ;is, knowing the
; RGB intensity of the starting pixel of a ;scan
; line and the incremental R,G,B values,
; calculating ;the RGB intensity for each pixel on
; the scan line by ;linear interpolation.
; INPUTS:

; Rgb1(DWORD): RGB intensity of the starting pixel on the scan line. Color values
;              of R1,G1,B1 (8 bits each) are stored in first 24 bits.
; dR(SWORD):   Incremental intensity of R for each successive pixel. dR is a 16-bit
;              fixed point notation with the first 8 bits representing the fractional
part
;              and the remaining 8 bits the integer part. In the procedure,
;              accumulation of dR includes both nteger part and fractional parts,
but
;              only the integer part is used as when assigning RGB intensity
;              to a pixel.

; dG(SWORD):   refer to dR
; dB(SWORD):   refer to dR
; NumP(WORD):  number of pixels on the scan line.
;
; OUTPUTS:
; Rgbs(PTR DWORD):  RGB intensities of all pixels on the scan line. Intensities
;                   R,G,B(each 8 bits)are stored in first 24 bits in a DWORD.
In
;                   the procedure, R,G,B are calculated by linear interpolation,
eg.
```

```
R(i) = R1 + i*dR,
G(i) = G1 + i*dG,
B(i) = B1 + i*dB.
i = 0, ..., NumP – 1.

; Assumption:      Input 'NumP' is greater than 3.
;                  The case that NumP is fewer than 4, is handled in C code.

;*************************************************
.486P
.MODEL FLAT, C include iammx.inc
.Data
.Code
Public Scan_In_RGB_MMX

Scan_In_RGB_MMX Proc C Public USES eax ebx ecx edi,
Rgb1: DWORD,              ; RGB intensities of the start pixel.
Rgbs: PTR DWORD,          ; output, RGB intensities of all pixels.

dR:       SWORD,          ; increment of red intensity.
dG:       SWORD,          ; increment of green intensity.
dB1:      SWORD,          ; increment of blue intensity.
NumP:     WORD            ; number of pixels on the scan line.

;Declare local variables:
LOCAL dRed[4]: SWORD      ; reserve 4 words for holding
                          ; increment of red.
LOCAL dGreen[4]: SWORD    ; reserve 4 words for holding
                          ; increment of green.
LOCAL dBlue[4]: SWORD     ; reserve 4 words for holding
                          ; increment of blue.
LOCAL factor: QWORD       ; reserve 4 words for holding
                          ; 4 factor values

; load dR to 4 words of mm3,
; load dG to 4 words of mm4,
; load dB to 4 words of mm5
; Pack R1,G1,B1 from input Rgb1
; Load R1, R1+dR, R1+2*dR, R1+3*dR as 4 words in mm2
; Load G1, G1+dG, G1+2*dG, G1+3*dG as 4 words in mm1
; Load B1, B1+dB, B1+2*dB, B1+3*dB as 4 words in mm0

movdt     mm3, [ebp + 16]  ; load dR in the lowest
                           ; word in mm3
mm3       pxor mm2,mm2     ; clear mm2

movdt     mm4, [ebp + 20]  ; load dG in the lowest
                           ; word in mm4
```

```
punpcklwd   mm3,mm3         ; unpack dR to lower 2
                            ; words in mm3

movdt       mm5, [ebp + 24] ; load dB in the lowest
                            ; word in mm5
punpcklwd   mm4,mm4         ; unpack dG to lower 2
                            ; words in mm4

lea         ecx, factor     ; Load offset address of dRed.
Punpckldq   mm3,mm3         ; unpack dR to 4 words in mm3

mov         eax, 010000h
punpcklwd   mm5,mm5         ; unpack dB to lower 2
                            ; words in mm5

movdt       mm6, Rgb1       ; load R1,G1,B1 in lower three bytes to mm6
punpckldq   mm4,mm4         ; unpack dG to 4 words in mm4

mov         [ecx], eax      ; store values
                            ; 0,1(each 2 bytes) in
                            ; memory location 'ecx'

punpcklbw   mm6,mm6         ; unpack R1R1,G1G1,B1B1 to the first 3
                            ; words of mm6.
mov         eax, 030002h
punpckldq   mm5,mm5         ; unpack dB to 4 words in mm5

movq        mm7, mm6        ; load R1R1,G1G1,B1B1 to the first
                            ; three words of mm7.

pxor        mm1,mm1         ; clear mm2

mov         [ecx + 4], eax  ; store values
                            ; 2,3(each 2 bytes) in
                            ; memory location 'ecx + 4'

punpcklwd   mm7,mm7         ; unpack R1R1,R1R1,G1G1,G1G1 to 4
                            ; words of mm7.

movq        mm0, mm3        ;unpack R1 to the higher bytes of the
                            ; 4 words of mm7

punpcklbw   mm2,mm7         ;unpack R1 to the higher bytes of
                            ; the 4 words of mm2

pmullw      mm0, [ecx]      ; multiplied by [3,2,1,0], mm0
                            ; becomes [3dR,2dR,dR,0]

punpckhbw   mm1,mm7         ;unpack G1 to the higher bytes of
```

```
                                ; the 4 words of mm1

lea         edi, dRed           ; Load offset address of dRed.
psllw       mm3, 2              ; mulitiply each dR in 4
                                ; words of mm3 by 4

paddsw      mm2, mm0            ; 4 words in mm2
                                ; becomes R1, R1+dR,
                                ; R1+2dR, R1+3dR

punpckhbw   mm6,mm6             ; unpack B1 to the
                                ; first 4 bytes of mm6

movq        [edi], mm3          ; store 4 dR to consecutive memory
                                ; location: dRed[4]

movq        mm0, mm4            ; load dG to 4 words of mm0

pmullw      mm0, [ecx]          ; multiplied
                                ; by [3,2,1,0], mm0 becomes
                                ; [3dG,2dG,dG,0]

psllw       mm4, 2              ; mulitiply each dG in 4
                                ;words of mm4 by 4

mov         ebx, Rgbs           ; Load address
                                ; of *Rgbs for storing
                                ; intensities movq mm7, mm5
                                ; load dB to 4 words of mm7

mov         eax, 0ff00ff00h
paddsw      mm1, mm0            ; 4 words in mm2
                                ; becomes G1, G1+dG,
                                ; G1+2dG, G1+3dG

pmullw      mm7, [ecx]          ; multiplied by [3,2,1,0], mm7
                                ; becomes [3dB,2dB,dB,0]

pxor        mm0,mm0             ; clear mm0

movq        [edi – 8], mm4      ; store 4 dG to
                                ; consecutive memory
                                ; location: dGreen[4]

psllw       mm5, 2              ; mulitiply each dB in 4
                                ; words of mm5 by 4

xor         ecx, ecx            ; clear ecx as a counter of j
punpcklbw   mm0,mm6             ; unpack B1 to the
```

```
                                ; higher bytes of the
                                ; 4 words of mm0

movq        [edi – 16], mm5     ; store 4 dB to consecutive
                                ; memory location: dBlue[4]

paddsw      mm0, mm7            ; 4 words in mm0
                                ; becomes B1, B1+dB,
                                ; B1+2dB, B1+3dB

movdt       mm7, eax            ; load 0ff00ff00h to the
                                ; first dword of mm7
movq        mm3,mm2             ; load R1,R2,R3,R4 to 4 words of mm3

xor         eax, eax            ; clear eax
movq        mm4,mm1             ; load G1,G2,G3,G4 to 4 words of mm4

mov         ax, numP            ; load numP to eax
punpckldq   mm7,mm7             ; mm7(0ff00ff00ff00ff00h)
                                ; will serve as
                                ; a mask in J loop
;For each pass through of loopJ, RGB intensities of four
;pixels will be calculated and stored to the destination
;memory location of Rgbs[]. If numP is not a multitple of
;4, there will be up to 3 extra intensities assignment in
;the last pass of the loop. Caller should allocate
;enough ;memory for holding the extra intensities.

loopJ:

pand        mm4, mm7            ; clear the fractional part of
                                ; G1,G2,G3,G4 in mm4

psrlw       mm3, 8              ; shift the integer parts of R1,R2,R3,R4
                                ; to lower byte

movq        mm5,mm0             ; copy B1,B2,B3,B4 into
                                ; 4 words of mm5

por         mm3,mm4             ; combine G,R and load G1R1, G2R2,
                                ; G3R3, G4R4 to 4 ;words of mm3

psrlw       mm5, 8              ; shift the integer parts of
                                ; B1,B2,B3,B4 to lower bytes
movq        mm6,mm3             ; copy G1R1,G2R2,G3R3,G4R4 to mm6

paddw       mm2, [edi]          ; add (4dR,4dR,4dR,4dR) to
                                ; (R1,R2,R3,R4) in mm2.
punpcklwd   mm3,mm5             ; unpack B1G1R1,B2G2R2
```

```
                                    ; to 2 dwords of mm3

paddw       mm1, [edi – 8]          ; add (4dG,4dG,4dG,4dG) to  (G1,G2,G3,G4) in
mm1

punpckhwd   mm6,mm5                 ; unpack B3G3R3, B4G4R4 to 2 dwords of mm6

paddw       mm0, [edi – 16]         ; add (4dB,4dB,4dB,4dB) to (B1,B2,B3,B4) in mm0

movq        mm4,mm1                 ; copy G1,G2,G3,G4 into 4 words of mm4

movq        [ebx +4*ecx],mm3        ; store two B1G1R1, B2G2R2 to memory location

movq        mm3,mm2                 ; copy R1,R2,R3,R4 into 4 words of mm3

movq        8 [ebx +4 * ecx], mm6   ; store B3G3R3,  B4G4R4 to memory location

add         ecx, 4                  ; increase the J-counter by 4

cmp         ecx, eax                ; compare counter ecx with num_Pjl
                                    ; if ecx < numP,
                                    ; continue for next four pixels

emms                                ; clear floating point stack

ret
Scan_In_RGB_MMX EndP
END
```

3.4. Преобразование цветового пространства RGB в цветовое пространство YUV

Для изображения точки на экране монитора используется формат RGB. Каждая компонента задает интенсивность красного (R), зеленого (G) и синего (B) цветов. Комбинация этих трех компонент определяет яркость точки, а также ее цвет. Изменение любого из трех значений скажется одновременно на яркости и на цвете точки. Это бывает не всегда удобно.

В цветовом пространстве YUV цвет точки определяется компонентами U и V, а яркость – компонентой Y. Таким образом, становится возможным изменять цвет и яркость точки независимо друг от друга. Такая возможность оказывается полезной для алгоритмов сжатия с потерями информации. Так как глаз человека более чувствителен к изменению яркости, чем к изменению цвета, при сжатии с потерями информации можно сохранить больше данных о яркости (Y), чем о цвете (UV). Например, если на каждую выборку цвета делать 4 выборки данных о яркости

(YUV411), то размер изображения уменьшится в два раза еще до алгоритма сжатия. Величина потерь выбирается исходя из желаемого качества изображения.

Уравнения преобразования:

Y = 0.299R 0.587G + 0.114B
U = -0.146 R – 0.288 G + 0.434 B
V = 0.617 R – 0.517 G – 0.100 G

Для того чтобы можно было применить ММХ-технологию, произведем масштабирование. Умножим коэффициенты на 32768:

Y = [(9798 R + 19235G + 3736 B)/32768]
U = [(-4784 R – 9437 G + 4221 B)/32768] + 128
V = [(20218R – 16941G – 3277 B)/32768] + 128

Процедуре _rgbtoyuv через стек передаются следующие параметры: адрес RGB-буфера, количество строк, количество столбцов, адрес Y-буфера, адрес U-буфера, адрес V-буфера. Компоненты в RGB-буфере чередуются. Размер RGB-буфера равен (количество строк) x (количество столбцов) x 3. Компоненты Y, U и V записываются в разные буферы. Размер каждого буфера для компонентов YUV равен (количество строк) x (количество столбцов).

Оптимизированный алгоритм преобразования RGB в YUV

```
;rgbtoyuv.asm
;The loop processes interleaved RGB values for 8 pixels.
;The notation in the comments which describe the data locate
;the first byte on the right. For example in a register containing
;G2R2B1G1R1B0G0R0, R0 is in the position of the
;lease significant
;byte and G2 is in the position of the most significant byte.
;The output is to separate Y, U and V buffers. Both input and
;output data are bytes.

TITLE rgbtoyuv
.486P
.model FLAT

PUBLIC _rgbtoyuv

_DATA SEGMENT

ALIGN       8
ZEROSX      dw 0,0,0,0
ZEROS       dd ?,?
OFFSETDX    dw 0,64,0,64 ;offset used before shift
```

```
OFFSETD         dd ?,?
OFFSETWX        dw 128,0,128,0 ;offset used before pack 32
OFFSETW         dd ?,?
OFFSETBX        dw 128,128,128,128
OFFSETB         dd ?,?

TEMP0       dd ?,?
TEMPY       dd ?,?
TEMPU       dd ?,?
TEMPV       dd ?,?

YR0GRX          dw 9798,19235,0,9798
YBG0BX          dw 3736,0,19235,3736
YR0GR       dd ?,?
YBG0B       dd ?,?
UR0GRX          dw -4784,-9437,0,-4784
UBG0BX          dw 14221,0,-9437,14221
UR0GR       dd ?,?
UBG0B       dd ?,?
VR0GRX          dw 20218,-16941,0,20218
VBG0BX          dw -3277,0,-16941,-3277
VR0GR       dd ?,?
VBG0B       dd ?,?

_DATA ENDS


_TEXT SEGMENT

_inPtr$ = 8
_rows$ = 12
_columns$ = 16
_outyPtr$ = 20
_outuPtr$ = 24
_outvPtr$ = 28

_rgbtoyuv PROC NEAR

push    ebp
mov     ebp, esp
push    eax
push    ebx
push    ecx
push    edx
push    esi
push    edi

lea         eax, ZEROSX     ;This section gets around a bug
movq        mm0, [eax]      ;unlikely to persist
```

```
movq    ZEROS, mm0
lea     eax, OFFSETDX
movq    mm0, [eax]
movq    OFFSETD, mm0
lea     eax, OFFSETWX
movq    mm0, [eax]
movq    OFFSETW, mm0
lea     eax, OFFSETBX
movq    mm0, [eax]
movq    OFFSETB, mm0
lea     eax, YR0GRX
movq    mm0, [eax]
movq    YR0GR, mm0
lea     eax, YBG0BX
movq    mm0, [eax]
movq    YBG0B, mm0
lea     eax, UR0GRX
movq    mm0, [eax]
movq    UR0GR, mm0
lea     eax, UBG0BX
movq    mm0, [eax]
movq    UBG0B, mm0
lea     eax, VR0GRX
movq    mm0, [eax]
movq    VR0GR, mm0
lea     eax, VBG0BX
movq    mm0, [eax]
movq    VBG0B, mm0
mov     eax, _rows$[ebp]
mov     ebx, _columns$[ebp]
mul     ebx             ;number pixels
shr     eax, 3          ;number of loops
mov     edi, eax        ;loop counter in edi
mov     eax, _inPtr$[ebp]
mov     ebx, _outyPtr$[ebp]
mov     ecx, _outuPtr$[ebp]
mov     edx, _outvPtr$[ebp]
sub     edx, 8          ;incremented before write

RGBtoYUV:
movq    mm1, [eax]      ;load G2R2B1G1R1B0G0R0
pxor    mm6, mm6        ;0 -> mm6

movq    mm0, mm1        ;G2R2B1G1R1B0G0R0 -> mm0
psrlq   mm1, 16         ;00G2R2B1G1R1B0-> mm1

punpcklbw mm0, ZEROS    ;R1B0G0R0 -> mm0
movq    mm7, mm1        ;00G2R2B1G1R1B0-> mm7
```

```
punpcklbw   mm1, ZEROS      ;B1G1R1B0 -> mm1
movq        mm2, mm0        ;R1B0G0R0 -> mm2

pmaddwd     mm0, YR0GR      ;yrR1,ygG0+yrR0 -> mm0
movq        mm3, mm1        ;B1G1R1B0 -> mm3

pmaddwd     mm1, YBG0B      ;ybB1+ygG1,ybB0 -> mm1
movq        mm4, mm2        ;R1B0G0R0 -> mm4

pmaddwd     mm2, UR0GR      ;urR1,ugG0+urR0 -> mm2
movq        mm5, mm3        ;B1G1R1B0 -> mm5

pmaddwd     mm3, UBG0B      ;ubB1+ugG1,ubB0 -> mm3
punpckhbw   mm7, mm6        ;00G2R2 -> mm7

pmaddwd     mm4, VR0GR      ;vrR1,vgG0+vrR0 -> mm4
paddd       mm0, mm1        ;Y1Y0 -> mm0

pmaddwd      mm5, VBG0B     ;vbB1+vgG1,vbB0 -> mm5

movq        mm1, 8[eax]     ;R5B4G4R4B3G3R3B2 -> mm1
paddd       mm2, mm3        ;U1U0 -> mm2

movq        mm6, mm1        ;R5B4G4R4B3G3R3B2 -> mm6

punpcklbw   mm1, ZEROS      ;B3G3R3B2 -> mm1
paddd       mm4, mm5        ;V1V0 -> mm4

movq        mm5, mm1        ;B3G3R3B2 -> mm5
psllq       mm1, 32         ;R3B200 -> mm1

paddd       mm1, mm7        ;R3B200+00G2R2=R3B2G2R2->mm1

punpckhbw   mm6, ZEROS      ;R5B4G4R3 -> mm6
movq        mm3, mm1        ;R3B2G2R2 -> mm3

pmaddwd     mm1, YR0GR      ;yrR3,ygG2+yrR2 -> mm1
movq        mm7, mm5        ;B3G3R3B2 -> mm7

pmaddwd     mm5, YBG0B      ;ybB3+ygG3,ybB2 -> mm5
psrad       mm0, 15         ;32-bit scaled Y1Y0 -> mm0

movq        TEMP0, mm6      ;R5B4G4R4 -> TEMP0
movq        mm6, mm3        ;R3B2G2R2 -> mm6

pmaddwd     mm6, UR0GR      ;urR3,ugG2+urR2 -> mm6
psrad       mm2, 15         ;32-bit scaled U1U0 -> mm2

paddd       mm1, mm5        ;Y3Y2 -> mm1
```

```
movq    mm5, mm7        ;B3G3R3B2 -> mm5

pmaddwd mm7, UBG0B      ;ubB3+ugG3,ubB2
psrad   mm1, 15         ;32-bit scaled Y3Y2 -> mm1

pmaddwd mm3, VR0GR      ;vrR3,vgG2+vgR2
packssdw mm0, mm1       ;Y3Y2Y1Y0 -> mm0

pmaddwd mm5, VBG0B      ;vbB3+vgG3,vbB2 -> mm5
psrad   mm4, 15         ;32-bit scaled V1V0 -> mm4

movq    mm1, 16[eax]    ;B7G7R7B6G6R6B5G5 -> mm7
paddd   mm6, mm7        ;U3U2 -> mm6

movq    mm7, mm1        ;B7G7R7B6G6R6B5G5 -> mm1
psrad   mm6, 15         ;32-bit scaled U3U2 -> mm6

paddd   mm3, mm5        ;V3V2 -> mm3
psllq   mm7, 16         ;R7B6G6R6B5G500 -> mm7

movq    mm5, mm7        ;R7B6G6R6B5G500 -> mm5
psrad   mm3, 15         ;32-bit scaled V3V2 -> mm3

movq    TEMPY, mm0      ;32-bit scaled Y3Y2Y1Y0 -> TEMPY
packssdw mm2, mm6       ;32-bit scaled U3U2U1U0 -> mm2

movq    mm0, TEMP0      ;R5B4G4R4 -> mm0

punpcklbw mm7, ZEROS    ;B5G500 -> mm7
movq    mm6, mm0        ;R5B4G4R4 -> mm6

movq    TEMPU, mm2      ;32-bit scaled U3U2U1U0 -> TEMPU
psrlq   mm0, 32         ;00R5B4 -> mm0

paddw   mm7, mm0        ;B5G5R5B4 -> mm7
movq    mm2, mm6        ;B5B4G4R4 -> mm2

pmaddwd mm2, YR0GR      ;yrR5,ygG4+yrR4 -> mm2
movq    mm0, mm7        ;B5G5R5B4 -> mm0

pmaddwd mm7, YBG0B      ;ybB5+ygG5,ybB4 -> mm7
packssdw mm4, mm3       ;32-bit scaled V3V2V1V0 -> mm4

add     eax, 24         ;increment RGB count
add     edx, 8          ;increment V count

movq    TEMPV, mm4      ;(V3V2V1V0)/256 -> mm4
movq    mm4, mm6        ;B5B4G4R4 -> mm4
```

```
pmaddwd   mm6, UR0GR      ;urR5,ugG4+urR4
movq      mm3, mm0        ;B5G5R5B4 -> mm0

pmaddwd   mm0, UBG0B      ;ubB5+ugG5,ubB4
paddd     mm2, mm7        ;Y5Y4 -> mm2

pmaddwd   mm4, VR0GR      ;vrR5,vgG4+vrR4 -> mm4
pxor      mm7, mm7        ;0 -> mm7

pmaddwd   mm3, VBG0B      ;vbB5+vgG5,vbB4 -> mm3
punpckhbw mm1, mm7        ;B7G7R7B6 -> mm1

paddd     mm0, mm6        ;U5U4 -> mm0
movq      mm6, mm1        ;B7G7R7B6 -> mm6

pmaddwd   mm6, YBG0B      ;ybB7+ygG7,ybB6 -> mm6
punpckhbw mm5, mm7        ;R7B6G6R6 -> mm5

movq      mm7, mm5        ;R7B6G6R6 -> mm7
paddd     mm3, mm4        ;V5V4 -> mm3

pmaddwd   mm5, YR0GR      ;yrR7,ygG6+yrR6 -> mm5
movq      mm4, mm1        ;B7G7R7B6 -> mm4

pmaddwd   mm4, UBG0B      ;ubB7+ugG7,ubB6 -> mm4
psrad     mm0, 15         ;32-bit scaled U5U4 -> mm0

paddd     mm0, OFFSETW    ;add offset to U5U4 -> mm0
psrad     mm2, 15         ;32-bit scaled Y5Y4 -> mm2

paddd     mm6, mm5        ;Y7Y6 -> mm6
movq      mm5, mm7        ;R7B6G6R6 -> mm5

pmaddwd   mm7, UR0GR      ;urR7,ugG6+ugR6 -> mm7
psrad     mm3, 15         ;32-bit scaled V5V4 -> mm3

pmaddwd   mm1, VBG0B      ;vbB7+vgG7,vbB6 -> mm1
psrad     mm6, 15         ;32-bit scaled Y7Y6 -> mm6

paddd     mm4, OFFSETD    ;add offset to U7U6
packssdw  mm2, mm6        ;Y7Y6Y5Y4 -> mm2

pmaddwd   mm5, VR0GR      ;vrR7,vgG6+vrR6 -> mm5
paddd     mm7, mm4        ;U7U6 -> mm7

psrad     mm7, 15         ;32-bit scaled U7U6 -> mm7

movq      mm6, TEMPY      ;32-bit scaled Y3Y2Y1Y0 -> mm6
packssdw  mm0, mm7         ;32-bit scaled U7U6U5U4 -> mm0
```

```
movq      mm4, TEMPU      ;32-bit scaled U3U2U1U0 -> mm4
packuswb  mm6, mm2        ;all 8 Y values -> mm6

movq      mm7, OFFSETB    ;128,128,128,128 -> mm7
paddd     mm1, mm5        ;V7V6 -> mm1

paddw     mm4, mm7        ;add offset to U3U2U1U0/256
psrad     mm1, 15         ;32-bit scaled V7V6 -> mm1

movq      [ebx], mm6      ;store Y
packuswb  mm4, mm0        ;all 8 U values -> mm4

movq      mm5, TEMPV      ;32-bit scaled V3V2V1V0 -> mm5
packssdw  mm3, mm1        ;V7V6V5V4 -> mm3

paddw     mm5, mm7        ;add offset to V3V2V1V0
paddw     mm3, mm7        ;add offset to V7V6V5V4

movq      [ecx], mm4      ;store U
packuswb  mm5, mm3        ;ALL 8 V values -> mm5

add       ebx, 8          ;increment Y count
add       ecx, 8          ;increment U count

movq      [edx], mm5      ;store V

dec       edi             ;decrement loop counter
jnz       RGBtoYUV        ;do 24 more bytes if not 0

pop       edi
pop       esi
pop       edx
pop       ecx
pop       ebx
pop       eax
pop       ebp

ret       0

_rgbtoyuv ENDP
_TEXT ENDS
END
```

3.5. Преобразование данных из формата 24-Bit True Color в формат 16-Bit High Color

Преобразование данных из формата True Color в формат High Color показано на рис. 3.1.

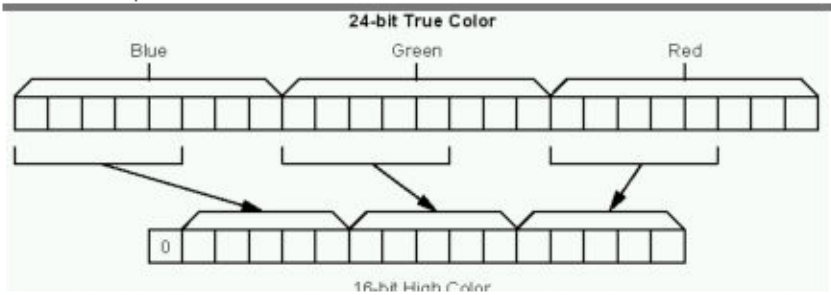


Рис. 3.1. Преобразование данных из формата True Color в формат High Color

Далее будут рассмотрены два алгоритма преобразования. Первый алгоритм – Mask-Shift-Or. Второй алгоритм – PMADD.

Mask-Shift-Or Algorithm

```
;*
;* Description:
;* The purpose of this file is to provide the MMX  code for the
;* RGB24to16 algorithm as an instructional example
;* to those who are just beginning
;* to code using MMX  instructions.
;*
;* Assumptions:
;* 1. The number of elements allocated for the
;* source (src) must be divisible by 8.
;* The number of rows X the number of columns does
;* not need to be divisible by 8. This is to allow
;* working on 8
;* pixels within the inner loop without having to
;* post-process pixels after the loop.
;* 2. The number of elements allocated for the
;* destination (dest) must
;* be divisible by 8. The number of rows X the
;* number of columns
;* does not need to be divisible by 8. This is to
;* allow working on 8 pixels within the inner loop
;* without having to post-process
;* pixels after the loop.
;*
;*************************************************
;


.586
.MODEL FLAT, C

PD      EQU     <DWORD PTR>
PW      EQU     <WORD PTR>
```

```
PB      EQU     <BYTE PTR>
;------------------------- --------------------

.CODE
RGB24to16 PROC C PUBLIC USES esi edi ebp ebx ecx,
src:PTR DWORD,
dest:PTR DWORD,
nRows:PTR DWORD,
nCols:PTR DWORD

; Locals (on local stack frame)

saveesp         EQU     PD [esp+0]
EndOfLocals EQU         PD [esp+4]

LocalFrameSize = 4

;constants for MMX register initialization

.DATA

blues   dq 0f8000000f8h         ;mask for 5 MSbits of blue data
greens  dq 0f8000000f800h       ;mask for 5 MSbits of green data
reds    dq 0f8000000f80000h ;mask for 5 MSbits of red data

.CODE

MEM_MASK_BLUES EQU DWORD PTR blues
MEM_MASK_GREENS EQU DWORD PTR greens
MEM_MASK_REDS EQU DWORD PTR reds

;************************************************
;*
;* Procedure: rgb24to16 Date: 12/08/95
;*
;* Author: Patricia L. Gray File: rgb24to16.asm
;*
;* Description:
;* rgb24to16 is an optimized MMX routine to convert
;* RGB data from 24 bit true color to 16 bit high color.
;* The inner loop processes
;* 8 pixels at a time and packs the 8 pixels
;* represented as 8 DWORDs
;* into 8 WORDs. The algorithm used for each 2
;* pixels is as follows:
;* Step 1: read in 2 pixels as a quad word
;* Step 2: make a copy of the two pixels
;* Step 3: AND the 2 pixels with 0x00f8000000f8 to
;* obtain a quad word of:
```

```
;* 00000000000000000000000000BBBBB000
;* 00000000000000000000000000bbbbb000
;*
;* Step 4: PSRLD quad word by 3 to obtain a quad
;* word of:
;* 0000000000000000000000000000BBBBB
;* 0000000000000000000000000000bbbbb
;*
;* Step 5: make a copy of the original again
;* Step 6: AND the copy of the original pixels with
;* 0x0000f8000000f800 to obtain
;* 00000000000000000GGGGG00000000000
;* 00000000000000000ggggg00000000000
;*
;* Step 7 PSRLD quad word by 6 to obtain a quad
;* word of:
;* 0000000000000000000000GGGGG00000
;* 0000000000000000000000ggggg00000
;*
;* Step 8 OR the results of Step 4 and 7 to obtain
;* a quad word of:
;* 0000000000000000000000GGGGGBBBBB
;* 0000000000000000000000gggggbbbbb
;*
;* Step 9 AND the last copy of the original with
;* 0x00f8000000f80000h to obtain
;* 00000000RRRRR000000000000000000000
;* 00000000rrrrr000000000000000000000
;*
;* Step 10 PSRLD quad word by 9 to obtain a quad word of:
;* 0000000000000000RRRRR0000000000
;* 0000000000000000rrrrr0000000000
;*
;* Step 11 OR the results of Step 8 and 10 to
;* obtain a quad word of:
;* 0000000000000000RRRRRGGGGGBBBBB
;* 0000000000000000rrrrrgggggbbbbb
;*
;* Step 8: When two pairs of pixels are converted,
;* pack the results into one register and then store them into
;* the destination.
;*
;* Inputs:
;* src long int * a pointer to the first element of the input source
;*
;* dest short int * a pointer to the destination
;* of the converted RGB data
;* nRows short int * the number of rows in the src/dest bitmap
;*
```

```
;* nCols short int * the number of columns in the
;* src/dest bitmap
;*

mov     ecx,esp
sub     esp,LocalFrameSize
and     esp, 0fffffff8h             ;8-byte align start of local stack frame
mov     saveesp, ecx                ;save original esp to restore in epilgue
mov     eax, nRows                  ;multiply nRows
mov     ebx, nCols                  ;with nCols
imul    ebx                         ;to get the size of the source
mov     ecx, eax                    ;compare with loop counter
sub     ecx, 8

mov     eax, src                    ;load the src and
mov     ebx, dest                   ;and dest pointers

inner_loop:

movq    mm0, [eax+4*ecx]            ;get the first 2 RGB elements

movq    mm3, [eax+4*ecx+16]         ;get the third 2 RGB elements
movq    mm1, mm0                    ;save original first 2

pand    mm0, MEM_MASK_BLUES         ;mask out all but the 5 MSB blue data
movq     mm4, mm3                   ;save the original third 2

pand    mm3, MEM_MASK_BLUES         ;mask out all but the 5 MSB blue data
psrld   mm0, 3                      ;shift blues right 3 which is now the result reg 1

psrld   mm3, 3                      ;shift blues right 3 which is now the result reg 3
movq    mm2, mm1                    ;save the original again

pand    mm1, MEM_MASK_GREENS        ;mask out all but the 5MSB green data
movq    mm6, mm4                    ;save the original again

pand    mm4, MEM_MASK_GREENS        ;mask out all but the 5MSB green data
psrld   mm1, 6                      ;shift greens to bits 5-9

por     mm0, mm1                    ;combine with the blue data in result reg 1
psrld   mm4, 6                      ;shift greens to bits 5-9

pand    mm2, MEM_MASK_REDS          ;mask out all but the 5MSB red data
por     mm3, mm4                    ;combine with the green data in result reg 2

pand    mm6, MEM_MASK_REDS          ;mask out all but the 5MSB red data
psrld   mm2, 9                      ;shift reds to bits 10-14

por     mm0, mm2                    ;combine with the blue and green data in
result reg
psrld   mm6, 9                      ;shift reds to bits 10-14
```

```
movq    mm1, [eax+4*ecx+24]     ;get fourth 2 RGB elements
por     mm6, mm3                ;combine with the blue and green data in
result reg

movq    mm3, [eax+4*ecx+8]      ;get second 2 RGB elements
movq    mm2, mm1                ;save the original fourth 2

pand    mm1, MEM_MASK_BLUES     ;mask out all but the 5 MSB blue data
movq     mm4, mm3               ;save original second 2

pand    mm3, MEM_MASK_BLUES     ;mask out all but the 5 MSB blue data
psrld   mm1, 3                  ;shift blues right 3 which is now the result reg 4

psrld   mm3, 3                  ;shift blues right 3 which is now the result reg 2
movq    mm5, mm4                ;save the original again

pand    mm4, MEM_MASK_GREENS        ;mask out all but the 5 MSB green
data
movq    mm7, mm2                ;save the original again

pand     mm2, MEM_MASK_GREENS       ;mask out all but the 5 MSB green
data
psrld   mm4, 6                  ;shift greens to bits 5-9

por     mm3, mm4                ;combine with the blue data in result reg 2
psrld   mm2, 6                  ;shift greens to bits 5-9

pand    mm5, MEM_MASK_REDS      ;mask out all but the 5 MSB red data
por     mm1, mm2                ;combine with the blue data in result reg 4

pand    mm7, MEM_MASK_REDS      ;mask out all but the 5 MSB red data
psrld   mm5, 9                  ;shift reds to bits 10-14

por     mm3, mm5                ;combine with the blue and
                                ;green data in result reg 2
psrld   mm7, 9                  ;shift reds to bits 10-14

packssdw mm0, mm3               ;pack result 1 and 2 into one qword
por     mm7, mm1                ;combine with the blue and
                                ;green data in result reg 4

packssdw mm6, mm7               ;pack result 3 and 4 into one qword
movq    [ebx+2*ecx], mm0        ;store the result

movq    [ebx+2*ecx+8], mm6      ;store the result
sub     ecx, 8
jae     inner_loop              ;go get some more if not done

DONE:
emms                            ;function epilogue
mov     esp, saveesp
ret
```

```
RGB24to16 ENDP
END
```

PMADD Algorithm

На рис. 3.2 приведена диаграмма PMADD-алгоритма.

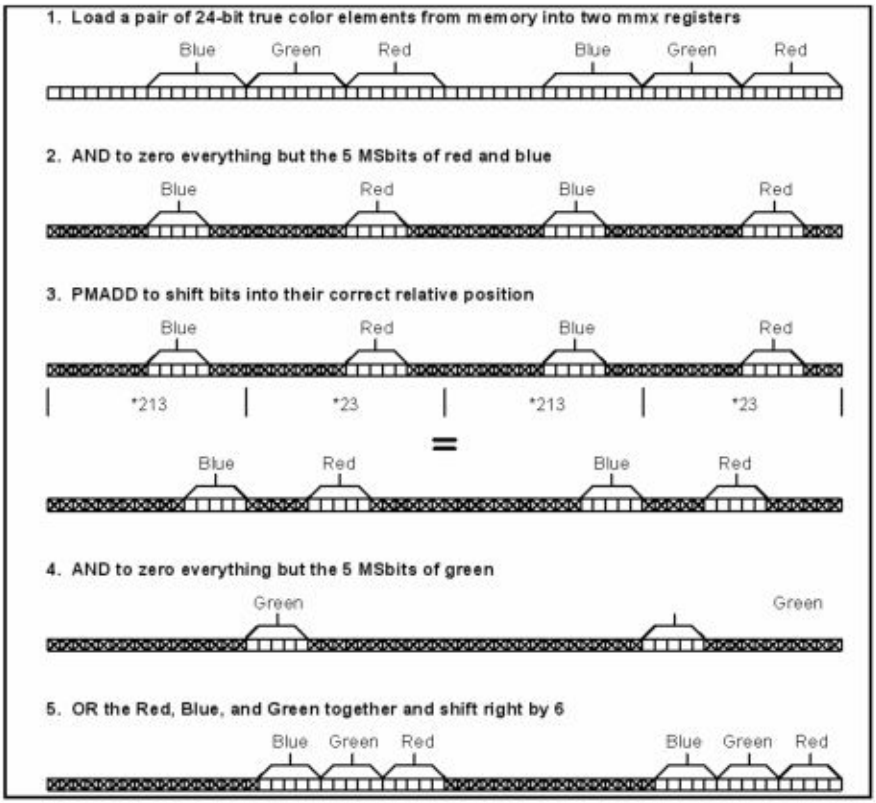


Рис. 3.2. PMADD-алгоритм для преобразования цвета из формата True Color в формат High Color

```
;* Description:
;* The purpose of this file is to provide the MMX code for the
;* RGB24to16 algorithm as an instructional example to those who
;* are just beginning to code using MMX instructions.
;*
;* Assumptions:
;* 1. The number of elements allocated for the pMatrix must be
;* divisible by 8. The number of rows X the number
;* of columns does
;* not need to be divisible by 8. This is to allow working on 8
```

```
;* pixels within the inner loop without having to
;* post-process pixelsafter the loop.
;* 2. The number of elements allocated for qMatrix must be divisible
;* by 8. The number of rows X the number of columns does not need
;* to be divisible by 8. This is to allow working on 8 pixels within
;* the inner loop without having to post-process pixels after the loop.
;*
;*************************************************/

TITLE Convert RGB 24 to 16
.486P

.model FLAT

;************************************************
;* DATA SEGMENT
;************************************************

_DATA SEGMENT
rgbMulFactor    dq 2000000820000008H ; RGB quad word multiplier
rgbMask1        dq 00f800f800f800f8H
rgbMask2        dq 0000f8000000f800H

_DATA ENDS
;************************************************
;* TEXT SEGMENT
;************************************************
_TEXT SEGMENT

; Declare rgb24to16 as a public routine to allow
; the 'C' code to
; call it.

PUBLIC RGB24to16
;* Description:
;* rgb24to16 is an optimized MMX routine to convert
;* RGB data from
;* 24 bit true color to 16 bit high color. The inner loop processes
;* 8 pixels at a time and packs the 8 pixels
;* represented as 8 DWORDs
;* into 8 WORDs. The algorithm used for each 2
;* pixels is as follows:
;* Step 1: read in 2 pixels as a quad word
;* Step 2: make a copy of the two pixels
;* Step 3: AND the 2 pixels with 0x00f800f800f800f8
;* to obtain a quad word of:
;* 00000000RRRRR00000000000BBBBB000
;* 00000000rrrrr00000000000bbbbb000
;*
```

```
;* Step 4: PMADDWD this quad word by
;* 0x2000000820000008 to obtain
;* a quad word of:
;* 00000000000RRRRR00000BBBBB000000
;* 00000000000rrrrr00000bbbbb000000
;*
;* Step 5: AND the copy of the original pixels with
;* 0x0000f8000000f800 to obtain
;* 0000000000000000GGGGG00000000000
;* 0000000000000000ggggg00000000000
;*
;* Step 6: OR the results of step 4 and step 5 to
;* obtain
;* 00000000000RRRRRGGGGGBBBBB000000
;* 00000000000rrrrrgggggbbbbb000000
;*
;* Step 7: SHIFT RIGHT by 6 bits to obtain
;* 0000000000000000RRRRRGGGGGBBBBB
;* 0000000000000000rrrrrgggggbbbbb
;*
;* Step 8: When two pairs of pixels are converted, pack the
;* results into one register and then store them
;* into
;* the q Matrix.
;*
;* Inputs:
;* pPtr long int * a pointer to the first element of the input 'p' matrix
;* qPtr short int * a pointer to the output RGB converted matrix
;* nRows short int * the number of rows in the p/q matrix
;* nCols short int * the number of columns in the p/q matrix
;*
;*************************************************/

RGB24to16 PROC C USES ebx ecx edx,
pMatrix:PTR DWORD,
qMatrix:PTR WORD,
nRows:DWORD,
nCols:DWORD
;
; This calculates the number of elements in the 'p'
; matrix and assigns it to
; EAX. EAX is then adjusted to contain the index to
; the last 8 pixel aligned
; pixel by performing ((nRows  * nCols) – 8 + 7) &
; 0xfffffff8. Pointers to the
; arrays 'p' and 'q' are also set up
;
mov         eax, nRows
mov         ebx, nCols
```

```
imul    ebx             ; EAX = total number of pixels to process

mov     ebx, pMatrix    ; EBX points to 'pMatrix'
sub     eax, 1          ; align index EAX to the last 8 pixel boundary

mov     edx, qMatrix    ; EDX points to 'qMatrix'
and     eax, 0fffffff8H ; finish the index EAX alignment

;
; This section performs up to and including step 4
; on pixels 0 and 1. It also
; performs up to and including step 5 on pixels 2 and 3.
; This is done prior to
; entering the loop so that better loop efficiency
; is achieved. Better loop
; efficiency is achieved because these instructions
; are paired with other
; instruction at the end of the loop which could
; not be previously paired.
;

movq    mm7, DWORD PTR rgbMulFactor         ; MM7 = pixel multiplication
factor

movq    mm6, DWORD PTR rgbMask2     ; MM6 = green pixel mask

movq    mm2, 8[ebx][eax*4]          ; get pixels 2 and 3

movq    mm0, [ebx][eax*4]           ; get pixels 0 and 1
movq    mm3, mm2                    ; copy pixels 2 and 3

pand    mm3, DWORD PTR rgbMask1     ; get R and B of pixels 2 and 3
movq    mm1, mm0                    ; copy pixels 0 and 1

pand    mm1, DWORD PTR rgbMask1     ; get R and B of pixels 0 and 1
pmaddwd mm3, mm7                    ; SHIFT-OR pixels 2 and 3

pmaddwd mm1, mm7                    ; SHIFT-OR pixels 0 and 1
pand    mm2, mm6                    ; get G of pixels 2 and 3

;
; This section performs steps 1 through 8 for 4
; pairs of pixels (or for a total
; of 8 pixels).
;

inner_loop:

movq    mm4, 24[ebx][eax*4] ; get pixels 6 and 7
```

```
pand      mm0, mm6              ; get G of pixels 0 and 1

movq      mm5, 16[ebx][eax*4] ; get pixels 4 and 5
por       mm3, mm2              ; OR to get RGB of pixels 2 and 3

psrld     mm3, 6                ; SHIFT pixels 2 and 3 (step 7)
por       mm1, mm0              ; OR to get RGB of pixels 0 and 1

movq      mm0, mm4              ; copy pixels 6 and 7
psrld     mm1, 6                ; SHIFT pixels 0 and 1 (step 7)

pand      mm0, DWORD PTR rgbMask1     ; get R and B of pixels 6 and 7
packssdw mm1, mm3              ; combine pixels 0, 1, 2 and 3

movq      mm3, mm5              ; copy pixels 4 and 5
pmaddwd mm0, mm7              ; SHIFT-OR pixels 6 and 7

pand      mm3, DWORD PTR rgbMask1     ; get R and B of pixels 4 and 5
pand      mm4, mm6              ; get G of pixels 6 and 7

movq      [edx][eax*2], mm1   ; store pixels 0, 1, 2 and 3
pmaddwd mm3, mm7              ; SHIFT-OR pixels 4 and 5

sub       eax, 8                ; subtract 8 pixels from the index
por       mm4, mm0              ; OR to get RGB of pixels 6 and 7

pand      mm5, mm6              ; get G of pixels 4 and 5
psrld     mm4, 6                ; loop iteration; SHIFT pixels 6

movq      mm2, 8[ebx][eax*4]  ; get pixels 2 and 3 for the next
por       mm5, mm3              ; loop iteration OR to get RGB of and 7 (step 7)

movq      mm0, [ebx][eax*4]   ; get pixels 0 and 1 for the next
psrld     mm5, 6                ; SHIFT pixels 4 and 5 (step 7) pixels 4 and 5

movq      mm3, mm2              ; copy pixels 2 and 3
movq      mm1, mm0              ; copy pixels 0 and 1

pand      mm3, DWORD PTR rgbMask1     ; get R and B of pixels 2 and 3
packssdw mm5, mm4              ; combine pixels 4, 5, 6 and 7

pand      mm1, DWORD PTR rgbMask1     ; get R and B of pixels 0 and 1
pand      mm2, mm6              ; get G of pixels 2 and 3

movq      24[edx][eax*2], mm5 ; store pixels 4, 5, 6 and 7
pmaddwd mm3, mm7              ; SHIFT-OR pixels 2 and 3

pmaddwd mm1, mm7              ; SHIFT-OR pixels 0 and 1
jge       inner_loop            ; if we need to do more jump to
```

```
                                    ; the beginning of the loop
;
; We have converted 24-bit true color to 16-bit
; high color for the given data!
;

rgb24to16_done:

emms
ret 0                   ;we are done!

