دانشگاه اصفهان

دانشکدهٔ فنی و مهندسی

درس: مباحث ویژه در ریزپردازنده‌های پیشرفته

موضوع:

بررسی تکنولوژی‌های MMX و SSE

استاد درس:

دکتر عباس وفایی

علی بهلولی

اردیبهشت 1386

فهرست مطالب

1 چکیده

شرکت اینتل در مسیر توسعهٔ ریز پردازنده های خود، دستورات جدیدی را برای کاربردهای مالتی مدیا به ریزپردازنده های خود اضافه کرد و آن را اصطلاحا قابلیت MMX نامید. این دستورات بعد از معرفی پردازنده‌های نسل پنجم[1] ارائه و در نسلهای بعدی کاملتر شدند. بنابراین پردازندهای اولیه پنتیوم و همچنین پنتیوم پرو از این قابلیت پشتیبانی نمی‌کردند و بعدها به این پردازنده‌ها اضافه شد. این دستورات قادر بودند یک دستور را روی چندین داده به صورت همزمان اجرا کنند(Single Instruction Multiple Data).

در فوریهٔ 1999 شرکت اینتل پردازندهٔ Pentium 3 را تولید کرد. در این پردازنده، قابلیت MMX پردازنده‌های قبلی کامل‌تر شد و SSE[2] نامیده شد.SSE دارای 70 دستور جدید برای پردازش صوت و گرافیک علاوه بر دستورات MMX می‌باشد.دستورات SSE امکان انجام محاسبات اعشاری را فراهم کرده است. در این تکنولوژی جدید برخلاف MMX به جای استفاده مشترک از رجیسترهای واحد Floating Point پردازنده از واحد مجزایی برای این منظور استفاده شده است. در نوامبر 2000، تکنولوژی SSE2 معرفی شد. در این تکنولوژی 144 دستور SIMD به پردازنده‌های P4 اضافه شدند. این تکنولوژی شامل تمام دستورات SSE و MMX پردازنده‌های قبلی می‌بود. در فوریهٔ 2004، تکنولوژی SSE3 معرفی شد. این تکنولوژی روی پردازندهٔ Pentium 4 Prescott پیاده‌سازی شد. در این تکنولوژی، 13 دستور SIMD جدید برای بهبود در محاسبات ریاضی پیچیده، گرافیک، کدگشایی ویدیو و سنکرون کردن threadها به تکنولوژی قبلی اضافه گشت. تکنولوژی SSE3 شامل تمامی دستورات MMX، SSE، SSE2 می‌باشد.

---

[1] Pentium

[2] Streaming SIMD Extention

2 مقدمه

شرکت اینتل از پیشگامان تولید پردازنده در جهان می‌باشد، این شرکت پس از اینکه پردازندهٔ 8088 در کامپیوترهای PC بکارگرفته شد با سرعت زیادی تکنولوژی طراحی و ساخت پردازنده‌های جدید را شروع کرد و نسل‌های متعددی از پردازنده‌ها را تولید کرد. در هر نسل از پردازنده‌ها علاوه بر اینکه سرعت اجرای برنامه‌ها بیشتر می‌شد قابلیت‌های جدیدی به پردازنده‌ها اضافه می‌شد. یکی از رویکردهای شرکت اینتل، مجتمع کردن برخی از قسمتهای PC در داخل پردازنده می‌باشد. به عنوان مثال کمک پردازندهٔ حسابی و حافظهٔ کش را به داخل پردازنده‌ها منتقل کرد. رویکرد دیگر اینتل جوابگویی به نیازهای جدید نرم‌افزار. یکی از این نیازها محاسبات سنگین ریاضی می‌باشد که برای برنامه‌های پردازش صوت و گرافیک مورد نیاز می‌باشد. تولید کنندگان کامپیوتر برای رفع این نیازمندی غالبا از پردازنده‌های DSP استفاده می‌کردند. با ورود مالتی مدیا به دنیای کامپیوترها کمبود دستورات مورد نیاز برای پردازش آنها به صورت چشمگیری احساس شد. از آنجا که اغلب پردازش‌های مالتی مدیا باید به صورت بلادرنگ انجام شود امکان پیاده سازی آنها توسط CPUهایی که مخصوص این کار نیستند، امکانپذیر نبود. شرکت اینتل در سال 1997 برای پاسخ به این نیاز و ارائهٔ را حلی برای آن، دستورات جدیدی را برای کاربردهای مالتی مدیا به ریزپردازنده های خود اضافه کرد. این دستورات بعد از معرفی پردازنده‌های نسل پنجم ارائه و در نسلهای بعدی کاملتر شدند. بنابراین پردازندهای اولیه پنتیوم و همچنین پنتیوم پرو از این قابلیت پشتیبانی نمی‌کردند و بعدها به این پردازنده‌ها اضافه شد. این دستورات قادر بودند یک دستور را روی چندین داده به صورت همزمان اجرا کنند(Single Instruction Multiple Data). شرکت اینتل این تکنولوژی جدید را MMX[3] نامید.

