УДК 681.3.06

М.А. КРИВОШЛЫК, Е.Г. КАЧКО, канд. техн. наук

ПРИМЕНЕНИЕ ТЕХНОЛОГИИ ММХ ДЛЯ ВЫПОЛНЕНИЯ ЦЕЛОЧИСЛЕННЫХ АРИФМЕТИЧЕСКИХ ОПЕРАЦИЙ НАД ЧИСЛАМИ МНОГОКРАТНОЙ ТОЧНОСТИ

Объем и сложность данных, обрабатываемых современными компьютерами, стремительно увеличиваются. Постоянно возрастающая мощность аппаратных средств, используемых злоумышленником для реализации угроз нарушения конфиденциальности, целостности, аутентичности, доступности информации приводит к необходимости создания все более совершенных систем защиты. Такие системы могут быть созданы как за счет применения более совершенных алгоритмов, так и за счет увеличения размера используемых ключевых параметров. Последнее приводит к усложнению вычислений и возрастанию времени криптопреобразований, однако сложность криптоанализа увеличивается при этом по экспоненциальному закону.

Решение задачи криптографической защиты информации требует выполнения большого объема целочисленных вычислений, таких как арифметические действия над числами многократной точности. Данные операции приводят к сильной загрузке центрального процессора. В связи с этим целесообразно было бы попытаться переложить часть вычислений с CPU на другие вычислительные устройства компьютера. Такими устройствами являются блок с плавающей точкой (FPU) и устройство для выполнения ММХ команд.

Технология ММХ была разработана фирмой Intel для ускорения мультимедиа и коммуникационных программ. Она включает в себя новые команды и типы данных и основана на их параллельной обработке. При этом сохраняется полная совместимость с существующими операционными системами и программным обеспечением. ММХ-технология - одно из самых значительных усовершенствований со времени создания 32-разрядной архитектуры. В основе ММХ лежит принцип SIMD (Single Instruction Multiple Data), т.е. одной командой можно обработать сразу несколько единиц информации.

Эффективное использование возможностей FPU и ММХ-технологии при выполнении целочисленных вычислений над числами многократной точности может привести к повышению производительности компьютера и значительно уменьшить время выполнения криптопреобразований.

ММХ-регистры отображены на поля мантиссы в FPU-регистрах. Значение, записываемое в ММХ-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) слова состояния FPU устанавливается в 0 после выполнения каждой ММХ-команды. Значение мантиссы, записываемое в FPU-регистр с помощью FPU-команды, автоматически появляется в соответствующем ММХ-регистре [1].

Отображение ММХ-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, в отличие от регистров FPU, где в обозначении Stn n указывает на относительный номер регистра (относительно поля TOS).

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

Так как ММХ и FPU используют физически одни и те же регистры, для сохранения и восстановления контекста ММХ используются команды FSAVE (Store FP state) и FRSTOR(Restore FP state).

ММХ-технология вводит 4 новых 64-разрядных типа данных :

- упакованные байты;
- упакованные слова;
- упакованные двойные слова;
- учетверенное слово.

В памяти новые типы данных располагаются так, как это принято в Intel-архитектуре, то есть по принципу младший байт первым .

MMX поддерживает новую арифметику, называемую арифметикой с насыщением (Saturation arithmetic). Сравним ее с привычной арифметикой с циклическим переносом (Wraparound arithmetic).

В режиме циклического переполнения в случае переполнения результат обрезается и возвращаются только младшие биты результата. Данный метод 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.

Большую проблему при использовании MMX для целочисленных вычислений представляет то, что MMX-команды не сообщают о переполнении с помощью генерации исключений или установки каких-либо флагов.

Эмуляция MMX-команд невозможна.

Система команд MMX состоит из 57 инструкций, которые можно подразделить на следующие классы:

- арифметические команды;
- команды преобразования;
- команды сравнения;
- команды логических операций;
- команды переноса данных;
- команды сдвига;
- команда EMMS.

Задача использования технологии MMX для целочисленных арифметических вычислений разбивается на 2 подзадачи:

- использование MMX совместно с целочисленными командами процессора;
- использование MMX и команд FPU параллельно с целочисленными командами.

Первая подзадача состоит в попытке использования 64-разрядной архитектуры и нового набора арифметических команд MMX.

Вторая подзадача - в разбиении кода приложения на 2 части. Первая часть при этом выполняется в центральном процессоре, вторая параллельно в MMX и FPU . Это возможно благодаря независимости конвейеров CPU и FPU.

Рассмотрим возможности решения первой задачи. MMX позволяет выполнить такие целочисленные операции, как сложение, вычитание и умножение. При их выполнении возникают следующие проблемы.

1. Неприменима арифметика с насыщением, поскольку в случае переполнения или заема результат автоматически округляется, что недопустимо для целочисленных вычислений. В результате система команд MMX сокращается для нас почти в 2 раза.

2. Арифметика с циклическим переносом может применяться, однако в случае переноса или заема не формируются никакие флаги, а результат обрезается. Это приводит к тому, что, например, в случае сложения 2-х чисел максимальной для MMX длины мы не можем быть уверены в старшем бите результата.

3. Большое число MMX команд предназначено для работы со знаковыми числами, то есть старший бит операндов недоступен для вычисления этими командами.

Из 2-й и 3-й проблем можно найти следующий выход: необходимо использовать числа не максимальной длины. Однако разрядность MMX-операндов выровнена на границе слова, а это значит, что уменьшение длины значащей части слагаемых (множителей) на 1 бит сделает ее не кратной степени двойки и приведет с одной стороны к получению правильного результата, а с другой стороны к следующему:

- большие затраты времени, связанные с переводом исходных чисел в такой формат;
- значительно усложняется синтез результата из промежуточных значений в случае работы с числами многократной точности.