RGB24to16 ENDP
_TEXT ENDS
END
```

3.6. Наложение изображения на поверхность

Предположим, что у нас есть некоторая фоновая поверхность и нам надо наложить на нее изображение, некоторые точки которого являются прозрачными (рис. 3.3). Прозрачные точки имеют значение, равное нулю.

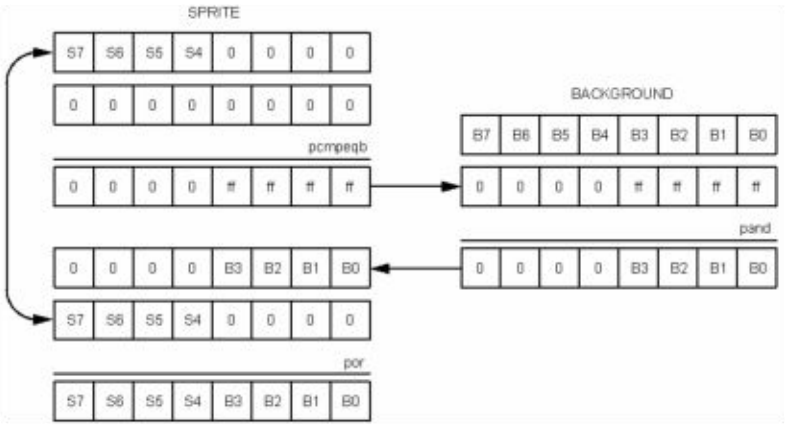


Рис. 3.3. Наложение изображения на поверхность

Оптимизированный алгоритм наложения изображения на поверхность

```
;sprite overlay
;This function assumes that the width of the sprite
;is divisible by 8

TITLE testmmx
.486
```

```
.model FLAT PUBLIC _sprite_overlay

_DATA SEGMENT

_DATA ENDS

_TEXT SEGMENT

_spritePtr$ = 8                 ;the sprite
_backPtr$ = 12                  ;buffer to save region under sprite
_videoPtr$ = 16                 ;video buffer with large background
_offset_video$ = 20             ;offset from beginning of video buffer
_width_screen$ = 24             ;width of large background image
_width_sprite$ = 28
_height_sprite$ = 32

_sprite_overlay PROC NEAR

push    ebp
mov     ebp, esp
push    eax
push    ebx
push    ecx
push    edx
push    esi
push    edi

mov     ecx, _videoPtr$[ebp]
mov     edi, _offset_video$[ebp]

mov     edx, _width_screen$[ebp]
add     ecx, edi

mov     eax, _spritePtr$[ebp]
mov     ebx, _backPtr$[ebp]

mov     edi, _width_sprite$[ebp]
mov     esi, _height_sprite$[ebp]

sub     edx, edi
xor     ebp, ebp

sub     ebx, 8
sub     ecx, 8
```

```
LoopTop:

add     ebx, 8          ;counter for buffer region under sprite add ecx, 8
                        ;counter for video address to write sprite

movq    mm0, [eax]      ;read sprite
pxor    mm7, mm7        ;zero in mm7

movq    mm1, [ecx]      ;read background under sprite pcmpeqb mm7, mm0
                        ;make mask where sprite is transparent

add     eax, 8          ;counter for sprite buffer
pand    mm7, mm1        ;pixels of region under sprite to draw

add     ebp, 8          ;inner loop counter
por     mm7, mm0        ;combine sprite and background

movq    [ebx], mm1      ;save background overwritten by sprite

movq    [ecx], mm7      ;write sprite overlay results

cmp     ebp, edi        ;done with current sprite row? jne LoopTop

add     ecx, edx        ;advance region to next buffer row
xor     ebp, ebp        ;reset inner loop counter

dec     esi             ;last row of sprite ?
jnz     LoopTop

pop     edi
pop     esi
pop     edx
pop     ecx
pop     ebx
pop     eax
pop     ebp

ret     0
_sprite_overlay ENDP
_TEXT ENDS
END
```

3.7. Наложение изображений с использованием альфа-канала

Канал альфа используется для наложения (смешивания) двух изображений. При этом изображения смешиваются в разных пропорциях (в зависимости от значения альфа). Благодаря этому можно получать различные видеоэффекты, например плавный переход от одного изображения к другому или эффект полупрозрачности.

Уравнение альфа-наложения можно представить в следующем виде:

s[r,g,b] = alpha * p[r,g,b] + (1 – alpha) * q[r,g,b]

где: **alpha** – весовой коэффициент; **p и q** – значения цвета входных точек; **s –** значение цвета выходной точки.

Представление данных (рис. 3.4):

Массив **p** – это массив, состоящий из 32-разрядных элементов, где младшие 8 бит содержат компоненту синего цвета (pb), следующие 8 бит содержат компоненту зеленого цвета (pg), следующие 8 бит содержат компоненту красного цвета (pr) и старшие 8 бит содержат значение коэффициента альфа.

Массив **q** – это массив 15-разрядных значений, где младшие 5 бит содержат компоненту синего цвета (qb), следующие 5 бит содержат компоненту зеленого цвета (qg) и старшие 5 бит содержат компоненту красного цвета (qr).

Массив **s** – это массив 15-разрядных значений, построенный по принципу массива **q**.

Рис. 3.4. Формат представления данных для альфа-наложения

Оптимизируем уравнение для использования ММХ-технологии:

s = alpha * p + (1 – alpha) * q
s = alpha * p + q – alpha * q
s = alpha * (p – q) + q

На рис. 3.5 показана диаграмма вычисления цвета точки.

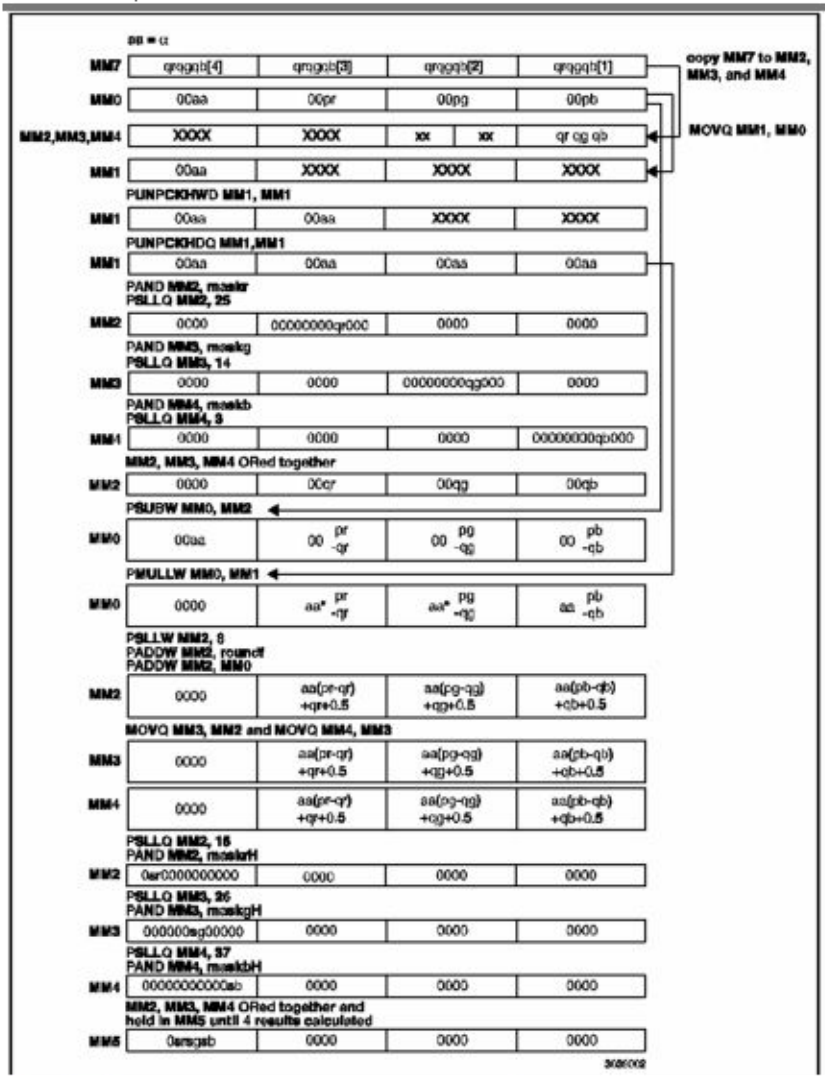


Рис. 3.5. Вычисление цвета точки при альфа-наложении

Оптимизированный алгоритм альфа-наложения

```
title alphaB
include iammx.inc

.486P
.model flat,c
```

```
.data
zeros   dq      0h
roundf  dd      800080h, 80h

maskr   dd      7c00h, 0h
maskg   dd      3e0h, 0h
maskb   dd      1fh, 0h

maskrH  dd      0h, 7c000000h
maskgH  dd      0h, 3e00000h
maskbH  dd      0h, 1f0000h

.code
;-----------------------------------------------
; input:  p_ptr pointer to array
;                 of [aa:8 pr:8 pg:8 pb:8]
;                 q_ptr pointer to array
;                 of [qr:5 qg:5 qb:5]
;                 nx number of rows
;                 ny number of columns
;
; output: q_ptr pointer to array of [sr:5 sg:5 sb:5]

alphaB proc near C uses esi edi eax ebx ecx edx,
p_ptr           : ptr dword,
q_ptr           : ptr word,
nx              : ptr word,
ny              : ptr word
;-----------------------------------------------

mov     esi, p_ptr          ; esi = p
xor     eax, eax            ; eax = 0
mov     edi, q_ptr          ; edi = q
xor     ecx, ecx            ; ecx = 0
mov     ax, word ptr nx     ; ax = nx
mov     cx, word ptr ny     ; cx = ny
imul    eax, ecx            ; eax = nx*ny

L1:
movq    MM7, mmword ptr [edi]   ; MM7 = qqq4 qqq3 qqq2 qqq1
pxor    MM5, MM5                ; MM5 = 0000 0000 0000 0000

L2:
movq    MM6, mmword ptr [esi]   ; MM6 = a2r2 g2b2 a1r1 g1b1
movq    MM0, MM6                ; MM0 = xxxx xxxx aarr ggbb
add     esi, 8                  ; esi += 8

L3:
punpcklbw   MM0, mmword ptr zeros       ; MM0 = 00aa 00rr 00gg 00bb
```

```
movq    MM2, MM7        ; MM2 = xxxx xxxx xxxx qqqq

pand    MM2, maskr      ; MM2 = 0000 0000 0000 qq00 movq MM1, MM0
                        ; MM1 = 00aa xxxx xxxx xxxx

movq    MM3, MM7        ; MM3 = xxxx xxxx xxxx qqqq punpckhwd MM1, MM1
                        ; MM1 = 00aa 00aa xxxx xxxx

pand    MM3, maskg      ; MM3 = 0000 0000 0000 0qq0
psllq   MM2, 25         ; MM2 = 0000 00qr 0000 0000

movq    MM4, MM7        ; MM4 = xxxx xxxx xxxx qqqq psllq MM3, 14
                        ; MM3 = 0000 0000 00qg 0000

pand    MM4, maskb      ; MM4 = 0000 0000 0000 00qq punpckhdq MM1, MM1
                        ; MM1 = 00aa 00aa 00aa 00aa

por     MM2, MM3        ; MM2 = 0000 00qr 00qg 0000 psllq MM4, 3
                        ; MM4 = 0000 0000 0000 00qb

por     MM2, MM4        ; MM2 = 0000 00qr 00qg 00qb psrlq MM6, 32
                        ; MM6 >>= 32

psubw   MM0, MM2        ; MM0 = p – q
psllw   MM2, 8          ; MM2 = 0000 qr00 qg00 qb00

paddw   MM2, roundf     ; MM2 = q + round'g factor
pmullw  MM0, MM1        ; MM0 = (p – q)*aa

psrlq   MM5, 16         ; MM5 >>= 16

psrlq   MM7, 16         ; MM7 >>= 16
nop

paddw   MM2, MM0        ; MM2 = (p – q)*aa + q + round'g factor
movq    MM0, MM6        ; MM0 = xxxx xxxx aarr ggbb

movq    MM3, MM2        ; MM3 = MM2
psllq   MM2, 15         ; MM2 = rrxx xxxx xxxx xxxx

movq    MM4, MM3        ; MM4 = MM3
psllq   MM3, 26         ; MM3 = xggx xxxx xxxx xxxx

pand    MM2, maskrH     ; MM2 = rr00 0000 0000 0000
psllq   MM4, 37         ; MM4 = xxbb xxxx xxxx xxxx

pand MM3, maskgH        ; MM3 = 0gg0 0000 0000 0000
por     MM5, MM2        ; MM2 = sss0 0000 0000 0000
```

```
pand MM4, maskbH          ; MM4 = 00bb 0000 0000 0000
por MM5, MM3              ; MM5 = ssss SSSS SSSS SSSS

por MM5, MM4
dec     eax               ; -- eax

test    eax, 1
jnz     L3
test    eax, 2
jnz     L2
movq    mmword ptr [edi], MM5     ; save 4 alpha
                                  ;blendedwords in q
cmp     eax, 0
je      L4
add     `edi, 8           ; edi += 8
jmp L1

L4:
emms
ret

alphaB endp
end
```

4. СПРАВОЧНИК ПО СИСТЕМЕ КОМАНД MMX

4.1. Синтаксис команд

Типичная MMX-команда имеет следующий синтаксис:

⇒ префикс: **P –** Packed. Означает использование упакованных данных;

⇒ операция: например, ADD, CMP или XOR;

⇒ суффикс:

- **US** – Unsigned Saturation. Означает использование беззнакового насыщения;
- S – Signed saturation. Означает использование знакового насыщения;
- **B, W, D, Q –** Byte, Word, Doubleword, Quadword. Означает используемый тип данных.

Команды, которые имеют различный тип входных и выходных элементов, имеют два суффикса. Например, команда преобразует данные из одного типа в другой. Такая команда будет иметь два суффикса: один для первоначального типа данных, другой для преобразованного типа данных.

4.2. Формат команд

MMX-команды используют существующий формат команд. Все команды, кроме EMMS, используют ModR/M-формат. Все команды начинаются с байта 0Fh. Поддерживаются все существующие режимы адресации с использованием формата SIB (Scale Index Base).

Для команд переноса данных операнд-приемник и операнд-источник могут располагаться в памяти, целочисленном регистре или MMX-регистре. Для всех остальных MMX-команд операнд-приемник находится в MMX-регистре, операнд-источник в памяти, MMX-регистре или является непосредственным операндом.

4.3. Обозначения, используемые при описании команд

СТОЛБЕЦ КОДА ОПЕРАЦИИ. В этом столбце приведен полный объектный код, формируемый для каждой формы команды. По

возможности коды приводятся как шестнадцатеричные байты в том порядке, в каком они хранятся в памяти. Применяются следующие элементы, отличающиеся от шестнадцатеричных байтов:

⇒ **/digit –** (в диапазоне 0-7 показывает, что байт ModR/M использует только операнд r/m (регистр или память). Поле reg содержит цифру (digit), являющуюся расширением кода операции;

⇒ **/r** – показывает, что байт ModR/M содержит операнд-регистр и операнд r/m

⇒ **ib** – однобайтный непосредственный операнд, находящийся после кода операции, и байт ModR/M и SIB.

СТОЛБЕЦ КОМАНДЫ. Этот столбец содержит синтаксис команды. Для представления операндом применяются следующие обозначения:

⇒ **imm8 –** непосредственное знаковое значение длиной байт с диапазоном значений от -128 до +127;

⇒ **r/m32** – операнд длиной двойное слово из регистра или памяти, используемый в командах с атрибутом размера операнда 32 бита;

⇒ **mm/m32** – обозначает младшие 32 разряда MMX-регистра или 32-разрядную ячейку памяти;

⇒ **mm/m64** – обозначает целый MMX-регистр или 64-разрядную ячейку памяти.

ОПЕРАЦИЯ. Здесь содержится алгоритмическое описание команды. При этом используются следующие функции:

⇒ **ZeroExtend (value)** – возвращает значение, расширенное нулем до размера операнда команды. Если, например, OperandSize = 32, то ZeroExtend (-10) превращает байт F6h в двойное слово 000000F6h. Если переданное функции ZeroExtend значение и размер операнда одинаковые, функция возвращает значение неизменным;

⇒ **SingExtend (value) –** возвращает значение, расширенное знаком до размера операнда команды. Если, например, OperandSize = 32, то SingExtend
(-10) превращает байт F6h в двойное слово FFFFFFF6h. Если переданное функции SingExtend значение и размер операнда одинаковые, функция возвращает значение неизменным;

⇒ **SaturateSignedWordToSignedByte** – преобразовывает знаковое 16-разрядное значение в знаковое 8-разрядное значение. Если исходное значение меньше -128, то оно насыщается до значения -128 (0x80). Если больше 127, то насыщается до значения 127 (0x7F);

⇒ **SaturateSignedDwordToSignedWord** – преобразовывает знаковое 32-разрядное значение в знаковое 16-разрядное значение. Если исходное значение меньше -32768, то оно насыщается до значения -32768 (0x8000). Если больше 32767, то насыщается до значения 32767 (0x7FFF);

⇒ **SaturateSignedWordToUnsignedByte** – преобразовывает знаковое 16-разрядное значение в беззнаковое 8-разрядное значение. Если исходное значение меньше нуля, то оно насыщается до значения 0 (0x00). Если больше 255, то насыщается до значения 255 (0xFF);

⇒ **SaturateToSignedByte** – представляет результат операции как знаковое 8-разрядное значение. Если результат меньше -128, то оно насыщается до значения -128 (0x80). Если больше 127, то насыщается до значения 127 (0x7F);

⇒ **SaturateToSignedWord** – представляет результат операции как знаковое 16-разрядное значение. Если результат меньше -32768, то оно насыщается до значения -32768 (0x8000). Если больше 32767, то насыщается до значения 32767 (0x7FFF);

⇒ **SaturateToUnsignedByte** – представляет результат операции как беззнаковое 8-разрядное значение. Если результат меньше нуля, то оно насыщается до значения 0 (0x00). Если больше 255, то насыщается до значения 255 (0xFF);

⇒ **SaturateToUnsignedWord** – представляет результат операции как беззнаковое 16-разрядное значение. Если результат меньше нуля, то оно насыщается до значения 0 (0x00). Если больше 65535, то насыщается до значения 65535 (0xFFFF).

ВОЗДЕЙСТВИЕ НА ФЛАЖКИ. Все MMX-команды никогда не изменяют регистр EFLAGS, поэтому данный пункт в описании команд отсутствует.

4.4. Перечень MMX-команд

EMMS – Empty MMX™ State

Код	*Команда*	*Описание*
0F 77	EMMS	Очищает регистр тегов

Операция

TW ← 0xFFFF;

Описание

Команда EMMS заносит во все биты регистра тегов единицы, помечая регистры FPU как свободные. Далее они могут быть использованы FPU-командами. Если FPU-команда попытается ис-

пользовать "грязные" регистры, то это может привести к переполнению стека регистров, генерации исключений или получению неверного результата.

Все MMX-команды заносят во все поля регистра тегов 0.

Команда EMMS должна быть исполнена в конце MMX подпрограмм, перед обращением к другим подпрограммам, в которых могут содержаться FPU-команды.

Пример

Особые случаи P-режима

#UD, если CR0.EM=1.
#NM, если CR0.TS=1.
#MF, если обрабатывается исключение FPU.

Особые случаи R-режима

#UD, если CR0.EM=1.
#NM, если CR0.TS=1.
#MF, если обрабатывается исключение FPU.

Особые случаи V-режима

#UD, если CR0.EM=1.
#NM, если CR0.TS=1.
#MF, если обрабатывается исключение FPU.

MOVD – Move 32 Bits

Код	*Команда*	*Описание*
0F 6E /r	MOVD mm, r/m32	Передача двойного слова из целочисленного регистра или памяти в MMX-регистр
0F 7E /r	MOVD r/m32, mm	Передача двойного слова из MMX-регистра в целочисленный регистр или память

Операция

IF destination = mm
THEN
mm(63…0) ← ZeroExtend(r/m32);
ELSE
r/m32 ← mm(31…0);

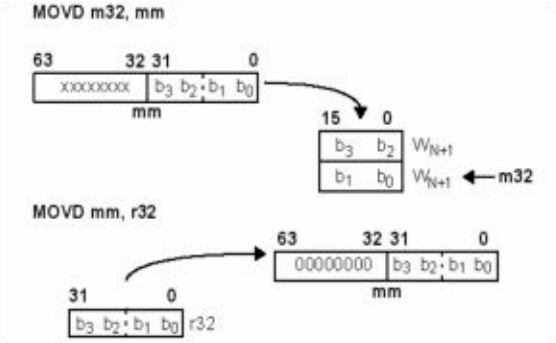
Описание

Команда MOVD копирует 32 разряда из операнда-источника в операнд-приемник. Операнды могут быть MMX-регистрами, целочисленными регистрами или находиться в памяти. Команда MOVD не может переносить данные из MMX-регистра в MMX-регистр, из памяти в память или из целочисленного регистра в целочисленный регистр.

Когда операнд-приемник является MMX-регистром, то 32-разрядный операнд-источник записывается в младшие 32 разряда 64-разрядного регистра-приемника, при этом в старшие 32 разряда заносятся нули.

Когда операндом-источником является MMX-регистр, то младшие 32 разряда MMX-регистра записываются в память или 32-разрядный целочисленный регистр.

Пример



Особые случаи P-режима

#GP(0), если операнд-приемник в незаписываемом сегменте.
#GP(0), если эффективный адрес операнда выходит за пределы сегмента CS, DS, ES, FS или GS. Если DS, ES, FS или GS равны нулю.
#SS(0), если эффективный адрес операнда выходит за пределы сегмента SS.
#PF(код-нарушение), при страничном нарушении.
#AC(0), при невыравненном обращении к памяти, если CPL=3 и данное исключение разрешено.
#UD, если CR0.EM=1.
#NM, если CR0.TS=1.
#MF, если обрабатывается исключение FPU.

Особые случаи R-режима

#GP(0), если эффективный адрес операнда выходит за пределы сегмента CS, DS, ES, FS или GS.
#UD, если CR0.EM=1.
#NM, если CR0.TS=1.
#MF, если обрабатывается исключение FPU.

Особые случаи V-режима

Такие же, как в R-режиме.
#PF(код-нарушение), при страничном нарушении.
#AC(0), при невыравненном обращении к памяти, если CPL=3 и данное исключение разрешено.

MOVQ – Move 64 Bits

Код	Команда	Описание
0F 6F /r	MOVQ mm, mm/m64	Передача учетверенного слова из MMX-регистра или памяти в MMX-регистр
0F 7F /r	MOVQ mm/m64, mm	Передача учетверенного слова из MMX-регистра в MMX-регистр или память

Операция

```
IF destination = mm
THEN
mm ← mm/m64;
ELSE
mm/m64 ← mm;;
```

Описание

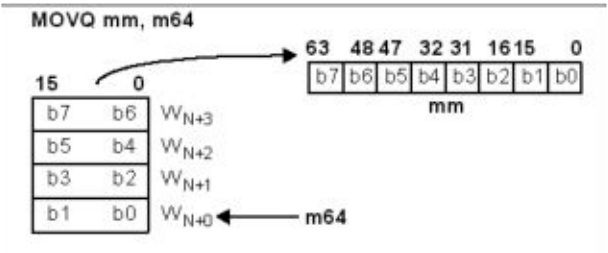
Команда MOVD копирует 64 разряда из операнда-источника в операнд-приемник. Операнды могут быть MMX-регистрами или находиться в памяти. Команда MOVQ не может переносить данные из памяти в память.

Когда операнд-приемник является MMX-регистром, а операнд-источник находится в памяти, то 64-разрядный операнд в памяти копируется в MMX-регистр.

Когда операндом-источником является MMX-регистр, а операнд-приемник находится в памяти, то MMX-регистр копируется в 64-разрядный операнд в памяти.

Когда оба операнда являются MMX-регистрами, то регистр-источник копируется в регистр-приемник.

Пример



Особые случаи P-режима

#GP(0), если операнд-приемник в незаписываемом сегменте.
#GP(0), если эффективный адрес операнда выходит за пределы сегмента CS, DS, ES, FS или GS. Если DS, ES, FS или GS равны нулю.
#SS(0), если эффективный адрес операнда выходит за пределы сегмента SS.
#PF(код-нарушение), при страничном нарушении.

#AC(0), при невыравненном обращении к памяти, если CPL=3 и данное исключение разрешено.
#UD, если CR0.EM=1.
#NM, если CR0.TS=1.
#MF, если обрабатывается исключение FPU.

Особые случаи R-режима

#GP(0), если эффективный адрес операнда выходит за пределы сегмента CS, DS, ES, FS или GS.
#UD, если CR0.EM=1.
#NM, если CR0.TS=1.
#MF, если обрабатывается исключение FPU.

Особые случаи V-режима

Такие же, как в R-режиме.
#PF(код-нарушение), при страничном нарушении.
#AC(0), при невыравненном обращении к памяти, если CPL=3 и данное исключение разрешено.

PACKSSWB/PACKSSDW – Pack with Signed Saturation

Код	*Команда*	*Описание*
0F 63 /r	PACKSSWB mm, mm/m64	Упаковывает с насыщением знаковые слова из MMX-регистра и MMX-регистра/памяти в знаковые байты MMX-регистра
0F 6B /r	PACKSSDW mm, mm/m64	Упаковывает с насыщением знаковые двойные слова из MMX-регистра и из MMX-регистра или памяти в знаковые слова MMX-регистра

Операция

```
IF instruction is PACKSSWB
THEN {
    mm(7...0) ← SaturateSignedWordToSignedByte mm(15...0);
    mm(15...8) ← SaturateSignedWordToSignedByte mm(31...16);
    mm(23...16) ← SaturateSignedWordToSignedByte mm(47...32);
    mm(31...24) ← SaturateSignedWordToSignedByte mm(63...48);
    mm(39...32) ← SaturateSignedWordToSignedByte mm/m64(15...0);
    mm(47...40) ← SaturateSignedWordToSignedByte mm/m64(31...16);
    mm(55...48) ← SaturateSignedWordToSignedByte mm/m64(47...32);
    mm(63...56) ← SaturateSignedWordToSignedByte mm/m64(63...48);
    }
ELSE { (* instruction is PACKSSDW *)
    mm(15...0) ← SaturateSignedDwordToSignedWord mm(31...0);
    mm(31...16) ← SaturateSignedDwordToSignedWord mm(63...32);
    mm(47...32) ← SaturateSignedDwordToSignedWord mm/m64(31...0);
    mm(63...48) ← SaturateSignedDwordToSignedWord mm/m64(63...32);
    }
```

Описание

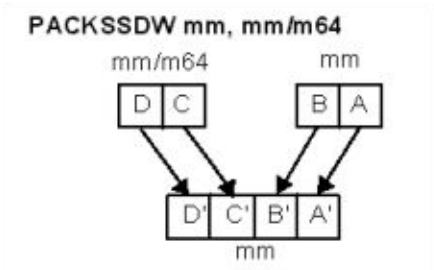
Команда PACKSS упаковывает с насыщением элементы данных со знаком из обоих операндов и записывает результат в операнд-приемник. Операндом-приемником является MMX-регистр. Операндом-источником может быть как MMX-регистр, так и операнд в памяти.

Элементы операндов преобразовываются в элементы с в два раза меньшей длины, используя принцип насыщения. В младшую часть результата записывается упакованный операнд-приемник, а в старшую часть – упакованный операнд-источник.

Команда PACKSSWB упаковывает 4 знаковых слова из обоих операндов в 8 знаковых байт. Если значение слова больше или меньше границ диапазона знакового байта, то результат упаковки насыщается соответственно до 0x7F или до 0x80.

Команда PACKSSDB упаковывает два знаковых двойных слова из обоих операндов в 4 знаковых слова. Если значение двойного слова больше или меньше границ диапазона знакового слова, то результат упаковки насыщается соответственно до 0x7FFF или до 0x8000.

Пример



Особые случаи P-режима

#GP(0), если эффективный адрес операнда выходит за пределы сегмента CS, DS, ES, FS или GS. Если DS, ES, FS или GS равны нулю.
#SS(0), если эффективный адрес операнда выходит за пределы сегмента SS.
#PF(код-нарушение), при страничном нарушении.
#AC(0), при невыравненном обращении к памяти, если CPL=3 и данное исключение разрешено.
#UD, если CR0.EM=1.
#NM, если CR0.TS=1.
#MF, если обрабатывается исключение FPU.

Особые случаи R-режима

#GP(0), если эффективный адрес операнда выходит за пределы сегмента CS, DS, ES, FS или GS.
#UD, если CR0.EM=1.
#NM, если CR0.TS=1.
#MF, если обрабатывается исключение FPU.

Особые случаи V-режима

Такие же, как в R-режиме.
#PF(код-нарушение), при страничном нарушении.
#AC(0), при невыравненном обращении к памяти, если CPL=3 и данное исключение разрешено.

PACKUSWB – Pack with Unsigned Saturation

Код	*Команда*	*Описание*
0F 67 /r	PACKUSWB mm, mm/m64	Упаковывает с насыщением знаковые слова из MMX-регистра и MMX-регистра/памяти в беззнаковые байты MMX-регистра

Операция

mm(7...0) ← SaturateSignedWordToUnsignedByte mm(15...0);
mm(15...8)← SaturateSignedWordToUnsignedByte mm(31...15);
mm(23...16) ← SaturateSignedWordToUnsignedByte mm(47...32);
mm(31...24) ← SaturateSignedWordToUnsignedByte mm(63...48);
mm(39...32) ← SaturateSignedWordToUnsignedByte mm/m64(15...0);
mm(47...40) ← SaturateSignedWordToUnsignedByte mm/m64(31...16);
mm(55...48) ← SaturateSignedWordToUnsignedByte mm/m64(47...32);
mm(63...56) ← SaturateSignedWordToUnsignedByte mm/m64(63...48);

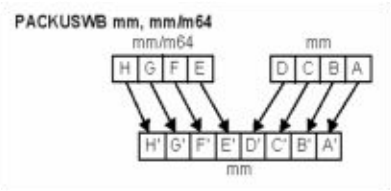
Описание

Операндом-приемником является MMX-регистр. Операндом-источником может быть как MMX-регистр так и операнд в памяти.

Элементы операндов преобразовываются в элементы в два раза меньшей длины, используя принцип насыщения. В младшую часть результата записывается упакованный операнд-приемник, а в старшую часть – упакованный операнд-источник.

Команда PACKUSWB упаковывает 4 знаковых слова из обоих операндов в 8 беззнаковых байт. Если значение слова больше или меньше границ диапазона беззнакового байта, то результат упаковки насыщается соответственно до 0xFF или до 0x00.

Пример



Особые случаи P-режима

#GP(0), если эффективный адрес операнда выходит за пределы сегмента CS, DS, ES, FS или GS. Если DS, ES, FS или GS равны нулю.
#SS(0), если эффективный адрес операнда выходит за пределы сегмента SS.
#PF(код-нарушение), при страничном нарушении.
#AC(0), при невыравненном обращении к памяти, если CPL=3 и данное исключение разрешено.
#UD, если CR0.EM=1.
#NM, если CR0.TS=1.
#MF, если обрабатывается исключение FPU.

Особые случаи R-режима

#GP(0), если эффективный адрес операнда выходит за пределы сегмента CS, DS, ES, FS или GS.
#UD, если CR0.EM=1.
#NM, если CR0.TS=1.
#MF, если обрабатывается исключение FPU.

Особые случаи V-режима

Такие же, как в R-режиме.
#PF(код-нарушение), при страничном нарушении.
#AC(0), при невыравненном обращении к памяти, если CPL=3 и данное исключение разрешено.

PADDB/PADDW/PADDD – Packed Add

Код	*Команда*	*Описание*
0F FC /r	PADDB mm, mm/m64	Складывает упакованные байты из MMX-регистра или памяти с упакованным байтом в MMX-регистре
0F FD /r	PADDW mm, mm/m64	Складывает упакованные слова из MMX-регистра или памяти с упакованными словами в MMX-регистре
0F FE /r	PADDD mm, mm/m64	Складывает упакованные двойные слова из MMX-регистра или памяти с упакованными двойными словами в MMX-регистре

Операция

IF instruction is PADDB
THEN {

```
    mm(7...0) ← mm(7...0) + mm/m64(7...0);
    mm(15...8) ← mm(15...8) + mm/m64(15...8);
    mm(23...16) ← mm(23...16) + mm/m64(23...16);
    mm(31...24) ← mm(31...24) + mm/m64(31...24);
    mm(39...32) ← mm(39...32) + mm/m64(39...32);
    mm(47...40) ← mm(47...40) + mm/m64(47...40);
    mm(55...48) ← mm(55...48) + mm/m64(55...48);
    mm(63...56) ← mm(63...56) + mm/m64(63...56);
    }
IF instruction is PADDW
THEN {
    mm(15...0) ← mm(15...0) + mm/m64(15...0);
    mm(31...16) ← mm(31...16) + mm/m64(31...16);
    mm(47...32) ← mm(47...32) + mm/m64(47...32);
    mm(63...48) ← mm(63...48) + mm/m64(63...48);
    }
ELSE { (* instruction is PADDD *)
    mm(31...0) ← mm(31...0) + mm/m64(31...0);
    mm(63...32) ← mm(63...32) + mm/m64(63...32);
    }
```

Описание

Команда PADD производит сложение элементов операнда-источника с элементами операнда-приемника, используя принцип циклического переноса. Результат операции записывается в регистр-приемник.

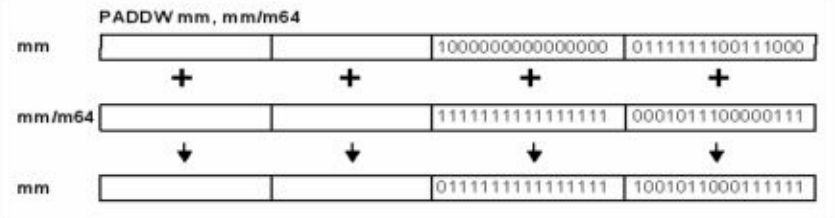
Операндом-приемником является MMX-регистр. Операндом-источником может быть как MMX-регистр, так и операнд в памяти.

Команда PADDB производит сложение байтов.

Команда PADDW производит сложение слов.

Команда PADDD производит сложение двойных слов.

Пример



Особые случаи P-режима

#GP(0), если эффективный адрес операнда выходит за пределы сегмента CS, DS, ES, FS или GS. Если DS, ES, FS или GS равны нулю.

#SS(0), если эффективный адрес операнда выходит за пределы сегмента SS.
#PF(код-нарушение), при страничном нарушении.
#AC(0), при невыравненном обращении к памяти, если CPL=3 и данное исключение разрешено.
#UD, если CR0.EM=1.
#NM, если CR0.TS=1.
#MF, если обрабатывается исключение FPU.

Особые случаи R-режима

#GP(0), если эффективный адрес операнда выходит за пределы сегмента CS, DS, ES, FS или GS.
#UD, если CR0.EM=1.
#NM, если CR0.TS=1.
#MF, если обрабатывается исключение FPU.

Особые случаи V-режима

Такие же, как в R-режиме.
#PF(код-нарушение), при страничном нарушении.
#AC(0), при невыравненном обращении к памяти, если CPL=3 и данное исключение разрешено.

PADDSB/PADDSW – Packed Add with Saturation

Код	*Команда*	*Описание*
0F EC /r	PADDSB mm, mm/m64	Складывает с насыщением знаковые упакованные байты из MMX-регистра или памяти со знаковыми упакованным байтам в MMX-регистре
0F ED /r	PADDSW mm, mm/m64	Складывает с насыщением знаковые упакованные слова из MMX-регистра или памяти со знаковыми упакованным словами в MMX-регистре

Операция

```
IF instruction is PADDSB
THEN{
    mm(7...0) ← SaturateToSignedByte (mm(7...0) + mm/m64 (7...0)) ;
    mm(15...8) ← SaturateToSignedByte (mm(15...8) + mm/m64(15...8) );
    mm(23...16) ← SaturateToSignedByte (mm(23...16) + mm/m64(23...16) );
    mm(31...24) ← SaturateToSignedByte (mm(31...24) + mm/m64(31...24) );
    mm(39...32) ← SaturateToSignedByte (mm(39...32) + mm/m64(39...32) );
    mm(47...40) ← SaturateToSignedByte (mm(47...40) + mm/m64(47...40) );
    mm(55...48) ← SaturateToSignedByte (mm(55...48) + mm/m64(55...48) );
    mm(63...56) ← SaturateToSignedByte (mm(63...56) + mm/m64(63...56) );
    }
ELSE { (* instruction is PADDSW *)
    mm(15...0) ← SaturateToSignedWord (mm(15...0) + mm/m64(15...0) );
    mm(31...16) ← SaturateToSignedWord (mm(31...16) + mm/m64(31...16) );
    mm(47...32) ← SaturateToSignedWord (mm(47...32) + mm/m64(47...32) );
    mm(63...48) ← SaturateToSignedWord (mm(63...48) + mm/m64(63...48) );
```

}

Описание

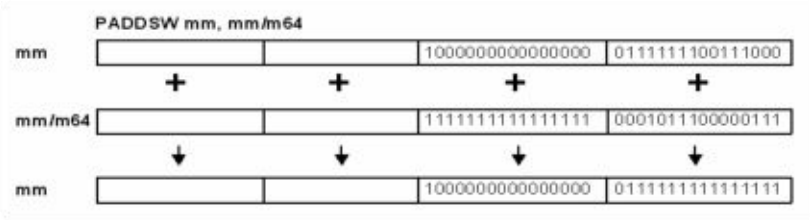
Команда PADDS производит сложение знаковых элементов операнда-источника со знаковыми элементами операнда-приемника, используя принцип насыщения. Результат операции записывается в регистр-приемник.

Операндом-приемником является MMX-регистр. Операндом-источником может быть как MMX-регистр, так и операнд в памяти.

Команда PADDB производит сложение знаковых байтов. Если значение результата больше или меньше границ диапазона знакового байта, то результат операции насыщается соответственно до 0x7F или до 0x80.

Команда PADDW производит сложение знаковых слов. Если значение результата больше или меньше границ диапазона знакового слова, то результат операции насыщается соответственно до 0x7FFF или до 0x8000.

Пример



Особые случаи P-режима

#GP(0), если эффективный адрес операнда выходит за пределы сегмента CS, DS, ES, FS или GS. Если DS, ES, FS или GS равны нулю.
#SS(0), если эффективный адрес операнда выходит за пределы сегмента SS.
#PF(код-нарушение), при страничном нарушении.
#AC(0), при невыравненном обращении к памяти, если CPL=3 и данное исключение разрешено.
#UD, если CR0.EM=1.
#NM, если CR0.TS=1.
#MF, если обрабатывается исключение FPU.

Особые случаи R-режима

#GP(0), если эффективный адрес операнда выходит за пределы сегмента CS, DS, ES, FS или GS.
#UD, если CR0.EM=1.
#NM, если CR0.TS=1.
#MF, если обрабатывается исключение FPU.

Особые случаи V-режима

Такие же, как в R-режиме.
#PF(код-нарушение), при страничном нарушении.
#AC(0), при невыравненном обращении к памяти, если CPL=3 и данное исключение разрешено.

PADDUSB/PADDUSW – Packed Add Unsigned with Saturation

Код	*Команда*	*Описание*
0F DC/r	PADDUSB mm, mm/m64	Складывает с насыщением беззнаковые упакованные байты из MMX-регистра или памяти с беззнаковыми упакованными байтами в MMX-регистре
0F DD/r	PADDUSW mm, mm/m64	Складывает с насыщением беззнаковые упакованные слова из MMX-регистра или памяти с беззнаковыми упакованными словами в MMX-регистре

Операция