در فوریهٔ 1999 شرکت اینتل پردازندهٔ Pentium 3 را تولید کرد. در این پردازنده، قابلیت MMX پردازنده‌های قبلی کامل‌تر شد و SSE[4] نامیده شد.SSE دارای 70 دستور جدید برای پردازش صوت و گرافیک علاوه بر دستورات MMX می‌باشد.دستورات SSE امکان انجام محاسبات اعشاری را فراهم کرده است. در این تکنولوژی جدید برخلاف MMX به جای استفاده مشترک از رجیسترهای واحد Floating Point پردازنده از واحد مجزایی برای این منظور استفاده شده است. در نوامبر 2000، تکنولوژی SSE2 معرفی شد. در این تکنولوژی 144 دستور SIMD به پردازنده‌های P4 اضافه شدند. این تکنولوژی شامل تمام دستورات SSE و MMX پردازنده‌های قبلی می‌بود. در فوریهٔ 2004، تکنولوژی SSE3 معرفی شد. این تکنولوژی روی پردازندهٔ Pentium 4 Prescott پیاده‌سازی شد. در این تکنولوژی، 13 دستور SIMD جدید

---

[3] Multi Media eXtention

[4] Streaming SIMD Extention

برای بهبود در محاسبات ریاضی پیچیده، گرافیک، کدگشایی ویدیو و سنکرون کردن threadها به تکنولوژی قبلی اضافه گشت. تکنولوژی SSE3 شامل تمامی دستورات MMX، SSE، SSE2 می‌باشد.
در این گزارش به بررسی این دو تکنولوژی شرکت اینتل پرداخته می‌شود و نحوهٔ استفاده از این قابلیتها شرح داده می‌شود و در نهایت کارایی این تکنولوژها بررسی و مقایسه می‌شوند.