Поскольку мы работаем исключительно с такими числами, то данный метод неприменим.

Реальным выходом в данной ситуации оказывается применение операндов с разрядностью в 2 раза меньшей, нежели исходная. Это требует в 2 раза больше выполнений команд, но при этом операнды выровнены на границе слова, что облегчает анализ результата (то есть данный метод требует меньше пред и поствычислений).

Далее приведен пример программы сложения чисел многократной точности длиной 1024 бита с использованием команд MMX.

```
.586
.MMX
model       flat
extrn       ExitProcess:proc
data
x dd    32      dup (0ffffffffh)
y dd    32      dup (0ffffffffh)
z dd    33      dup (?)
.code
begin:  lea         ebx,offset x
lea         esi,offset y
lea         edi,offset z

  mov       cx,32
  mov       eax,1
movd        mm3,eax
movd    mm7,eax
psllq       mm3,32
mov     edx,0ffffh
movd        mm4,edx

for1:mov        eax,[ebx]               ;Загрузка операндов
mov     edx,eax
and     eax,0ffffh
shr         edx,16
movd        mm0,edx
psllq   mm0,32
movd        mm1,eax
por     mm0,mm1
mov     eax,[esi]
mov     edx,eax
and     eax,0ffffh
shr         edx,16
movd        mm1,edx
psllq   mm1,32
movd        mm2,eax
por     mm1,mm2

paddd   mm0,mm1             ;Выполнение сложения
movq    mm2,mm0
movq    mm1,mm0
psrlq   mm1,16
pand        mm1,mm3
```

```
paddd   mm1,mm2
pand    mm0,mm4
psrlq   mm1,16
pxor    mm1,mm7
por     mm0,mm1

movd    edx,mm0             ;Запись результата
  add   [edi],edx
  xor   eax,eax
l2:   jnc   l1
  lea   eax,[eax+4]
  adc   [edi+eax],0
  jmp   l2
l1:   psrlq   mm0,32
movd    eax,mm0
  xor   edx,edx
  add   [edi+4],eax
l3:   jnc   l4
  lea   edx,[edx+4]
  adc   [edi+edx+4],0
  jmp   l3

l4:   add   ebx,4
  add   esi,4
  add   edi,4
  loop  for1

call    ExitProcess
```

Анализ временных параметров программ, выполняющих сложение и умножение целых чисел многократной точности показал их неэффективность. Так сложение MMX выполняется на 75% медленнее целочисленного, а умножение MMX - почти в 4 раза медленнее. Исходя из этого можно сделать вывод о неприменимости 64-разрядной архитектуры MMX и нового набора команд вместо целочисленных арифметических вычислений многократной точности.

Рассмотрим теперь возможности решения задачи разбиения кода приложения на две части с целью параллельного выполнения целочисленных команд и команд MMX и FPU.

К сожалению в данном случае также возникает ряд сложностей. Основной проблемой является то, что при разработке MMX-технологии фирма Intel не предусматривала необходимость совместной работы с данными в FPU и MMX. Сама Intel в технической документации по своим процессорам неоднократно подчеркивала необходимость обнулять регистры FPU при переходе к командам MMX и регистры MMX с помощью команды EMMS при переходе к FPU. Это связано с тем, что хотя по сути дела данные находятся в одних и тех же физических регистрах, реально они имеют разный формат. Как уже было описано выше, при занесении значения в регистр MMX в поле порядка соответствующего ему регистра FPU (биты 78-64) и в знаковый бит (бит 79) заносятся единицы, а значение поля TOS (Top Of Stack) слова состояния FPU устанавливается в 0 после выполнения каждой MMX-команды. Таким образом сложно рассчитать, какой FPU-регистр соответствует конкретному MMX-регистру. Кроме того, для команд FPU данные MMX в FPU-регистрах видны как вещественные знаковые. И наоборот: после выполнения операций над целыми числами в регистрах FPU в MMX-регистрах окажется бесполезный набор цифр.

Конечно, из данной проблемы можно найти выход путем преобразования данных MMX перед выполнением действий в FPU в формат, понятный командам FPU (и наоборот), однако для этого необходимо использовать целочисленные команды процессора, что неприменимо по условию поставленной задачи.

Итак, приходится констатировать невозможность совместного использования данных FPU и MMX. Это фактически означает, что все вычисления придется проводить только силами MMX. А это, к сожалению, невозможно ввиду ограниченности набора команд MMX. Таким образом, данную задачу решить невозможно.

Кроме того, в ходе исследований было определено, что время переключения процессора из режима FPU в режим MMX (и наоборот) очень велико. Это значит, что выигрыш в производительности может быть получен только в том случае, когда используются большие непрерывные участки MMX-кода. Однако при реализации алгоритмов алгоритмов многократной точности этого достичь не удалось.

Мы рассмотрели 2 попытки использования технологии MMX для целочисленных арифметических вычислений, однако они оказались неудачными. Так как других возможностей использования MMX для данной цели не видно, можно сделать вывод о неприменимости MMX для снижения нагрузки на целочисленный процессор.

Некоторые проблемы, описанные в данной статье, на момент ее написания уже удачно решены в технологии Streaming SIMD Extention, реализованной фирмой Intel в процессорах Pentium III и являющейся логичным продолжением технологии MMX. Анализ возможностей использования этой технологии для выполнения арифметических операций над числами многократной точности будет произведен в последующих исследованиях.

Список литературы: **1.** *Бердышев Е.* Технология MMX. Новые возможности процессоров P5 и P6. М.: Диалог-МИФИ, 1998. – 234 с.

Харьковский государственный технический университет радиоэлектроники

Поступила в редколлегию 15.03.2000