```
IF instruction is PADDUSB
THEN{
    mm(7...0) ← SaturateToUnsignedByte (mm(7...0) + mm/m64 (7...0) );
    mm(15...8) ← SaturateToUnsignedByte (mm(15...8) + mm/m64(15...8) );
    mm(23...16) ← SaturateToUnsignedByte (mm(23...16) + mm/m64(23...16) );
    mm(31...24) ← SaturateToUnsignedByte (mm(31...24) + mm/m64(31...24) );
    mm(39...32) ← SaturateToUnsignedByte (mm(39...32) + mm/m64(39...32) );
    mm(47...40) ← SaturateToUnsignedByte (mm(47...40) + mm/m64(47...40) );
    mm(55...48) ← SaturateToUnsignedByte (mm(55...48) + mm/m64(55...48) );
    mm(63...56) ← SaturateToUnsignedByte (mm(63...56) + mm/m64(63...56) );
}
ELSE { (* instruction is PADDUSW *)
    mm(15...0) ← SaturateToUnsignedWord (mm(15...0) + mm/m64(15...0) );
    mm(31...16) ← SaturateToUnsignedWord (mm(31...16 ) + mm/m64(31...16) );
    mm(47...32) ← SaturateToUnsignedWord (mm(47...32 ) + mm/m64(47...32) );
    mm(63...48) ← SaturateToUnsignedWord (mm(63...48) + mm/m64(63...48) );
    }
```

Описание

Команда PADDUS производит сложение беззнаковых элементов операнда-источника с беззнаковыми элементами операнда-приемника, используя принцип насыщения. Результат операции записывается в регистр-приемник.

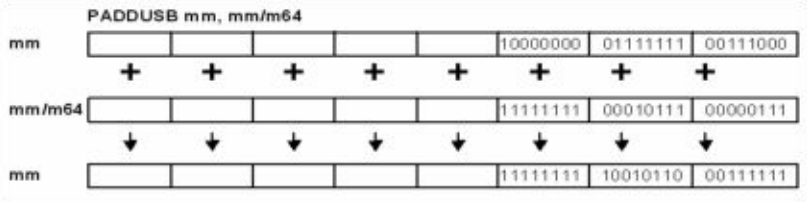
Операндом-приемником является MMX-регистр. Операндом-источником может быть как MMX-регистр, так и операнд в памяти.

Команда PADDUSB производит сложение беззнаковых байтов. Если значение результата больше или меньше границ диапазо-

на беззнакового байта, то результат операции насыщается соответственно до 0xFF или до 0x00.

Команда PADDUSW производит сложение беззнаковых слов. Если значение результата больше или меньше границ диапазона беззнакового слова, то результат операции насыщается соответственно до 0xFFFF или до 0x0000.

Пример



Особые случаи P-режима

#GP(0), если эффективный адрес операнда выходит за пределы сегмента CS, DS, ES, FS или GS. Если DS, ES, FS или GS равны нулю.
#SS(0), если эффективный адрес операнда выходит за пределы сегмента SS.
#PF(код-нарушение), при страничном нарушении.
#AC(0), при невыравненном обращении к памяти, если CPL=3 и данное исключение разрешено.
#UD, если CR0.EM=1.
#NM, если CR0.TS=1.
#MF, если обрабатывается исключение FPU.

Особые случаи R-режима

#GP(0), если эффективный адрес операнда выходит за пределы сегмента CS, DS, ES, FS или GS.
#UD, если CR0.EM=1.
#NM, если CR0.TS=1.
#MF, если обрабатывается исключение FPU.

Особые случаи V-режима

Такие же, как в R-режиме.
#PF(код-нарушение), при страничном нарушении.
#AC(0), при невыравненном обращении к памяти, если CPL=3 и данное исключение разрешено.

PAND – Bitwise Logical And

Код	*Команда*	*Описание*
0F DB /r	PAND mm, mm/m64	Логическое И ММХ-регистра или памяти с ММХ-регистром

Операция

mm ← mm AND mm/m64;

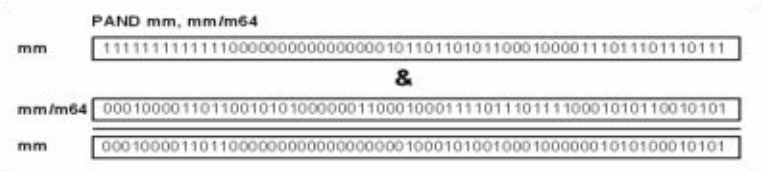
Описание

Команда PAND производит побитную операцию логическое И над 64 битами операнда-источника и операнда-приемника. Результат операции записывается в операнд-приемник.

Каждый бит результата устанавливается в единицу, если соответствующие биты операндов равны единице, в противном случаи бит результата устанавливается в 0.

Операндом-приемником является MMX-регистр. Операндом-источником может быть как MMX-регистр, так и операнд в памяти.

Пример



Особые случаи P-режима

#GP(0), если эффективный адрес операнда выходит за пределы сегмента CS, DS, ES, FS или GS. Если DS, ES, FS или GS равны нулю.
#SS(0), если эффективный адрес операнда выходит за пределы сегмента SS.
#PF(код-нарушение), при страничном нарушении.
#AC(0), при невыравненном обращении к памяти, если CPL=3 и данное исключение разрешено.
#UD, если CR0.EM=1.
#NM, если CR0.TS=1.
#MF, если обрабатывается исключение FPU.

Особые случаи R-режима

#GP(0), если эффективный адрес операнда выходит за пределы сегмента CS, DS, ES, FS или GS.
#UD, если CR0.EM=1.
#NM, если CR0.TS=1.
#MF, если обрабатывается исключение FPU.

Особые случаи V-режима

Такие же, как в R-режиме.
#PF(код-нарушение), при страничном нарушении.
#AC(0), при невыравненном обращении к памяти, если CPL=3 и данное исключение разрешено.

PANDN – Bitwise Logical And Not

Код	Команда	Описание
0F DF /r	PANDN mm, mm/m64	Логическое И-НЕ MMX-регистра или памяти с MMX-регистром

Операция

mm ←mm ANDN mm/m64;

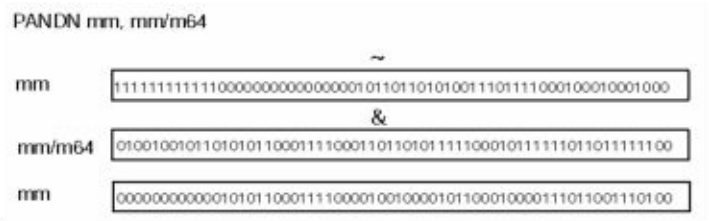
Описание

Команда PANDN производит побитную операцию логическое И-НЕ над 64 битами операнда-источника и операнда-приемника. Результат операции записывается в операнд-приемник.

Каждый бит результата устанавливается в 0, если соответствующие биты операндов равны нулю, в противном случае бит результата устанавливается в единицу.

Операндом-приемником является MMX-регистр. Операндом-источником может быть как MMX-регистр, так и операнд в памяти.

Пример



Особые случаи P-режима

#GP(0), если эффективный адрес операнда выходит за пределы сегмента CS, DS, ES, FS или GS. Если DS, ES, FS или GS равны нулю.
#SS(0), если эффективный адрес операнда выходит за пределы сегмента SS.
#PF(код-нарушение), при страничном нарушении.
#AC(0), при невыравненном обращении к памяти, если CPL=3 и данное исключение разрешено.
#UD, если CR0.EM=1.
#NM, если CR0.TS=1.
#MF, если обрабатывается исключение FPU.

Особые случаи R-режима

#GP(0), если эффективный адрес операнда выходит за пределы сегмента CS, DS, ES, FS или GS.
#UD, если CR0.EM=1.
#NM, если CR0.TS=1.
#MF, если обрабатывается исключение FPU.

Особые случаи V-режима

Такие же, как в R-режиме.
#PF(код-нарушение), при страничном нарушении.
#AC(0), при невыравненном обращении к памяти, если CPL=3 и данное исключение разрешено.

PCMPEQB/PCMPEQW/PCMPEQD – Packed Compare for Equal

Код	*Команда*	*Описание*
0F 74 /r	PCMPEQB mm, mm/m64	Сравнивает на равенство упакованные байты в MMX-регистре или памяти с упакованными байтами в MMX-регистре
0F 75 /r	PCMPEQW mm, mm/m64	Сравнивает на равенство упакованные слова в MMX-регистре или памяти с упакованными словами в MMX-регистре
0F 76 /r	PCMPEQD mm, mm/m64	Сравнивает на равенство упакованные двойные слова в MMX-регистре или памяти с упакованными двойными словами в MMX-регистре

Операция

```
IF instruction is PCMPEQB
THEN {
    IF mm(7...0) = mm/m64(7...0)
    THEN mm(7 0) ← 0xFF;
    ELSE mm(7...0) ← 0;
    IF mm(15...8) = mm/m64(15... 8)
    THEN mm(15...8) ← 0xFF;
    ELSE mm(15...8) ← 0;
     ...
    IF mm(63...56) = mm/m64(63...56)
    THEN mm(63...56) ← 0xFF;
    ELSE mm(63...56) ← 0;
    }
ELSE IF instruction is PCMPEQW
THEN {
    IF mm(15...0) = mm/m64(15...0)
    THEN mm(15...0) ← 0xFFFF;
    ELSE mm(15...0) ← 0;
    IF mm(31...16) = mm/m64(31...16)
    THEN mm(31...16) ← 0xFFFF;
    ELSE mm(31...16) ← 0;
     ...
    IF mm(63...48) = mm/m64(63...48)
    THEN mm(63...48) ← 0xFFFF;
    ELSE mm(63...48) ← 0;
    }
ELSE { (* instruction is PCMPEQD *)
```

```
IF mm(31...0) = mm/m64(31...0)
THEN mm(31...0) ← 0xFFFFFFFF;
ELSE mm(31...0) ← 0;
IF mm(63...32) = mm/m64(63...32)
THEN mm(63...32) ← 0xFFFFFFFF;
ELSE mm(63...32) ← 0;
}
```

Описание

Команда PCMPEQ сравнивает элементы операнда-приемника с соответствующими элементами операнда-источника. Если значения элементов равны между собой, то в соответствующий элемент регистра-приемника записываются все единицы, в противном случае записываются все нули.

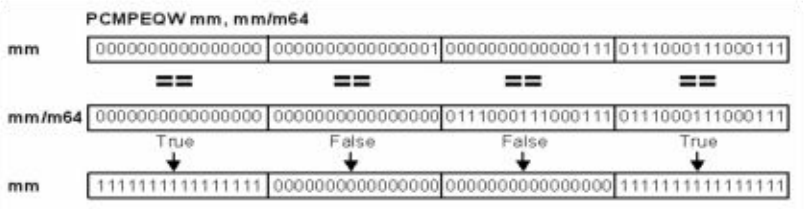
Операндом-приемником является MMX-регистр. Операндом-источником может быть как MMX-регистр, так и операнд в памяти.

Команда PCMPEQB сравнивает байты операнда-приемника с байтами операнда-источника и устанавливает байты операнда-приемника в соответствии с результатом.

Команда PCMPEQW сравнивает слова операнда-приемника со словами операнда-источника и устанавливает слова операнда-приемника в соответствии с результатом.

Команда PCMPEQD сравнивает двойные слова операнда-приемника с двойными словами операнда-источника и устанавливает двойные слова операнда-приемника в соответствии с результатом.

Пример



Особые случаи P-режима

#GP(0), если эффективный адрес операнда выходит за пределы сегмента CS, DS, ES, FS или GS. Если DS, ES, FS или GS равны нулю.
#SS(0), если эффективный адрес операнда выходит за пределы сегмента SS.
#PF(код-нарушение), при страничном нарушении.
#AC(0), при невыравненном обращении к памяти, если CPL=3 и данное исключение разрешено.
#UD, если CR0.EM=1.

#NM, если CR0.TS=1.
#MF, если обрабатывается исключение FPU.

Особые случаи R-режима

#GP(0), если эффективный адрес операнда выходит за пределы сегмента CS, DS, ES, FS или GS.
#UD, если CR0.EM=1.
#NM, если CR0.TS=1.
#MF, если обрабатывается исключение FPU.

Особые случаи V-режима

Такие же, как в R-режиме.
#PF(код-нарушение), при страничном нарушении.
#AC(0), при невыравненном обращении к памяти, если CPL=3 и данное исключение разрешено.

PCMPGTB/PCMPGTW/PCMPGTD – Packed Compare for Greater Than

Код	*Команда*	*Описание*
0F 64 /r	PCMPGTB mm, mm/m64	Сравнивает на "больше" упакованные байты в MMX-регистре или памяти с упакованными байтами в MMX-регистре
0F 65 /r	PCMPGTW mm, mm/m64	Сравнивает на "больше" упакованные слова в MMX-регистре или памяти с упакованными словами в MMX-регистре
0F 66 /r	PCMPGTD mm, mm/m64	Сравнивает на "больше" упакованные двойные слова в MMX-регистре или памяти с упакованными двойными словами в MMX-регистре

Операция

```
IF instruction is PCMPGTB
THEN {
    IF mm(7...0) > mm/m64(7...0)
    THEN mm(7 0) ← 0xFF;
    ELSE mm(7...0) ← 0;
    IF mm(15...8) > mm/m64(15... 8)
    THEN mm(15...8) ← 0xFF;
    ELSE mm(15...8) ← 0;
     ...
    IF mm(63...56) > mm/m64(63...56)
    THEN mm(63...56) ← 0xFF;
    ELSE mm(63...56) ← 0;
    }
ELSE IF instruction is PCMPGTW
THEN {
    IF mm(15...0) > mm/m64(15...0)
    THEN mm(15...0) ← 0xFFFF;
```

```
    ELSE mm(15...0) ←0;
    IF mm(31...16) > mm/m64(31...16)
    THEN mm(31...16) ← 0xFFFF;
    ELSE mm(31...16) ← 0;
     ...
    IF mm(63...48) > mm/m64(63...48)
    THEN mm(63...48) ← 0xFFFF;
    ELSE mm(63...48) ← 0;
    }
ELSE { (* instruction is PCMPGTD *)
    IF mm(31...0) > mm/m64(31...0)
    THEN mm(31...0) ← 0xFFFFFFFF;
    ELSE mm(31...0) ← 0; IF mm(63...32) > mm/m64(63...32)
    THEN mm(63...32) ← 0xFFFFFFFF;
    ELSE mm(63...32) ← 0;
    }
```

Описание

Команда PCMPGT сравнивает знаковые элементы операнда-приемника с соответствующими знаковыми элементами операнда-источника. Если значение элемента приемника больше, чем соответствующее значение элемента источника, то в соответствующий элемент регистра-приемника записываются все единицы, в противном случае записываются все нули.

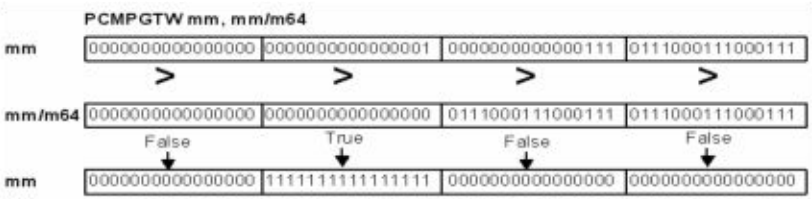
Операндом-приемником является MMX-регистр. Операндом-источником может быть как MMX-регистр, так и операнд в памяти.

Команда PCMPGTB сравнивает знаковые байты операнда-приемника со знаковыми байтами операнда-источника и устанавливает байты операнда-приемника в соответствии с результатом.

Команда PCMPGTW сравнивает знаковые слова операнда-приемника со знаковыми словами операнда-источника и устанавливает слова операнда-приемника в соответствии с результатом.

Команда PCMPGTD сравнивает знаковые двойные слова операнда-приемника со знаковыми двойными словами операнда-источника и устанавливает двойные слова операнда-приемника в соответствии с результатом.

Пример



Особые случаи P-режима

#GP(0), если эффективный адрес операнда выходит за пределы сегмента CS, DS, ES, FS или GS. Если DS, ES, FS или GS равны нулю.
#SS(0), если эффективный адрес операнда выходит за пределы сегмента SS.
#PF(код-нарушение), при страничном нарушении.
#AC(0), при невыравненном обращении к памяти, если CPL=3 и данное исключение разрешено.
#UD, если CR0.EM=1.
#NM, если CR0.TS=1.
#MF, если обрабатывается исключение FPU.

Особые случаи R-режима

#GP(0), если эффективный адрес операнда выходит за пределы сегмента CS, DS, ES, FS или GS.
#UD, если CR0.EM=1.
#NM, если CR0.TS=1.
#MF, если обрабатывается исключение FPU.

Особые случаи V-режима

Такие же, как в R-режиме.
#PF(код-нарушение), при страничном нарушении.
#AC(0), при невыравненном обращении к памяти, если CPL=3 и данное исключение разрешено.

PMADDWD – Packed Multiply and Add

Код	*Команда*	*Описание*
0F F5 /r	PMADDWD mm, mm/m64	Умножает упакованные слова в MMX-регистре на упакованные слова в MMX-регистре или памяти. Складывает попарно промежуточные 32-разрядные результаты и записывает результат в виде двух двойных слов в MMX-регистр

Операция

mm(31...0) ← mm(15...0) * mm/m64(15...0) + mm(31...16) * mm/m64(31...16);
mm(63...32) ← mm(47...32) * mm/m64(47...32) + mm(63...48) * mm/m64(63...48);

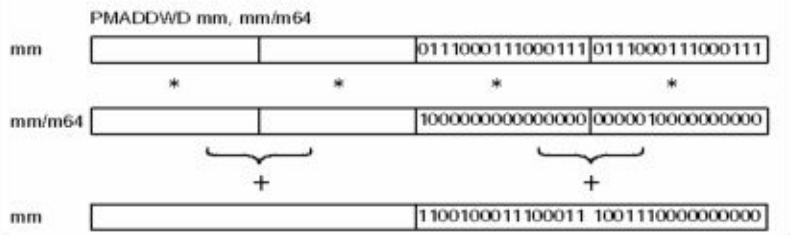
Описание

Команда PMADDWD умножает 4 знаковых слова операнда-приемника на 4 знаковых слова операнда-источника. В результате получается 4 двойных слова. Затем два верхних двойных слова суммируются между собой и результат записывается в верхнее двойное слово операнда-приемника, два нижних двойных слова также суммируются между собой, и результат записывается в нижнее двойное слово операнда-приемника.

Операндом-приемником является MMX-регистр. Операндом-источником может быть как MMX-регистр, так и операнд в памяти.

В результате выполнения команды PMADD произойдет циклический перенос, если все 4 слова обоих операндов равны 0x8000.

Пример



Особые случаи P-режима

#GP(0), если эффективный адрес операнда выходит за пределы сегмента CS, DS, ES, FS или GS. Если DS, ES, FS или GS равны нулю.
#SS(0), если эффективный адрес операнда выходит за пределы сегмента SS.
#PF(код-нарушение), при страничном нарушении.
#AC(0), при невыравненном обращении к памяти, если CPL=3 и данное исключение разрешено.
#UD, если CR0.EM=1.
#NM, если CR0.TS=1.
#MF, если обрабатывается исключение FPU.

Особые случаи R-режима

#GP(0), если эффективный адрес операнда выходит за пределы сегмента CS, DS, ES, FS или GS.
#UD, если CR0.EM=1.
#NM, если CR0.TS=1.
#MF, если обрабатывается исключение FPU.

Особые случаи V-режима

Такие же, как в R-режиме.
#PF(код-нарушение), при страничном нарушении.
#AC(0), при невыравненном обращении к памяти, если CPL=3 и данное исключение разрешено.

PMULHW – Packed Multiply High

Код	*Команда*	*Описание*
0F E5 /r	PMULHW mm, mm/m64	Умножает упакованные знаковые слова в MMX-регистре на упакованные знаковые слова в MMX-регистре или памяти и возвращает старшие 16 бит результата в MMX-регистре

Операция

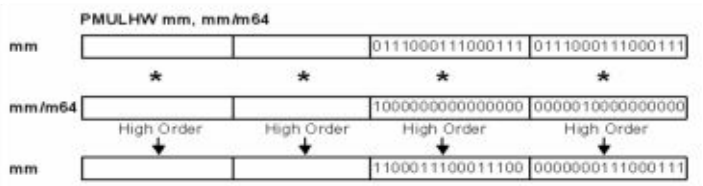
mm(15...0) ← (mm(15...0) * mm/m64(15...0)) (31...16);
mm(31...16)← (mm(31...16) * mm/m64(31...16)) (31...16);
mm(47...32) ← (mm(47...32) * mm/m64(47...32)) (31...16);
mm(63...48) ← (mm(63...48) * mm/m64(63...48)) (31...16);

Описание

Команда PMULHW умножает 4 знаковых слова операнда-приемника на 4 знаковых слова операнда-источника. Старшие 16 бит 32-разрядного результата записываются в соответствующие слова операнда-приемника.

Операндом-приемником является MMX-регистр. Операндом-источником может быть как MMX-регистр, так и операнд в памяти.

Пример



Особые случаи P-режима

#GP(0), если эффективный адрес операнда выходит за пределы сегмента CS, DS, ES, FS или GS. Если DS, ES, FS или GS равны нулю.
#SS(0), если эффективный адрес операнда выходит за пределы сегмента SS.
#PF(код-нарушение), при страничном нарушении.
#AC(0), при невыравненном обращении к памяти, если CPL=3 и данное исключение разрешено.
#UD, если CR0.EM=1.
#NM, если CR0.TS=1.
#MF, если обрабатывается исключение FPU.

Особые случаи R-режима

#GP(0), если эффективный адрес операнда выходит за пределы сегмента CS, DS, ES, FS или GS.
#UD, если CR0.EM=1.
#NM, если CR0.TS=1.

#MF, если обрабатывается исключение FPU.

Особые случаи V-режима

Такие же, как в R-режиме.
#PF(код-нарушение), при страничном нарушении.
#AC(0), при невыравненном обращении к памяти, если CPL=3 и данное исключение разрешено.

PMULLW – Packed Multiply Low

Код	*Команда*	*Описание*
0F D5 /r	PMULLW mm, mm/m64	Умножает упакованные знаковые слова в MMX-регистре на упакованные знаковые слова в MMX-регистре или памяти и возвращает младшие 16 бит результата в MMX-регистре

Операция

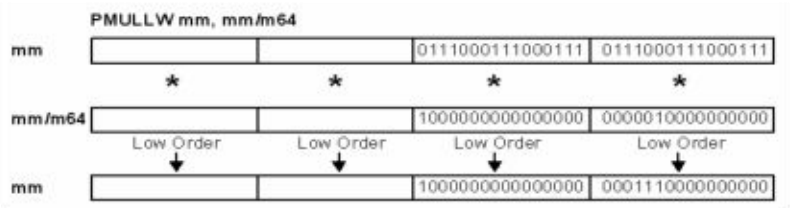
mm(15...0) ← (mm(15...0) * mm/m64(15...0)) (15...0);
mm(31...16) ← (mm(31...16) * mm/m64(31...16)) (15...0);
mm(47...32) ← (mm(47...32) * mm/m64(47...32)) (15...0);
mm(63...48) ← (mm(63...48) * mm/m64(63...48)) (15...0);

Описание

Команда PMULLW умножает 4 знаковых слова операнда-приемника на 4 знаковых слова операнда-источника. Младшие 16 бит 32-разрядного результата записываются в соответствующие слова операнда-приемника.

Операндом-приемником является MMX-регистр. Операндом-источником может быть как MMX-регистр, так и операнд в памяти.

Пример



Особые случаи P-режима

#GP(0), если эффективный адрес операнда выходит за пределы сегмента CS, DS, ES, FS или GS. Если DS, ES, FS или GS равны нулю.
#SS(0), если эффективный адрес операнда выходит за пределы сегмента SS.
#PF(код-нарушение), при страничном нарушении.
#AC(0), при невыравненном обращении к памяти, если CPL=3 и данное исключение разрешено.

#UD, если CR0.EM=1.
#NM, если CR0.TS=1.
#MF, если обрабатывается исключение FPU.

Особые случаи R-режима

#GP(0), если эффективный адрес операнда выходит за пределы сегмента CS, DS, ES, FS или GS.
#UD, если CR0.EM=1.
#NM, если CR0.TS=1.
#MF, если обрабатывается исключение FPU.

Особые случаи V-режима

Такие же, как в R-режиме.
#PF(код-нарушение), при страничном нарушении.
#AC(0), при невыравненном обращении к памяти, если CPL=3 и данное исключение разрешено.

POR – Bitwise Logical Or

Код	*Команда*	*Описание*
0F EB /r	POR mm, mm/m64	Логическое ИЛИ MMX-регистра или памяти с MMX-регистром

Операция

mm ← mm OR mm/m64;

Описание

Команда POR производит побитную операцию логическое ИЛИ над 64 битами операнда-источника и операнда-приемника. Результат операции записывается в операнд-приемник.

Каждый бит результата устанавливается в 0, если соответствующие биты операндов равны нулю, в противном случае бит результата устанавливается в единицу.

Операндом-приемником является MMX-регистр. Операндом-источником может быть как MMX-регистр, так и операнд в памяти.

Пример

Особые случаи P-режима

#GP(0), если эффективный адрес операнда выходит за пределы сегмента CS, DS, ES, FS или GS. Если DS, ES, FS или GS равны нулю.
#SS(0), если эффективный адрес операнда выходит за пределы сегмента SS.

#PF(код-нарушение), при страничном нарушении.
#AC(0), при невыравненном обращении к памяти, если CPL=3 и данное исключение разрешено.
#UD, если CR0.EM=1.
#NM, если CR0.TS=1.
#MF, если обрабатывается исключение FPU.

Особые случаи R-режима

#GP(0), если эффективный адрес операнда выходит за пределы сегмента CS, DS, ES, FS или GS.
#UD, если CR0.EM=1.
#NM, если CR0.TS=1.
#MF, если обрабатывается исключение FPU.

Особые случаи V-режима

Такие же, как в R-режиме.
#PF(код-нарушение), при страничном нарушении.
#AC(0), при невыравненном обращении к памяти, если CPL=3 и данное исключение разрешено.

PSLLW/PSLLD/PSLLQ – Packed Shift Left Logical

Код	*Команда*	*Описание*
0F F1 /r	PSLLW mm, mm/m64	Логический сдвиг слов MMX-регистра влево на число позиций, указанное в MMX-регистре или памяти
0F 71 /6 ib	PSLLW mm, imm8	Логический сдвиг слов MMX-регистра влево на число позиций imm8
0F F2 /r	PSLLD mm, mm/m64	Логический сдвиг двойных слов MMX-регистра влево на число позиций, указанное в MMX-регистре или памяти
0F 72 /6 ib	PSLLD mm, imm8	Логический сдвиг двойных слов MMX-регистра влево на число позиций imm8
0F F3 /r	PSLLQ mm, mm/m64	Логический сдвиг MMX-регистра влево на число позиций, указанное в MMX-регистре или памяти
0F 73 /6 ib	PSLLQ mm, imm8	Логический сдвиг MMX-регистра влево на число позиций imm8

Операция

```
IF the second operand is imm8
THEN
    temp ← imm8;
  ELSE (* second operand is mm/m64 *)
    temp ← mm/m64;
  IF instruction is PSLLW
  THEN {
    mm(15...0) ← mm(15...0) << temp;
    mm(31...16) ← mm(31...16) << temp;
```

```
    mm(47...32) ← mm(47...32) << temp;
    mm(63...48) ← mm(63...48) << temp;
    }
  ELSE IF instruction is PSLLD
  THEN {
    mm(31...0) ← mm(31...0) << temp;
    mm(63...32) ← mm(63...32) << temp;
    }
  ELSE (* instruction is PSLLQ *)
    mm ← mm << temp;
```

Описание

Команда PSLL производит сдвиг влево битов первого операнда на число позиций, указанное во втором операнде. Результат сдвига записывается в регистр-приемник. Пустые младшие биты каждого элемента заполняются нулями. Если значение второго операнда больше 15 (для слов), 31 (для двойных слов) или 63 (для учетверенного слова), то результат обращается в нуль.

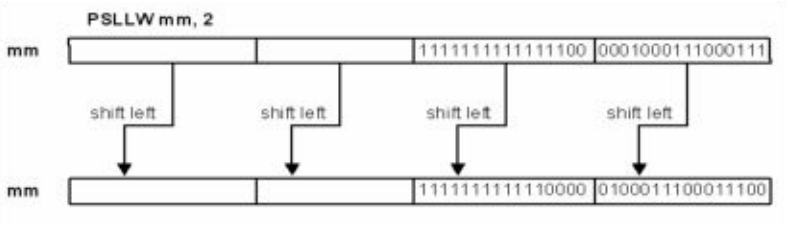
Операндом-приемником является MMX-регистр. Операндом-источником может быть MMX-регистр, 64-разрядный операнд в памяти или непосредственное 8-разрядное значение.

Команда PSLLW сдвигает каждое из четырех слов регистра-приемника влево на число позиций, указанное в операнде-источнике. Младшие биты каждого слова заполняются нулями.

Команда PSLLD сдвигает каждое из двух двойных слов регистра-приемника влево на число позиций, указанное в операнде-источнике. Младшие биты каждого двойного слова заполняются нулями.

Команда PSLLQ сдвигает учетверенное слово регистра-приемника влево на число позиций, указанное в операнде-источнике. Младшие биты заполняются нулями.

Пример



Особые случаи P-режима

#GP(0), если эффективный адрес операнда выходит за пределы сегмента CS, DS, ES, FS или GS. Если DS, ES, FS или GS равны нулю.

#SS(0), если эффективный адрес операнда выходит за пределы сегмента SS.
#PF(код-нарушение), при страничном нарушении.
#AC(0), при невыравненном обращении к памяти, если CPL=3 и данное исключение разрешено.
#UD, если CR0.EM=1.
#NM, если CR0.TS=1.
#MF, если обрабатывается исключение FPU.

Особые случаи R-режима

#GP(0), если эффективный адрес операнда выходит за пределы сегмента CS, DS, ES, FS или GS.
#UD, если CR0.EM=1.
#NM, если CR0.TS=1.
#MF, если обрабатывается исключение FPU.

Особые случаи V-режима

Такие же, как в R-режиме.
#PF(код-нарушение), при страничном нарушении.
#AC(0), при невыравненном обращении к памяти, если CPL=3 и данное исключение разрешено.

PSRAW/PSRAD – Packed Shift Right Arithmetic

Код	*Команда*	*Описание*
0F E1 /r	PSRAW mm, mm/m64	Арифметический сдвиг слов MMX-регистра влево на число позиций, указанное в MMX-регистре или памяти
0F 71 /4 ib	PSRAW mm, imm8	Арифметический сдвиг слов MMX-регистра влево на число позиций imm8
0F E2 /r	PSRAD mm, mm/m64	Арифметический сдвиг двойных слов MMX-регистра влево на число позиций, указанное в MMX-регистре или памяти
0F 72 /4 ib	PSRAD mm, imm8	Арифметический сдвиг двойных слов MMX-регистра влево на число позиций imm8

Операция

```
IF the second operand is imm8
THEN
    temp ← imm8;
ELSE (* second operand is mm/m64 *)
    temp ← mm/m64;
IF instruction is PSRAW
THEN {
    mm(15...0) ← SignExtend (mm(15...0) >>temp);
    mm(31...16) ← SignExtend (mm(31...16) >> temp);
    mm(47...32) ← SignExtend (mm(47...32) >> temp);
    mm(63...48) ← SignExtend (mm(63...48) >> temp);
    }
ELSE { (*instruction is PSRAD *)
```

```
mm(31...0) ← SignExtend (mm(31...0) >> temp);
mm(63...32) ← SignExtend (mm(63...32) >> temp);
}
```

Описание

Команда PSRA производит сдвиг вправо битов первого операнда на число позиций, указанное во втором операнде. Результат сдвига записывается в регистр-приемник. Пустые старшие биты каждого элемента заполняются знаковым битом. Если значение второго операнда больше 15 (для слов), 31 (для двойных слов) или 63 (для учетверенного слова), то весь элемент будет заполнен знаковым битом.

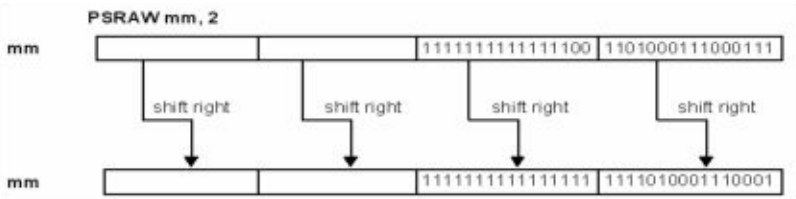
Операндом-приемником является MMX-регистр. Операндом-источником может быть MMX-регистр, 64-разрядный операнд в памяти или непосредственное 8-разрядное значение.

Команда PSLLW сдвигает каждое из четырех слов регистра-приемника влево на число позиций, указанное в операнде-источнике. Младшие биты каждого слова заполняются нулями.

Команда PSLLD сдвигает каждое из двух двойных слов регистра-приемника влево на число позиций, указанное в операнде-источнике. Младшие биты каждого двойного слова заполняются нулями.

Команда PSLLQ сдвигает учетверенное слово регистра-приемника влево на число позиций, указанное в операнде-источнике. Младшие биты заполняются нулями.

Пример



Особые случаи P-режима

#GP(0), если эффективный адрес операнда выходит за пределы сегмента CS, DS, ES, FS или GS. Если DS, ES, FS или GS равны нулю.
#SS(0), если эффективный адрес операнда выходит за пределы сегмента SS.
#PF(код-нарушение), при страничном нарушении.
#AC(0), при невыравненном обращении к памяти, если CPL=3 и данное исключение разрешено.
#UD, если CR0.EM=1.
#NM, если CR0.TS=1.
#MF, если обрабатывается исключение FPU.

Особые случаи R-режима

#GP(0), если эффективный адрес операнда выходит за пределы сегмента CS, DS, ES, FS или GS.
#UD, если CR0.EM=1.
#NM, если CR0.TS=1.
#MF, если обрабатывается исключение FPU.

Особые случаи V-режима

Такие же, как в R-режиме.
#PF(код-нарушение), при страничном нарушении.
#AC(0), при невыравненном обращении к памяти, если CPL=3 и данное исключение разрешено.

PSRLW/PSRLD/PSRLQ – Packed Shift Right Logical

Код	*Команда*	*Описание*
0F D1 /r	PSRLW mm, mm/m64	Логический сдвиг слов MMX-регистра вправо на число позиций, указанное в MMX-регистре или памяти
0F 71 /2 ib	PSRLW mm, imm8	Логический сдвиг слов MMX-регистра вправо на число позиций imm8
0F D2 /r	PSRLD mm, mm/m64	Логический сдвиг двойных слов MMX-регистра вправо на число позиций, указанное в MMX-регистре или памяти
0F 72 /2 ib	PSRLD mm, imm8	Логический сдвиг двойных слов MMX-регистра вправо на число позиций imm8
0F D3 /r	PSRLQ mm, mm/m64	Логический сдвиг MMX-регистра вправо на число позиций, указанное в MMX-регистре или памяти
0F 73 /2 ib	PSRLQ mm, imm8	Логический сдвиг MMX-регистра вправо на число позиций imm8

Операция

```
IF the second operand is imm8
THEN
    temp ← imm8;
ELSE (* second operand is mm/m64 *)
    temp ← mm/m64;
IF instruction is PSRLW
THEN {
    mm(15...0) ← mm(15...0) >> temp;
    mm(31...16) ← mm(31...16) >> temp;
    mm(47...32) ← mm(47...32) >> temp;
    mm(63...48) ← mm(63...48) >> temp;
    }
ELSE IF instruction is PSRLD
THEN {
    mm(31...0) ← mm(31...0) >> temp;
    mm(63...32) ← mm(63...32) >> temp;
```

```
    }
ELSE (* instruction is PSRLQ *)
    mm ← mm >> temp;
```

Описание

Команда PSRL производит сдвиг вправо битов первого операнда на число позиций, указанное во втором операнде. Результат сдвига записывается в регистр-приемник. Пустые старшие биты каждого элемента заполняются нулями. Если значение второго операнда больше 15 (для слов), 31 (для двойных слов) или 63 (для учетверенного слова), то результат обращается в 0.

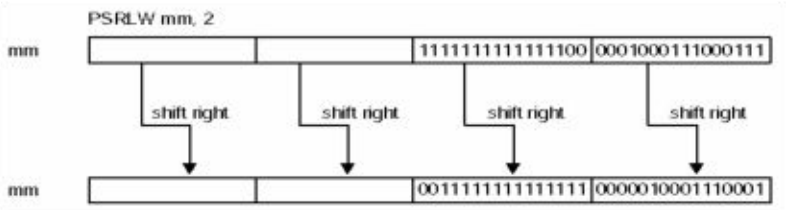
Операндом-приемником является MMX-регистр. Операндом-источником может быть MMX-регистр, 64-разрядный операнд в памяти или непосредственное 8-разрядное значение.

Команда PSRLW сдвигает каждое из четырех слов регистра-приемника вправо на число позиций, указанное в операнде-источнике. Младшие биты каждого слова заполняются нулями.

Команда PSRLD сдвигает каждое из двух двойных слов регистра-приемника вправо на число позиций, указанное в операнде-источнике. Младшие биты каждого двойного слова заполняются нулями.

Команда PSRLQ сдвигает учетверенное слово регистра-приемника вправо на число позиций, указанное в операнде-источнике. Младшие биты заполняются нулями.

Пример



Особые случаи P-режима

#GP(0), если эффективный адрес операнда выходит за пределы сегмента CS, DS, ES, FS или GS. Если DS, ES, FS или GS равны нулю.
#SS(0), если эффективный адрес операнда выходит за пределы сегмента SS.
#PF(код-нарушение), при страничном нарушении.
#AC(0), при невыравненном обращении к памяти, если CPL=3 и данное исключение разрешено.
#UD, если CR0.EM=1.
#NM, если CR0.TS=1.
#MF, если обрабатывается исключение FPU.

Особые случаи R-режима

#GP(0), если эффективный адрес операнда выходит за пределы сегмента CS, DS, ES, FS или GS.
#UD, если CR0.EM=1.
#NM, если CR0.TS=1.
#MF, если обрабатывается исключение FPU.

Особые случаи V-режима

Такие же, как в R-режиме.
#PF(код-нарушение), при страничном нарушении.
#AC(0), при невыравненном обращении к памяти, если CPL=3 и данное исключение разрешено.

PSUBB/PSUBW/PSUBD – Packed Subtract

Код	*Команда*	*Описание*
0F F8 /r	PSUBB mm, mm/m64	Вычитает упакованные байты в MMX-регистре или памяти из упакованных байтов в MMX-регистре
0F F9 /r	PSUBW mm, mm/m64	Вычитает упакованные слова в MMX-регистре или памяти из упакованных слов в MMX-регистре
0F FA /r	PSUBD mm, mm/m64	Вычитает упакованные двойные слова в MMX-регистре или памяти из упакованных двойных слов в MMX-регистре

Операция