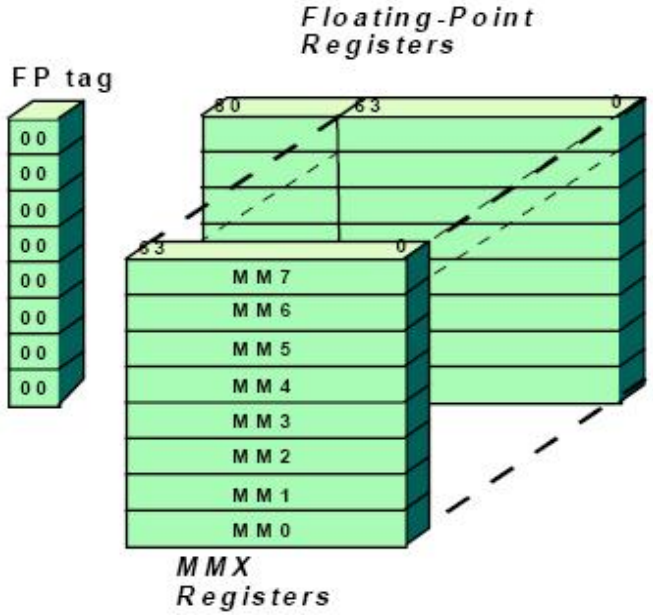
3 تکنولوژی MMX

در سال 1997 شرکت اینتل به پردازنده‌های پنتیوم خود 57 دستورالعمل جدید برای استفاده در پردازشهای graphics، video، compress/decompress، audio و signal processing اضافه کرد. این عمل در پاسخ به نیاز نرم‌افزارهای مالتی‌مدیا بوجود آمد. شرکت اینتل به این منظور تغییر مهمی در معماری پردازنده‌هایش بوجود آورد و این تغییر قابلیت اجرای دستورات به صورت SIMD (Single Instruction Multiple Data) می باشد. این تکنولوژی MMX نامیده شد. MMX را بعضا مخفف Matrix Math Extensions نیز می‌گویند که با توجه به نوع دستوراتی که در این معماری اضافه گردیده، اسم مناسبی است. چون، این دستورات بر روی ماتریسی از داده‌ها اجرا می‌گردند و در کاربردهایی که در بالا نام بردیم بسیار مفید هستند. از این رو انتخاب اسم Multi Media eXtensions نیز برای آن درست است. در پردازنده‌های قبلی ساختار به صورت SISD (Single Instruction Single Data) بود یعنی هر دستور فقط روی یک سری داده عمل می‌کرد و قادر نبودند در یک زمان روی ماتریس عمل کنند. در SIMD یک دستور می‌تواند همزمان روی چندین داده عمل کند. در واقع روش SIMD یک روش موازی سازی در داخل پردازنده می‌باشد که باعث بالا رفتن سرعت اجرای برخی از برنامه‌ها می‌گردد. افزایش سرعت اجرای برنامه‌های multimedia با معماری MMX قابل توجه است. اما مسلما برنامه‌های کاربردی دیگر مثل برنامه های تجاری استفاده چندانی از MMX نمی‌کنند. در برنامه‌هایی که هدف پردازش Video streaming، audio file editing، graphics manipulation، game و برنامه های کاربردی مشابه بیشترین استفاده را از MMX می‌برند.
در ادامه به شرح تکنولوژی MMX پرداخته می‌شود.

3-1 رجیسترهای MMX

در تکنولوژی MMX از هشت رجیستر استفاده می‌شود. این رجیسرها،64 بیتی می‌باشند. دستورات MMX به این رجیسترها به صورت مستقیم و با نامهای MM0 تا MM7 دسترسی دارند. در واقع این رجیسترها به صورت مجزا برای MMX ایجاد نشده‌اند و اینتل از هشت رجیستر 80 بیتی خود که در واحد FPU ریزپردازنده به کار گرفته شده‌اند استفاده می‌کند اینتل 64 بیت کم ارزش این رجیسترها را در تکنولوژی

MMX استفاده کرده و آنها را MM0 تا MM7 نامگذاری کرده است. به دلیل مشترک بودن این رجیسترها نمی‌توان همزمان از دستورات FPU و دستورات MMX استفاده کرد.



شکل 1: رجیسترهای MMX

3-2 نوع داده‌ها در MMX

چهار نوع دادهٔ جدید در این تکنولوژی تعریف شده‌است. این نوع‌ها همگی integer می‌باشند که عبارتند از:

آرایهٔ 8 تایی از بایت، در این نوع داده، رجیستر 64 بیتی به 8 قسمت تقسیم می‌شود که هر یک از این قسمت‌ها می‌تواند یک بایت داده را نگهداری کند.

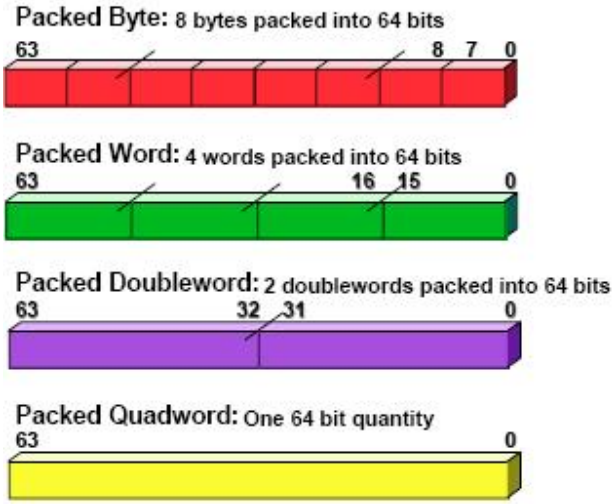
آرایهٔ 4 تایی از Word ، در این نوع داده، رجیستر 64 بیتی به 4 قسمت تقسیم می‌شود که هر یک از این قسمت‌ها می‌تواند یک دادهٔ شانزده بیتی را نگهداری کند.