```
IF instruction is PSUBB
THEN {
    mm(7...0) ← mm(7...0) – mm/m64(7...0);
    mm(15...8) ← mm(15...8) – mm/m64(15...8);
    mm(23...16) ← mm(23...16) – mm/m64(23...16);
    mm(31...24) ← mm(31...24) – mm/m64(31...24);
    mm(39...32) ← mm(39...32) – mm/m64(39...32);
    mm(47...40) ← mm(47...40) – mm/m64(47...40);
    mm(55...48) ← mm(55...48) – mm/m64(55...48);
    mm(63...56) ← mm(63...56) – mm/m64(63...56);
    }
IF instruction is PSUBW
THEN {
    mm(15...0) ← mm(15...0) – mm/m64(15...0);
    mm(31...16) ← mm(31...16) – mm/m64(31...16);
    mm(47...32) ← mm(47...32) – mm/m64(47...32);
    mm(63...48) ← mm(63...48) – mm/m64(63...48);
    }
ELSE { (* instruction is PSUBD *)
    mm(31...0) ← mm(31...0) – mm/m64(31...0);
```

mm(63...32) ← mm(63...32) – mm/m64(63...32);
}

Описание

Команда PSUB производит вычитание элементов операнда-источника из элементов операнда-приемника, используя принцип циклического переноса. Результат операции записывается в регистр-приемник.

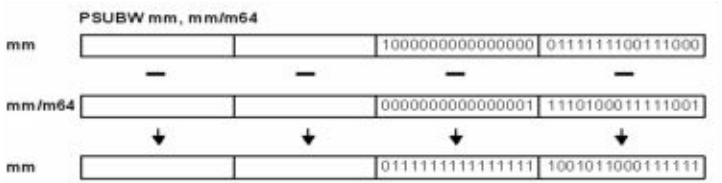
Операндом-приемником является MMX-регистр. Операндом-источником может быть как MMX-регистр, так и операнд в памяти.

Команда PSUBB производит вычитание байтов.

Команда PSUBW производит вычитание слов.

Команда PSUBD производит вычитание двойных слов.

Пример



Особые случаи P-режима

#GP(0), если эффективный адрес операнда выходит за пределы сегмента CS, DS, ES, FS или GS. Если DS, ES, FS или GS равны нулю.
#SS(0), если эффективный адрес операнда выходит за пределы сегмента SS.
#PF(код-нарушение), при страничном нарушении.
#AC(0), при невыравненном обращении к памяти, если CPL=3 и данное исключение разрешено.
#UD, если CR0.EM=1.
#NM, если CR0.TS=1.
#MF, если обрабатывается исключение FPU.

Особые случаи R-режима

#GP(0), если эффективный адрес операнда выходит за пределы сегмента CS, DS, ES, FS или GS.
#UD, если CR0.EM=1.
#NM, если CR0.TS=1.
#MF, если обрабатывается исключение FPU.

Особые случаи V-режима

Такие же, как в R-режиме.
#PF(код-нарушение), при страничном нарушении.

#AC(0), при невыравненном обращении к памяти, если CPL=3 и данное исключение разрешено.

PSUBSB/PSUBSW – Packed Subtract with Saturation

Код	Команда	Описание
0F E8 /r	PSUBSB mm, mm/m64	Вычитает с насыщением упакованные знаковые байты в MMX-регистре или памяти из упакованных знаковых байтов в MMX-регистре
0F E9 /r	PSUBSW mm, mm/m64	Вычитает с насыщением упакованные знаковые слова в MMX-регистре или памяти из упакованных знаковых слов в MMX-регистре

Операция

```
IF instruction is PSUBSB
THEN{
    mm(7...0) ← SaturateToSignedByte (mm(7...0) – mm/m64 (7...0));
    mm(15...8) ← SaturateToSignedByte (mm(15...8) – mm/m64(15...8));
    mm(23...16) ← SaturateToSignedByte (mm(23...16) – mm/m64(23...16));
    mm(31...24) ← SaturateToSignedByte (mm(31...24) – mm/m64(31...24));
    mm(39...32) ← SaturateToSignedByte (mm(39...32) – mm/m64(39...32));
    mm(47...40) ← SaturateToSignedByte (mm(47...40) – mm/m64(47...40));
    mm(55...48) ← SaturateToSignedByte (mm(55...48) – mm/m64(55...48));
    mm(63...56) ← SaturateToSignedByte (mm(63...56) – mm/m64(63...56))
    }
ELSE { (* instruction is PSUBSW *)
    mm(15...0) ← SaturateToSignedWord (mm(15...0) – mm/m64(15...0));
    mm(31...16) ← SaturateToSignedWord (mm(31...16) – mm/m64(31...16));
    mm(47...32) ← SaturateToSignedWord (mm(47...32) – mm/m64(47...32));
    mm(63...48) ← SaturateToSignedWord (mm(63...48) – mm/m64(63...48));
    }
```

Описание

Команда PSUBS производит вычитание знаковых элементов операнда-источника из знаковых элементов операнда-приемника, используя принцип насыщения. Результат операции записывается в регистр-приемник.

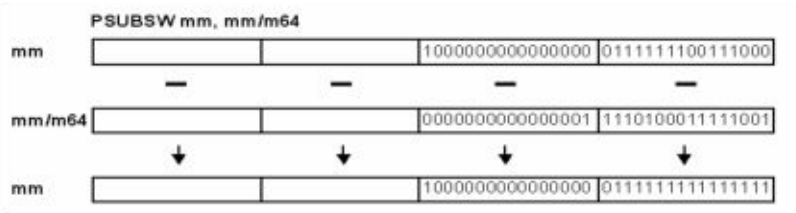
Операндом-приемником является MMX-регистр. Операндом-источником может быть как MMX-регистр, так и операнд в памяти.

Команда PSUBB производит вычитание знаковых байтов. Если значение результата больше или меньше границ диапазона знакового байта, то результат операции насыщается соответственно до 0x7F или до 0x80.

Команда PSUBW производит вычитание знаковых слов. Если значение результата больше или меньше границ диапазона

знакового слова, то результат операции насыщается соответственно до 0x7FFF или до 0x8000.

Пример



Особые случаи P-режима

#GP(0), если эффективный адрес операнда выходит за пределы сегмента CS, DS, ES, FS или GS. Если DS, ES, FS или GS равны нулю.
#SS(0), если эффективный адрес операнда выходит за пределы сегмента SS.
#PF(код-нарушение), при страничном нарушении.
#AC(0), при невыравненном обращении к памяти, если CPL=3 и данное исключение разрешено.
#UD, если CR0.EM=1.
#NM, если CR0.TS=1.
#MF, если обрабатывается исключение FPU.

Особые случаи R-режима

#GP(0), если эффективный адрес операнда выходит за пределы сегмента CS, DS, ES, FS или GS.
#UD, если CR0.EM=1.
#NM, если CR0.TS=1.
#MF, если обрабатывается исключение FPU.

Особые случаи V-режима

Такие же, как в R-режиме.
#PF(код-нарушение), при страничном нарушении.
#AC(0), при невыравненном обращении к памяти, если CPL=3 и данное исключение разрешено.

PSUBUSB/PSUBUSW – Packed Subtract Unsigned with Saturation

Код	*Команда*	*Описание*
0F D8 /r	PSUBUSB mm, mm/m64	Вычитает с насыщением упакованные беззнаковые байты в MMX-регистре или памяти из упакованных беззнаковых байтов в MMX-регистре
0F D9 /r	PSUBUSW mm, mm/m64	Вычитает с насыщением упакованные беззнаковые слова в MMX-регистре или памяти из упакованных беззнаковых слов в MMX-регистре

Операция

```
IF instruction is PSUBUSB
THEN{
    mm(7...0) ← SaturateToUnsignedByte (mm(7...0) – mm/m64 (7...0) );
    mm(15...8) ← SaturateToUnsignedByte ( mm(15...8) – mm/m64(15...8) );
    mm(23...16) ← SaturateToUnsignedByte (mm(23...16) – mm/m64(23...16) );
    mm(31...24) ← SaturateToUnsignedByte (mm(31...24) – mm/m64(31...24) );
    mm(39...32) ← SaturateToUnsignedByte (mm(39...32) – mm/m64(39...32) );
    mm(47...40) ← SaturateToUnsignedByte (mm(47...40) – mm/m64(47...40) );
    mm(55...48) ← SaturateToUnsignedByte (mm(55...48) – mm/m64(55...48) );
    mm(63...56) ← SaturateToUnsignedByte (mm(63...56) – mm/m64(63...56) ); }
ELSE { (* instruction is PSUBUSW *)
    mm(15...0) ← SaturateToUnsignedWord (mm(15...0) – mm/m64(15...0) );
    mm(31...16) ← SaturateToUnsignedWord (mm(31...16) – mm/m64(31...16) );
    mm(47...32) ← SaturateToUnsignedWord (mm(47...32) – mm/m64(47...32) );
    mm(63...48) ← SaturateToUnsignedWord (mm(63...48) – mm/m64(63...48) );
    }
```

Описание

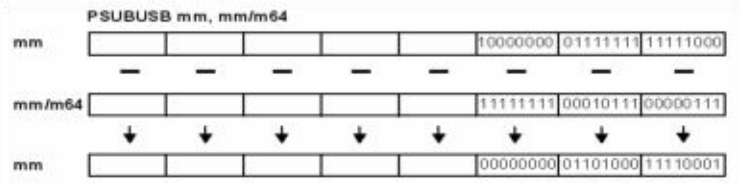
Команда PSUBS производит вычитание беззнаковых элементов операнда-источника из беззнаковых элементов операнда-приемника, используя принцип насыщения. Результат операции записывается в регистр-приемник.

Операндом-приемником является MMX-регистр. Операндом-источником может быть как MMX-регистр, так и операнд в памяти.

Команда PSUBUSB производит вычитание беззнаковых байтов. Если значение результата меньше нуля, то результат операции насыщается 0x00.

Команда PSUBUSW производит вычитание беззнаковых слов. Если значение результата меньше нуля, то результат операции насыщается 0x0000.

Пример



Особые случаи P-режима

#GP(0), если эффективный адрес операнда выходит за пределы сегмента CS, DS, ES, FS или GS. Если DS, ES, FS или GS равны нулю.
#SS(0), если эффективный адрес операнда выходит за пределы сегмента SS.

#PF(код-нарушение), при страничном нарушении.
#AC(0), при невыравненном обращении к памяти, если CPL=3 и данное исключение разрешено.
#UD, если CR0.EM=1.
#NM, если CR0.TS=1.
#MF, если обрабатывается исключение FPU.

Особые случаи R-режима

#GP(0), если эффективный адрес операнда выходит за пределы сегмента CS, DS, ES, FS или GS.
#UD, если CR0.EM=1.
#NM, если CR0.TS=1.
#MF, если обрабатывается исключение FPU.

Особые случаи V-режима

Такие же, как в R-режиме.
#PF(код-нарушение), при страничном нарушении.
#AC(0), при невыравненном обращении к памяти, если CPL=3 и данное исключение разрешено.

PUNPCKHBW/PUNPCKHWD/PUNPCKHDQ – Unpack High Packed Data

Код	*Команда*	*Описание*
0F 68 /r	PUNPCKHBW mm, mm/m64	Распаковывает байты из верхней половины MMX-регистра и верхней половины MMX-регистра/памяти в MMX-регистр
0F 69 /r	PUNPCKHWD mm, mm/m64	Распаковывает слова из верхней половины MMX-регистра и верхней половины MMX-регистра/памяти в MMX-регистр
0F 6A /r	PUNPCKHDQ mm, mm/m64	Распаковывает двойные слова из верхней половины MMX-регистра и верхней половины MMX-регистра/памяти в MMX-регистр

Операция

```
IF instruction is PUNPCKHBW
THEN {
    mm(63...56) ← mm/m64(63...56);
    mm(55...48) ← mm(63...56);
    mm(47...40) ← mm/m64(55...48);
    mm(39...32) ← mm(55...48);
    mm(31...24) ← mm/m64(47...40);
    mm(23...16) ← mm(47...40);
    mm(15...8) ← mm/m64(39...32);
    mm(7...0) ← mm(39...32);
ELSE IF instruction is PUNPCKHW
THEN {
    mm(63...48) ← mm/m64(63...48);
    mm(47...32) ← mm(63...48);
```

```
    mm(31...16) ← mm/m64(47...32);
    mm(15...0) ← mm(47...32);
    }
ELSE { (* instruction is PUNPCKHDQ *)
    mm(63...32) ← mm/m64(63...32);
    mm(31...0) ← mm(63...32)
    }
```

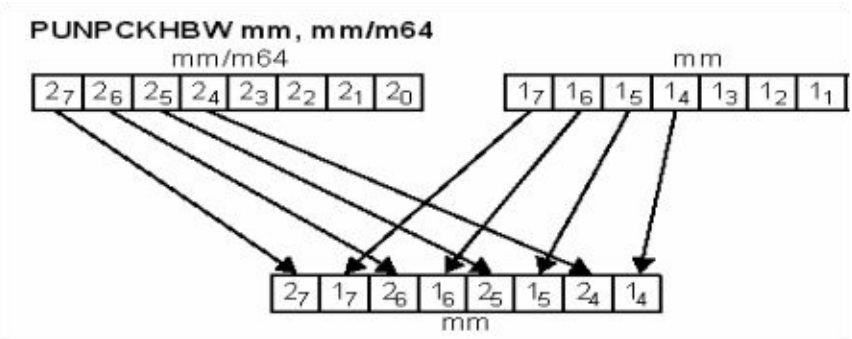
Описание

Команда PUNPCKH распаковывает верхние элементы операнда-источника и операнда-приемника в операнд-приемник. Элементы двух операндов записываются в результат через один, т. е. на нулевое место помещается восьмой/четвертый/второй элемент[1] операнда-приемника, на первое – восьмой/четвертый/второй элемент операнда-источника, на второе – девятый/пятый/третий элемент операнда-приемника, на третье – девятый/пятый/третий элемент операнда-источника и т. д.

Операндом-приемником является MMX-регистр. Операндом-источником может быть как MMX-регистр, так и операнд в памяти. Операнд в памяти является 64-разрядным, но используются только старшие 32 разряда.

Команда PUNPCKHBW распаковывает байты.
Команда PUNPCKHWD распаковывает слова.
Команда PUNPCKHDQ распаковывает двойные слова.

Пример



Особые случаи P-режима

#GP(0), если эффективный адрес операнда выходит за пределы сегмента CS, DS, ES, FS или GS. Если DS, ES, FS или GS равны нулю.
#SS(0), если эффективный адрес операнда выходит за пределы сегмента SS.

[1] *Элементами являются байты/слова/двойные слова*

#PF(код-нарушение), при страничном нарушении.
#AC(0), при невыравненном обращении к памяти, если CPL=3 и данное исключение разрешено.
#UD, если CR0.EM=1.
#NM, если CR0.TS=1.
#MF, если обрабатывается исключение FPU.

Особые случаи R-режима

#GP(0), если эффективный адрес операнда выходит за пределы сегмента CS, DS, ES, FS или GS.
#UD, если CR0.EM=1.
#NM, если CR0.TS=1.
#MF, если обрабатывается исключение FPU.

Особые случаи V-режима

Такие же, как в R-режиме.
#PF(код-нарушение), при страничном нарушении.
#AC(0), при невыравненном обращении к памяти, если CPL=3 и данное исключение разрешено.

PUNPCKLBW/PUNPCKLWD/PUNPCKLDQ – Unpack Low Packed Data

Код	*Команда*	*Описание*
0F 60 /r	PUNPCKLBW mm, mm/m32	Распаковывает байты из нижней половины MMX-регистра и нижней половины MMX-регистра/памяти в MMX-регистр
0F 61 /r	PUNPCKLWD mm, mm/m32	Распаковывает слова из нижней половины MMX-регистра и нижней половины MMX-регистра/памяти в MMX-регистр
0F 62 /r	PUNPCKLDQ mm, mm/m32	Распаковывает двойные слова из нижней половины MMX-регистра и нижней половины MMX-регистра/памяти в MMX-регистр

Операция

```
IF instruction is PUNPCKLBW
THEN {
    mm(63...56) ←mm/m32(31...24);
    mm(55...48) ← mm(31...24);
    mm(47...40) ← mm/m32(23...16);
    mm(39...32) ← mm(23...16);
    mm(31...24) ← mm/m32(15...8);
    mm(23...16) ←mm(15...8);
    mm(15...8) ← mm/m32(7...0);
    mm(7...0) ← mm(7...0);
    }
ELSE IF instruction is PUNPCKLWD
THEN {
```

```
    mm(63...48) ← mm/m32(31...16);
    mm(47...32) ← mm(31...16);
    mm(31...16) ← mm/m32(15...0);
    mm(15...0) ← mm(15...0);
    }
ELSE{ (* instruction is PUNPCKLDQ *)
    mm(63...32) ← mm/m32(31...0);
    mm(31...0) ← mm(31...0);
    }
```

Описание

Команда PUNPCKL распаковывает нижние элементы операнда-источника и операнда-приемника в операнд-приемник. Элементы двух операндов записываются в результат через один, т. е. на нулевое место помещается нулевой элемент операнда-приемника, на первое – нулевой элемент операнда-источника, на второе – третий элемент операнда-приемника, на третье – первый элемент операнда-источника и т. д.

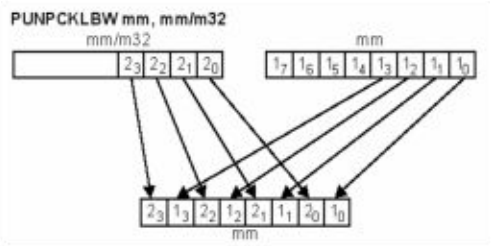
Операндом-приемником является MMX-регистр. Операндом-источником может быть как MMX-регистр так и операнд в памяти. Когда операндом-источником является MMX-регистр, то его верхние биты игнорируются.

Команда PUNPCKLBW распаковывает байты.

Команда PUNPCKLWD распаковывает слова.

Команда PUNPCKLDQ распаковывает двойные слова.

Пример



Особые случаи P-режима

#GP(0), если эффективный адрес операнда выходит за пределы сегмента CS, DS, ES, FS или GS. Если DS, ES, FS или GS равны нулю.
#SS(0), если эффективный адрес операнда выходит за пределы сегмента SS.
#PF(код-нарушение), при страничном нарушении.
#AC(0), при невыравненном обращении к памяти, если CPL=3 и данное исключение разрешено.
#UD, если CR0.EM=1.

#NM, если CR0.TS=1.
#MF, если обрабатывается исключение FPU.

Особые случаи R-режима

#GP(0), если эффективный адрес операнда выходит за пределы сегмента CS, DS, ES, FS или GS.
#UD, если CR0.EM=1.
#NM, если CR0.TS=1.
#MF, если обрабатывается исключение FPU.

Особые случаи V-режима

Такие же, как в R-режиме.
#PF(код-нарушение), при страничном нарушении.
#AC(0), при невыравненном обращении к памяти, если CPL=3 и данное исключение разрешено.

PXOR – Bitwise Logical Exclusive OR

Код	*Команда*	*Описание*
0F EF/r	PXOR mm, mm/m64	Логическое ИСКЛЮЧАЮЩЕЕ ИЛИ MMX-регистра или памяти с MMX-регистром

Операция

mm ← mm XOR mm/m64;

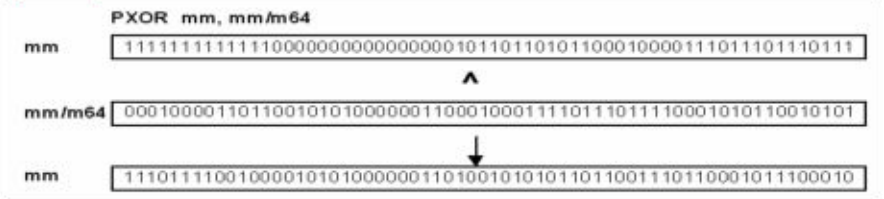
Описание

Команда PXOR производит побитную операцию логическое ИСКЛЮЧАЮЩЕЕ ИЛИ над 64 битами операнда-источника и операнда-приемника. Результат операции записывается в операнд-приемник.

Каждый бит результата устанавливается в единицу, если соответствующие биты операндов различны, в противном случае бит результата устанавливается в 0.

Операндом-приемником является MMX-регистр. Операндом-источником может быть как MMX-регистр, так и операнд в памяти.

Пример



Особые случаи P-режима

#GP(0), если эффективный адрес операнда выходит за пределы сегмента CS, DS, ES, FS или GS. Если DS, ES, FS или GS равны нулю.

#SS(0), если эффективный адрес операнда выходит за пределы сегмента SS.
#PF(код-нарушение), при страничном нарушении.
#AC(0), при невыравненном обращении к памяти, если CPL=3 и данное исключение разрешено.
#UD, если CR0.EM=1.
#NM, если CR0.TS=1.
#MF, если обрабатывается исключение FPU.

Особые случаи R-режима

#GP(0), если эффективный адрес операнда выходит за пределы сегмента CS, DS, ES, FS или GS.
#UD, если CR0.EM=1.
#NM, если CR0.TS=1.
#MF, если обрабатывается исключение FPU.

Особые случаи V-режима

Такие же, как в R-режиме.
#PF(код-нарушение), при страничном нарушении.
#AC(0), при невыравненном обращении к памяти, если CPL=3 и данное исключение разрешено.

5. ПРОЦЕССОРЫ СЕМЕЙСТВА P5

В этой главе рассматривается архитектура конвейера процессоров семейства P5 (Pentium и Pentium MMX), а также их новые возможности по сравнению с Intel486.

5.1. Новые возможности

В процессоры P5 были добавлены следующие команды:

⇒ команда CMPXCHG8B (compare and exchange 8 bytes);

⇒ команда CPUID (CPU identification);

⇒ команда RDTSC (read time-stamp counter);

⇒ команда RDMSR (read model-specific register);

⇒ команда WRMSR (write model-specific register);

⇒ команда RSM (resume from SSM).

Подробнее о новых командах см. гл. – "Новые команды P5 и P6".

Форма команды MOV, которая обращалась к регистрам тестирования, удалена из процессоров P5 и всех последующих. Функции регистров тестирования теперь выполняют регистры MSR (Model Specific Registers).

Задействован новый регистр управления CR4; подробнее см. в прил. Г – "Формат регистра CR4".

Введен новый класс регистров – регистры, специфические для конкретной модели процессора (MSR – Model Specific Registers). Эти регистры используются для тестирования внутренних устройств процессора, а также для других целей.

В регистр EFLAGS добавлены следующие флажки (рис. 5.1):

⇒ VIF (virtual interrupt flag), бит 19;

⇒ VIP (virtual interrupt pending), бит 20;

⇒ ID (identification flag), бит 21. Если программа способна устанавливать и сбрасывать этот флаг, то процессор поддерживает команду CPUID.

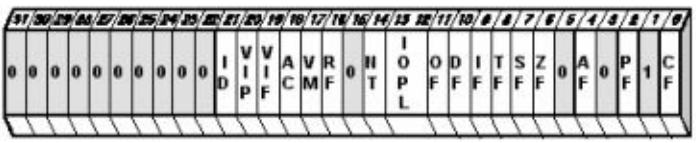


Рис. 5.1. Формат регистра EFLAGS в процессорах P5

Изменения в прерываниях:

При попытке записать единицу в зарезервированный бит специальных регистров генерируется исключение #GP – нарушение общей защиты.

При обнаружении единицы в зарезервированном бите элемента каталога страниц или элемента таблицы страниц генерируется исключение #PG – страничное нарушение.

Добавлено новое исключение #18 – Machine Check Exception. Это исключение предназначено для сообщения об аппаратных ошибках. Исключение является специфическим для данной модели процессора и может быть изменено в последующих моделях. Управление исключением осуществляется через MSR-регистры.

Новые функциональные возможности:

Добавлен новый режим работы – System Management Mode (SMM) (см. 11 – "Системный режим").

Добавлены новые возможности в механизм страничного преобразования (см. 8 – "Страничное преобразование в процессорах P5 и P6").

Добавлены новые возможности в механизм обработки прерываний (см. 9 – "Виртуальные прерывания").

5.2. Конвейер процессоров семейства P5

Конвейер Pentium (рис. 5.2) построен так, что позволяет выполнять одновременно до двух команд. Прозрачный для программ механизм предсказания ветвлений позволяет уменьшить задержки конвейера при переходах. В процессоре Pentium MMX в конвейер добавлены новые стадии. P5 может декодировать до двух инструкций за один такт и направлять их по двум логическим каналам. Будем называть их U-каналом и V-каналом. На этапе декодирования процессор проверяет, могут ли две команды выполняться параллельно. Если да, то первая команда направляется в U-канал конвейера, а вторая – в V-канал. В противном случае только одна команда направляется в U-канал и ничего не поступает в V-канал.

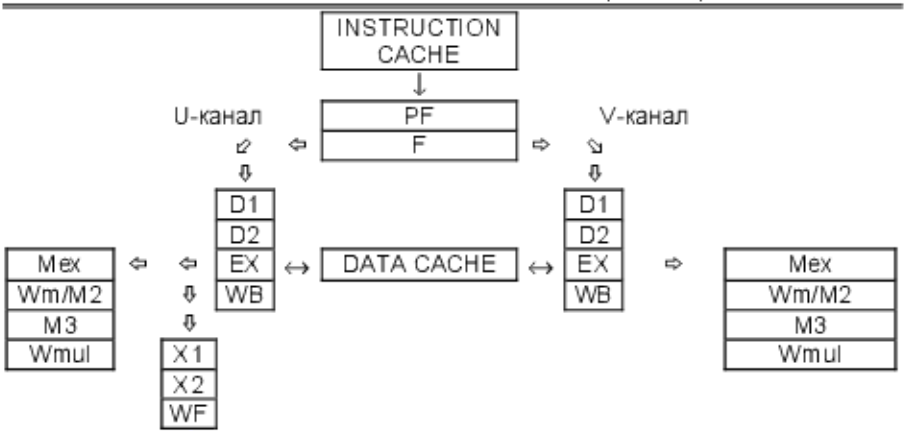


Рис. 5.2. Конвейер процессоров семейства P5

На стадии предвыборки (PF – prefetch) команды выбираются из кэша команд. Далее они поступают на стадию выборки (F – fetch). Здесь происходит разделение выбранной порции кода на отдельные команды, а также декодирование любых префиксов. Между стадией (F) и стадией (D1) находится FIFO-буфер. В нем может содержаться до четырех инструкций. FIFO-буфер прозрачен, т. е. он не отнимает времени, когда пуст. В каждом такте из стадии (F) в FIFO-буфер может выпускаться до двух команд. Пара инструкций поступает (если это возможно) из FIFO на стадию (D1). Так как средняя скорость выполнения команд меньше чем две команды за такт, то FIFO обычно заполнен. Следовательно, FIFO может буферировать задержки, возникающие на стадиях (PF) и (F), тем самым предотвращая по возможности обеднение (когда в FIFO находится только одна команда) или полную остановку конвейера.

Если в одном из каналов возникла задержка, то команды, следующие за застрявшей командой, не могут продвигаться дальше, даже если застрявшая команда находится в другом канале. Например, параллельно по двум каналам следуют две команды, одна из которых требует один такт на стадии (EX), а другая – два такта. Пусть первая команда находится в V-канале, а вторая – в U-канале. Попав на стадию (EX), первый такт эти команды выполняют вместе. В следующем такте команда в U-канале остается на стадии (EX), а команда в V-канале переходит на следующую стадию, при этом на ее место ничего не поступает, т. е. параллельно с двухтактной командой не могут выполняться две однотактные. Решение о спаривании команд принимается

только один раз при входе в конвейер. Это один из главных недостатков архитектуры P5.

PF – предвыборка команд.

F – определение границ команд.

D1 – декодирование команд.

D2 – генерация линейного адреса.

EX: INT – чтение операндов из памяти, выполнение команды, запись операндов в память;

MMX – чтение операндов из памяти, далее переход на стадию Mex;

FPU – чтение операндов из памяти и регистров, далее переход на стадию X1; преобразование данных ко внешнему формату, запись в память (FST).

WB - запись результата в регистр.

Mex - выполнение MMX-команд. Первый такт команды умножения.

Wm/M2 - запись результата однотактных команд. Второй такт умножения.

M3 - третий такт умножения.

Wmul - запись результата умножения.

X1 - преобразование данных ко внутреннему формату, запись в регистр.

X2 - выполнение FPU-команд.

WF - округление и запись результата в регистр.

В отличие от целочисленных команд, которые целиком выполняются на стадии (EX), команды FPU и MMX начинают выполняться на стадии (EX), а затем уходят на свои стадии, где продолжают выполняться дальше. FPU-команды не могут спариваться с целочисленными командами в начале конвейера, но, после того как FPU-команда уйдет на стадию (X1), следующие за ней целочисленные команды смогут продвигаться дальше. Например, если запустить в конвейер сначала команду FMUL, то следующие за ней целочисленные команды смогут продолжать выполняться параллельно с FMUL. Если же сначала запустить команду MUL, то она застрянет на стадии (EX), блокировав дальнейшее продвижение следующих за ней команд по обоим каналам.

5.2.1. Блокировка генерации адреса

В случае, если одна команда использует какой-либо регистр в качестве операнда-приемника, а следующая за ней (по конвейеру, а не по тексту программы) команда использует этот же регистр для адресации, то возникает блокировка генерации адреса (Address Generation Interlock – AGI) на стадии (D2). Действительно, когда первая команда находится на стадии (EX), вторая находится на стадии (D2) и хочет вычислить линейный адрес, но это невозможно, так как первая команда еще не получила результата. В этом случае на стадии (D2) вводится дополнительный такт. Вторая команда приступит к генерации адреса, когда первая перейдет на стадию (WB).

add esi,eax	**D2**	**EX**	**WB**				
inc edi	**D2**	**EX**	**WB**				
add ebx,2	**D1**	**D2**	**EX**	**WB**			
mov eax,[esi]	**D1**	D2	**D2**	**EX**	**WB**		
por mm0,mm1		**D1**	D1	**D2**	**EX**	**Mex**	**Wm/M2**

Команды PUSH, POP, RET, CALL используют регистр ESP для формирования адреса. Явление AGI в данном случае также имеет место.

```
sub   esp,55   ; Добавляется один лишний такт
push ecx
```

Команды PUSH и POP также используют ESP в качестве операнда-приемника. Однако если следующая команда использует для адресации регистр ESP, блокировки генерации адреса не происходит. Процессор “переименовывает” регистр ESP при выполнении команд PUSH и POP.

```
push    edi             ; Нет дополнительного такта
mov     eax,[esp]
```

MMX-регистры не могут быть использованы для адресации; следовательно, рассмотренное явление не имеет места для MMX-команд.

5.2.2. Обращение к регистрам разного размера

В процессоре Intel486 вводится один дополнительный такт, если после записи в часть регистра следует чтение из целого регистра. В процессорах семейства P5 дополнительный такт не требуется.

```
mov     al,4            ; Нет задержки на P5
mov     [ebp],eax
```

Внимание! Процессоры семейства P6, также как и Intel486, чувствительны к таким случаям, но в P6 задержка много больше одного такта.

5.2.3. Оптимизация предвыборки команд

Процессоры семейства P5 имеют длину строки кэш-памяти, равную 32 байтам. Устройство предвыборки команд производит выборку кода 128-разрядными блоками по 16-байтным границам. Выравнивание кода повышает эффективность предвыборки, а значит, помогает избежать обеднения конвейера.

Рекомендации:

⇒ Вход в цикл следует выравнивать по 16-байтным границам.

⇒ Метки условных переходов выравнивать не стоит.

⇒ Метки безусловных переходов следует выравнивать по 16-байтным границам.

5.2.4. Предсказание ветвлений

Для динамического предсказания ветвлений используется устройство, называемое Branch Target Buffer – BTB. Оно представляет собой кэш-память, в которой хранится информация о сделанных ранее переходах. Когда команда выбирается из памяти, ее адрес транслируется через BTB и BTB выдает устройству предвыборки адрес следующей команды.

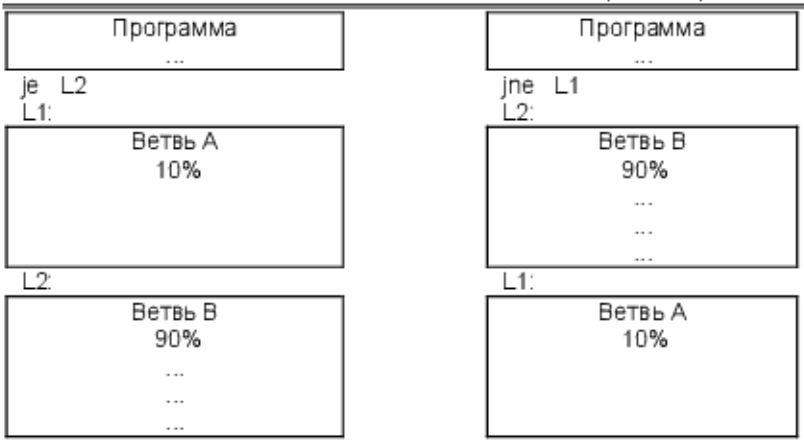
BTB работает по такому принципу:

⇒ если адрес команды отсутствует в BTB, то предвыборка продолжается дальше со следующего адреса;

⇒ если адрес команды находится в BTB и переход предсказывается как сделанный, то это требует одного лишнего такта;

⇒ BTB хранит 4-битную историю переходов.

Из принципа работы BTB следует, что наиболее вероятные ветви программы необходимо располагать непосредственно после команды перехода (переход не выполнен). Кроме того, такой способ расположения кода поможет сохранить место в BTB, так как невыполненные переходы не записываются в BTB. Неверно предсказанные переходы из-за необходимого сброса конвейера приводят к задержке в 3 или 4 такта.



Неоптимизированный код **Оптимизированный код**

5.2.5. Выравнивание данных

Так как длина строки кэш-памяти составляет 32 байта, то блоки данных (массивы, структуры), размер которых кратен 32 байтам, следует выравнивать по границам строк кэш-памяти. Блоки, размер которых не кратен 32 байтам, следует выравнивать по 16-байтным границам.

Одиночные данные, а также данные внутри структур, следует выравнивать по следующему принципу:

⇒ 2-байтные данные должны целиком содержаться внутри выравненного двойного слова, т. е. адрес может быть xxx00, xxx01, xxx10, но не xxx11;

⇒ 4-байтные данные должны быть выравнены по 4-байтной границе;

⇒ 8-байтные данные (например, MMX-регистры) должны быть выравнены по 8-байтной границе.

Обращение к невыравненным данным приводит к введению дополнительных тактов, так как в этом случае требуется два обращения к кэш-памяти вместо одного, когда данные выравнены. Невыравненное обращение к кэш-памяти требует четырех тактов вместо одного.

В случае кэш-промаха процессор считывает всю строку из внешней памяти. Процессор может заполнять строку в следующем порядке:

1-й адрес	2-й адрес	3-й адрес	4-й адрес
0h	8h	10h	18h
8h	0h	18h	10h
10h	18h	0h	8h
18h	10h	8h	0h

Данные становятся доступными для использования по мере их поступления. Предпочтительно обращаться к данным одной строки в порядке их поступления из внешней памяти.

5.2.6. Правила спаривания двух команд

Разделим все команды на 4 класса:

Класс (NP). Эти команды не могут спариваться ни при каких условиях. К ним относятся:

⇒ команды сдвига со счетчиком в регистре cl;

⇒ длинные арифметические команды, например MUL, DIV;

⇒ дополнительные команды, например RET, ENTER, PUSHA, MOVS, STOS, LOOPNZ;

⇒ некоторые команды FPU, например FSCALE, FLDCW, FST;

⇒ межсегментные команды, например PUSH sreg, CALL far;

⇒ EMMS.

Класс (UV). Эти команды могут спариваться, направляясь в любой из каналов (U или V). К ним относятся следующие команды:

⇒ большинство ALU-команд, например ADD, INC, XOR;

⇒ все команды сравнения, например CMP, TEST;

⇒ все стековые команды с регистром, например PUSH reg, POP reg;

⇒ MMX-команды, которые не обращаются к памяти или целочисленному регистру (могут спариваться как с MMX-командами, так и с не MMX-командами, но так как процессор содержит только одно устройство MMX-умножения и одно устройство MMX-сдвига, две MMX-команды, использующие одно и то же устройство, не могут быть выполнены одновременно), например POR mmxreg, mmreg.

Класс (PU). Эти команды не могут исполняться в V-канале, но могут спариваться с другими командами, которые могут направляться в V-канал. Сюда относятся:

⇒ команды с переносом, например ADC, SBB;

⇒ команды с префиксами (кроме 0Fh, 66h, 67h);

⇒ команды сдвига с непосредственным операндом;

⇒ некоторые FPU-команды (FPU-команды могут спариваться только с командой FXCH), например FADD, FMUL, FLD;

⇒ MMX-команды, которые обращаются к памяти или целочисленному регистру (такие команды не могут спариваться с не MMX-командами).

Класс (PV). Эти команды могут выполняться в любом из каналов (U или V), но имеют возможность спариваться с другими командами, только когда направляются в V-канал. К ним относятся:

⇒ команды немежсегментной передачи управления, например CALL near, JMP near, Jcc;

⇒ FXCH.

Итак, для того чтобы две команды могли выполниться одновременно, они должны принадлежать соответствующим классам (команда, которая направляется в U-канал, должна принадлежать классу (UV) или (PU), а команда, которая направляется в V-канал, – классу (UV) или (PV)). Кроме того, Pentium MMX не спаривает команды, если первая команда длиннее 11 байт или вторая длиннее 7 байт (префиксы не учитываются).

На спаривание команд также влияют их операнды. Рассмотрим возможные случаи взаимозависимости операндов.

Истинная взаимозависимость (flow dependency). Какой-либо регистр служит операндом-приемником для первой команды и операндом-источником для второй команды.

```
mov  eax,100          ; Такие команды не могут спариваться
mov  [ebp],eax
```

Исключение составляют специальные пары:

взаимозависимость по ESP:

⇒ PUSH reg/imm ⇔ PUSH reg/imm;

⇒ PUSH reg/imm ⇔ CALL;

⇒ POP reg ⇔ POP reg.

взаимозависимость по коду условия:

⇒ CMP ⇔ Jcc;

⇒ ADD ⇔ JN.

Взаимозависимость по выходу (output dependency). Обе команды имеют один и тот же регистр-приемник.

```
mov  eax,5            ; Такие команды не могу спариваться
mov  eax,[ebp]
```

Исключение составляют команды, которые записывают в регистр EFLAGS (например, две ALU-команды, которые изменяют флажки).

Антивзаимозависимость (antidependency). Какой-либо регистр служит операндом-источником для первой команды и операндом-приемником для второй команды.

```
mov eax,ebx          ; Такие команды могут спариваться
mov ebx,[ebp]
```

Далее рассмотрим ограничения, которые могут встретиться при параллельном выполнении. Существуют случаи, когда команды, объединившись в пару при входе в конвейер, не могут быть выполнены одновременно на стадии (EX):

⇒ Еесли команды обращаются к одному банку кэш-памяти, то вторая команда (V-канал) будет ждать, пока не закончится обращение первой команды (U-канал). Конфликт возникает, когда биты со второго по четвертый совпадают в двух физических адресах;

⇒ многотактная команда в U-канале будет исполняться одна до последнего обращения к памяти.

Пример

```
add eax,mem1
add ebx,mem2
```

EX	**EX**	**WB**
EX	**EX**	**WB**

Приведенные выше команды складывают содержимое регистра со значением в памяти и записывают результат в регистр. Эта операция требует 2 такта на выполнение. В первом такте считывается значение из памяти, а во втором такте осуществляется сложение. Так как в U-канале только одно обращение к памяти, то операция в V-канале может начаться в том же такте. Следовательно, на выполнение этих двух команд потребуется два такта.

```
add mem1,eax
add mem2,ebx
```

EX	**EX**	**EX**	**WB**		
EX	EX	**EX**	**EX**	**EX**	**WB**

Приведенные выше команды складывают содержимое регистра со значением в памяти и записывают результат в память. Эта операция требует 3 такта на выполнение. В первом такте считывается значение из памяти, во втором такте осуществляется сложение, в третьем – результат записывается в память. Последний такт выполнения команды в U-канале перекрывается с первым тактом выполнения команды в V-канале. Таким образом, выполнение такой пары команд потребует пяти тактов.

5.2.7. Оптимизация для FPU

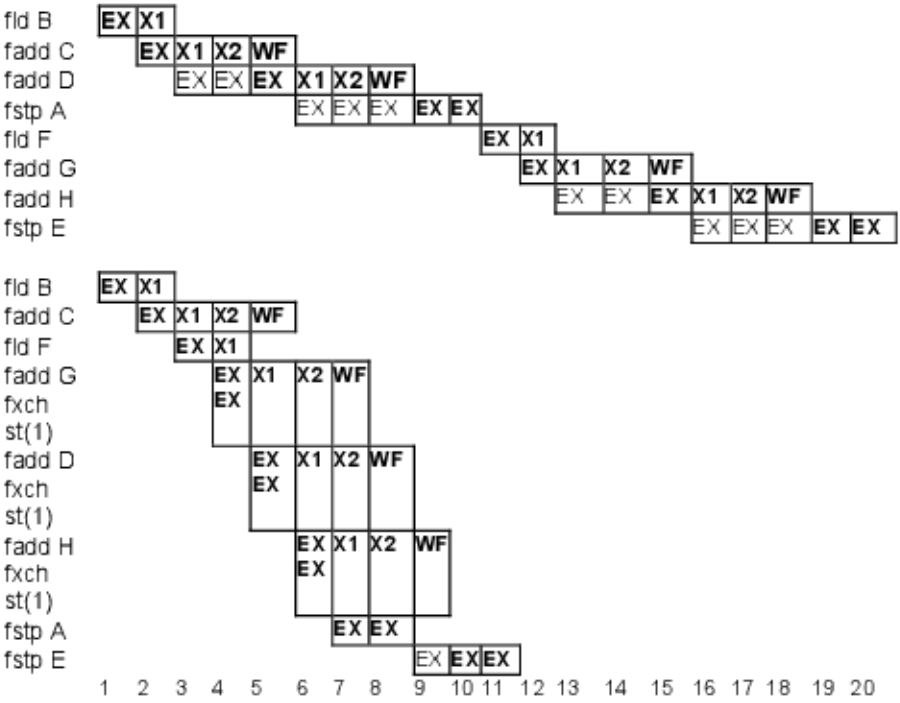
Большинство команд FPU требует, чтобы один из операндов и результат использовали вершину стека FPU-регистров. Это делает каждую команду зависящей от предыдущей и не дает возможности использовать преимущества конвейерного выполнения.

Команда FXCH дает возможность нейтрализовать привязанность FPU-команд к вершине стека регистров. Команда FXCH может спариваться с наиболее общими FPU-командами. Для того чтобы FXCH могла исполнятся в V-канале параллельно с другой командой, она (FXCH) должна быть обязательно второй в паре. Спаривания не произойдет, если FXCH будет первой в паре, так как все FPU-команды (кроме FXCH) не могут исполняться в V-канале.

Пример

Пусть нам нужно произвести следующие вычисления:

A = B + C + D
E = F + G + E



Как видно из рисунка, использование команды FXCH сократило время выполнения почти в два раза. Стоит обратить внимание на выполнение команды FST. Если FST должна сохранить в памяти результат предыдущей операции, то выполнение FST начинается не во время записи результата в регистр на стадии (WF), а на такт позже. Это связано с тем, что на стадии (WF) происходит сначала округление и затем в конце такта запись результата в регистр. Команда FST в первом такте своего выполнения должна успеть не только сосчитать значение результата, но и преобразовать его во внешний формат. Поэтому FST ждет один лишний такт.

Команды FPU, которые работают с целыми операндами (FIADD, FISUB ...), лучше разбить на две команды:

```
fiadd [ebp]     ; 4          fild    [ebp]    ; 1
                             faddp   st(1)    ; 2
```

Трансцендентные команды выполняются одни, и никакие команды не могут выполняться параллельно с ними. Целочисленные команды, следующие за трансцендентной командой, будут ждать, пока последняя не завершится.

6. ПРОЦЕССОРЫ СЕМЕЙСТВА P6

В этой главе рассматривается архитектура конвейера процессоров семейства P6 (Pentium Pro и Pentium II), а также их новые возможности по сравнению с P5.

6.1. Новые возможности

В процессоры P6 добавлены следующие команды:

⇒ CMOVcc (Conditional Move). Выполняет условную передачу данных;

⇒ FCMOVcc (Floating-point Conditional Move). Выполняет условную передачу FPU-регистра в вершину стека [ST(0)];

⇒ FCOMI (Floating-point Compare and set EFLAGS). Сравнивает значения двух FPU-регистров и устанавливает флажки регистра EFLAGS в соответствии с результатом;

⇒ RDPMC (Read Performance Monitoring Counters). Считывает содержимое специфических счетчиков для мониторинга производительности процессора;

⇒ UD2 (Undefined). Генерирует исключение недействительной операции. Эта команда служит для тестирования обработчика исключения недействительной операции.

Подробнее о новых командах см. 7 – “Новые команды P5 и P6”.

Добавлены новые флажки в регистр управления CR4, подробнее см. в приложении Г – “Формат регистра CR4”.

Новые функциональные возможности добавлены в механизм страничного преобразования (см. 8 – “Страничное преобразование в процессорах P5 и P6”).

6.2. Конвейер процессоров семейства P6

Конвейер процессоров семейства P6 существенно отличается от конвейера процессоров семейства P5. В P6 используется принципиально новый подход к выполнению команд. Применен ряд новых приемов для предотвращения заторов конвейера, например внеочередное выполнение команд (out-of-order execution), переименование регистров. Конвейер P6 состоит из трех частей (рис. 6.1):

1. In-Order Issue Front End. На этом этапе происходит выборка команд из памяти и декодирование в микрооперации.
2. Out-of-Order Core. На этом этапе процессор выполняет микрооперации. Выполнение может происходить вне очереди.

3. In-Order Retirement unit. На этом этапе происходит удаление команд с конвейера.

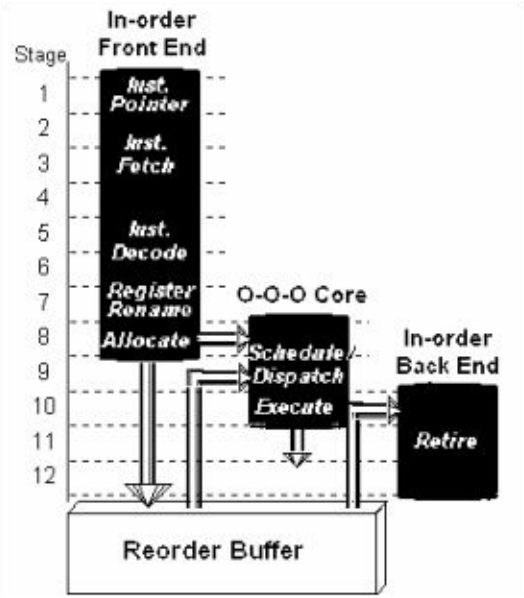


Рис. 6.1. Конвейер процессоров P6

Теперь рассмотрим процесс выполнения команд более подробно. P6 содержит 3 параллельно работающих декодера. Первый декодер способен декодировать макрокоманды, состоящие из четырех и более микроопераций. Сложные макрокоманды (более четырех микроопераций) требуют более одного такта для декодирования. Второй и третий декодеры могут декодировать макрокоманды, состоящие только из одной микрооперации. Таким образом, при составлении кода следует придерживаться последовательности 4-1-1 (первая макрокоманда состоит из четырех микроопераций, а остальные две – из одной микрооперации). Такое упорядочение макрокоманд позволит достичь максимальной производительности декодеров.

Пример

Первоначальный код

Команда	*Число микроопераций*
add eax,4	1
add eax,ecx	1
sub eax,mem2	2

Переупорядоченный код

Команда	*Число микроопераций*
sub eax,mem2	2
add ebx,edx	1
add eax,ecx	1

Первоначальный код

inc esi	1
add ebx,edx	1
sub ebx,mem3	2

Переупорядоченный код

sub ebx,mem3	2
inc esi	1
add eax,4	1

Горизонтальными линиями отделены группы команд, которые декодируются за один такт. Из рассмотренного примера видно, что правильно упорядоченный код декодируется быстрее.

В каждом такте декодеры могут производить до шести микроопераций, которые поступают в специальную очередь. Из очереди до трех микроопераций поступает на стадию (RAT) – Register Allocation Table. Здесь происходит переименование регистров и резервирование мест (регистров) в (ROB) – Re-order buffer. Переименование регистров позволяет нейтрализовать ложные взаимозависимости.

Следующий пример показывает суть механизма переименования регистров.

Пример

Команда до переименования	*Переименование регистров*	*Команда после переименования*
mov eax,17	EAX→r1	mov r1,17
mov mem1,eax		mov mem1,r1
mov eax,3	EAX→r6	mov r6,3
add mem2,eax		add mem2,r6
sub mem4,eax		sub mem4,r6
mov eax,555	EAX→r8	mov r8,555
mov eax,mem7	EAX→r12	mov r12,mem7

Из приведенного примера видно, что после переименования остались только истинные взаимозависимости, ложные взаимозависимости исчезли.

Далее микрооперации записываются в (ROB). ROB организован в виде кольцевого буфера на 40 мест (40 программно-прозрачных регистров, служащих для переименования). Микрооперации поступают в (ROB) в порядке очереди и удаляются в порядке очереди, а выполняться микрооперации могут вне очереди по мере готовности исходных данных и доступности исполнительных устройств процессора. До трех микроопераций могут передаваться на выполнение в каждом такте. В зависимости от выполняемой функции микрооперация направляется в один из пяти портов (конвейеров). К каждому порту прикреплены свои исполнительные устройства. После выполнения микрооперация возвращается обратно в (ROB), где ожидает удаления. После

того как результат микрооперации был записан в (ROB), он (результат) становится доступным другим микрооперациям. На рис. 6.2 показано состояние буфера (ROB) в одном отдельно взятом такте. Так как процессор выполняет команды вне очереди, то важным моментом является обеспечение достаточного числа микроопераций, готовых к выполнению. Это может быть достигнуто путем быстрого декодирования и оптимизации предсказания ветвлений. На рис. 6.2 виден принцип внеочередного выполнения микроопераций. Например, микрооперация под номером 9 еще не приступила к выполнению, так как она ждет результата микрооперации под номером 6, но в это время микрооперация 10 уже выполнилась.

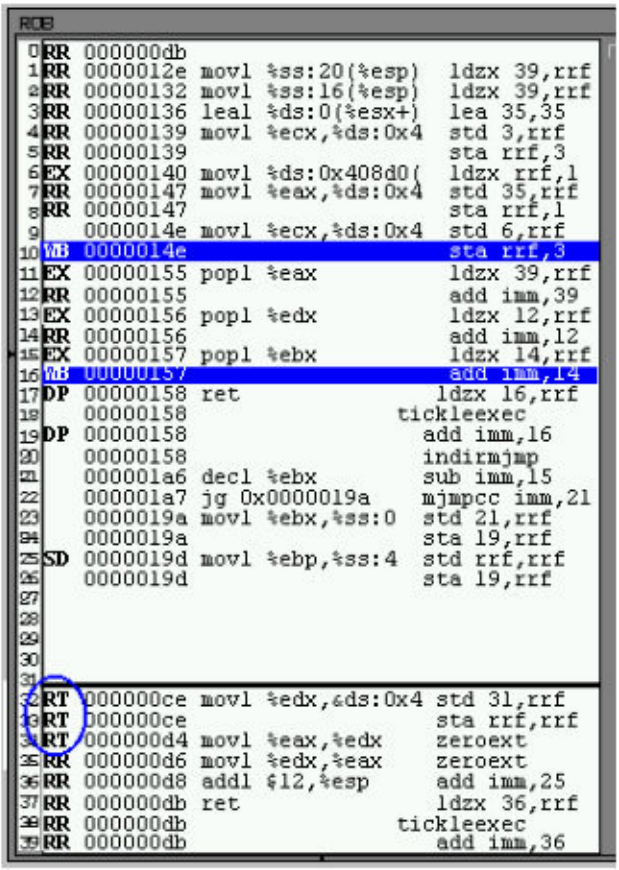


Рис. 6.2. Состояние буфера ROB

Двумя буквами на рисунке обозначено состояние команд:

Стадия выполнения	Обозначение
Ready for scheduling	RS
Scheduling	SD
Dispatching	DP
Executing	EX
Writing back the ROB	WB
Ready to retire	RR
Retiring	RT

В каждом такте могут удаляться до трех микроопераций. При удалении временные результаты, хранящиеся в (ROB), записываются в соответствующие программные регистры или память. Удаление происходит в порядке очереди, что обеспечивает правильное ведение контекста.

6.2.1. Предсказание ветвлений

Предсказанию ветвлений стоит уделbть особое внимание, так как неверное предсказание в процессорах P6 обходится очень дорого, намного дороже, чем в P5. Задержка при ошибочном предсказании составляет минимум 12 тактов, но может быть и больше.

Когда команда перехода после выполнения возвращается в ROB, процессор узнает, правильно ли было сделано предсказание. Если «да», то работа конвейера продолжается как обычно. Если «нет», то процессор запрещает поступление новых команд в (ROB), сбрасывает команды, находящиеся на стадиях от начала конвейера до (RAT) включительно, так как они соответствуют неверной ветви программы. Затем процессор продолжает выполнять команды, оставшиеся в (ROB) до удаления команды перехода, вызвавшей неправильное предсказание. После чего процессор сбрасывает команды, успевшие проникнуть в (ROB) после команды перехода, и разрешает поступление в (ROB) новых команд. Если такие ситуации будут встречаться достаточно часто, то производительность процессора резко снизится!

Для динамического предсказания ветвлений используется устройство, называемое Branch Target Buffer – BTB. Оно представляет собой кэш-память, в которой хранится информация о сделанных ранее переходах. Когда команда выбирается из памяти, ее адрес транслируется через BTB и BTB выдает устройству предвыборки адрес следующей команды.

BTB работает по следующему принципу:

⇒ если адрес команды отсутствует в BTB, то предвыборка продолжается дальше со следующего адреса;

⇒ если адрес команды находится в BTB и переход предсказывается как сделанный, то это требует одного лишнего такта;

⇒ BTB хранит 4-битную историю переходов.

Переходы, которые отсутствуют в BTB, предсказываются с использованием статического алгоритма предсказания ветвлений:

⇒ безусловные переходы предсказываются как сделанные;

⇒ условные переходы назад предсказываются как сделанные;

⇒ условные переходы вперед предсказываются как не сделанные.

Задержка на статическое предсказание составляет 6 тактов.

С помощью BTB можно успешно предсказывать один и тот же переход, если он имеет регулярно повторяющуюся закономерность выполнения. Например, будет корректно предсказан условный переход, который выполняется при каждой четной итерации и не выполняется при каждой нечетной итерации. Переходы по указателю, значение которого не является постоянным для подавляющего большинства случаев, а также условные переходы, не имеющие закономерности выполнения, являются непредсказуемыми или плохо предсказуемыми. Если такие переходы встречаются внутри часто повторяющихся циклов, то это может существенно сказаться на производительности.

6.2.2. Обращение к регистрам разного размера

Если процессор считывает данные из 32-разрядного регистра (например, EAX) сразу после того, как была произведена запись во фрагмент этого регистра (например, AL, AH, AX), то микрооперация считывания не сможет проникнуть в (ROB) до того как микрооперация записи не исполнится и не будет удалена из (ROB). Это занимает минимум 7 тактов.

Задержка связана с тем, что после возвращения в (ROB) результата микрооперации, которая записывает в регистр, в (ROB) будет содержаться лишь фрагмент регистра, а микрооперации, которая считывает из регистра, нужен целый регистр (рис. 6.3).

Микрооперация на стадии (RAT) требует 2 байта из результата 15-й микрооперации и по 1 байту из результата 16-й и 17-й микрооперации. Процессор не может собирать один регистр по частям из разных мест. Микрооперация на стадии (RAT) будет ждать, пока из (ROB) не будут удалены микрооперации 16 и 17.

Rat/Alloc

0	addl %ebx, %eax add 0, 0
1	
2	

Schedul

0	
1	(16) add rr, 15
2	
3	
4	

ROB

10	.....			.....
11	..... ---			.
12	RR 000000090	pushl	%ebp	std rrf, rrf
13	EX 000000090			sta rrf, imm
14	RR 000000090			sub imm, rrf
15	DP 000000091		%esp, %ebp	zeroext 13, rrf
16	WB 000000093	addl	%ebx, %eax	add rrf, rrf
17	000000095	addw	%bx, %ax	add rrf, 15
18	000000098	addb	%bl, %al	add rrf, 16
19	DP 00000009a	incl	%ebx	add imm, rrf
20	--- ---			---

Dispatch

0	(14) zeroext 13, rrf
1	(18) add imm, rrf
2	
3	
4	

WB

0	(15) add rrf, rrf
1	
2	
3	
4	

Рис. 6.3. Обращение к регистрам разного размера

Аналогична ситуация с флажками: если одна команда модифицирует какие-либо флажки (но не все), а другая читает комбинацию из модифицированных и немодифицированных флажков, то возникает задержка конвейера. Задержки не возникает, если команда читает только модифицированные или только немодифицированные флажки.

Для избежания задержек следует использовать команды XOR и SUB. С помощью этих команд необходимо очистить целый регистр

перед записью во фрагмент регистра. В этом случае последующее считывание из целого регистра не приведет к задержке.

Пример

```
xor  eax,eax
mov  al,mem8
use  eax              ; Нет задержки

xor  eax,eax
mov  ax,mem16
use  eax              ; Нет задержки

sub  ax,ax
mov  al,mem8
use  ax               ; Нет задержки

sub  eax,eax
mov  al,mem8
use  ax               ; Нет задержки

xor  ah,ah
mov  al,mem8
use  ax               ; Нет задержки
```

Естественно, все вышеизложенное справедливо не только для регистра EAX, но и для всех остальных регистров: EAX, EBX, ECX, EDX, EBP, ESP, EDI и ESI.

6.2.3. Обращение к памяти

Кэш данных P6 позволяет обращаться к нему по двум портам: один порт – для записи, другой – для чтения. Процессор P6 способен производить одновременно считывание и запись данных по одному и тому же адресу. При этом возможны условия, которые будут приводить к задержкам конвейера.

Если по одному и тому же адресу или по перекрывающимся адресам следует сначала короткая запись, а затем длинное чтение, то такая ситуация приводит к задержке:

```
mov word ptr [ebp],10
mov ecx,[ebp]                 ; Задержка
```

Если по перекрывающимся адресам (но не по одому и тому же адресу) следует сначала длинная запись, а затем короткое чтение, то такая ситуация тоже приводит к задержке. Но если обращение происходит по одному и тому же адресу, то это не приводит к задержке:

```
mov [ebp-2],eax
mov cx,[ebp]                  ; Задержка
```

```
mov  [esi],eax
mov  dx,[esi]                    ; Нет задержки
```

Для избежания задержки можно заменить фрагмент кода, например:

```
movq    mem,mm0          movq    mem,mm0
...                      ...
mov     bx,mem+2         movq    mm1,mem
mov     cx,mem+4         movd    eax,mm1
                         psrlq   mm1,32
                         shr     eax,16
                         movd    ebx,mm1
                         and     ebx,0ffffh
```

Для P6 такая замена команд позволит избежать задержки при обращении к памяти. Но такое увеличение числа команд скажется негативно на производительности процессоров P5.

7. НОВЫЕ КОМАНДЫ P5 И P6

7.1. Новые команды процессоров семейства P5

В процессоры P5 были добавлены следующие команды:

⇒ команда CMPXCHG8B (Compare and Exchange 8 Bytes). Сравнивает и обменивает 8-байтные значения;

⇒ команда CPUID (CPU Identification). Выдает информацию о процессоре;

⇒ команда RDTSC (Read Time-stamp Counter). Считывает содержимое регистра TSC;

⇒ команда RDMSR (Read Model-specific Register). Считывает из MSR-регистра;

⇒ команда WRMSR (Write Model-specific Register). Записывает в MSR-регистр;

⇒ команда RSM (Resume from SMM). Возврат из режима SMM.

CMPXCHG8B – Compare and Exchange 8 Bytes

Код	*Команда*	*Описание*
0F C7/1m64	CMPXCHG8B r/m64	Сравнивает EDX:EAX с r/m64 если равно: ZF=1, r/m64=ECX:EBX; если не равно: ZF=0, EDX:EAX=r/m64

Операция

```
IF EDX:EAX=DEST
    ZF ← 1
    DEST ← ECX:EBX
ELSE
    ZF ← 0
    EDX:EAX ← DEST
```

Описание

Команда CMPXCHG8B сравнивает 64-разрядное значение в паре регистров EDX:EAX с операндом-приемником. Если значения равны, то пара регистров ECX:EBX записывается в операнд-приемник, в противном случае операнд-приемник загружается в пару регистров EDX:EAX.

Влияние на флажки

ZF=1, если значения операнда-приемника и пары регистров EDX:EAX равны, в противном случае ZF=0.
Значение флагов CF, PF, AF, SF и OF не изменяется.

Особые случаи P-режима

#GP(0), если операнд-приемник в незаписываемом сегменте.
#GP(0), если эффективный адрес операнда выходит за пределы сегмента CS, DS, ES, FS или GS. Если DS, ES, FS или GS равны нулю.
#SS(0), если эффективный адрес операнда выходит за пределы сегмента SS.
#PF(код-нарушение), при страничном нарушении.
#AC(0), при невыравненном обращении к памяти, если CPL=3 и данное исключение разрешено.
#UD, если в байте modr/m указан регистр в качестве приемника.

Особые случаи R-режима

#GP, если эффективный адрес операнда выходит за пределы сегмента CS, DS, ES, FS или GS.
#SS, если эффективный адрес операнда выходит за пределы сегмента SS.
#UD, если в байте modr/m указан регистр в качестве приемника.

Особые случаи V-режима

Те же, что и в R-режиме.
#PF(код-нарушение), при страничном нарушении.
#AC(0), при невыравненном обращении к памяти, если CPL=3 и данное исключение разрешено.

CPUID – CPU Identification

Opcode Instruction Description
0F A2 CPUID EAX ← Processor identification information

Операция

```
CASE (EAX) OF
  EAX = 0:
    EAX ← highest input value understood by CPUID;
          (* 2 for P6 processor *)
          (* 1 for P5 processor *)
    EBX ← Vendor identification string;
    EDX ← Vendor identification string;
    ECX ← Vendor identification string;
BREAK;
EAX = 1:
    EAX[3:0] ← Stepping ID;
    EAX[7:4] ← Model;
    EAX[11:8] ← Family;
    EAX[13:12] ← Processor type;
    EAX[31:12] ← Reserved;
    EBX ← Reserved;
    ECX ← Reserved;
    EDX ← Feature flags;
BREAK;
EAX = 2:
    EAX ← Cache information;
```

```
    EBX ← Cache information;
    ECX ← Cache information;
    EDX ← Cache information;
BREAK;
DEFAULT: (* EAX > highest value recognized by CPUID *)
    EAX ← reserved, undefined;
    EBX ← reserved, undefined;
    ECX ← reserved, undefined;
    EDX ← reserved, undefined;
BREAK;
ESAC;
```

Описание

Команда CPUID позволяет узнать различную информацию о процессоре. Входное значение в регистре EAX определяет, какую информацию необходимо выдать (табл. 7.1). Информация возвращается в регистрах EAX, EBX, ECX и EDX.

Таблица 7.1. Информация, возвращаемая командой CPUID

Входное значение EAX	*Информация, выдаваемая процессором*
0	EAX – максимальное входное значение регистра EAX, распознаваемое командой CPUID; EBX – Genu; ECX – inel; EDX – ntel
1	EAX – версия процессора (Type, Family, Model, Stepping); EBX – зарезервировано; ECX – зарезервировано; EDX – информация о возможностях процессора
2	EAX – информация о кэше; EBX – информация о кэше; ECX – информация о кэше; EDX – информация о кэше

Входное значение EAX=0:

Процессор возвращает в регистре EAX максимальное входное значение, распознаваемое командой CPUID.

В регистры EBX, ECX и EDX заносится строка-идентификатор процессора (для процессоров Intel это строка GenuineIntel).

```
EBX ← 756e6547h
EDX ← 49656e69h
ECX ← 6c65746eh
```

Входное значение EAX=1:

Процессор возвращает в регистре EAX информацию о версии процессора, в регистре EDX – информацию о возможностях процессора (рис. 7.1).

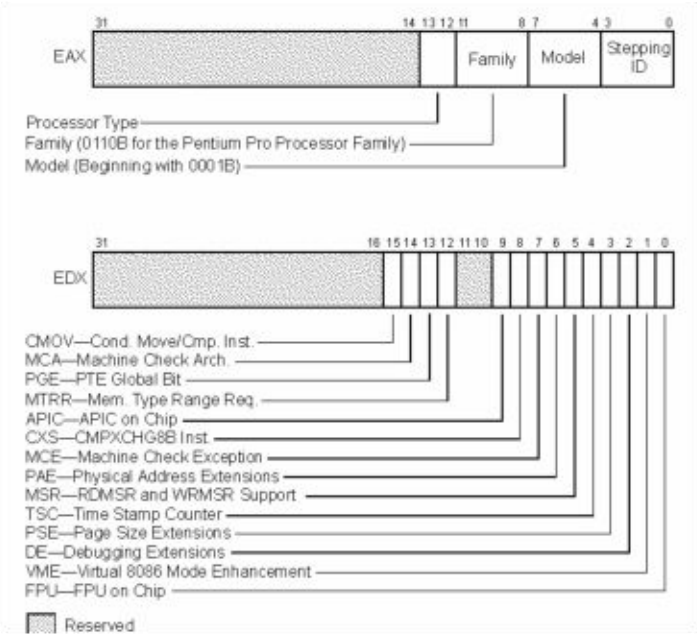


Рис. 7.1. Информация о версии и возможностях процессора

0 FPU – в процессор интегрировано устройство FPU, поддерживающее систему команд Intel387.

1 VME – процессор поддерживает механизм виртуальных прерываний. Поддерживаются: биты CR4.VME, CR4.PVI, EFLAGS.VIF, EFLAGS.VIP, карта перенаправления прерывания.

2 DE – процессор поддерживает расширенные возможности отладки (точки останова для операций ввода-вывода). Поддерживается бит CR4.DE.

3 PSE – процессор поддерживает страницы размером 4 MB. Поддерживаются: бит CR4.PSE, бит PS в элементах каталога страниц.

4 TSC – процессор поддерживает команду RDTSC. Поддерживается бит CR4.TSD.

5 MSR – процессор поддерживает команды RDMSR и WRMSR.

6 PAE – процессор поддерживает расширение физического адреса (адреса больше 32 разрядов). Точное количество разрядов адреса зависит от конкретной модели процессора. Поддерживаются: расширенный формат элементов страниц, страницы размером 2 MB, бит CR4.PAE.

7 MCE – процессор поддерживает исключение #18 – Machine Check Exception. Это исключение предназначено для оповещения об аппаратных ошибках. Поддерживается бит CR4.MCE.

8 CX8 – процессор поддерживает команду CMPXCHG8B.

9 APIC процессор содержит Advanced Programmable Interrupt Controller (APIC).

10,11 Зарезервировано.

12 MTRR – процессор поддерживает Memory Type Range (MTRRs)Registers. Эти регистры относятся к MSR-регистрам.

13 PGE – процессор поддерживает глобальные страницы. Поддерживаются: бит CR4.PGE, бит PG в элементах таблиц страничного преобразования.

14 MCA – процессор поддерживает MSR-регистр MCG_CAP (machine check global capability).

15 CMOV – процессор поддерживает команды CMOV, FCMOV и FCOMI.

16-31 Зарезервировано.

Входное значение EAX=1

Процессор возвращает информацию о внутренней кэш-памяти и буферах TLB.

Младший байт регистра EAX (регистр AL) содержит значение, которое показывает, сколько раз необходимо исполнить команду CPUID с входным значением 2, чтобы получить всю информацию о кэш-памяти и буферах TLB.

Старший бит (бит 31) каждого регистра показывает, содержит ли данный регистр достоверную информацию (бит равен нулю) или является зарезервированным (бит равен единице).

Если регистр содержит достоверную информацию, то она представляется в виде 1-байтных дескрипторов (табл. 7.2).

Таблица 7.2. Дескрипторы для описания кэш-памяти и буферов TLB

Значение дескриптора	*Описание*
00H	Нуль-дескриптор
01H	TLB команд: 4 KB страницы, 4-направленный ассоциативный, 64 элемента
02H	TLB команд: 4 MB страницы, 4-направленный ассоциативный, 4 элемента
03H	TLB данных: 4 KB страницы, 4-направленный ассоциативный, 64 элемента
04H	TLB данных: 4 MB страницы, 4-направленный ассоциативный, 8 элементов
06H	Кэш команд: размер 8 KB, 4-направленный ассоциативный, длина строки 32 байта
0AH	Кэш данных: размер 8 KB, 2-направленный ассоциативный, длина строки 32 байта
41H	Объединенный кэш: размер 128 KB, 4-направленный ассоциативный, длина строки 32 байта
42H	Объединенный кэш: размер 256 KB, 4-направленный ассоциативный, длина строки 32 байта
43H	Объединенный кэш: размер 512 KB, 4-направленный ассоциативный, длина строки 32 байта

Например, первые процессоры Pentium Pro возвращают следующие значения:

```
EAX 03 02 01 01H
EBX 0H
ECX 0H
EDX 06 04 0A 42H
```

Регистр AL содержит удиницу, следовательно, команду CPUID необходимо выполнить только один раз. Остальные байты раскодируются по табл. 7.2.

Влияние на флажки

Нет.

Особые случаи P-режима

Нет.

Особые случаи R-режима

Нет.

Особые случаи V-режима

Нет.

RDTSC – Read from Time Stamp Counter

Код	*Команда*	*Описание*
0F 31	RDTSC	Считывает TSC в EDX:EAX

Операция

EDX:EAX ← Time Stamp Counter;

Описание

Процессор содержит 64-разрядный счетчик времени – Time Stamp Counter (принадлежит к MSR-регистрам), который увеличивается на единицу в каждом такте. Команда RDTSC копирует содержимое этого счетчика в пару регистров EDX:EAX. Если текущий уровень привилегий равен нулю, то состояние бита TSD в регистре управления CR4 не влияет на выполнение команды. Если CPL > 0, то Time Stamp Counter может быть прочитан только при CR4.TSD = 0. Модифицировать счетчик можно, только когда CPL = 0.

Влияние на флажки

Нет.

Особые случаи P-режима

#GP(0), если CR4.TSD=1 и CPL<>0.

Особые случаи R-режима

Нет.

Особые случаи V-режима

#GP(0), при попытке выполнить команду.

RDMSR – Read from Model Specific Register

Код	*Команда*	*Описание*
0F 32	RDMSR	Считывает содержимое MSR с номером ECX в пару EDX:EAX

Операция

EDX:EAX ← MSR[ECX];

Описание

Регистр ECX содержит номер одного из 64-разрядных MSR-регистров. Содержимое этого MSR копируется в пару EDX:EAX.

Влияние на флажки

Нет.

Особые случаи P-режима

#GP(0), если CPL<>0 или регистр ECX содержит недопустимое значение.

Особые случаи R-режима

#GP(0), если регистр ECX содержит недопустимое значение.

Особые случаи V-режима

#GP(0), при попытке выполнить команду.

WRMSR – Write to Model Specific Register

Код	*Команда*	*Описание*
0F 30	WRMSR	Записывает содержимое пары EDX:EAX в MSR с номером ECX

Операция

MSR[ECX] ←EDX:EAX;

Описание

Регистр ECX содержит номер одного из 64-разрядных MSR-регистров. Содержимое пары EDX:EAX копируется в MSR.

Влияние на флажки

Нет.

Особые случаи P-режима

#GP(0), если CPL<>0 или регистр ECX содержит недопустимое значение.

Особые случаи R-режима

#GP, если регистр ECX содержит недопустимое значение.

Особые случаи V-режима

#GP(0), при попытке выполнить команду.

RSM – Resume from System Management Mode

Код	*Команда*	*Описание*
0F AA	RSM	Возврат из S-режима

Операция

```
ReturnFromSSM;
ProcessorState ← Restore(SSMDump);
```

Описание

Команда осуществляет возврат процессора из S-режима в тот режим, в котором он находился до получения сигнала SMI. Контекст процессора восстанавливается из области, куда он был записан при входе в S-режим. Если процессор обнаружит недопустимую информацию в восстанавливаемом контексте, то он

(процессор) перейдет в состояние отключения. Неверная информация в контексте:

⇒ любой зарезервированный бит в регистре управления CR4 установлен в единицу;

⇒ любая недопустимая комбинация битов в регистре управления CR0. Например, (PG=1 и PE=0) или (NW=1 и CD=0);

⇒ если значение SMBASE не выравнено по 32-KB границе (для процессоров P5). Для процессоров P6 это ограничение снято.

Для более подробной информации см. гл. 11.

Влияние на флажки

Все.

Особые случаи P-режима

#UD, при попытке выполнить команду, когда процессор находится не в S-режиме.

Особые случаи R-режима

#UD, при попытке выполнить команду, когда процессор находится не в S-режиме.

Особые случаи V-режима

#UD, при попытке выполнить команду, когда процессор находится не в S-режиме.

7.2. Новые команды процессоров семейства P6

В процессоре Pentium Pro добавлены следующие команды:

⇒ **CMOVcc (conditional move)**. Выполняет условную передачу данных;

⇒ **FCMOVcc (floating-point conditional move)**. Выполняет условную передачу FPU-регистра в вершину стека [ST(0)];

⇒ **FCOMI (floating-point compare and set EFLAGS)**. Сравнивает значения двух FPU-регистров и устанавливает флажки регистра EFLAGS в соответствии с результатом;

⇒ **RDPMC (read performance monitoring counters)**. Считывает содержимое специфических счетчиков для мониторинга производительности процессора;

⇒ **UD2 (undefined)**. Генерирует исключение недействительной операции. Эта команда служит для тестирования обработчика исключения недействительной операции.

CMOV cc – Conditional Move

Код	*Команда*	*Описание*
0F 47 cw/cd	CMOVA r16, r/m16	Если выше (CF=0 и ZF=0)
0F 47 cw/cd	CMOVA r32, r/m32	Если выше (CF=0 и ZF=0)
0F 43 cw/cd	CMOVAE r16, r/m16	Если выше или равно (CF=0)
0F 43 cw/cd	CMOVAE r32, r/m32	Если выше или равно (CF=0)
0F 42 cw/cd	CMOVB r16, r/m16	Если ниже (CF=1)
0F 42 cw/cd	CMOVB r32, r/m32	Если ниже (CF=1)
0F 46 cw/cd	CMOVBE r16, r/m16	Если ниже или равно (CF=1 или ZF=1)
0F 46 cw/cd	CMOVBE r32, r/m32	Если ниже или равно (CF=1 или ZF=1)
0F 42 cw/cd	CMOVC r16, r/m16	Если перенос (CF=1)
0F 42 cw/cd	CMOVC r32, r/m32	Если перенос (CF=1)
0F 44 cw/cd	CMOVE r16, r/m16	Если равно (ZF=1)
0F 44 cw/cd	CMOVE r32, r/m32	Если равно (ZF=1)
0F 4F cw/cd	CMOVG r16, r/m16	Если больше (ZF=0 и SF=OF)
0F 4F cw/cd	CMOVG r32, r/m32	Если больше (ZF=0 и SF=OF)
0F 4D cw/cd	CMOVGE r16, r/m16	Если больше или равно (SF=OF)
0F 4D cw/cd	CMOVGE r32, r/m32	Если больше или равно (SF=OF)
0F 4C cw/cd	CMOVL r16, r/m16	Если меньше (SF<>OF)
0F 4C cw/cd	CMOVL r32, r/m32	Если меньше (SF<>OF)
0F 4E cw/cd	CMOVLE r16, r/m16	Если меньше или равно (ZF=1 или SF<>OF)
0F 4E cw/cd	CMOVLE r32, r/m32	Если меньше или равно (ZF=1 или SF<>OF)
0F 46 cw/cd	CMOVNA r16, r/m16	Если не выше (CF=1 или ZF=1)
0F 46 cw/cd	CMOVNA r32, r/m32	Если не выше (CF=1 или ZF=1)
0F 42 cw/cd	CMOVNAE r16, r/m16	Если не выше или равно (CF=1)
0F 42 cw/cd	CMOVNAE r32, r/m32	Если не выше или равно (CF=1)
0F 43 cw/cd	CMOVNB r16, r/m16	Если не ниже (CF=0)
0F 43 cw/cd	CMOVNB r32, r/m32	Если не ниже (CF=0)
0F 47 cw/cd	CMOVNBE r16, r/m16	Если не ниже или равно (CF=0 и ZF=0)
0F 47 cw/cd	CMOVNBE r32, r/m32	Если не ниже или равно (CF=0 и ZF=0)
0F 43 cw/cd	CMOVNC r16, r/m16	Если не перенос (CF=0)
0F 43 cw/cd	CMOVNC r32, r/m32	Если не перенос (CF=0)
0F 45 cw/cd	CMOVNE r16, r/m16	Если не равно (ZF=0)
0F 45 cw/cd	CMOVNE r32, r/m32	Если не равно (ZF=0)
0F 4E cw/cd	CMOVNG r16, r/m16	Если не больше (ZF=1 или SF<>OF)
0F 4E cw/cd	CMOVNG r32, r/m32	Если не больше (ZF=1 или SF<>OF)
0F 4C cw/cd	CMOVNGE r16, r/m16	Если не больше или равно (SF<>OF)
0F 4C cw/cd	CMOVNGE r32, r/m32	Если не больше или равно (SF<>OF)
0F 4D cw/cd	CMOVNL r16, r/m16	Если не меньше (SF=OF)

Код	Команда	Описание
0F 4D cw/cd	CMOVNL r32, r/m32	Если не меньше (SF=OF)
0F 4F cw/cd	CMOVNLE r16, r/m16	Если не меньше или равно (ZF=0 и SF=OF)
0F 4F cw/cd	CMOVNLE r32, r/m32	Если не меньше или равно (ZF=0 и SF=OF)
0F 41 cw/cd	CMOVNO r16, r/m16	Если не переполнение (OF=0)
0F 41 cw/cd	CMOVNO r32, r/m32	Если не переполнение (OF=0)
0F 4B cw/cd	CMOVNP r16, r/m16	Если не паритет (PF=0)
0F 4B cw/cd	CMOVNP r32, r/m32	Если не паритет (PF=0)
0F 49 cw/cd	CMOVNS r16, r/m16	Если не знак (SF=0)
0F 49 cw/cd	CMOVNS r32, r/m32	Если не знак (SF=0)
0F 45 cw/cd	CMOVNZ r16, r/m16	Если не нуль (ZF=0)
0F 45 cw/cd	CMOVNZ r32, r/m32	Если не нуль (ZF=0)
0F 40 cw/cd	CMOVO r16, r/m16	Если переполнение (OF=0)
0F 40 cw/cd	CMOVO r32, r/m32	Если переполнение (OF=0)
0F 4A cw/cd	CMOVP r16, r/m16	Если паритет (PF=1)
0F 4A cw/cd	CMOVP r32, r/m32	Если паритет (PF=1)
0F 4A cw/cd	CMOVPE r16, r/m16	Если паритет четный (PF=1)
0F 4A cw/cd	CMOVPE r32, r/m32	Если паритет четный (PF=1)
0F 4B cw/cd	CMOVPO r16, r/m16	Если паритет нечетный (PF=0)
0F 4B cw/cd	CMOVPO r32, r/m32	Если паритет нечетный (PF=0)
0F 48 cw/cd	CMOVS r16, r/m16	Если знак (SF=1)
0F 48 cw/cd	CMOVS r32, r/m32	Если знак (SF=1)
0F 44 cw/cd	CMOVZ r16, r/m16	Если нуль (ZF=1)
0F 44 cw/cd	CMOVZ r32, r/m32	Если нуль (ZF=1)

Операция

```
temp ← DEST
IF condition TRUE THEN
        DEST ← SRC
ELSE
        DEST ← temp
FI;
```

Описание

Команда CMOVcc проверяет состояние регистра EFLAGS и производит операцию переноса данных, если флажки находятся в состоянии, специфицированном командой CMOVcc. Если условие (cc) не выполняется, то выполнение программы продолжается со следующей после CMOV-команды. Команды CMOVcc могут переносить 16- или 32-разрядные данные из памяти в регистр общего назначения или из одного регистра общего назначения в другой. Перенос 8-разрядных данных не поддерживается.

Команды CMOV могут поддерживаться не всеми процессорами семейства P6. Для определения того, поддерживает ли конкрет-

ная модель процессора команду CMOV, следует воспользоваться командой CPUID.

Влияние на флажки

Нет.

Особые случаи P-режима

#GP(0), если эффективный адрес операнда выходит за пределы сегмента CS, DS, ES, FS или GS. Если DS, ES, FS или GS равны нулю.
#SS(0), если эффективный адрес операнда выходит за пределы сегмента SS.
#PF(код-нарушение), при страничном нарушении.
#AC(0), при невыравненном обращении к памяти, если CPL=3 и данное исключение разрешено.

Особые случаи R-режима

#GP, если эффективный адрес операнда выходит за пределы сегмента CS, DS, ES, FS или GS.
#SS, если эффективный адрес операнда выходит за пределы сегмента SS.

Особые случаи V-режима

#GP(0), если эффективный адрес операнда выходит за пределы сегмента CS, DS, ES, FS или GS.
#SS(0), если эффективный адрес операнда выходит за пределы сегмента SS.
#PF(код-нарушение), при страничном нарушении.
#AC(0), при невыравненном обращении к памяти, если CPL=3 и данное исключение разрешено.

FCMOV cc – Floating-Point Conditional Move

Код	*Команда*	*Описание*
DA C0+i	FCMOVB ST(0), ST(i)	Move if below (CF=1)
DA C8+i	FCMOVE ST(0), ST(i)	Move if equal (ZF=1)
DA D0+i	FCMOVBE ST(0), ST(i)	Move if below or equal (CF=1 or ZF=1)
DA D8+i	FCMOVU ST(0), ST(i)	Move if unordered (PF=1)
DB C0+i	FCMOVNB ST(0), ST(i)	Move if not below (CF=0)
DB C8+i	FCMOVNE ST(0), ST(i)	Move if not equal (ZF=0)
DB D0+i	FCMOVNBE ST(0), ST(i)	Move if not below or equal (CF=0 and ZF=0)
DB D8+i	FCMOVNU ST(0), ST(i)	Move if not unordered (PF=0)

Операция