آرایهٔ 2 تایی از Dword ، در این نوع داده، رجیستر 64 بیتی به 2 قسمت تقسیم می‌شود که هر یک از این قسمت‌ها می‌تواند یک دادهٔ 32 بیتی را نگهداری کند.

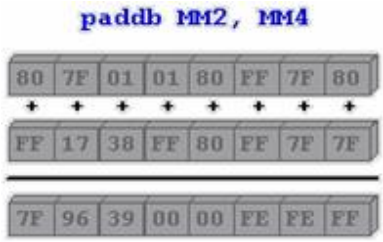
و آخرین نوع Qword می‌باشد. در این نوع داده، کل رجیستر به صورت یک دادهٔ 64 بیتی در نظر گرفته می‌شود.

در شکل 2 انواع داده‌هایی که توسط MMX پشتیبانی می‌شوند نمایش داده شده است. دستورات MMX قادرند روی داده‌های فوق عمل کنند. به عنوان مثال در شکل 3، نحوهٔ جمع زدن دو آرایهٔ هشت‌تایی از اعداد هشت بیتی را نمایش داده است. همانطور که در شکل 3 ملاحظه می‌شود ا زدو رجیستر MM2 و MM4 استفاده شده است. یکی از آرایه‌ها در رجیستر MM2 و دیگری در رجیستر MM4 قرار داده شده است و با

استفاده از دستور paddb MM2,M4 همزمان هشت عدد صحیح با هم جمع شده‌اند. دستورات MMX ، برای مشخص کردن نوع داده‌ای که با آن کار می‌کنند از علایم b، w، d و q استفاده می‌کنند که به ترتیب نشان دهنده byte، word، double word و quad word می باشد.



شکل 2: **نوع داده هایی که توسط MMX پشتیبانی می شوند**



شکل 3: **جمع همزمان هشت عدد با استفاده از تکنولوژی MMX**

3-3 دستورات MMX

MMX دارای 57 دستور می‌باشد که شامل دستورات محاسباتی، مقایسه، تبدیل، منطقی، انتقال و مدیریت حالت می‌باشد. که در ادامه به توضیح آنها پرداخته می‌شود.

3-3-1 دستورات محاسباتی

دستورات محاسباتی شامل جمع، تفریق، ضرب، شیفت حسابی و ضرب-جمع می‌باشد. با توجه به اینکه در هنگام انجام محاسبات ممکن است Overfllow یا Underfllow رخ دهد و امکان تصحیح وجود ندارد، دستورات MMX هنگام انجام محاسبات می توانند به روشهای unsigned saturation، signed saturation

و wraparound کار کنند. در روش unsigned saturation پس از انجام محاسبات اگر نتیجه محاسبات بیشتر از حداکثر یا کمتر از حداقل شد (حداقل در این حالت صفر است)، نتیجه محاسبات به ترتیب برابر با حداکثر یا حداقل آن نوع داده قرار داده می‌شود. در روش signed saturation پس از انجام محاسبات اگر نتیجه محاسبات بیشتر از حداکثر یا کمتر از حداقل شد، نتیجه محاسبات به ترتیب برابر با حداکثر یا حداقل آن نوع داده قرار داده می‌شود. در روش wraparound پس از انجام محاسبات اگر نتیجه محاسبات بیشتر از حداکثر شد، به همان اندازه‌ای که از حداکثر آن نوع داده رد کرده است به حداقل نوع داده اضافه شده و ثبت می‌شود. به همین طریق اگر نتیجه محاسبات کمتر از حداقل شد، به همان اندازه‌ای که از حداقل پایینتر رفته است از حداکثر نوع داده کاسته شده و ثبت می‌شود.

دستورات MMX برای نشان دادن اینکه از کدامیک از روشهای محاسبات unsigned saturation یا signed saturation استفاده می کنند، به ترتیب علایم us و ss (گاهی هم s) را به کار می‌برند. عدم وجود هر یک از این علایم نیز نشان دهنده محاسبات wraparound است. به عنوان مثال در دستور paddb ، عمل جمع به صورت wraparound انجام می‌شود.

قالب دستورات MMX به صورت زیر می‌باشد:

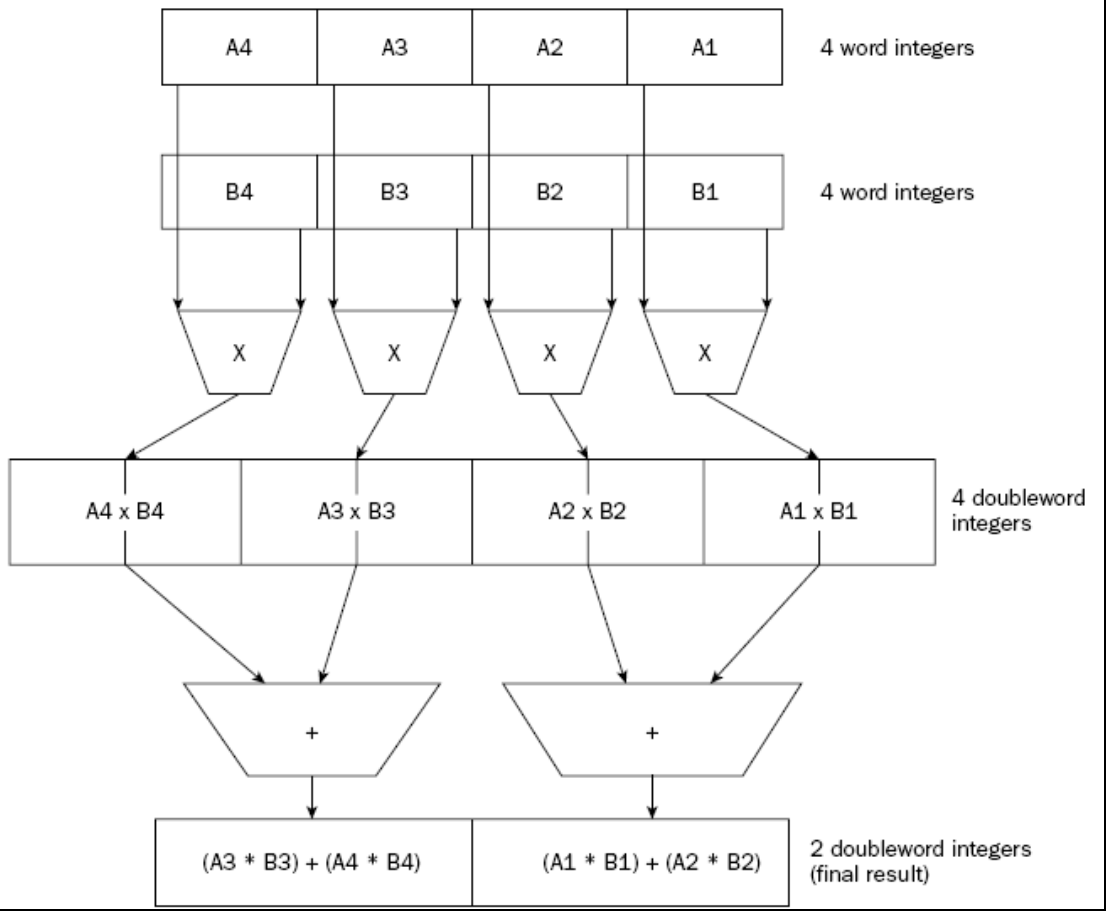
- پیشوند P
- چندین حرف برای تعیین عملکرد دستور(به عنوان مثال add, cmp, xor)
- پسوند برای تعیین نوع انجام محاسبات(US برای Unsigned saturation و S برای Signed aturation)
- پسوند برای تعیین نوع داده‌ها (b، w، d، q به ترتیب برای داده‌های ، 8، 16، 32 و 64 بیتی)

دستورات محاسباتی MMX در جدول 1 لیست شده‌اند.