```
IF condition TRUE
        ST(0) ← ST( i)
FI;
```

Описание

Команда проверяет состояние регистра EFLAGS и переносит операнд-источник в операнд-приемник, если условие, специфициро-

ванное командой FCMOV, истинно. Операнд-источник всегда регистр ST(i), операнд-приемник всегда регистр ST(0).

Команда FCMOV может поддерживаться не всеми процессорами семейства P6. Для определения того, поддерживает ли конкретная модель процессора команду FCMOV, следует воспользоваться командой CPUID.

Влияние на флажки FPU

C1=0, если произошло антипереполнение стека регистров FPU.
C0, C2, C3 Неопределенны.

Особые случаи FPU

#IS, если произошло антипереполнение стека регистров FPU.

Особые случаи P-режима

#NM, если CR.EM=1 или CR.TS=1.

Особые случаи R-режима

#NM, если CR.EM=1 или CR.TS=1.

Особые случаи V-режима

#NM, если CR.EM=1 или CR.TS=1.

FCOMI/FCOMIP / FUCOMI/FUCOMIP – Compare Real and Set EFLAGS

Код	*Команда*	*Описание*
DB F0+i	FCOMI ST, ST(i)	Сравнивает ST(0) с ST(i) и устанавливает флажки в соответствии с результатом
DF F0+i	FCOMIP ST, ST(i)	Сравнивает ST(0) с ST(i) и устанавливает флажки в соответствии с результатом. Выталкивает из стека
DB E8+i	FUCOMI ST, ST(i)	Сравнивает ST(0) с ST(i) и устанавливает флажки в соответствии с результатом
DF E8+i	FUCOMIP ST, ST(i)	Сравнивает ST(0) с ST(i) и устанавливает флажки в соответствии с результатом. Выталкивает из стека.

Операция

```
CASE (relation of operands) OF
        ST(0) > ST( i): ZF, PF, CF ← 000;
        ST(0) < ST( i): ZF, PF, CF ← 001;
        ST(0) = ST( i): ZF, PF, CF ← 100;
ESAC;
IF instruction is FCOMI or FCOMIP THEN
        IF ST(0) or ST( i) = NaN or unsupported format THEN
                #IA
                IF FPUControlWord.IM = 1 THEN
```

```
                        ZF, PF, CF ← 111;
                FI;
        FI;
FI;
IF instruction is FUCOMI or FUCOMIP THEN
        IF ST(0) or ST( i) = QNaN, but not SNaN or unsupported format THEN
                ZF, PF, CF ← 111;
        ELSE (* ST(0) or ST( i) is SNaN or unsupported format *)
                #IA;
                IF FPUControlWord.IM = 1 THEN
                        ZF, PF, CF ← 111;
                FI;
        FI;
FI;
IF instruction is FCOMIP or FUCOMIP THEN
        PopRegisterStack;
FI;
```

Описание

Сравнивает содержимое регистров ST(0) и ST(i) и устанавливает флажки ZF, PF и CF в регистре EFLAAGS в соответствии с результатом сравнения. Знак нуля игнорируется (-0.0 = +0.0).

Результат сравнения	*ZF*	*PF*	*CF*
ST0>ST(i)	0	0	0
ST0<ST(i)	0	0	1
ST0=ST(i)	1	0	0
Неупорядоченные	1	1	1

Флажки не устанавливаются, если генерируется #IA.

Команды FCOMI/FCOMIP производят те же действия, что и команды FUCOMI/FUCOMIP. Единственная разница в том, как они обрабатывают QNaN-операнды. Команды FCOMI/FCOMIP устанавливают флажки как "неупорядоченные" и генерируют исключение недействительной арифметической операции (#IA), когда один или оба операнда являются NaN-значениями (SNaN или QNaN). Команды FUCOMI/FUCOMIP не генерируют исключение недействительной арифметической операции для QNaN-операндов. Если генерируется исключение (#IA) и оно незамаскированно, то флажки не устанавливаются.

Влияние на флажки FPU

C1=0, если произошло антипереполнение стека регистров FPU, в противном случае C1=0.
C0, C2, C3 Не изменяются.

Особые случаи FPU

#IS, если произошло антипереполнение стека регистров FPU.
#IA (команда FCOMI или FCOMIP), если один или оба операнда являются NaN или представлены в неподдерживаемом формате.
(команда FUCOMI или FUCOMIP), если один или оба операнда являются SNaN (но не QNaNs) или представлены в неподдерживаемом формате. Обнаружение QNaN не приводит к исключению недействительной операции.

Особые случаи P-режима

#NM, если CR.EM=1 или CR.TS=1.

Особые случаи R-режима

#NM, если CR.EM=1 или CR.TS=1.

Особые случаи V-режима

#NM, если CR.EM=1 или CR.TS=1.

RDPMC – Read Performance-Monitoring Counters

Код	*Команда*	*Описание*
0F 33	RDPMC	Считывает счетчик производительности процессора, заданный регистром ECX, в регистры EDX:EAX

Операция

```
IF (ECX = 0 OR 1) AND ((CR4.PCE = 1) OR ((CR4.PCE = 0) AND (CPL=0)))
        THEN
                EDX:EAX ← PMC[ECX];
        ELSE (* ECX is not 0 or 1 and/or CR4.PCE is 0 and CPL is 1, 2, or 3 *)
                #GP(0)
FI;
```

Описание

Считывает в EDX:EAX счетчик событий, номер которого задан в регистре ECX. Команда RDPMC дает возможность программам на уровне привилегий больше нуля считывать счетчики событий, если установлен флаг PCE в регистре CR4.

Влияние на флажки

Нет.

Особые случаи P-режима

#GP(0), если текущий уровень привилегий не равен нулю и PCE-флаг в регистре CR4 сброшен в 0. Если значение ECX не равно нулю или единице.

Особые случаи R-режима

#GP, если PCE-флаг в регистре CR4 сброшен в 0. Если значение ECX не равно нулю или единице.

Особые случаи V-режима

#GP(0), если PCE-флаг в регистре CR4 сброшен в 0. Если значение ECX не равно нулю или единице.

UD2 – Undefined Instruction

Код	*Команда*	*Описание*
0F 0B	UD2	Генерирует исключение #UD – недействительный код операции

Операция

```
#UD (* Generates invalid opcode exception *);
```

Описание

Команда UD2 генерирует исключение #UD – недействительный код операции. Данная команда введена для тестирования обработчика исключения #UD. Кроме генерации исключения команда UD2 не выполняет никаких действий.

Влияние на флажки

Нет.

Особые случаи P-режима

#UD.

Особые случаи R-режима

#UD.

Особые случаи V-режима

#UD.

8. СТРАНИЧНОЕ ПРЕОБРАЗОВАНИЕ В ПРОЦЕССОРАХ P5 И P6

8.1. Управление страничным преобразованием

Управление страничным преобразованием осуществляется с помощью 3 битов в регистрах управления (табл. 8.1):

⇒ PG (Paging)-флаг, бит 31 в CR0. Разрешает механизм страничного преобразования;

⇒ PSE (Page Size Extension)-флаг, бит 4 в CR4. Разрешает страницы размером 4 MB (или размером 2 MB, когда установлен флаг PAE);

⇒ PAE (Physical Address Extension)-флаг, бит 5 в CR4. Разрешает 36-разрядный физический адрес. Расширенный физический адрес может быть использован только с разрешенным страничным преобразованием.

Таблица 8.1. Размеры страниц и физического адреса

CR0.PG	*CR4.PAE*	*CR4.PSE*	*PDE.PS*	*Размер страницы*	*Размер физического адреса*
0	x	x	x	-	Страничное преобразование запрещено
1	0	0	x	4 KB	32 (P5 & P6)
1	0	1	0	4 KB	32 (P5 & P6)
1	0	1	1	4 MB	32 (P5 & P6)
1	1	x	0	4 KB	36 (P6)
1	1	x	1	2 MB	36 (P6)

8.2. Таблицы страничного преобразования

Информация, которую использует процессор для преобразования линейного адреса в физический, находится в следующих таблицах:

⇒ каталоге страниц (Page Directory);

⇒ таблице страниц (Page Table);

⇒ таблице указателей на каталог страниц (Page Directory Pointer Table).

Когда используется 32-разрядный физический адрес, таблицы обеспечивают доступ к страницам размером 4 KB или 4 MB. При использовании расширенного 36-разрядного физического

адреса таблицы обеспечивают доступ к страницам размером 4 KB или 2 MB.

8.2.1. Преобразование линейного адреса для 4-KB страниц

Рис. 8.1 показывает иерархию таблиц при отображении линейного адреса на страницы размером 4 KB. Элементы каталога страниц указывают на таблицы страниц, элементы таблицы страниц – на страницы в физической памяти. Этот метод может быть использован для адресации до 2^{20} страниц, что покрывает линейное адресное пространство размером 2^{32} байт.

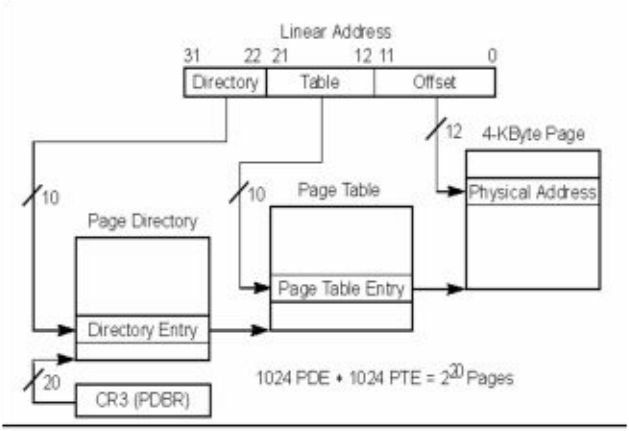


Рис. 8.1. Преобразование линейного адреса для 4-KB страниц

Для выбора различных элементов таблиц линейный адрес делится на 3 секции:

⇒ элемент каталога страниц – биты с 22-го по 31-й содержат смещение для входа в каталог страниц. Выбранный элемент содержит базовый физический адрес таблицы страниц;

⇒ элементы таблицы страниц – биты с 12-го по 21-й содержат смещение для входа в выбранную таблицу страниц. Этот элемент содержит базовый физический адрес страницы в физической памяти;

⇒ смещение – биты с 0-го по 11-й содержат смещение физического адреса внутри страницы.

8.2.2. Преобразование линейного адреса для 4-MB страниц

Рис. 8.2 показывает, как каталог страниц может быть использован для отображения линейного адреса на 4-MB страницы. Элементы каталога страниц указывают на страницы физической памяти. Этот метод страничного преобразования может быть использован для отображения до 1024 страниц на линейное адресное пространство размером 4 GB. Данный метод преобразования отсутствует в Intel486.

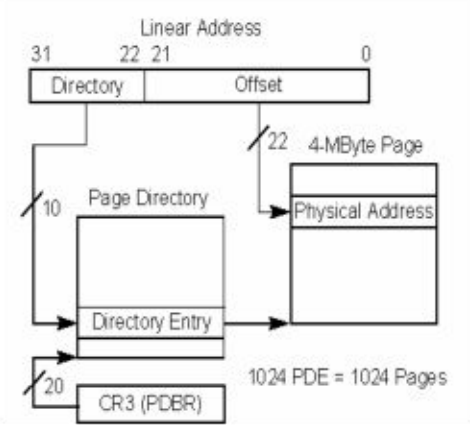


Рис. 8.2. Преобразование линейного адреса для 4 -MB страниц

Страницы размером 4 MB выбираются, когда установлен флаг PSE в регистре управления CR4 и когда установлен флаг PS в элементе каталога страниц. Когда эти флаги установлены, линейный адрес делится на две секции:

⇒ элемент каталога страниц – биты с 22-го по 31-й содержат смещение для входа в каталог страниц. Выбранный элемент содержит базовый физический адрес 4 MB-страницы.

⇒ смещение – биты с 0-го по 21-й содержат смещение физического адреса внутри страницы.

8.2.3. Элементы таблиц страничного преобразования

На рис. 8.3 показан формат элемента каталога страниц и элемента таблицы страниц, когда используются 4-KB страницы и 32-разрядный физический адрес. На рис. 8.4 показан формат элемента каталога страниц, когда используются 4 MB страницы и 32-разрядный физический адрес.

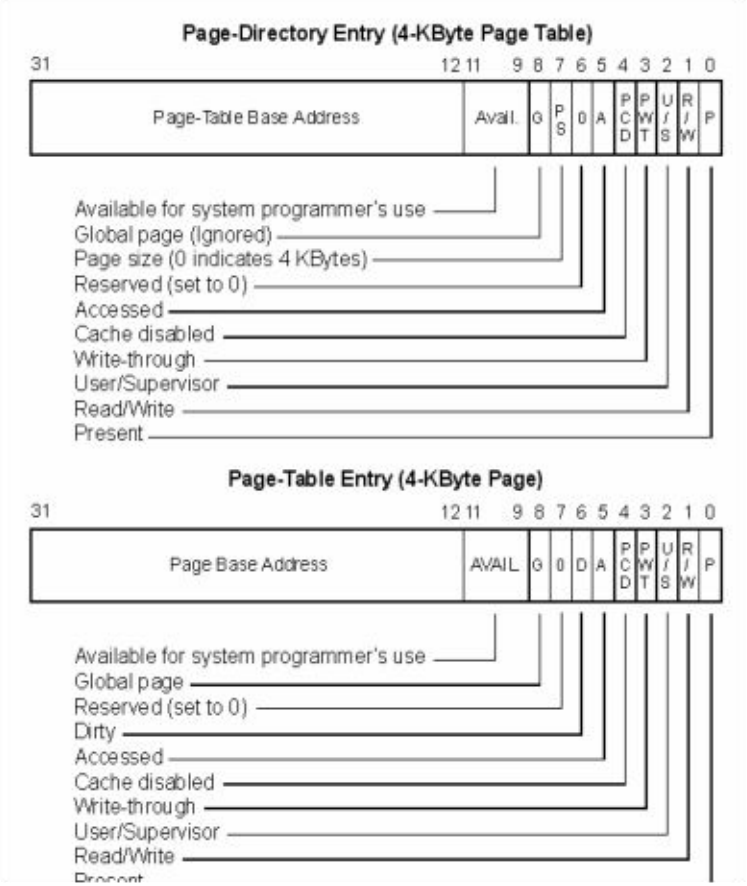


Рис. 8.3. Формат элемента каталога страниц и элемента таблицы страниц для 4-КВ страниц и 32-разрядного физического адреса

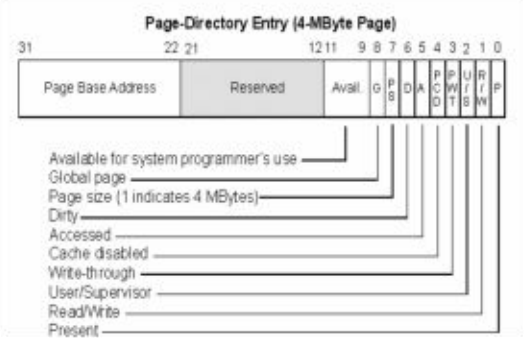


Рис. 8.4. Формат элемента каталога страниц для 4-МВ страниц и 32-разрядного физического адреса

Функции флагов и полей в элементах следующие:

⇒ **Page base address**. Базовый адрес страницы, биты с 12-го по 31-й.

- Для элемента таблицы 4-KB страниц: содержит физический адрес первого байта 4-KB страницы. Биты данного поля интерпретируются как 20 старших бит физического адреса, следовательно, страницы являются выравненными по 4-KB границам.
- Для элемента каталога 4-KB страниц: содержит физический адрес первого байта таблицы страниц. Биты данного поля интерпретируются как 20 старших бит физического адреса, следовательно, таблицы страниц являются выравненными по 4-KB границам.
- Для элемента каталога 4-MB страниц: содержит физический адрес первого байта 4-MB страницы. Используются только биты с 22-го по 32-й этого поля (биты с 12-го по 21-й зарезервированы). Эти биты интерпретируются как 11 старших бит физического адреса, следовательно, страницы являются выравненными по 4-MB границам.

⇒ **Present (P) flag. Бит присутствия, бит 0**. Показывает, находится ли страница или таблица страниц в физической памяти. Когда P=1, страница находится в памяти и при обращении к ней происходит преобразование линейного адреса в физический. Когда P=0, страницы в памяти нет, остальная часть элемента таблицы доступна для операционной системы. Если процессор попытается обратиться к странице, у которой P=0, то произойдет исключение #PF – страничное нарушение. Процессор никогда не изменяет этот флаг.

⇒ **Read/Write (R/W) flag. Бит чтение/запись, бит 1**. User/Supervisor (U/S) flag. Бит пользователь/супервизор, бит 2. Эти биты применяются для защиты по привилегиям на уровне страниц.

⇒ **Accessed (A) flag. Бит обращения к странице, бит 5**. Показывает, было ли обращение (чтение или запись) к странице или таблице страниц. Процессор устанавливает этот флаг при обращении к странице. Сброс флага осуществляется только программным путем, процессор никогда не сбрасывает этот флаг. Данный флаг совместно с флагом D используется операционной системой для управления памятью.

⇒ **Dirty (D) flag. Бит "грязный", бит 6**. Показывает, было ли обращение на запись к странице (данный флаг не используется в элементах каталога страниц). Процессор устанавливает этот флаг при записи в страницу. Сброс флага осуществляется только программным путем, процессор никогда не сбрасывает этот флаг. Данный флаг совместно с флагом A используется операционной системой для управления памятью.

⇒ **Page size (PS) flag. Размер страницы, бит 7**. Определяет размер страницы. Данный флаг используется только в элементах каталога страниц. Когда этот флаг не установлен, размер страницы 4 KB и элемент каталога страниц указывает на таблицу страниц. Когда этот флаг установлен, размер страницы равен 4 MB при использовании 32-разрядной адресации (и 2 MB при разрешенном расширенном физическом адресе) и элемент каталога страниц указывает на страницу. Когда элемент каталога страниц указывает на таблицу страниц, то размер всех страниц, содержащихся в этой таблице, равен 4 KB.

⇒ **Global (G) flag. Глобальная страница, бит 8**. Когда данный флаг установлен, это означает, что страница является глобальной. Когда страница помечена как глобальная и глобальные страницы разрешены (флаг PGE в регистре CR4 установлен), при перезагрузке регистра CR3 элемент таблицы страниц или элемент каталога страниц не объявляется недостоверным в TLB. Этот флаг служит для того, чтобы предотвратить удаление часто используемых страниц из TLB. Этот флаг может быть изменен только программным путем. Если данный флаг установлен в элементе каталога страниц, который ссылается на таблицу страниц, то он игнорируется и глобальная характеристика страницы определяется состоянием бита G в элементе таблицы страниц. Данный флаг не поддерживается процессорами семейства P5.

8.2.4. Смешивание 4-KB и 4-MB страниц

Если флаг PSE в регистре CR4 установлен, то оба метода преобразования могут использоваться из одного и того же каталога страниц. Если флаг PSE не установлен, то используется только

метод преобразования для 4-KB страниц (независимо от установки флага PS в элементе каталога страниц).

Типичным примером смешивания 4-KB и 4-MB страниц является размещение ядра операционной системы в большой странице с целью уменьшения TLB-промахов, что ведет к общему увеличению производительности системы. Процессор содержит элементы 4-KB и 4-MB страниц в разных TLB, таким образом, размещение часто используемого кода в большой странице освобождает элементы в TLB для 4-KB страниц.

8.3. Базовый адрес каталога страниц

Физический адрес текущего каталога страниц находится в регистре CR3. Если страничное преобразование будет использоваться, то процессор должен загрузить верное значение в CR3 до разрешения страничного преобразования. Значение регистра CR3 в дальнейшем может быть изменено с помощью загрузки нового значения команды MOV или в процессе переключения задач.

8.4. Расширение физического адреса

Флаг расширения физического адреса (Physical Address Extension – PAE) в регистре CR4 разрешает расширение физического адреса в процессоре Pentium Pro с 32 до 36 разрядов. Эта возможность может быть использована только при включенном страничном преобразовании (т. е. когда оба флага – PG в регистре CR0 и PAE в регистре CR4 – установлены). Процессорами семейства P5 расширение физического адреса НЕ ПОДДЕРЖИВАЕТСЯ.

Когда расширение физического адреса разрешено, процессор поддерживает страницы двух размеров: 4 KB и 2 MB. Как и при 32-разрядном адресе, оба размера страниц могут использоваться одновременно, т. е. элемент каталога страниц может ссылаться на 2-MB страницу или на каталог страниц, который содержит указатели на 4-KB страницы.

Для поддержки 36-разрядного физического адреса в таблицах для страничного преобразования сделаны следующие изменения:

⇒ Элементы таблиц стали 64-разрядными для размещения 36-разрядного физического адреса. Таким образом, каждый каталог страниц и каждая таблица страниц может содержать 512 элементов.

⇒ В иерархию страничного преобразования добавлена новая таблица – таблица указателей на каталог страниц. Эта таблица содержит

четыре 64-разрядных элемента и находится выше каталога страниц в иерархии. Когда расширение физического адреса разрешено, процессор поддерживает до четырех каталогов страниц.

⇒ Базовый 20-разрядный адрес каталога страниц в регистре CR3 заменен на 27-разрядный адрес таблицы указателей на каталог страниц (рис. 8.5). Это 27 старших разрядов физического адреса первого байта таблицы указателей на каталог страниц, следовательно, эта таблица является выравненной по 32-байтной границе.

⇒ Теперь 32-разрядный линейный адрес может отображаться на 36-разрядное пространство физических адресов.

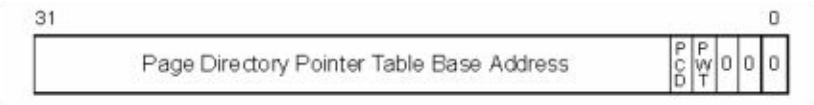


Рис. 8.5. Формат регистра CR3, когда разрешено расширение физического адреса

8.4.1. Преобразование линейного адреса с расширенной адресацией для 4-KB страниц

На рис. 8.6 показана иерархия таблиц, используемая для отображения линейного адреса на 4-KB страницы, когда разрешено расширение физического адреса. Этот метод преобразования может быть использован для адресации до 2^{20} страниц, что покрывает линейное адресное пространство размером 2^{32} байт.

Для выбора различных элементов таблиц линейный адрес делится на 4 секции:

⇒ элемент таблицы указателей на каталог страниц – биты 30 и 31 содержат смещение для входа в таблицу указателей на каталог страниц. Выбранный элемент содержит базовый физический адрес каталога страниц;

⇒ элемент каталога страниц – биты с 21-го по 29-й содержат смещение для входа в выбранный каталог страниц. Выбранный элемент содержит базовый физический адрес таблицы страниц;

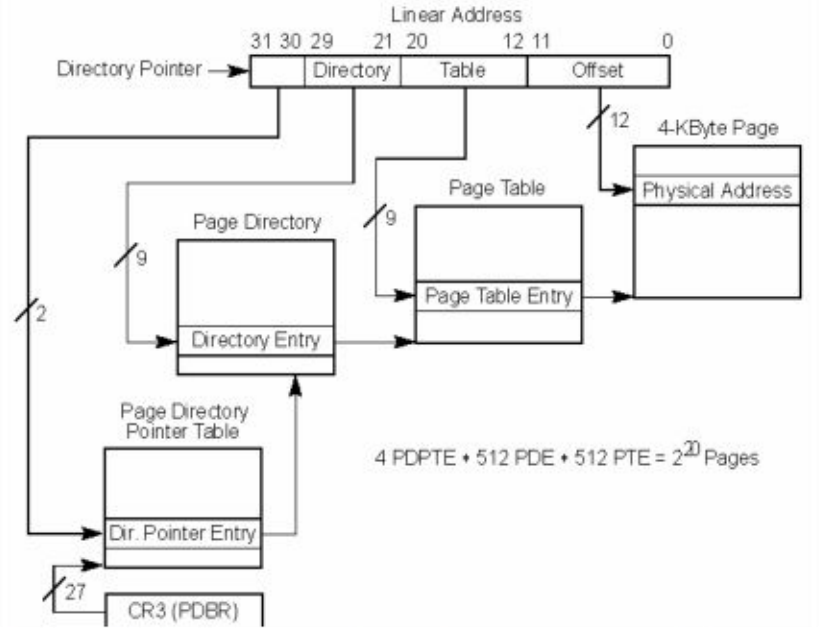


Рис. 8.6. Преобразование линейного адреса с расширенной адресацией для 4-КВ страниц

⇒ элемент таблицы страниц – биты с 12-го по 20-й содержат смещение для входа в выбранную таблицу страниц. Выбранный элемент содержит базовый физический адрес страницы в памяти;

⇒ смещение – биты с 0-го по 11-й содержат смещение физического адреса внутри страницы.

8.4.2. Преобразование линейного адреса с расширенной адресацией для 2-MB страниц

Рис. 8.7 показывает, как таблица указателей на каталог страниц и каталог страниц могут быть использованы для отображения линейного адреса на 2-MB страницы. Этот метод страничного преобразования может быть использован для отображения до 2048 страниц(4 элемента таблицы указателей на каталог страниц умножить на 512 элементов каталога страниц) на линейное адресное пространство размером 4 GB.

Размер страницы 2 MB выбирается, когда установлен флаг PSE в регистре CR4 и установлен флаг PS в элементе каталога страниц. Когда эти флаги установлены, линейный адрес делится на 3 секции:

⇒ элемент таблицы указателей на каталог страниц – биты 30 и 31 содержат смещение для входа в таблицу указателей на каталог страниц. Выбранный элемент содержит базовый физический адрес каталога страниц;

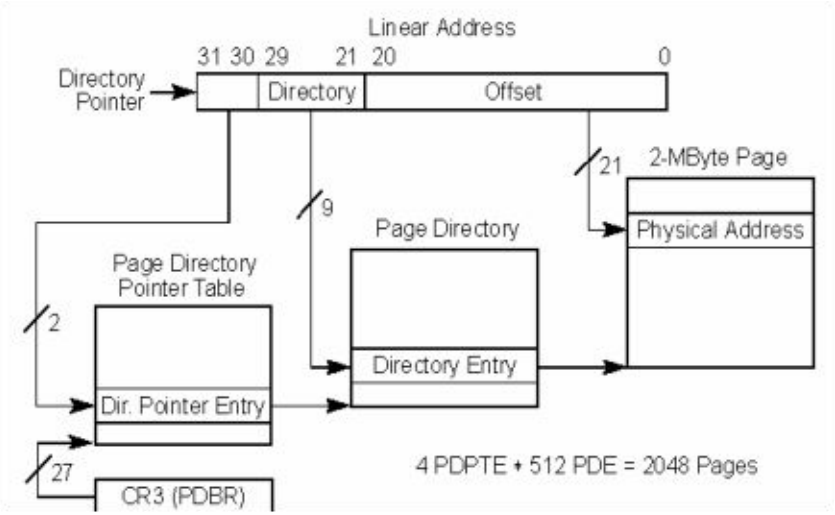


Рис. 8.7. Преобразование линейного адреса с расширенной адресацией для 2-MB страниц

⇒ элемент каталога страниц – биты с 21-го по 29-й содержат смещение для входа в выбранный каталог страниц. Выбранный элемент содержит базовый физический адрес страницы размером 2 MB;

⇒ смещение – биты с 0-го по 20-й содержат смещение физического адреса внутри страницы.

8.4.3. Элементы таблиц страничного преобразования с расширенной адресацией

На рис. 8.8 показан формат элемента таблицы указателей на каталог страниц, элемента каталога страниц и элемента таблицы страниц, когда используются 4-KB страницы и 36-разрядный физический адрес. На рис. 8.9 показан формат элемента таблицы указателей на каталог страниц и элемента каталога страниц, когда используются 2-MB страницы и 36-разрядный физический адрес.

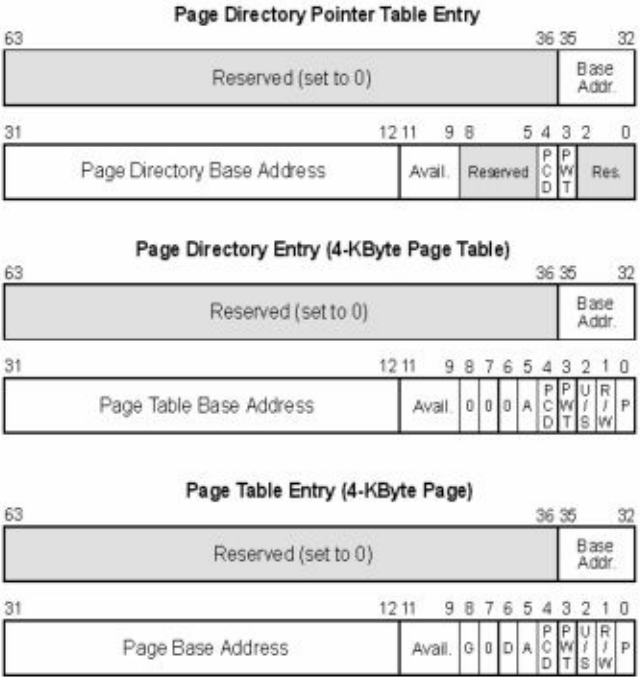


Рис. 8.8. Формат элемента таблицы указателей на каталог страниц, элемента каталога страниц и элемента таблицы страниц для 4-КВ страниц и 36-разрядного физического адреса

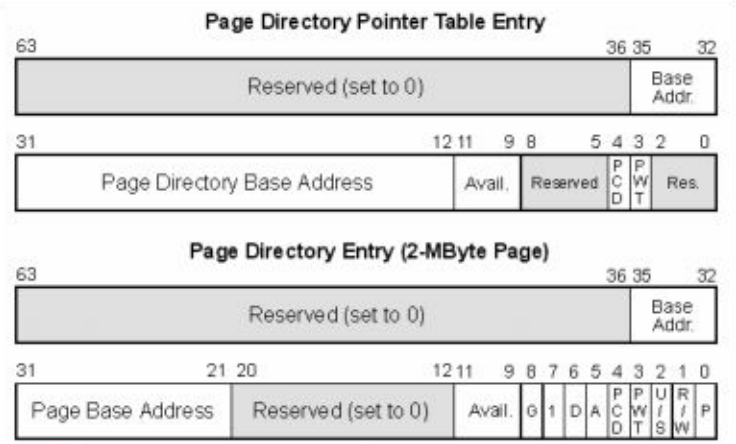


Рис. 8.9. Формат элемента таблицы указателей на каталог страниц и элемента каталога страниц для 2-МВ страниц и 36-разрядного физического адреса

Функции флагов и полей в элементах следующие:

⇒ **Page base address**. Базовый адрес страницы, биты с 12-го по 35-й.

- Для элементов всех таблиц (кроме элемента каталога страниц, когда он указывает на 2-MB страницу) содержит физический адрес первого байта 4-KB страницы или таблицы. Биты данного поля интерпретируются как 24 старших бита 36-разрядного физического адреса, следовательно, страницы и таблицы являются выравненными по 4-KB границам.
- Для элемента каталога 2-MB страниц содержит физический адрес первого байта 2-MB страницы. Используются только биты с 21-го по 35-й этого поля (биты с 12-го по 20-й зарезервированы). Биты данного поля интерпретируются как 15 старших бит 36-разрядного физического адреса, следовательно, 2-MB страницы являются выравненными по 2-MB границам.

⇒ Остальные флаги в элементах таблиц имеют те же значения, что и для страничного преобразования без расширения разрядности физического адреса (см. 8.2.3 – “Элементы таблиц страничного преобразования”).

9. ВИРТУАЛЬНЫЕ ПРЕРЫВАНИЯ

В процессорах семейства P5 в механизм обработки прерываний были добавлены некоторые новые возможности.

Рассмотрим изменения, касающиеся V-режима. Когда процессор находится в V-режиме, метод обработки прерываний зависит от состояния различных флагов и полей:

⇒ VME-флаг (бит 0 в регистре CR4) – когда WME=1, разрешаются расширенные возможности по обработке прерываний и исключений в V-режиме, а также разрешается аппаратная поддержка флага VIF; когда WME=0, расширенные возможности запрещены;

⇒ IOPL-поле (биты 12 и 13 в регистре EFLAGS) – управляет тем, как будут обрабатываться прерывания;

⇒ Interrupt redirection bit map – битовая карта перенаправления прерываний (32 байта в сегменте TSS) – содержит 256 флагов, которые показывают, как программные прерывания (прерывания, генерируемые командой INT n) будут обрабатываться в V-режиме. Программные прерывания могут направляться на обработку в текущую программу 8086 или обрабатываться в P-режиме. Аппаратные прерывания и исключения всегда направляются на обработку в P-режим.

В табл. 9.1 показано, как процессор обрабатывает прерывания и исключения в V-режиме в зависимости от состояния описанных выше флагов и полей.

Таблица 9.1. Обработка прерываний и исключений в V-режиме

Метод	*VME*	*IOPL*	*Бит в карте перенаправления*	*Действия процессора*
1	0	3	x	***Программные прерывания* (INT n), *исключения и маскируемые прерывания* (INTR#):** направляются в P-режим (через IDT)
2	0	< 3	x	***Программные прерывания* (INT n):** генерируется исключение #GP. ***Исключения и маскируемые прерывания* (INTR#):** направляются в P-режим (через IDT)
3	1	< 3	1	***Программные прерывания* (INT n):** генерируется исключение #GP. ***Исключения и маскируемые прерывания* (INTR#):** направляются в P-режим (через IDT)

Метод	VME	IOPL	Бит в карте перенаправления	Действия процессора
4	1	3	1	***Программные прерывания* (INT n), *исключения и маскируемые прерывания* (INTR#):** направляются в P-режим (через IDT)
5	1	3	0	***Программные прерывания* (INT n):** перенаправляются в V-режим по механизму R-режима. ***Исключения и маскируемые прерывания* (INTR#):** направляются в P-режим (через IDT)
6	1	< 3	0	***Программные прерывания* (INT n)**: генерируется исключение #GP. ***Исключения и маскируемые прерывания* (INTR#):** бработываются с поддержкой флагов VIF и VIP

Когда флаг WME=0, действия процессора P5 по обработке прерываний ничем не отличаются от действий Intel486. Когда флаг WME=1, к обработке прерываний привлекается новый объект – битовая карта перенаправления прерываний (рис. 9.1). Она располагается по базовому адресу карты ввода-вывода минус 32 байта в сегменте TSS. Каждый бит этой карты показывает, будет ли соответствующее прерывание обрабатываться как обычно в P-режиме (бит равен единице) или прерывание будет перенаправлено на обработку в программу V-режима через таблицу прерываний, расположенную по линейному адресу 0 (бит равен нулю).

Обработка прерываний по методу 5 (см. табл. 9.1). Процессор перенаправляет программные прерывания в программу V-режима через таблицу прерываний, находящуюся по линейному адресу 0 (как это происходит в R-режиме). Процессор производит следующую последовательность действий:

1. Записывает текущие значения регистров CS и IP в текущий стек.
2. Записывает в стек регистр FLAGS с очищенными NT и IOPL.
3. Очищает флаг IF в регистре EFLAGS, чтобы запретить прерывания.
4. Очищает флаг TF в регистре EFLAGS.
5. Загружает регистры CS и EIP из таблицы прерываний по линейному адресу 0 (загружаются младшие 16 бит регистра EIP, старшие 16 бит устанавливаются в 0).
6. Начинает выполнять программу прерывания.

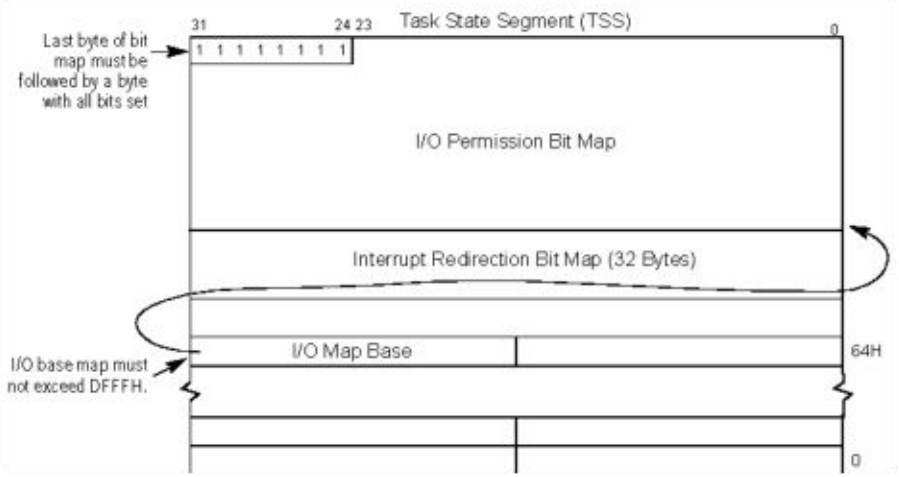


Рис. 9.1. Битовая карта перенаправления прерываний

При использовании этого метода обработки прерываний исключения не заносят в стек код ошибки.

Обработка прерываний по методу 6 (см. табл. 9.1). Когда процессор находится в состоянии, соответствующем методу 6, команды CLI и STI изменяют флаг VIF вместо флага IF. Когда программа 8086 исполняет команду CLI, процессор устанавливает VIF-флаг, когда команду STI – процессор сбрасывает флаг VIF. Процессор читает флаг VIF, но никогда не изменяет его. Процессор использует флаги VIF и VIP для определения того, как обрабатывать прерывания. Если произошло прерывание или исключение и флаг VIF сброшен в 0 (маскируемые аппаратные прерывания разрешены), то процессор производит те же действия, что и в методе 5 (перенаправляет прерывания в V-режим). Процессор также обрабатывает прерывания по методу 5, если флаг VIF установлен в единицу и произошло немаскируемое прерывание (NMI) или исключение. Если произошло маскируемое аппаратное прерывание (векторы с 32-го по 255-й) и VIF=1, то процессор производит следующую последовательность действий:

1. Процессор вызывает #GP:
   - Переключается в P-режим на CPL=0.
   - Сохраняет в PL0-стеке: EIP, CS, EFLAGS, ESP, SS, ES, DS, FS и GS. В стековом образе регистра EFLAGS поле IOPL устанавливается равным трем и флаг VIF копируется в флаг IF.

- Очищает сегментные регистры.
- Сбрасывает флаг VM в регистре EFLAGS.
- Приступает к выполнению выбранного прерывания.

2. Рекомендуемые действия обработчика #GP: прочитать VM флаг-регистра EFLAGS в стеке. Если этот флаг установлен, вызвать монитор V-режима.
3. Монитор V-режима читает флаг VIF в регистре EFLAGS. Если этот флаг установлен, монитор устанавливает флаг VIP в регистре EFLAGS, возвращает управление в обработчик #GP.
4. Обработчик #GP производит возвращение в V-режим.
5. Процессор продолжает выполнение программы 8086 без обработки прерывания.

Когда программа 8086 выполнит команду STI, процессор сделает следующее:

1. Проверит флаг VIP.
   - Если VIP=0, то процессор сбросит в 0 VIF-флаг.
   - Если VIP=1, то процессор генерирует #GP.
2. Рекомендуемые действия обработчика #GP: вызвать монитор V-режима, чтобы дать возможность обработать отложенное прерывание.

Типичные действия монитора V-режима – это очистить флаги VIF и VIP регистра EFLAGS в стеке и вернуть управление в V-режим (через обработчик #GP). Когда в следующий раз произойдет маскируемое аппаратное прерывание (при условии, что флаг VIF все еще сброшен в 0), оно будет обрабатываться по методу 5.

Состояние флагов VIF и VIP не изменяется в R-режиме или во время переключения между R-режимом и P-режимом.

Рассмотрим виртуальные прерывания в P-режиме. Для разрешения поддержки флагов VIF и VIP необходимо установить флаг PVI в регистре CR4.

Если PVI=1, CPL=3 и IOPL<3, то выполнение команд CLI и STI не приводит к генерации #GP. В этом случае команды CLI и STI воздействуют на флаг VIF вместо флага IF.

Команда CLI устанавливает флаг VIF, а команда STI сбрасывает его.

Если происходит маскируемое аппаратное прерывание и флаг VIF=1, то процессор вызывает #GP. Обработчик исключения

#GP может установить флаг VIP и вернуть управление PL3 программе, которая продолжит свою работу.

Когда программа выполнит команду STI, чтобы очистить флаг VIF, процессор автоматически осуществит вызов исключения #GP, которое сможет обработать отложенное прерывание.

Типичный метод обработки отложенного прерывания заключается в очистке флагов VIP и VIF в стековом образе регистра EFLAGS и возврате управления PL3-программе.

Когда в следующий раз произойдет аппаратное прерывание, процессор будет обрабатывать его нормальным образом.

Механизм обработки виртуальных прерываний применим только к аппаратным маскируемым прерываниям (векторы с 32-го по 255-й). Немаскируемые прерывания и исключения обрабатываются нормальным образом.

Когда виртуальные прерывания запрещены (CR4.PVI=0, или CPL<3, или IOPL=3), то команды CLI и STI выполняются так же, как в процессоре Intel486, т. е. если CPL>IOPL, то генерируется исключение #GP, если IOPL=3, то команды CLI и STI воздействуют на флаг IF.

Команды PUSHF, POPF и IRET выполняются так же, как в процессоре Intel486, независимо от разрешения виртуальных прерываний.

10. МОНИТОРИНГ ПРОИЗВОДИТЕЛЬНОСТИ

Мониторинг производительности осуществляется при помощи MSR регистров. Для доступа к MSR-регистрам используются команды RDMSR и WRMSR. В табл. 10.1 приведен список MSR регистров, относящихся к мониторингу производительности.

Таблица 10.1. MSR-регистры, относящиеся к мониторингу производительности

Значение ECX	*Регистр*
10h	Time Stamp Counter
11h	Control and Event Select
12h	Counter 0
13h	Counter 1

Регистр с номером 10h представляет собой 64-разрядный счетчик, который увеличивается в каждом такте, однако не гарантируется, что это будет так в последующих процессорах; гарантируется лишь, что значение TSC будет монотонно возрастать. Программы не должны использовать TSC для подсчета количества прошедших тактов. С помощью команды RDTSC счетчик может быть доступен для считывания с любого уровня привилегий, если CR4.TSD=0. Команда RDTSC и соответствующий регистр – TSC – будут поддерживаться всеми последующими моделями процессоров, в то время как доступ к TSC через команды RDMSR/WRMSR будет зависеть от конкретной реализации.

Регистры с номерами 12h и 13h представляют собой 40-разрядные счетчики CTR0 и CTR1. Каждый счетчик может быть запрограммирован для счета различных событий. Выбор события осуществляется при помощи регистра с номером 11h – Control and Event Select Register (CESR). Счетчики не изменяются при записи в регистр CESR и должны устанавливаться вручную при переключении на новое событие. Формат регистра CESR приведен на рис. 10.1.

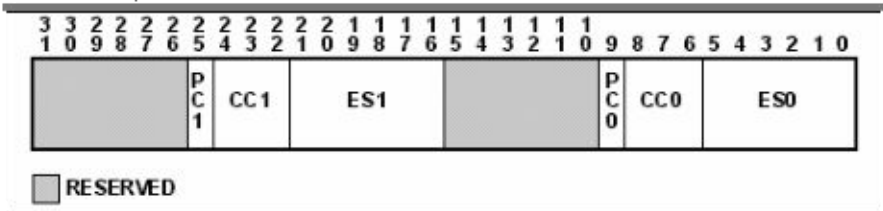


Рис. 10.1. Формат регистра CESR

Поля CC0 и CC1 служат для управления счетом. Функции этих полей приведены в табл. 10.2.

Таблица 10.2. Функции полей CC в регистре CESR

Поле CC	*Значение*
000	Запрещение счета
001	Считает выбранное событие, когда CPL<3
010	Считает выбранное событие, когда CPL=3
011	Считает выбранное событие при любом CPL
100	Запрещение счета
101	Считает такты, когда CPL<3
110	Считает такты, когда CPL=3
111	Считает такты при любом CPL

Биты PC служат для управления внешними выводами PM0/BP1 и PM1/BP1. Подробно назначение этих выводов здесь не рассматривается.

Поля ES0 и ES1 выбирают два независимых события, которые будут подсчитываться счетчиками CTR0 и CTR1. Список событий приведен в табл. 10.3.

Таблица 10.3. События для мониторинга производительности P5

Номер	*CTR0*	*CTR1*	*Событие*	*Количество или длительность*
0	Да	Да	Data Read - чтение данных	К
1	Да	Да	Data Write - запись данных	К
2	Да	Да	Data TLB Miss - TLB-промах при обращении к данным	К
3	Да	Да	Data Read Miss - кэш-промах при считывании данных	К
4	Да	Да	Data Write Miss - кэш-промах при записи данных	К

Номер	CTR 0	CTR1	Событие	Количество или длительность
5	Да	Да	Write (hit) to M or E state lines - кэш-попадание в M или E строку при записи	K
6	Да	Да	Data Cache Lines Written Back - записанные в память строки кэша данных	K
7	Да	Да	External Data Cache Snoops	K
8	Да	Да	External Data Cache Snoop Hits	K
9	Да	Да	Memory Accesses in Both Pipes - обращения к памяти из обоих каналов	K
10	Да	Да	Bank Conflicts – конфликт банков	K
11	Да	Да	Misaligned Data Memory or I/O References - невыравненное обращение к памяти или портам ввода-вывода	K
12	Да	Да	Code Read – считывание кода	K
13	Да	Да	Code TLB Miss – TLB-промах при считывании кода	K
14	Да	Да	Code Cache Miss – кэш-промах при считывании кода	K
15	Да	Да	Any Segment Register Loaded – загрузка любого сегментного регистра	K
16	Да	Да	Reserved – зарезервировано	
17	Да	Да	Reserved – зарезервировано	
18	Да	Да	Branches – ветвления	K
19	Да	Да	BTB Predictions – предсказания через BTB	K
20	Да	Да	Taken Branch or BTB hit – выполненный переход	K
21	Да	Да	Pipeline Flushes – сбросы конвейера	K
22	Да	Да	Instructions Executed – выполненные команды	K
23	Да	Да	Instructions Executed in the V-pipe e. g. parallelism/pairing – команды, выполненные в V-канале	K
24	Да	Да	Clocks while a bus cycle is in progress (bus utilization) – такты, в течение которых выполняются циклы шины	Д

Номер	CTR0	CTR1	Событие	Количество или длительность
25	Да	Да	Number of clocks stalled due to full write buffers – ожидание из-за переполнения буферов записи	Д
26	Да	Да	Pipeline stalled waiting for data memory read – ожидание из-за чтения данных из памяти	Д
27	Да	Да	Stall on write to an E or M state line – ожидание из-за записи в E или M строку	Д
29	Да	Да	I/O Read or Write Cycle циклы ввода-вывода	К
30	Да	Да	Non-cacheable memory reads – некэшируемые чтения из памяти	К
31	Да	Да	Pipeline stalled because of an address generation interlock – ожидание из-за блокировки генерации адреса	Д
32	Да	Да	Reserved – зарезервировано	
33	Да	Да	Reserved – зарезервировано	
34	Да	Да	FLOPs – команды FPU	К
35	Да	Да	Breakpoint match on DR0 Register – срабатывание контрольной точки в регистре DR0	К
36	Да	Да	Breakpoint match on DR1 Register – срабатывание контрольной точки в регистре DR1	К
37	Да	Да	Breakpoint match on DR2 Register - срабатывание контрольной точки в регистре DR2	К
38	Да	Да	Breakpoint match on DR3 Register – срабатывание контрольной точки в регистре DR3	К
39	Да	Да	Hardware Interrupts – аппаратные прерывания	К
40	Да	Да	Data Read or Data Write – чтение или запись данных	К
41	Да	Да	Data Read Miss or Data Write Miss – кэш-промахи при чтении или записи данных	К
43	Да	Нет	MMX TM instructions executed in U-pipe - команды MMX выполненные в U-канале	К
43	Нет	Да	MMX instructions executed in V-pipe – команды MMX выполненные в V-канале	К

Номер	CTR 0	CTR1	Событие	Количество или длительность
45	Да	Нет	EMMS instructions executed – выполненные команды EMMS	К
45	Нет	Да	Transition between MMX instructions and FP instructions – переход между MMX-командами и FP-командами	К
46	Нет	Да	Writes to Non-Cacheable Memory – записи в некэшируемую память	К
47	Да	Нет	Saturating MMX instructions executed – команды MMX с насыщением	К
47	Нет	Да	Saturations performed – сделанные насыщения	К
48	Да	Нет	Number of Cycles Not in HLT State – количество тактов, при котором процессор не выполняет команду HLT	Д
49	Да	Нет	MMX instruction data reads – чтение данных MMX-командами	К
50	Да	Нет	Floating Point Stalls – задержки из-за FPU команд	Д
50	Нет	Да	Taken Branches – Выполненные переходы	К
51	Нет	Да	D1 Starvation and one instruction in FIFO – обеднение на стадии D1	К
52	Да	Нет	MMX instruction data writes – запись данных MMX-командами	К
52	Нет	Да	MMX instruction data write misses – кэш-промахи при записи данных MMX-командами	К
53	Да	Нет	Pipeline flushes due to wrong branch prediction – сбросы конвейера из-за неверных предсказаний переходов	К
53	Нет	Да	Pipeline flushes due to wrong branch predictions resolved in WB-stage – сбросы конвейера из-за неверных предсказаний переходов разрешенных на стадии WB	К
54	Да	Нет	Misaligned data memory reference on MMX instruction – невыравненное обращение к данным MMX-командами	К
54	Нет	Да	Pipeline stalled waiting for MMX instruction data memory read – ожидание из-за чтения данных MMX-командами	Д

Номер	CTR 0	CTR1	Событие	Количество или длительность
55	Да	Нет	Returns Predicted Incorrectly – неверно предсказанные возвраты	К
55	Нет	Да	Returns Predicted (Correctly and Incorrectly) – предсказанные (верно и неверно) возвраты	К
56	Да	Нет	MMX instruction multiply unit interlock - блокировка устройством умножения ММХ	Д
56	Нет	Да	MOVD/MOVQ store stall due to previous operation – ожидание командами MOVD/MOVQ из-за предыдущей команды	Д
57	Да	Нет	Returns – возвраты (IRET не подсчитывается)	К
57	Нет	Да	RSB Overflows – переполнение RSB	К
58	Да	Нет	BTB false entries – BTB промахи	К
58	Нет	Да	BTB miss prediction on a Not-Taken Branch – BTB предсказывает невыполненные переходы как выполненные	К
59	Да	Нет	Number of clocks stalled due to full write buffers while executing MMX instructions – ожидание из-за переполнения буферов записи при выполнении ММХ-команд	Д
59	Нет	Да	Stall on MMX instruction write to E or M line – ожидание из-за записи ММХ- командой в E или M строку	Д

События с 43-го по 59-й добавлены в процессор Pentium MMX, эти события отсутствуют в процессоре Pentium.

11. СИСТЕМНЫЙ РЕЖИМ

Системный режим (SMM – System Management Mode) – это специальный режим, предназначенный для системных функций. Например, перевод всей системы в состояние с пониженным энергопотреблением. S-режим не используется прикладными программами. Особенностью S-режима является то, что вход и выход в него является прозрачным для операционной системы и приложений. Когда процессор переключается в S-режим, он сохраняет текущий контекст и переходит к выполнению кода, находящегося в специальной памяти SMRAM. Для завершения работы в S-режиме необходимо выполнить команду RSM. Эта команда приведет к тому, что процессор восстановит сохраненный контекст и переключится обратно в тот режим, в котором он находился до входа в S-режим. Далее процессор продолжит выполнение прерванной программы.

Особенности S-режима:

⇒ единственный способ переключиться в S-режим – это получение сигнала SMI;

⇒ процессор выполняет S-код в отдельном адресном пространстве (SMRAM), которое может быть сделано недоступным из других режимов;

⇒ при входе в S-режим процессор сохраняет контекст прерванной программы;

⇒ при входе в S-режим запрещается обработка прерываний;

⇒ команда RSM может быть выполнена только в S-режиме.

Функционирование процессора в S-режиме похоже на R-режим, где нет уровней привилегий и преобразования адресов. S-код может адресовать до 4 GB памяти, производить все операции ввода-вывода, а также работать с прерываниями.

11.1. Прерывание S-режима (SMI)

Единственный способ войти в S-режим – это получение сигнала SMI по линии SMI или сообщения SMI по шине APIC. SMI является немаскируемым внешним прерыванием, которое работает независимо от механизма обработки прерываний и исключений. SMI имеет приоритет перед NMI и маскируемыми прерываниями. Прерывание SMI запрещено, пока процессор находится в S-режиме.

11.2. Переключение между S-режимом и другими режимами

Сигнал SMI всегда приводит к переключению в S-режим, независимо от того, в каком режиме находится процессор (рис. 11.1). После выполнения команды RSM процессор всегда возвращается в тот режим, в котором он находился до возникновения сигнала SMI.

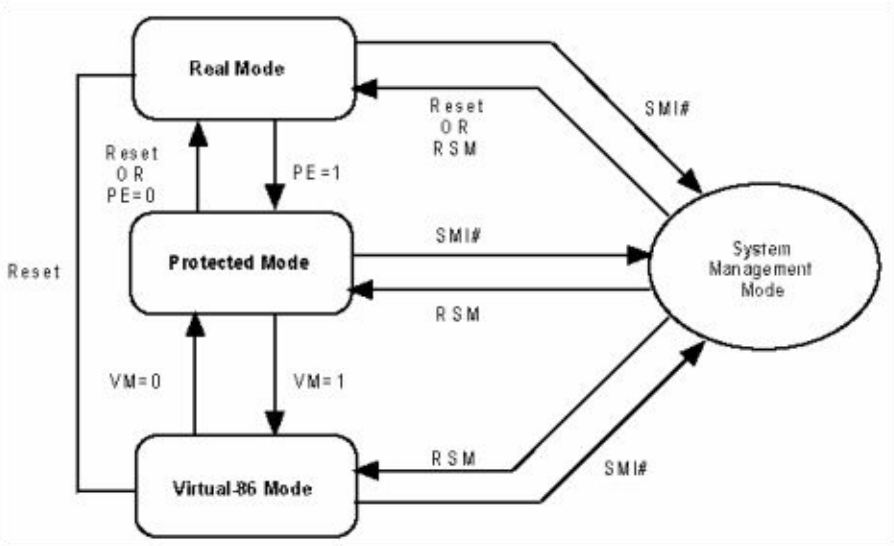


Рис. 11.1. Переключение между режимами

11.2.1. Вход в S-режим

После получения сигнала SMI процессор производит специальный цикл SMIACK на системной шине, сохраняет текущий контекст в SMRAM и начинает выполнять обработчик SMI.

SMI имеет больший приоритет, чем исключение отладки и внешние прерывания. Таким образом, если одновременно с SMI происходит NMI, маскируемое прерывание или исключение отладки, то обрабатывается только SMI. Процессор не реагирует на сигнал SMI, когда находится в S-режиме. Первое прерывание SMI, которое происходит, пока процессор в S-режиме, защелкивается и обрабатывается после выхода процессора из S-режима. Процессор защелкивает только одно прерывание SMI, пока находится в S-режиме.

11.2.2. Выход из S-режима

Единственный способ выхода из S-режима – это выполнить команду RSM. Она может быть выполнена только в S-режиме. Попытка выполнить команду RSM в любом другом режиме приводит к исключению UD – недействительный код операции.

Команда RSM восстанавливает контекст процессора путем загрузки сохраненных значений из RSRAM обратно в регистры процессора. После этого управление передается прерванной программе.

Если процессор обнаружит неверную информацию в сохраненном контексте, то он перейдет в состояние отключения, произведя специальный цикл шины, показывающий, что процессор перешел в состояние отключения. Отключение процессора происходит в следующих случаях:

⇒ при записи в регистр управления CR4 обнаружена единица в зарезервированных битах. Эта ошибка не возникнет, если обработчик SMI не модифицирует область сохраненного контекста в SMRAM;

⇒ неверная комбинация битов записывается в регистр управления CR0, например PG=1 и PE=0 или NW=1 и CD=0;

⇒ если значение SMBASE не выравнено по 32-KB границе (для процессоров P5). Для процессоров P6 это ограничение снято.

В состоянии отключения процессор прекращает выполнение команд до поступления сигналов RESET#, INIT# или NMI#. Сигнал FLUSH# распознается в состоянии отключения, а сигнал SMI# – не распознается.

Если процессор выполнял команду HALT при поступлении сигнала SMI, то возврат из S-режима может происходить по-разному. Также адрес SMBASE может быть изменен при выходе из S-режима.

11.3. Память SMRAM

Когда процессор находится в S-режиме, он исполняет код, который находится в специальной памяти SMRAM. Размер SMRAM может достигать 4 GB. По умолчанию размер SMRAM равен 64 KB. Память SMRAM начинается с физического адреса SMBASE (рис. 11.2). По умолчанию (после RESET) значение SMBASE равно 30000h. Процессор начинает выполнять обработчик SMI с адреса [SMBASE+8000h]. Контекст процессора сохраняется в области с [SMBASE+FE00h] по [SMBASE+FFFFh] (рис. 11.2).

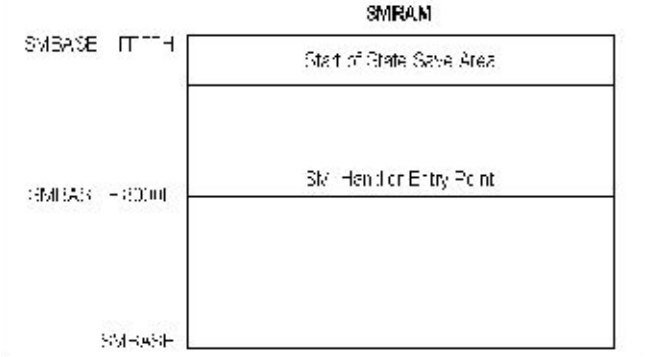


Рис. 11.2. Использование SMRAM

Местонахождение SMRAM может быть изменено с помощью изменения значения SMBASE. Следует отметить, что в мультипроцессорной системе процедура инициализации должна последовательно перевести каждый процессор в S-режим и изменить значение SMBASE так, чтобы области SMRAM различных процессоров не перекрывались.

Физически SMRAM может располагаться как в основном адресном пространстве процессора (SMRAM является частью основной памяти), так и в специальном отдельном пространстве физических адресов. При входе в S-режим процессор производит специальный цикл шины SMIACT, который может быть использован для распознавания обращения к SMRAM. Если SMRAM располагается в отдельном пространстве адресов, то логика системы должна предусматривать способ отображения SMRAM в основное адресное пространство для того, чтобы можно было произвести инициализацию SMRAM (загрузить обработчик SMI перед первым входом процессора в S-режим).

11.3.1. Область сохранения контекста

Когда процессор входит в S-режим, он сохраняет свой контекст в специальной области SMRAM. Эта область начинается с адреса [SMBASE+FFFFh]. Заполнение происходит сверху вниз до адреса [SMBASE+FF00h] (табл. 11.1). Зарезервированные поля области не должны использоваться программами.

Таблица 11.1. Область сохранения контекста

Смещение (относительно [SMBASE+8000h])	*Регистр*	*Возможна ли запись*
7FFCH	CR0	Нет
7FF8H	CR3	Нет
7FF4H	EFLAGS	Да
7FF0H	EIP	Да
7FECH	EDI	Да
7FE8H	ESI	Да
7FE4H	EBP	Да
7FE0H	ESP	Да
7FDCH	EBX	Да
7FD8H	EDX	Да
7FD4H	ECX	Да
7FD0H	EAX	Да
7FCCH	DR6	Нет
7FC8H	DR7	Нет
7FC4H	TR*	Нет
7FC0H	LDT Base*	Нет
7FBCH	GS*	Нет
7FB8H	FS*	Нет
7FB4H	DS*	Нет
7FB0H	SS*	Нет
7FACH	CS*	Нет
7FA8H	ES*	Нет
7FA7H – 7F98H	Reserved	Нет
7F94H	IDT Base	Нет
7F93H – 7F8CH	Reserved	Нет
7F88H	GDT Base	Нет
7F87H – 7F04H	Reserved	Нет
7F02H	Auto HALT Restart Field (Word)	Нет
7F00H	I/O Instruction Restart Field (Word)	Да
7EFCH	SMM Revision Identifier Field (Doubleword)	Нет
7EF8H	SMBASE Field (Doubleword)	Да
7EF7H – 7E00H	Reserved	Нет

** Верхнее слово является зарезервированным.*

Некоторые образы регистров в области сохранения контекста могут быть изменены обработчиком SMI. В этом случае после выполнения команды RSM измененные образы регистров будут записаны в регистры процессора. Некоторые образы регистров

не должны модифицироваться обработчиком SMI. Обработчик SMI не должен обращать внимание на значения зарезервированных полей области сохранения контекста.

Следующие регистры сохраняются (но являются недоступными для чтения) при входе в S-режим и восстанавливаются при выходе из S-режима:

⇒ регистр управления CR4;

⇒ теневые части регистров CS, DS, ES, FS, GS и SS.

Следующие регистры не сохраняются автоматически при входе в S-режим:

⇒ регистры отладки с DR0 по DR3;

⇒ регистры FPU;

⇒ регистры MTTR (процессоры P6);

⇒ регистр управления CR2.

При необходимости обработчик SMI должен сам сохранять и восстанавливать эти регистры.

11.3.2. Кэширование SMRAM

Если SMRAM не является частью основного адресного пространства, то следует уделить внимание управлению кэшированием, так как при переключении между основной памятью и SMRAM в кэш-памяти может оказаться недостоверная информация. Необходимо произвести сброс кэш-памяти при входе в S-режим и при выходе из него.

Если SMRAM является частью основного адресного пространства, то проблем с кэшированием не возникает.

11.4. Работа процессора в S-режиме

После сохранения контекста процессор инициализирует регистры значениями, показанными в табл. 11.2.

Таблица 11.2. Состояние регистров процессора после входа в S-режим

Регистры	*Значение*
Регистры общего назначения	Не определено
EFLAGS	00000002H
EIP	00008000H
CS selector	SMBASE, сдвинутое вправо на 4 разряда (после RESET 3000H)
CS base	SMBASE (после RESET 30000H)

Регистры	Значение
DS, ES, FS, GS, SS Selectors	0000H
DS, ES, FS, GS, SS Bases	000000000H
DS, ES, FS, GS, SS Limits	0FFFFFFFFH
CR0	Флажки PE, EM, TS и PG устанавливаются в 0; остальные не изменяются
DR6	Не определено
DR7	00000400H

После входа в S-режим флажки PE и PG в регистре управления CR0 устанавливаются в 0. Это переводит процессор в состояние, похожее на R-режим. Различия между работой процессора в R-режиме и S-режиме следующие:

⇒ диапазон адресуемой памяти SMRAM от 0h до 0FFFFFFFFh (4 GB). Расширение физического адреса (для процессоров P6) не поддерживается в S-режиме.

⇒ предел сегмента с 64 KB (R-режим) увеличен до 4 GB (S-режим).

⇒ по умолчанию размер операнда и размер адреса равен 16 битам, что ограничивает размер адресуемого пространства SMRAM до 1 MB. Однако префиксы изменения разрядности операнда и изменения разрядности адреса могут быть использованы для доступа к памяти за пределами 1 MB.

⇒ ближние переходы (JMP near, CALL near) могут быть сделаны в любое место 4-GB пространства, если использовать префикс изменения разрядности операнда. Так как формирование базового адреса сегмента происходит так же, как в R-режиме, то дальние переходы (JMP far, CALL far) не могут передавать управление в сегменты с базовым адресом больше 20 бит (1 MB). Но, так как предел сегмента в S-режиме равен 4 GB, то можно использовать 32-разрядное смещение внутри сегмента, указав префикс изменения разрядности операнда.