جدول1: لیست دستورات محاسباتی

PADD (Packed Add)	PADDB mm,mm/m64 PADDW mm,mm/m64 PADDD mm,mm/m64
PADDS (Packed Add with Saturation)	PADDSB mm, mm/m64 PADDSW mm, mm/m64
PADDUS (Packed Add Unsigned with Saturation)	PADDUSB mm, mm/m64 PADDUSW mm, mm/m64
PSUB (Packed Subtract)	PSUBB mm, mm/m64 PSUBW mm, mm/m64 PSUBD mm, mm/m64

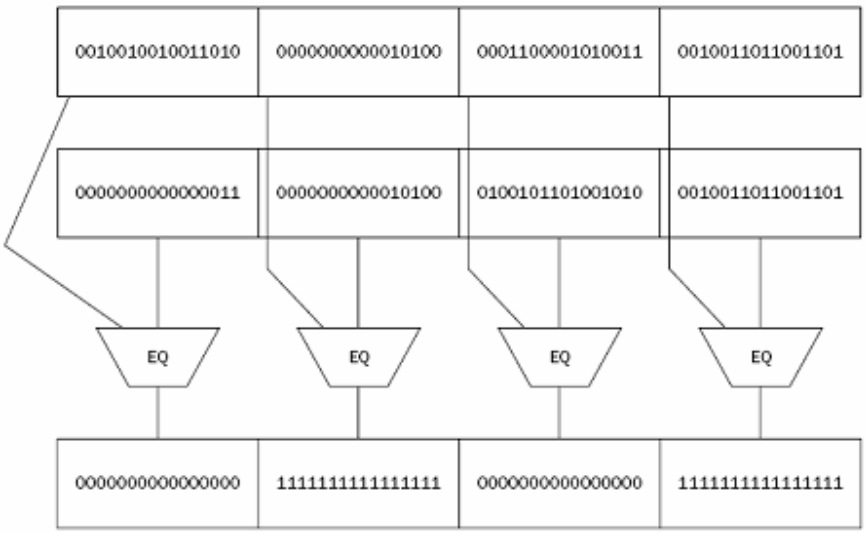
PSUBS (Packed Subtract with Saturation) PSUBUS (Packed Subtract Unsigned with Saturation)	PSUBSB mm, mm/m64 PSUBSW mm, mm/m64 PSUBUSB mm, mm/m64 PSUBUSW mm, mm/m64
MULTIPLY PMULHW (Packed Multiply High) PMULLW (Packed Multiply Low)	PMULHW mm, mm/m64 PMULLW mm, mm/m64 در دستور PMULHW 16 بیت پرارزش حاصلضرب در رجیستر مقصد قرار داده می‌شود در دستور PMULLW16 بیت کم ارزش حاصلضرب در رجیستر مقصد قرار داده می‌شود
PMADDWD (Packed Multiply and Add)	PMADDWD mm, mm/m64



2-3-3 دستورات مقایسه

تکنولوژی MMX دارای دستورات تساوی و بزرگتر می‌باشد. اگر شرط برقرار بود آنگاه تمام بیتهای مریوطه در رجیستر مقصد یک می‌شوند و اگر برقرار نبود آنگاه تمام بیتها صفر می‌شود. در جدول 2 لیست دستورات مقایسه نمایش داده شده است.

جدول2: لیست دستورات مقایسه‌ای

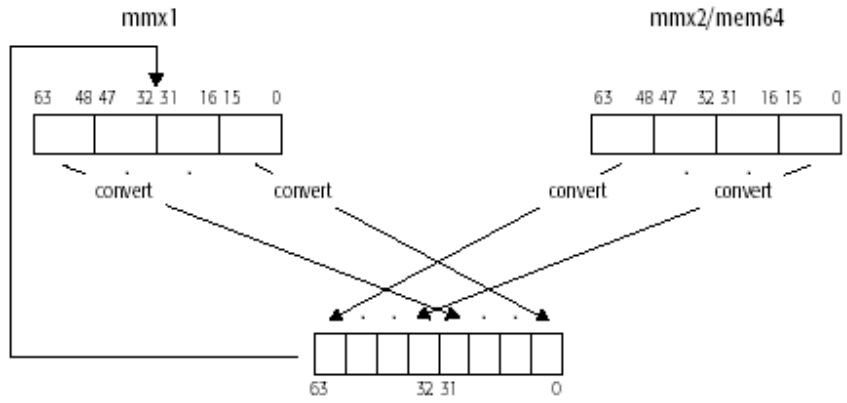
PCMPEQ (Packed Compare for Equal)	PCMPEQB mm, mm/m64 PCMPEQW mm, mm/m64 PCMPEQD mm, mm/m64
	