⇒ данные и стек могут находиться в любом месте 4-GB адресного пространства, но если они расположены выше 1 MB, то для доступа к ним необходимо использовать префикс изменения разрядности адреса. Так же как в случае сегмента кода, базовый адрес сегмента данных и сегмента стека не может быть больше 20 бит.

Значение регистра CS автоматически получается из значения SMBASE, сдвинутого на 4 разряда вправо. В регистр EIP загружается значение 8000h. Сегментные регистры DS, SS, ES, FS и GS устанавливаются в 0, их пределы устанавливаются равными 4 GB. Когда в сегментный регистр загружается 16-разрядное значение, в поле базового адреса (теневая часть сегментного

регистра) автоматически заносится новое значение сегментного регистра, сдвинутое на 4 разряда влево. Поле предела и атрибуты не изменяются.

Маскируемые прерывания, NMI-прерывания, SMI-прерывания, A20M-прерывания, пошаговое исполнение, точки останова – все это запрещается, когда процессор входит в S-режим. Маскируемые прерывания, пошаговое выполнение, точки останова могут быть разрешены в S-режиме, если будет создана таблица прерываний и необходимые обработчики прерываний.

11.5. Прерывания и исключения в S-режиме

Когда процессор входит в S-режим, все аппаратные прерывания запрещаются следующим образом:

⇒ флаг IF в режиме EFLAGS устанавливается в 0, что запрещает маскируемые прерывания;

⇒ флаг TF в режиме EFLAGS устанавливается в 0, что запрещает пошаговое исполнение;

⇒ регистр отладки DR7 очищается, что запрещает точки останова;

⇒ NMI, SMI и A20M прерывания блокируются внутренней логикой процессора.

Программные прерывания и исключения могут иметь место (они не запрещаются при входе в S-режим), маскируемые прерывания могут быть разрешены установкой флага IF в единицу. Intel рекомендует составлять S-код так, чтобы он не генерировал программных прерываний (команды INT n, INTO, INT3 или BOUND) или исключений. Если при работе в S-режиме все же требуется обрабатывать прерывания и исключения, то необходимо создать таблицу прерываний и правильно инициализировать ее. Следующие ограничения накладываются на обработку прерываний и исключений в S-режиме:

⇒ Таблица прерываний должна располагаться по линейному адресу 0.

⇒ Так как формирование базового адреса происходит по методу R-режима, прерывание или исключение не может передать управление в сегмент с базовым адресом больше 20 бит.

⇒ Прерывание или исключение не может передать управление в точку со смещением более 16 бит (64 KB).

⇒ Когда происходит прерывание или исключение, только 16 младших битов адреса возврата (EIP) заносятся в стек. Если смещение прерванной процедуры больше 64 KB, то становится невозможным вер-

нуть управление из обработчика прерывания/исключения назад в прерванную процедуру.

⇒ Если значение SMBASE указывает на область более 1 MB, то нормальный возврат из обработчика прерывания/исключения невозможен (в этом случае значение базового адреса сегмента НЕ ПОЛУЧАЕТСЯ из значения CS в адресе возврата).

⇒ Если нужно использовать возможности отладки, то необходимо создать обработчик исключения отладки и инициализировать регистры с DR0 по DR3 и регистр DR7 необходимыми значениями.

⇒ Если нужно использовать пошаговое выполнение, то необходимо создать обработчик исключения пошагового выполнения и затем установить флаг IF в регистре EFLAGS.

11.6. Обработка NMI в S-режиме

NMI-прерывания заблокированы в S-режиме. Если происходит NMI-прерывание пока процессор находится в S-режиме, то оно защелкивается и обрабатывается после выхода процессора из S-режима. Только одно прерывание может быть защелкнуто.

11.7. Идентификатор S-режима

Идентификатор S-режима (рис. 11.3) находится в SMRAM по адресу 7EFCH. Младшее слово идентификатора содержит версию базовой архитектуры S-режима.

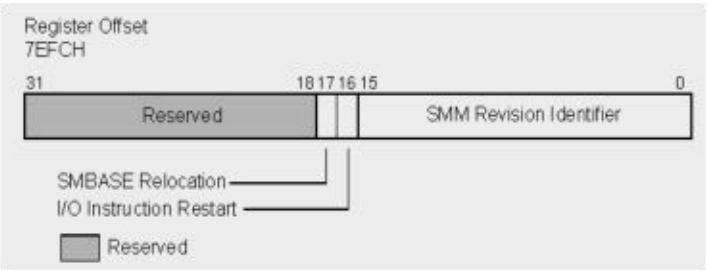


Рис. 11.3. Идентификатор S-режима

Старшее слово идентификатора содержит информацию о доступных расширениях S-режима. Если I/O instruction restart flag – флаг рестарта команд ввода-вывода (бит 16) установлен, то процессор поддерживает рестарт команд ввода-вывода. Если SMBASE relocation flag – флаг переопределения адреса SMBASE (бит 17) установлен, то процессор поддерживает переопределение адреса SMBASE.

11.8. Рестарт HALT

Если процессор находится в состоянии HALT (выполняет команду HLT) и в это время происходит SMI, то процессор устанавливает auto HALT restart flag – флаг рестарта HALT, который находится в области сохранения контекста (смещение 7F02H) (рис. 11.4)

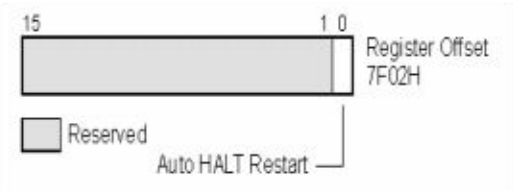


Рис. 11.4. Поле рестарта HALT

Обработчик SMI имеет две возможности:

⇒ обработчик может оставить флаг рестарта HALT установленным. Это приведет к тому, что после выполнения команды RSM управление будет передано на команду HLT. Процессор выполнит команду HLT и снова войдет в состояние HALT;

⇒ обработчик может очистить флаг рестарта HALT. Это приведет к тому, что после выполнения команды RSM управление будет передано на следующую команду после HLT.

Если процессор не находился в состоянии HALT, когда произошло SMI (флаг рестарта HALT равен нулю), то установка флага в единицу приведет к непредсказуемому поведению процессора после выполнения команды RSM. Все возможные состояния флага рестарта HALT приведены в табл. 11.3.

Таблица 11.3. Значения флага рестарта HALT

Значение флага после входа в S-режим	*Значение флага перед выходом из S-режима*	*Реакция процессора*
0	0	Возврат к следующей команде в прерванной программе
0	1	Непредсказуемые действия
1	0	Возврат к следующей команде после HLT
1	1	Возврат к команде HLT (возврат в состояние HALT)

11.8.1. Выполнение команды HLT в S-режиме

Команда HLT не должна выполняться в S-режиме до разрешения маскируемых прерываний. Если процессор переведен в состояние HALT в S-режиме, то единственное событие, которое может вывести его из этого состояния, – это маскируемое аппаратное прерывание, или RESET.

11.9. Переопределение адреса SMBASE

По умолчанию значение SMBASE равно 30000H (после RESET). Это значение хранится во внутреннем регистре процессора. Обработчик SMI может изменить значение SMBASE в области сохранения контекста (смещение 7EF8H) на новое значение. При выполнении команды RSM значение поля SMBASE в области сохранения контекста загружается во внутренний регистр SMBASE. Все последующие SMI будут использовать новое значение SMBASE. В мультипроцессорной системе программа инициализации должна изменить значение SMBASE каждого процессора так, чтобы области сохранения контекста для каждого процессора не перекрывались.

11.10. Рестарт команд ввода-вывода

Если флаг рестарта команд ввода-вывода в идентификаторе S-режима установлен, то процессор поддерживает рестарт команд ввода-вывода. Например, если команда ввода-вывода используется для доступа к устройству, находящемуся в режиме с пониженным энергопотреблением, то это устройство может ответить сигналом SMI#. Обработчик SMI выведет устройство из состояния с пониженным энергопотреблением и выполнит возврат из S-режима с рестартом команды ввода-вывода, вызвавшей SMI. Поле рестарта команды ввода-вывода (смещение 7F00H) управляет рестартом команд ввода-вывода. Если это поле содержит FFH при выполнении команды RSM, то происходит возврат управления на команду ввода-вывода. Если поле рестарта содержит 00H при выполнении команды RSM, то происходит возврат к команде, следующей за командой ввода-вывода (когда используется префикс повторения, следующей командой будет следующая итерация цикла). Процессор автоматически инициализирует поле рестарта команд ввода-вывода значением 00H при входе в S-режим. В табл. 11.4 приведены возможные состояния поля рестарта команд ввода-вывода.

Таблица 11.4. Состояния поля рестарта команд ввода-вывода

Значение поля после входа в S-режим	*Значение поля при выходе из S-режима*	*Действия процессора*
00h	00h	Не рестартовать прерванную команду ввода-вывода
00h	FFh	Рестартовать прерванную команду ввода-вывода

Обработчик SMI должен самостоятельно определять причину SMI. Механизм рестарта команд ввода-вывода не сообщает о причине SMI (процессор никак не сообщает, была ли прервана команда ввода-вывода).

Приложение А
Спаривание целочисленных команд

Обозначения

NP – команды не могут спариваться ни при каких условиях.

UV – команды могут спариваться, направляясь в любой из каналов (U или V).

PU – команды не могут исполняться в V-канале, но могут спариваться с другими командами, которые могут направляться в V-канал.

PV – команды могут выполняться в любом из каналов (U или V), но спариваются с другими командами только тогда, когда направляются в V-канал.

Команда	*Разновидность*	*Спаривание*
AAA – ASCII Adjust after Addition		NP
AAD – ASCII Adjust AX before Division		NP
AAM – ASCII Adjust AX after Multiply		NP
AAS – ASCII Adjust AL after Subtraction		NP
ADC – ADD with Carry		PU
ADD – Add		UV
AND – Logical AND		UV
ARPL – Adjust RPL Field of Selector		NP
BOUND – Check Array Against Bounds		NP
BSF – Bit Scan Forward		NP
BSR – Bit Scan Reverse		NP
BSWAP – Byte Swap		NP
BT – Bit Test		NP
BTC – Bit Test and Complement		NP
BTR – Bit Test and Reset		NP
BTS – Bit Test and Set		NP
CALL – Call Procedure (in same segment)		
direct	1110 1000 : full displacement	PV
register indirect	1111 1111 : 11 010 reg	NP
memory indirect	1111 1111 : mod 010 r/m	NP
CALL – Call Procedure (in other segment)		NP
CBW – Convert Byte to Word CWDE – Convert Word to oubleword		NP
CLC – Clear Carry Flag		NP

Команда	Разновидность	Спаривание
CLD – Clear Direction Flag		NP
CLI – Clear Interrupt Flag		NP
CLTS – Clear Task-Switched Flag in CR0		NP
CMC – Complement Carry Flag		NP
CMP – Compare Two Operands		UV
CMPS/CMPSB/CMPSW/CMPSD – Compare String Operands		NP
CMPXCHG – Compare and Exchange		NP
CMPXCHG8B – Compare and Exchange 8 Bytes		NP
CWD – Convert Word to Dword CDQ – Convert Dword to Qword		NP
DAA – Decimal Adjust AL after Addition		NP
DAS – Decimal Adjust AL after Subtraction		NP
DEC – Decrement by 1		UV
DIV – Unsigned Divide		NP
ENTER – Make Stack Frame for Procedure Parameters		NP
HLT – Halt		
IDIV – Signed Divide		NP
IMUL – Signed Multiply		NP
INC – Increment by 1		UV
INT n – Interrupt Type n		NP
INT – Single-Step Interrupt 3		NP
INTO – Interrupt 4 on Overflow		
NP		
INVD – Invalidate Cache		NP
INVLPG – Invalidate TLB Entry		NP
IRET/IRETD – Interrupt Return		NP
Jcc – Jump if Condition is Met		PV
JCXZ/JECXZ – Jump on CX/ECX Zero		NP
JMP – Unconditional Jump (to same segment)		
short	1110 1011 : 8-bit displacement	PV
direct	1110 1001 : full isplacement	PV
register indirect	1111 1111 : 11 100 reg	NP
memory indirect	1111 1111 : mod 100 r/m	NP
JMP – Unconditional Jump (to other segment)		NP
LAHF – Load Flags into AH Register		NP

Команда	Разновидност	Спари-ване
LAR – Load Access Rights Byte		NP
LDS – Load Pointer to DS		NP
LEA – Load Effective Address		UV
LEAVE – High Level Procedure Exit		NP
LES – Load Pointer to ES		NP
LFS – Load Pointer to FS		NP
LGDT – Load Global Descriptor Table Register		NP
LGS – Load Pointer to GS		NP
LIDT – Load Interrupt Descriptor Table Register		NP
LLDT – Load Local Descriptor Table Register		NP
LMSW – Load Machine Status Word		NP
LOCK – Assert LOCK# Signal Prefix		
LODS/LODSB/LODSW/LODSD – Load String Operand		NP
LOOP – Loop Count		NP
LOOPZ/LOOPE – Loop Count while Zero/Equal		NP
LOOPNZ/LOOPNE – Loop Count while not Zero/Equal		NP
LSL – Load Segment Limit		NP
LSS – Load Pointer to SS	0000 1111 : 1011 0010 : mod reg r/m	NP
LTR – Load Task Register		NP
MOV – Move Data		UV
MOV – Move to/from Control Registers		NP
MOV – Move to/from Debug Registers		NP
MOV – Move to/from Segment Registers		NP
MOVS/MOVSB/MOVSW/MOVSD – Move Data from String to String		NP
MOVSX – Move with Sign-Extend		NP
MOVZX – Move with Zero-Extend		NP
MUL – Unsigned Multiplication of AL, AX or EAX		NP
NEG – Two's Complement Negation		NP
NOP – No Operation 1001 0000		UV
NOT – One's Complement Negation		NP
OR – Logical Inclusive OR		UV
POP – Pop a Word from the Stack		
	reg 1000 1111 : 11 000 reg	UV

Команда	Разновидность	Спаривание
or	0101 1 reg	UV
memory	1000 1111 : mod 000 r/m	NP
POP – Pop a Segment Register from the Stack		NP
POPA/POPAD – Pop All General Registers		NP
POPF/POPFD – Pop Stack into FLAGS or EFLAGS Register		NP
PUSH – Push Operand onto the Stack		
	reg 1111 1111 : 11 110 reg	UV
or	0101 0 reg	UV
memory	1111 1111 : mod 110 r/m	NP
immediate	0110 10s0 : immediate data	UV
PUSH – Push Segment Register onto the Stack		NP
PUSHA/PUSHAD – Push All General Registers		NP
PUSHF/PUSHFD – Push Flags Register onto the Stack		NP
RCL – Rotate thru Carry Left		
reg by 1	1101 000w : 11 010 reg	PU
memory by 1	1101 000w : mod 010 r/m	PU
reg by CL	1101 001w : 11 010 reg	NP
memory by CL	1101 001w : mod 010 r/m	NP
reg by immediate count	1100 000w : 11 010 reg : imm8 data	PU
memory by immediate count	1100 000w : mod 010 r/m : imm8 data	PU
RCR – Rotate thru Carry Right		
reg by 1	1101 000w : 11 011 reg	PU
memory by 1	1101 000w : mod 011 r/m	PU
reg by CL	1101 001w : 11 011 reg	NP
memory by CL	1101 001w : mod 011 r/m	NP
reg by immediate count	1100 000w : 11 011 reg : imm8 data	PU
memory by immediate count	1100 000w : mod 011 r/m : imm8 data	PU
RDMSR – Read from Model-Specific Register		NP
REP LODS – Load String		NP
REP MOVS – Move String		NP
REP STOS – Store String		NP

Команда	Разновидность	Спари-вание
REPE CMPS – Compare String (Find Non-Match)		NP
REPE SCAS – Scan String (Find Non-AL/AX/EAX)		NP
REPNE CMPS – Compare String (Find Match)		NP
REPNE SCAS – Scan String (Find AL/AX/EAX)		NP
RET – Return from Procedure (to same segment)		NP
RET – Return from Procedure (to other segment)		NP
ROL – Rotate (not thru Carry) Left		
reg by 1	1101 000w : 11 000 reg	PU
memory by 1	1101 000w : mod 000 r/m	PU
reg by CL	1101 001w : 11 000 reg	NP
memory by CL	1101 001w : mod 000 r/m	NP
reg by immediate count	1100 000w : 11 000 reg : imm8 data	PU
memory by immediate count	1100 000w : mod 000 r/m : imm8 data	PU
ROR – Rotate (not thru Carry) Right		
reg by 1	1101 000w : 11 001 reg	PU
memory by 1	1101 000w : mod 001 r/m	PU
reg by CL	1101 001w : 11 001 reg	NP
memory by CL	1101 001w : mod 001 r/m	NP
reg by immediate count	1100 000w : 11 001 reg : imm8 data	PU
memory by immediate count	1100 000w : mod 001 r/m : imm8 data	PU
RSM – Resume from System Management Mode		NP
SAHF – Store AH into Flags		NP
SAL – Shift Arithmetic Left same instruction as SHL		
SAR – Shift Arithmetic Right		
reg by 1	1101 000w : 11 111 reg	PU
memory by 1	1101 000w : mod 111 r/m	PU
reg by CL	1101 001w : 11 111 reg	NP
memory by CL	1101 001w : mod 111 r/m	NP
reg by immediate count	1100 000w : 11 111 reg : imm8 data	PU
memory by immediate count	1100 000w : mod 111 r/m : imm8 data	PU

Команда	Разновидность	Спаривание
SBB – Integer Subtraction with Borrow		PU
SCAS/SCASB/SCASW/SCASD – Scan String		NP
SETcc – Byte Set on Condition		NP
SGDT – Store Global Descriptor Table Register		NP
SHL – Shift Left		
reg by 1	1101 000w : 11 100 reg	PU
memory by 1	1101 000w : mod 100 r/m	PU
reg by CL	1101 001w : 11 100 reg	NP
memory by CL	1101 001w : mod 100 r/m	NP
reg by immediate count	1100 000w : 11 100 reg : imm8 data	PU
memory by immediate count	1100 000w : mod 100 r/m : imm8 data	PU
SHLD – Double Precision Shift Left		
register by immediate count	0000 1111 : 1010 0100 : 11 reg2 reg1 : imm8	NP
memory by immediate count	0000 1111 : 1010 0100 : mod reg r/m : imm8	NP
register by CL	0000 1111 : 1010 0101 : 11 reg2 reg1	NP
memory by	CL 0000 1111 : 1010 0101 : mod reg r/m	NP
SHR – Shift Right		
reg by 1	1101 000w : 11 101 reg	PU
memory by 1	1101 000w : mod 101 r/m	PU
reg by CL	1101 001w : 11 101 reg	NP
memory by CL	1101 001w : mod 101 r/m	NP
reg by immediate count	1100 000w : 11 101 reg : imm8 data	PU
memory by immediate count	1100 000w : mod 101 r/m : imm8 data	PU
SHRD – Double Precision Shift Right		
register by immediate count	0000 1111 : 1010 1100 : 11 reg2 reg1 : imm8	NP
memory by immediate count	0000 1111 : 1010 1100 : mod reg r/m : imm8	NP
register by CL	0000 1111 : 1010 1101 : 11 reg2 reg1	NP
memory by CL	0000 1111 : 1010 1101 : mod reg r/m	NP
SIDT – Store Interrupt Descriptor Table Register		NP

Команда	Разновидность	Спари-вание
SLDT – Store Local Descriptor Table Register		NP
SMSW – Store Machine Status Word		NP
STC – Set Carry Flag		NP
STD – Set Direction Flag		NP
STI – Set Interrupt Flag		
STOS/STOSB/STOSW/STOSD – Store String Data		NP
STR – Store Task Register		NP
SUB – Integer Subtraction		UV
TEST – Logical Compare		
reg1 and reg2	1000 010w : 11 reg1 reg2	UV
memory and register	1000 010w : mod reg r/m	UV
immediate and register	1111 011w : 11 000 reg : immediate data	NP
immediate and accumulator	1010 100w : immediate data	UV
immediate and memory	1111 011w : mod 000 r/m : immediate data	NP
VERR – Verify a Segment for Reading		NP
VERW – Verify a Segment for Writing		NP
WAIT – Wait 1001 1011		NP
WBINVD – Write-Back and Invalidate Data Cache		NP
WRMSR – Write to Model-Specific Register		NP
XADD – Exchange and Add		NP
XCHG – Exchange Register/Memory with Register		NP
XLAT/XLATB – Table Look-up Translation		NP
XOR – Logical Exclusive OR		UV

Приложение Б
Спаривание FPU-команд

Обозначения

NP – команды не могут спариваться ни с какими другими командами.

FX – команды могут спариваться с командой FXCH, когда последняя направляется в V-канал.

Команда	Разновидность	Спаривание
F2XM1 – Compute 2 ST(0) – 1		NP
FABS – Absolute Value		FX
FADD – Add		FX
FADDP – Add and Pop		FX
FBLD – Load Binary Coded Decimal		NP
FBSTP – Store Binary Coded Decimal and Pop		NP
FCHS – Change Sign		FX
FCLEX – Clear Exceptions		NP
FCOM – Compare Real		FX
FCOMP – Compare Real and Pop		FX
FCOMPP – Compare Real and Pop Twice		
FCOS – Cosine of ST(0)		NP
FDECSTP – Decrement Stack-Top Pointer		NP
FDIV – Divide		FX
FDIVP – Divide and Pop		FX
FDIVR – Reverse Divide		FX
FDIVRP – Reverse Divide and Pop		FX
FFREE – Free ST(i) Register		NP
FIADD – Add Integer		NP
FICOM – Compare Integer		NP
FICOMP – Compare Integer and Pop		NP
FIDIV		NP
FIDIVR		NP
FILD – Load Integer		NP
FIMUL		NP
FINCSTP – Increment Stack Pointer		NP
FINIT – Initialize Floating-Point Unit		NP
FIST – Store Integer		NP
FISTP – Store Integer and Pop		NP
FISUB		NP

Команда	Разновидность	Спари-вание
FISUBR		NP
FLD – Load Real		
32-bit memory	11011 001 : mod 000 r/m	FX
64-bit memory	11011 101 : mod 000 r/m	FX
80-bit memory	11011 011 : mod 101 r/m	NP
	ST(i) 11011 001 : 11 000 ST(i)	FX
FLD1 – Load + 1.0 into ST(0)		NP
FLDCW – Load Control Word		NP
FLDENV – Load FPU Environment		NP
FLDL2E – Load log 2 (e) into ST(0)		NP
FLDL2T – Load log 2 (10) into ST(0)		NP
FLDLG2 – Load log 10 (2) into ST(0)		NP
FLDLN2 – Load log e (2) into ST(0)		NP
FLDPI – Load p into ST(0)		NP
FLDZ – Load + 0.0 into ST(0)		NP
FMUL – Multiply		FX
FMULP – Multiply		FX
FNOP – No Operation		NP
FPATAN – Partial Arctangent		NP
FPREM – Partial Remainder		NP
FPREM1 – Partial Remainder (IEEE)		NP
FPTAN – Partial Tangent		NP
FRNDINT – Round to Integer		
FRSTOR – Restore FPU State		NP
FSAVE – Store FPU State		NP
FSCALE – Scale		NP
FSIN – Sine		NP
FSINCOS – Sine and Cosine		NP
FSQRT – Square Root		NP
FST – Store Real		NP
FSTCW – Store Control Word		NP
FSTENV – Store FPU Environment		NP
FSTP – Store Real and Pop		NP
FSTSW – Store Status Word into AX		NP
FSTSW – Store Status Word into Memory		NP
FSUB – Subtract		FX
FSUBP – Subtract and Pop		FX
FSUBR – Reverse Subtract		FX
FSUBRP – Reverse Subtract and Pop		FX
FTST – Test		FX
FUCOM – Unordered Compare Real)		FX

Команда	Разновидность	Спаривание
FUCOMP – Unordered Compare and Pop		FX
FUCOMPP – Unordered Compare and Pop Twice		FX
FXAM – Examine		NP
FXCH – Exchange ST(0) and ST(i)		
FXTRACT – Extract Exponent and Significant		NP
FYL2X – ST(1) ґ log 2 (ST(0))		NP
FYL2XP1 – ST(1) ґ log 2 (ST(0) + 1.0)		NP
FWAIT – Wait until FPU Ready		

Приложение В
Декодирование команд в микрооперации

Макрокоманда	Число микро-операций	Макрокоманда	Число микро-операций
AAA	1	AAD	3
AAM	4	AAS	1
ADC AL,imm8	2	ADC eAX,imm16/32	2
ADC m16/32,imm16/32	4	ADC m16/32,r16/32	4
ADC m8,imm8	4	ADC m8,r8	4
ADC r16/32,imm16/32	2	ADC r16/32,m16/32	3
ADC r16/32,rm16/32	2	ADC r8,imm8	2
ADC r8,m8	3	ADC r8,rm8	2
ADC rm16/32,r16/32	2	ADC rm8,r8	2
ADD AL,imm8	1	ADD eAX,imm16/32	1
ADD m16/32,imm16/32	4	ADD m16/32,r16/32	4
ADD m8,imm8	4	ADD m8,r8	4
ADD r16/32,imm16/32	1	ADD r16/32,imm8	1
ADD r16/32,m16/32	2	ADD r16/32,rm16/32	1
ADD r8,imm8	1	ADD r8,m8	2
ADD r8,rm8	1	ADD rm16/32,r16/32	1
ADD rm8,r8	1	AND AL,imm8	1
AND eAX,imm16/32	1	AND m16/32,imm16/32	4
AND m16/32,r16/32	4	AND m8,imm8	4
AND m8,r8	4	AND r16/32,imm16/32	1
AND r16/32,imm8	1	AND r16/32,m16/32	2
AND r16/32,rm16/32	1	AND r8,imm8	1
AND r8,m8	2	AND r8,rm8	1
AND rm16/32,r16/32	1	AND rm8,r8	1
ARPL m16	complex	ARPL rm16, r16	complex
BOUND r16,m16/32&16/32	complex	BSF r16/32,m16/32	3
BSF r16/32,rm16/32	2	BSR r16/32,m16/32	3
BSR r16/32,rm16/32	2	BSWAP r32	2
BT m16/32, imm8	2	BT m16/32, r16/32	complex
BT rm16/32, imm8	1	BT rm16/32, r16/32	1
BTC m16/32, imm8	4	BTC m16/32, r16/32	complex
BTC rm16/32, imm8	1	BTC rm16/32, r16/32	1
BTR m16/32, imm8	4	BTR m16/32, r16/32	complex
BTR rm16/32, imm8	1	BTR rm16/32, r16/32	1
BTS m16/32, imm8	4	BTS m16/32, r16/32	complex
BTS rm16/32, imm8	1	BTS rm16/32, r16/32	1
CALL m16/32 near	complex	CALL m16	complex
CALL ptr16	complex	CALL r16/32 near	complex
CALL rel16/32 near	4	CBW	1

Макрокоманда	Число микро-операций	Макрокоманда	Число микро-операций
CLC	1	CLD	4
CLI	complex	CLTS	complex
CMC	1	CMOVB/NAE/C r16/32,m16/32	3
CMOVB/NAE/C r16/32,r16/32	2	CMOVBE/NA r16/32,m16/32	3
CMOVBE/NA r16/32,r16/32	2	CMOVE/Z r16/32,m16/32	3
CMOVE/Z r16/32,r16/32	2	CMOVL/NGE r16/32,m16/32	3
CMOVL/NGE r16/32,r16/32	2	CMOVLE/NG r16/32,m16/32	3
CMOVLE/NG r16/32,r16/32	2	CMOVNB/AE/NC r16/32,m16/32	3
CMOVNB/AE/NC r16/32,r16/32	2	CMOVNBE/A r16/32,m16/32	3
CMOVNBE/A r16/32,r16/32	2	CMOVNE/NZ r16/32,m16/32	3
CMOVNE/NZ r16/32,r16/32	2	CMOVNL/GE r16/32,m16/32	3
CMOVNL/GE r16/32,r16/32	2	CMOVNLE/G r16/32,m16/32	3
CMOVNLE/G r16/32,r16/32	2	CMOVNO r16/32,m16/32	3
CMOVNO r16/32,r16/32	2	CMOVNP/PO r16/32,m16/32	3
CMOVNP/PO r16/32,r16/32	2	CMOVNS r16/32,m16/32	3
CMOVNS r16/32,r16/32	2	CMOVO r16/32,m16/32	3
CMOVO r16/32,r16/32	2	CMOVP/PE r16/32,m16/32	3
CMOVP/PE r16/32,r16/32	2	CMOVS r16/32,m16/32	3
CMOVS r16/32,r16/32	2	CMP AL, imm8	1
CMP eAX,imm16/32	1	CMP m16/32, imm16/32	2
CMP m16/32, imm8	2	CMP m16/32,r16/32	2
CMP m8, imm8	2	CMP m8, imm8	2
CMP m8,r8	2	CMP r16/32,m16/32	2
CMP r16/32,rm16/32	1	CMP r8,m8	2
CMP r8,rm8	1	CMP rm16/32,imm16/32	1
CMP rm16/32,imm8	1	CMP rm16/32,r16/32	1
CMP rm8,imm8	1	CMP rm8,imm8	1
CMP rm8,r8	1	CMPSB/W/D m8/16/32,m8/16/32	complex

Макрокоманда	Число микро-операций	Макрокоманда	Число микро-операций
CMPXCHG m16/32,r16/32	complex	CMPXCHG m8,r8	complex
CMPXCHG rm16/32,r16/32	complex	CMPXCHG rm8,r8	complex
CMPXCHG8B rm64	complex	CPUID	complex
CWD/CDQ	1	CWDE	1
DAA	1	DAS	1
DEC m16/32	4	DEC m8	4
DEC r16/32	1	DEC rm16/32	1
DEC rm8	1	DIV AL,rm8	3
DIV AX,m16/32	4	DIV AX,m8	4
DIV AX,rm16/32	4	ENTER	complex
F2XM1	complex	FABS	1
FADD ST(i),ST	1	FADD ST,ST(i)	1
FADD m32real	2	FADD m64real	2
FADDP ST(i),ST	1	FBLD m80dec	complex
FBSTP m80dec	complex	FCHS	3
FCMOVB Sti	2	FCMOVBE Sti	2
FCMOVE Sti	2	FCMOVNB Sti	2
FCMOVNBE Sti	2	FCMOVNE Sti	2
FCMOVNU Sti	2	FCMOVU Sti	2
FCOM Sti	1	FCOM m32real	2
FCOM m64real	2	FCOM2 Sti	1
FCOMI Sti	1	FCOMIP Sti	1
FCOMP Sti	1	FCOMP m32real	2
FCOMP m64real	2	FCOMP3 Sti	1
FCOMP5 Sti	1	FCOMPP	2
FCOS	complex	FDECSTP	1
FDISI	1	FDIV ST(i),ST	1
FDIV ST,ST(i)	1	FDIV m32real	2
FDIV m64real	2	FDIVP ST(i),ST	1
FDIVR ST(i),ST	1	FDIVR ST,ST(i)	1
FDIVR m32real	2	FDIVR m64real	2
FDIVRP ST(i),ST	1	FENI	1
FFREE ST(i)	1	FFREEP ST(i)	2
FIADD m16int	complex	FIADD m32int	complex
FICOM m16int	complex	FICOM m32int	complex
FICOMP m16int	complex	FICOMP m32int	complex
FIDIV m16int	complex	FIDIV m32int	complex
FIDIVR m16int	complex	FIDIVR m32int	complex
FILD m16int	4	FILD m32int	4
FILD m64int	4	FIMUL m16int	complex
FIMUL m32int	complex	FINCSTP	1

Макрокоманда	Число микро-операций	Макрокоманда	Число микро-операций
FIST m16int	4	FIST m32int	4
FISTP m16int	4	FISTP m32int	4
FISTP m64int	4	FISUB m16int	complex
FISUB m32int	complex	FISUBR m16int	complex
FISUBR m32int	complex	FLD Sti	1
FLD m32real	1	FLD m64real	1
FLD m80real	4	FLD1	2
FLDCW m2byte	3	FLDENV m14/28byte	complex
FLDL2E	2	FLDL2T	2
FLDLG2	2	FLDLN2	2
FLDPI	2	FLDZ	1
FMUL ST(i),ST	1	FMUL ST,ST(i)	1
FMUL m32real	2	FMUL m64real	2
FMULP ST(i),ST	1	FNCLEX	3
FNINIT	complex	FNOP	1
FNSAVE m94/108byte	complex	FNSTCW m2byte	3
FNSTENV m14/28byte	complex	FNSTSW AX	3
FNSTSW m2byte	3	FPATAN	complex
FPREM	complex	FPREM1	complex
FPTAN	complex	FRNDINT	complex
FRSTOR m94/108byte	complex	FSCALE	
FSETPM	1	FSIN	complex
FSINCOS	complex	FSQRT	1
FST Sti	1	FST m32real	2
FST m64real	2	FSTP Sti	1
FSTP m32real	2	FSTP m64real	2
FSTP m80real	complex	FSTP1 Sti	1
FSTP8 Sti	1	FSTP9 Sti	1
FSUB ST(i),ST	1	FSUB ST,ST(i)	1
FSUB m32real	2	FSUB m64real	2
FSUBP ST(i),ST	1	FSUBR ST(i),ST	1
FSUBR ST,ST(i)	1	FSUBR m32real	2
FSUBR m64real	2	FSUBRP ST(i),ST	1
FTST	1	FUCOM Sti	1
FUCOMI Sti	1	FUCOMIP Sti	1
FUCOMP Sti	1	FUCOMPP	2
FWAIT	2	FXAM	1
FXCH Sti	1	FXCH4 Sti	1
FXCH7 Sti	1	FXTRACT	complex
FYL2X	complex	FYL2XP1	complex
HALT	complex	IDIV AL,rm8	3
IDIV AX,m16/32	4	IDIV AX,m8	4
IDIV eAX,rm16/32	4	IMUL m16	4

Макрокоманда	Число микро-операций	Макрокоманда	Число микро-операций
IMUL m32	4	IMUL m8	2
IMUL r16/32,m16/32	2	IMUL r16/32,rm16/32	1
IMUL r16/32,rm16/32,imm8/16/32	2	IMUL r16/32,rm16/32,imm8/16/32	1
IMUL rm16	3	IMUL rm32	3
IMUL rm8	1	IN eAX, DX	complex
IN eAX, imm8	complex	INC m16/32	4
INC m8	4	INC r16/32	1
INC rm16/32	1	INC rm8	1
INSB/W/D m8/16/32,DX	complex	INT1	complex
INT3	complex	INTN	3
INTO	complex	INVD	complex
INVLPG m	complex	IRET	complex
JB/NAE/C rel16/32	1	JB/NAE/C rel8	1
JBE/NA rel16/32	1	JBE/NA rel8	1
JCXZ/JECXZ rel8	2	JE/Z rel16/32	1
JE/Z rel8	1	JL/NGE rel16/32	1
JL/NGE rel8	1	JLE/NG rel16/32	1
JLE/NG rel8	1	JMP m16	complex
JMP near m16/32	2	JMP near reg16/32	1
JMP ptr16	complex	JMP rel16/32	1
JMP rel8	1	JNB/AE/NC rel16/32	1
JNB/AE/NC rel8	1	JNBE/A rel16/32	1
JNBE/A rel8	1	JNE/NZ rel16/32	1
JNE/NZ rel8	1	JNL/GE rel16/32	1
JNL/GE rel8	1	JNLE/G rel16/32	1
JNLE/G rel8	1	JNO rel16/32	1
JNO rel8	1	JNP/PO rel16/32	1
JNP/PO rel8	1	JNS rel16/32	1
JNS rel8	1	JO rel16/32	1
JO rel8	1	JP/PE rel16/32	1
JP/PE rel8	1	JS rel16/32	1
JS rel8	1	LAHF	1
LAR m16	complex	LAR rm16	complex
LDS r16/32,m16	complex	LEA r16/32,m	1
LEAVE	3	LES r16/32,m16	complex
LFS r16/32,m16	complex	LGDT m16&32	complex
LGS r16/32,m16	complex	LIDT m16&32	complex
LLDT m16	complex	LLDT rm16	complex
LMSW m16	complex	LMSW r16	complex
LOCK ADC m16/32,imm16/32	complex	LOCK ADC m16/32,r16/32	complex
LOCK ADC m8,imm8	complex	LOCK ADC m8,r8	complex

Макрокоманда	Число микро-операций	Макрокоманда	Число микро-операций
LOCK ADD m16/32,imm16/32	complex	LOCK ADD m16/32,r16/32	complex
LOCK ADD m8,imm8	complex	LOCK ADD m8,r8	complex
LOCK AND m16/32,imm16/32	complex	LOCK AND m16/32,r16/32	complex
LOCK AND m8,imm8	complex	LOCK AND m8,r8	complex
LOCK BTC m16/32, imm8	complex	LOCK BTC m16/32, r16/32	complex
LOCK BTR m16/32, imm8	complex	LOCK BTR m16/32, r16/32	complex
LOCK BTS m16/32, imm8	complex	LOCK BTS m16/32, r16/32	complex
LOCK CMPXCHG m16/32,r16/32	complex	LOCK CMPXCHG m8,r8	complex
LOCK CMPXCHG8B rm64	complex	LOCK DEC m16/32	complex
LOCK DEC m8	complex	LOCK INC m16/32	complex
LOCK INC m8	complex	LOCK NEG m16/32	complex
LOCK NEG m8	complex	LOCK NOT m16/32	complex
LOCK NOT m8	complex	LOCK OR m16/32,imm16/32	complex
LOCK OR m16/32,r16/32	complex	LOCK OR m8,imm8	complex
LOCK OR m8,r8	complex	LOCK SBB m16/32,imm16/32	complex
LOCK SBB m16/32,r16/32	complex	LOCK SBB m8,imm8	complex
LOCK SBB m8,r8	complex	LOCK SUB m16/32,imm16/32	complex
LOCK SUB m16/32,r16/32	complex	LOCK SUB m8,imm8	complex
LOCK SUB m8,r8	complex	LOCK XADD m16/32,r16/32	complex
LOCK XADD m8,r8	complex	LOCK XCHG m16/32,r16/32	complex
LOCK XCHG m8,r8	complex	LOCK XOR m16/32,imm16/32	complex
LOCK XOR m16/32,r16/32	complex	LOCK XOR m8,imm8	complex
LOCK XOR m8,r8	complex	LODSB/W/D m8/16/32,m8/16/32	2
LOOP rel8	4	LOOPE rel8	4
LOOPNE rel8	4	LSL m16	complex
LSL rm16	complex	LSS r16/32,m16	complex
LTR m16	complex	LTR rm16	complex
MOV AL,moffs8	1	MOV CR0, r32	complex

Макрокоманда	Число микро-операций	Макрокоманда	Число микро-операций
MOV CR2, r32	complex	MOV CR3, r32	complex
MOV CR4, r32	complex	MOV DRx, r32	complex
MOV DS,m16	4	MOV DS,rm16	4
MOV ES,m16	4	MOV ES,rm16	4
MOV FS,m16	4	MOV FS,rm16	4
MOV GS,m16	4	MOV GS,rm16	4
MOV SS,m16	4	MOV SS,rm16	4
MOV eAX,moffs16/32	1	MOV m16,CS	3
MOV m16,DS	3	MOV m16,ES	3
MOV m16,FS	3	MOV m16,GS	3
MOV m16,SS	3	MOV m16/32,imm16/32	2
MOV m16/32,r16/32	2	MOV m8,imm8	2
MOV m8,r8	2	MOV moffs16/32,eAX	2
MOV moffs8,AL	2	MOV r16/32,imm16/32	1
MOV r16/32,m16/32	1	MOV r16/32,rm16/32	1
MOV r32, CR0	complex	MOV r32, CR2	complex
MOV r32, CR3	complex	MOV r32, CR4	complex
MOV r32, DRx	complex	MOV r8,imm8	1
MOV r8,m8	1	MOV r8,rm8	1
MOV rm16,CS	1	MOV rm16,DS	1
MOV rm16,ES	1	MOV rm16,FS	1
MOV rm16,GS	1	MOV rm16,SS	1
MOV rm16/32,imm16/32	1	MOV rm16/32,r16/32	1
MOV rm8,imm8	1	MOV rm8,r8	1
MOVSB/W/D m8/16/32,m8/16/32	complex	MOVSX r16,m8	1
MOVSX r16,rm8	1	MOVSX r16/32,m16	1
MOVSX r32,m8	1	MOVSX r32,rm16	1
MOVSX r32,rm8	1	MOVZX r16,m8	1
MOVZX r16,rm8	1	MOVZX r32,m16	1
MOVZX r32,m8	1	MOVZX r32,rm16	1
MOVZX r32,rm8	1	MUL AL,m8	2
MUL AL,rm8	1	MUL AX,m16	4
MUL AX,rm16	3	MUL EAX,m32	4
MUL EAX,rm32	3	NEG m16/32	4
NEG m8	4	NEG rm16/32	1
NEG rm8	1	NOP	1
NOT m16/32	4	NOT m8	4
NOT rm16/32	1	NOT rm8	1
OR AL,imm8	1	OR eAX,imm16/32	1
OR m16/32,imm16/32	4	OR m16/32,r16/32	4
OR m8,imm8	4	OR m8,r8	4
OR r16/32,imm16/32	1	OR r16/32,imm8	1

Макрокоманда	Число микро-операций	Макрокоманда	Число микро-операций
OR r16/32,m16/32	2	OR r16/32,rm16/32	1
OR r8,imm8	1	OR r8,m8	2
OR r8,rm8	1	OR rm16/32,r16/32	1
OR rm8,r8	1	OUT DX, eAX	complex
OUT imm8, eAX	complex	OUTSB/W/D DX,m8/16/32	complex
POP DS	complex	POP ES	complex
POP FS	complex	POP GS	complex
POP SS	complex	POP eSP	3
POP m16/32	complex	POP r16/32	2
POP r16/32	2	POPA/POPAD	complex
POPF	complex	POPFD	complex
PUSH CS	4	PUSH DS	4
PUSH ES	4	PUSH FS	4
PUSH GS	4	PUSH SS	4
PUSH imm16/32	3	PUSH imm8	3
PUSH m16/32	4	PUSH r16/32	3
PUSH r16/32	3	PUSHA/PUSHAD	complex
PUSHF/PUSHFD	complex	RCL m16/32,1	4
RCL m16/32,CL	complex	RCL m16/32,imm8	complex
RCL m8,1	4	RCL m8,CL	complex
RCL m8,imm8	complex	RCL rm16/32,1	2
RCL rm16/32,CL	complex	RCL rm16/32,imm8	complex
RCL rm8,1	2	RCL rm8,CL	complex
RCL rm8,imm8	complex	RCR m16/32,1	4
RCR m16/32,CL	complex	RCR m16/32,imm8	complex
RCR m8,1	4	RCR m8,CL	complex
RCR m8,imm8	complex	RCR rm16/32,1	2
RCR rm16/32,CL	complex	RCR rm16/32,imm8	complex
RCR rm8,1	2	RCR rm8,CL	complex
RCR rm8,imm8	complex	RDMSR	complex
RDPMC	complex	RDTSC	complex
REP CMPSB/W/D m8/16/32,m8/16/32	complex	REP INSB/W/D m8/16/32,DX	complex
REP LODSB/W/D m8/16/32,m8/16/32	complex	REP MOVSB/W/D m8/16/32,m8/16/32	complex
REP OUTSB/W/D DX,m8/16/32	complex	REP SCASB/W/D m8/16/32,m8/16/32	complex
REP STOSB/W/D m8/16/32,m8/16/32	complex	RET	4
RET	complex	RET near	4
RET near iw	complex	ROL m16/32,1	4
ROL m16/32,CL	4	ROL m16/32,imm8	4
ROL m8,1	4	ROL m8,CL	4
ROL m8,imm8	4	ROL rm16/32,1	1

Макрокоманда	Число микро-операций	Макрокоманда	Число микро-операций
ROL rm16/32,CL	1	ROL rm16/32,imm8	1
ROL rm8,1	1	ROL rm8,CL	1
ROL rm8,imm8	1	ROR m16/32,1	4
ROR m16/32,CL	4	ROR m16/32,imm8	4
ROR m8,1	4	ROR m8,CL	4
ROR m8,imm8	4	ROR rm16/32,1	1
ROR rm16/32,CL	1	ROR rm16/32,imm8	1
ROR rm8,1	1	ROR rm8,CL	1
ROR rm8,imm8	1	RSM	complex
SAHF	1	SALC	2
SAR m16/32,1	4	SAR m16/32,CL	4
SAR m16/32,imm8	4	SAR m8,1	4
SAR m8,CL	4	SAR m8,imm8	4
SAR rm16/32,1	1	SAR rm16/32,CL	1
SAR rm16/32,imm8	1	SAR rm8,1	1
SAR rm8,CL	1	SAR rm8,imm8	1
SBB AL,imm8	2	SBB eAX,imm16/32	2
SBB m16/32,imm16/32	4	SBB m16/32,r16/32	4
SBB m8,imm8	4	SBB m8,r8	4
SBB r16/32,imm16/32	2	SBB r16/32,m16/32	3
SBB r16/32,rm16/32	2	SBB r8,imm8	2
SBB r8,m8	3	SBB r8,rm8	2
SBB rm16/32,r16/32	2	SBB rm8,r8	2
SCASB/W/D m8/16/32,m8/16/32	3	SETB/NAE/C m8	3
SETB/NAE/C rm8	1	SETBE/NA m8	3
SETBE/NA rm8	1	SETE/Z m8	3
SETE/Z rm8	1	SETL/NGE m8	3
SETL/NGE rm8	1	SETLE/NG m8	3
SETLE/NG rm8	1	SETNB/AE/NC m8	3
SETNB/AE/NC rm8	1	SETNBE/A m8	3
SETNBE/A rm8	1	SETNE/NZ m8	3
SETNE/NZ rm8	1	SETNL/GE m8	3
SETNL/GE rm8	1	SETNLE/G m8	3
SETNLE/G rm8	1	SETNO m8	3
SETNO rm8	1	SETNP/PO m8	3
SETNP/PO rm8	1	SETNS m8	3
SETNS rm8	1	SETO m8	3
SETO rm8	1	SETP/PE m8	3
SETP/PE rm8	1	SETS m8	3
SETS rm8	1	SGDT m16&32	4
SHL/SAL m16/32,1	4	SHL/SAL m16/32,1	4
SHL/SAL m16/32,CL	4	SHL/SAL m16/32,CL	4

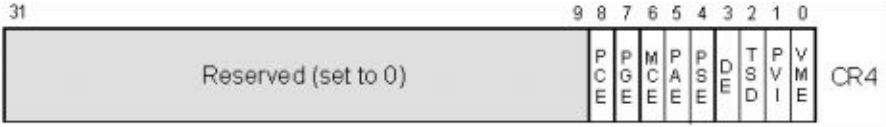
Макрокоманда	Число микро-операций	Макрокоманда	Число микро-операций
SHL/SAL m16/32,imm8	4	SHL/SAL m16/32,imm8	4
SHL/SAL m8,1	4	SHL/SAL m8,1	4
SHL/SAL m8,CL	4	SHL/SAL m8,CL	4
SHL/SAL m8,imm8	4	SHL/SAL m8,imm8	4
SHL/SAL rm16/32,1	1	SHL/SAL rm16/32,1	1
SHL/SAL rm16/32,CL	1	SHL/SAL rm16/32,CL	1
SHL/SAL rm16/32,imm8	1	SHL/SAL rm16/32,imm8	1
SHL/SAL rm8,1	1	SHL/SAL rm8,1	1
SHL/SAL rm8,CL	1	SHL/SAL rm8,CL	1
SHL/SAL rm8,imm8	1	SHL/SAL rm8,imm8	1
SHLD m16/32,r16/32,CL	4	SHLD m16/32,r16/32,imm8	4
SHLD rm16/32,r16/32,CL	2	SHLD rm16/32,r16/32,imm8	2
SHR m16/32,1	4	SHR m16/32,CL	4
SHR m16/32,imm8	4	SHR m8,1	4
SHR m8,CL	4	SHR m8,imm8	4
SHR rm16/32,1	1	SHR rm16/32,CL	1
SHR rm16/32,imm8	1	SHR rm8,1	1
SHR rm8,CL	1	SHR rm8,imm8	1
SHRD m16/32,r16/32,CL	4	SHRD m16/32,r16/32,imm8	4
SHRD rm16/32,r16/32,CL	2	SHRD rm16/32,r16/32,imm8	2
SIDT m16&32	<4	SLDT m16	<4
SLDT rm16	4	SMSW m16	<4
SMSW rm16	4	STC	1
STD	4	STI	<4
STOSB/W/D m8/16/32,m8/16/32	3	STR m16	<4
STR rm16	4	SUB AL,imm8	1
SUB eAX,imm16/32	1	SUB m16/32,imm16/32	4
SUB m16/32,r16/32	4	SUB m8,imm8	4
SUB m8,r8	4	SUB r16/32,imm16/32	1
SUB r16/32,imm8	1	SUB r16/32,m16/32	2
SUB r16/32,rm16/32	1	SUB r8,imm8	1
SUB r8,m8 2 SUB r8,rm8	1	SUB rm16/32,r16/32	1
SUB rm8,r8	1	TEST AL,imm8	1
TEST eAX,imm16/32	1	TEST m16/32,imm16/32	2
TEST m16/32,imm16/32	2	TEST m16/32,r16/32	2
TEST m8,imm8	2	TEST m8,imm8	2
TEST m8,r8	2	TEST rm16/32,imm16/32	1
TEST rm16/32,r16/32	1	TEST rm8,imm8	1
TEST rm8,r8	1	VERR m16	<4

Макрокоманда	Число микро-операций	Макрокоманда	Число микро-операций
VERR rm16	<4	VERW m16	<4
VERW rm16	<4	WBINVD	<4
WRMSR	<4	XADD m16/32,r16/32	<4
XADD m8,r8	>4	XADD rm16/32,r16/32	4
XADD rm8,r8	4	XCHG eAX,r16/32	3
XCHG m16/32,r16/32	<4	XCHG m8,r8	<4
XCHG rm16/32,r16/32	3	XCHG rm8,r8	3
XLAT/B	2	XOR AL,imm8	1
XOR eAX,imm16/32	1	XOR m16/32,imm16/32	4
XOR m16/32,r16/32	4	XOR m8,imm8	4
XOR m8,r8	4	XOR r16/32,imm16/32	1
XOR r16/32,imm8	1	XOR r16/32,m16/32	2
XOR r16/32,rm16/32	1	XOR r8,imm8	1
XOR r8,m8	2	XOR r8,rm8	1
XOR rm16/32,r16/32	1	XOR rm8,r8	1

Команды MMX, которые не обращаются к памяти, декодируются в одну микрооперацию. Команды MMX, которые обращаются к памяти, декодируются в две микрооперации.

Приложение Г
Формат регистра CR4

В процессорах P5 был впервые задействован регистр управления CR4, в процессорах P6 в регистр CR4 были добавлены новые флажки. На рисунке приведен формат регистра CR4 для процессоров семейства P6. В описании флажки, добавленные в процессорах P6, помечены символами (P6). В процессорах P5 эти флажки являются зарезервированными и должны содержать 0.



Формат регистра CR4 для процессоров P6

Описание битов регистра CR4:

⇒ **WME – Virtual-8086 Mode Extensions**. Когда WME=1, разрешаются расширенные возможности по обработке прерываний и исключений в V-режиме, а также разрешается аппаратная поддержка флага VIF. Когда WME=0, расширенные возможности запрещены.

⇒ **PVI – Protected-Mode Virtual Interrupts.** Когда PVI=1, разрешена аппаратная поддержка флага VIF в P-режиме. Когда PVI=0, поддержка флага VIF запрещена.

⇒ **TSD – Time Stamp Disable.** Когда TDS=1, разрешено выполнение команды RDTSC только на нулевом уровне привилегий. Когда TDS=0, выполнение команды RDTSC разрешено на любом уровне привилегий.

⇒ **DE – Debugging Extensions.** Когда DE=1, обращение к регистрам DR4 и DR5 вызывает исключение неверного кода операции (#UD). Когда DE=0, обращение к регистрам DR4 и DR5 отображается на регистры DR6 и DR7, как в ранних процессорах.

⇒ **PSE – Page Size Extensions.** Когда PSE=1, разрешена поддержка 4-MB страниц. Когда PSE=0, поддержка 4-MB страниц запрещена.

⇒ **PAE – Physical Address Extension. (P6)** Когда PAE=1, механизм страничного преобразования использует 36-разрядные физические адреса. Когда PAE=0, механизм страничного преобразования использует 32-разрядные физические адреса.

⇒ **MCE – Machine Check Enable.** Когда MCE=1, разрешена генерация исключения #MC. Когда MCE=0, генерация этого исключения запрещена.

⇒ **PGE – Page Global Enable. (P6)** Когда PGE=1, разрешена поддержка глобальных страниц. Когда PGE=0, поддержка глобальных страниц запрещена.

⇒ **PCE – Performance-monitoring Counter Enable. (P6)** Когда PCE=1, разрешено выполнение команды RDPMC на любом уровне привилегий. Когда PCE=0, выполнение команды RDPMC разрешено только на нулевом уровне привилегий.

Приложение Д
Кодирование MMX-команд

Обозначения

"+" – данный размер элементов поддерживается командой.

"-" – данный размер элементов не поддерживается командой.

"I" – размер элементов на входе.

"O" – размер элементов на выходе.

"n/a" – не применяется.

Кодирование поля gg

Поле gg	*Размер данных*
00	packed bytes
01	packed words
10	packed doublewords
11	quadword

Кодирование поля reg

Поле reg	*Регистр*
000	EAX
001	ECX
010	EDX
011	EBX
100	ESP
101	EBP
110	ESI
111	EDI

Кодирование поля mmxreg

Поле mmxreg	*Регистр*
000	mm0
001	mm1
010	mm2
011	mm3
100	mm4
101	mm5

Поле mmxreg	Регистр
110	mm6
111	mm7

Кодирование ММХ-команд

Формат команды	Размер элемента			
	B	W	DW	DQ
EMMS – Empty MMX state 0000 1111:01110111	n/a	n/a	n/a	n/a
MOVD – Move doubleword reg to mmxreg 0000 1111:01101110: 11 mmxreg reg reg from mmxreg 0000 1111:01111110: 11 mmxreg reg mem to mmxreg 0000 1111:01101110: mod mmxreg r/m mem from mmxreg 0000 1111:01111110: mod mmxreg r/m				
MOVQ – Move quadword mmxreg2 to mmxreg1 0000 1111:01101111: 11 mmxreg1 mmxreg2 mmxreg2 from mmxreg1 0000 1111:01111111: 11 mmxreg1 mmxreg2 mem to mmxreg 0000 1111:01101111: mod mmxreg r/m mem from mmxreg 0000 1111:01111111: mod mmxreg r/m	-	-	-	+
PACKSSDW - Pack dword to word data (signed with saturation) mmxreg2 to mmxreg1 0000 1111:01101011: 11 mmxreg1 mmxreg2 memory to mmxreg 0000 1111:01101011: mod mmxreg r/m	n/a	O	I	n/a
PACKSSWB - Pack word to byte data (signed	O	I	n/a	n/a

Формат команды	Размер элемента			
	B	W	DW	DQ
with saturation) mmxreg2 to mmxreg1 0000 1111:01100011: 11 mmxreg1 mmxreg2 memory to mmxreg 0000 1111:01100011: mod mmxreg r/m				
PACKUSWB - Pack word to byte data (unsigned with saturation) mmxreg2 to mmxreg1 0000 1111:01100111: 11 mmxreg1 mmxreg2 memory to mmxreg 0000 1111:01100111: mod mmxreg r/m	O	I	n/a	n/a
PADD – Add with wrap-around mmxreg2 to mmxreg1 0000 1111: 111111gg: 11 mmxreg1 mmxreg2 memory to mmxreg 0000 1111: 111111gg: mod mmxreg r/m	+	+	+	-
PADDS – Add signed with saturation mmxreg2 to mmxreg1 0000 1111: 111011gg: 11 mmxreg1 mmxreg2 memory to reg 0000 1111: 111011gg: mod reg r/m	+	+	-	-
PADDUS – Add unsigned with saturation mmxreg2 to mmxreg1 0000 1111: 110111gg: 11 mmxreg1 mmxreg2 memory to mmxreg 0000 1111: 110111gg: mod mmxreg r/m	+	+	-	-
PAND – Bitwise And mmxreg2 to mmxreg1 0000 1111:11011011: 11 mmxreg1 mmxreg2 memory to mmxreg 0000 1111:11011011: mod mmxreg r/m	-	-	-	+

Формат команды	Размер элемента			
	B	W	DW	DQ
PANDN – Bitwise AndNot mmxreg2 to mmxreg1 0000 1111:11011111: 11 mmxreg1 mmxreg2 memory to mmxreg 0000 1111:11011111: mod mmxreg r/m	-	-	-	+
PCMPEQ – Packed compare for equality mmxreg2 with mmxreg1 0000 1111:011101gg: 11 mmxreg1 mmxreg2 memory with mmxreg 0000 1111:011101gg: mod mmxreg r/m	+	+	+	-
PCMPGT - Packed compare greater (signed) mmxreg2 with mmxreg1 0000 1111:011001gg: 11 mmxreg1 mmxreg2 memory with mmxreg 0000 1111:011001gg: mod mmxreg r/m	+	+	+	-
PMADD – Packed multiply add mmxreg2 to mmxreg1 0000 1111:11110101: 11 mmxreg1 mmxreg2 memory to mmxreg 0000 1111:11110101: mod mmxreg r/m	n/a	I	O	n/a
PMULH – Packed multiplication mmxreg2 to mmxreg1 0000 1111:11100101: 11 mmxreg1 mmxreg2 memory to mmxreg 0000 1111:11100101: mod mmxreg r/m	-	+	-	-
PMULL – Packed multiplication mmxreg2 to mmxreg1 0000 1111:11010101: 11 mmxreg1 mmxreg2 memory to mmxreg 0000 1111:11010101: mod mmxreg r/m	-	+	-	-

Формат команды	Размер элемента			
	B	W	DW	DQ
POR – Bitwise Or mmxreg2 to mmxreg1 0000 1111:11101011: 11 mmxreg1 mmxreg2 memory to mmxreg 0000 1111:11101011: mod mmxreg r/m	-	-	-	+
PSLL – Packed shift left logical mmxreg2 by mmxreg1 0000 1111:111100gg: 11 mmxreg1 mmxreg2 mmxreg by memory 0000 1111:111100gg: 11 mmxreg r/m mmxreg by immediate 0000 1111:011100gg: 11 110 mmxreg: imm8	-	+	+	+
PSRA – Packed shift right arithmetic mmxreg2 by mmxreg1 0000 1111:111000gg: 11 mmxreg1 mmxreg2 mmxreg by memory 0000 1111:111000gg: 11 mmxreg r/m mmxreg by immediate 0000 1111:011100gg: 11 100 mmxreg: imm8	-	+	+	-
PSRL – Packed shift right logical mmxreg2 by mmxreg1 0000 1111:110100gg: 11 mmxreg1 mmxreg2 mmxreg by memory 0000 1111:110100gg: 11 mmxreg r/m mmxreg by immediate 0000 1111:011100gg: 11 010 mmxreg: imm8	-	+	+	+
PSUB – Subtract with wrap-around mmxreg2 to mmxreg1 0000 1111:111110gg: 11 mmxreg1 mmxreg2 memory to mmxreg 0000 1111:111110gg: mod mmxreg r/m	+	+	+	-
PSUBS -	+	+	-	-

Формат команды	Размер элемента			
	B	W	DW	DQ
Subtract signed with saturation mmxreg2 to mmxreg1 0000 1111:111010gg: 11 mmxreg1 mmxreg2 memory to mmxreg 0000 1111:111010gg: mod mmxreg r/m				
PSUBUS - Subtract unsigned with saturation mmxreg2 to mmxreg1 0000 1111:110110gg: 11 mmxreg1 mmxreg2 memory to mmxreg 0000 1111:110110gg: mod mmxreg r/m	+	+	-	-
PUNPCKH – Unpack high data to next larger type mmxreg2 to mmxreg1 0000 1111:011010gg: 11 mmxreg1 mmxreg2 memory to mmxreg 0000 1111:011010gg: mod mmxreg r/m	+	+	+	-
PUNPCKL – Unpack low data to next larger type mmxreg2 to mmxreg1 0000 1111:011000gg: 11 mmxreg1 mmxreg2 memory to mmxreg 0000 1111:011000gg: mod mmxreg r/m	+	+	+	-
PXOR – Bitwise Xor mmxreg2 to mmxreg1 0000 1111:11101111: 11 mmxreg1 mmxreg2 memory to mmxreg 0000 1111:11101111: mod mmxreg r/m	-	-	-	+

СОДЕРЖАНИЕ