PCMPGT (Packed Compare for Greater Than)	PCMPGTB mm, mm/m64 PCMPGTW mm, mm/m64 PCMPGTD mm, mm/m64

3-3-3 دستورات تبدیل

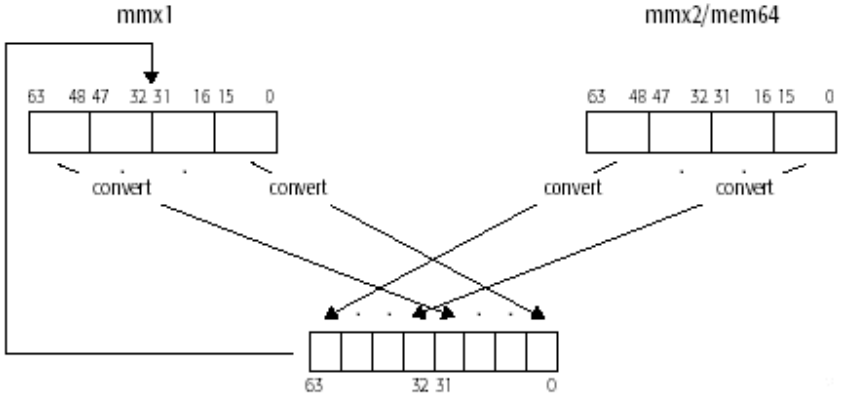
در تکنولوژی MMX دستوراتی برای تبدیل نوع داده‌ها و ساخت داده‌های سازگار با تکنولوژی MMX وجود دارد این دستورات در جدول 3 لیست شده‌اند.

جدول3: لیست دستورات تبدیل

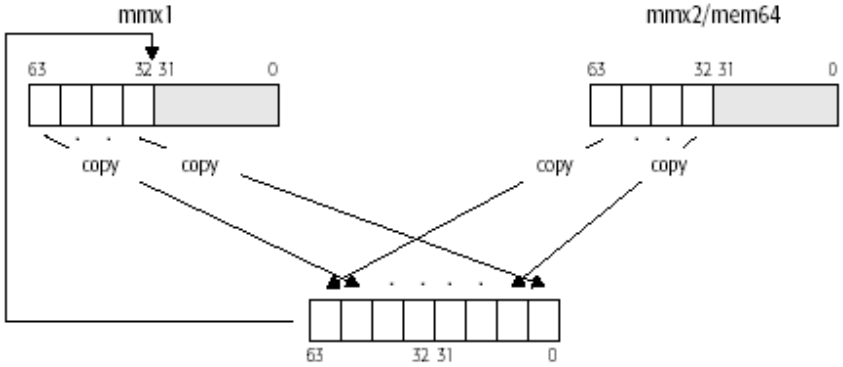
PACKSS (Packed with Signed Saturation)	PACKSSWB mm, mm/m64 PACKSSDW mm, mm/m64



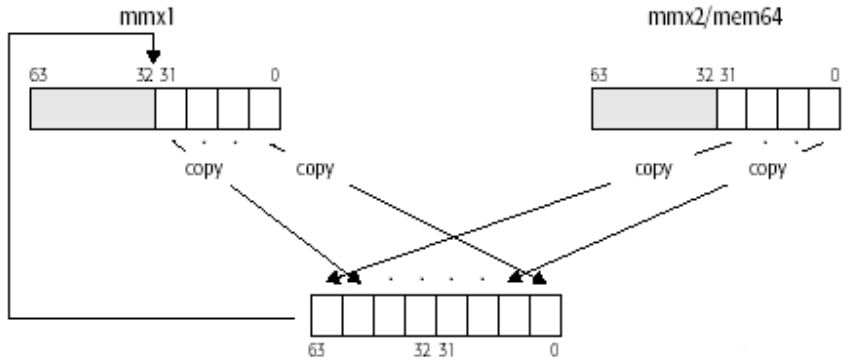
PACKUSWB (Packed with Unsigned Saturation)	PACKUSWB mm, mm/m64



PUNPCKH (Unpack High Packed Data)	PUNPCKHBW mm, mm/m64 PUNPCKHWD mm, mm/m64 PUNPCKHDQ mm, mm/m64



PUNPCKL (Unpack Low Packed Data)	PUNPCKLBW mm, mm/m32 PUNPCKLWD mm, mm/m32 PUNPCKLDQ mm, mm/m32

3-3-4 دستورات منطقی

دستورات منطقی MMX روی 64 بیت عمل می کنند. لیست این دستورات در جدول 4 ارائه گردیده است.

جدول4: لیست دستورات منطقی

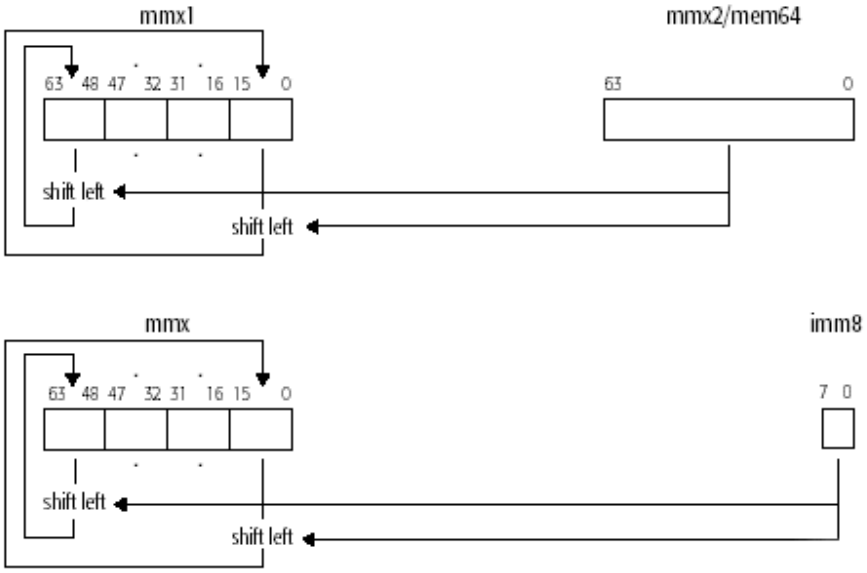
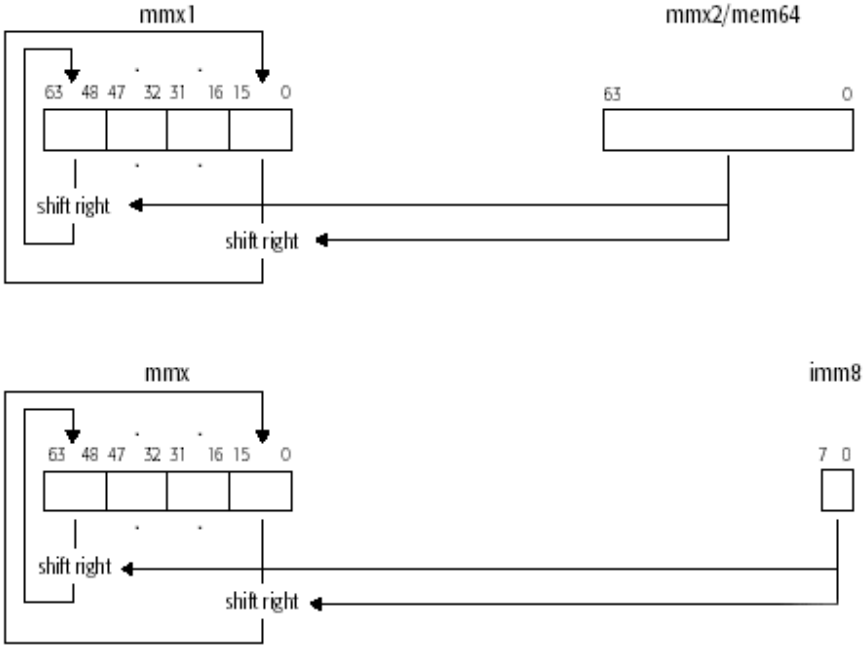
PAND (Bitwise Logical And), PANDN (Bitwise Logical And Not), POR (Bitwise Logical OR), and PXOR (Bitwise Logical Exclusive OR)	PAND mm, mm/m64 PANDN mm, mm/m64 POR mm, mm/m64 PXOR mm, mm/m64
	

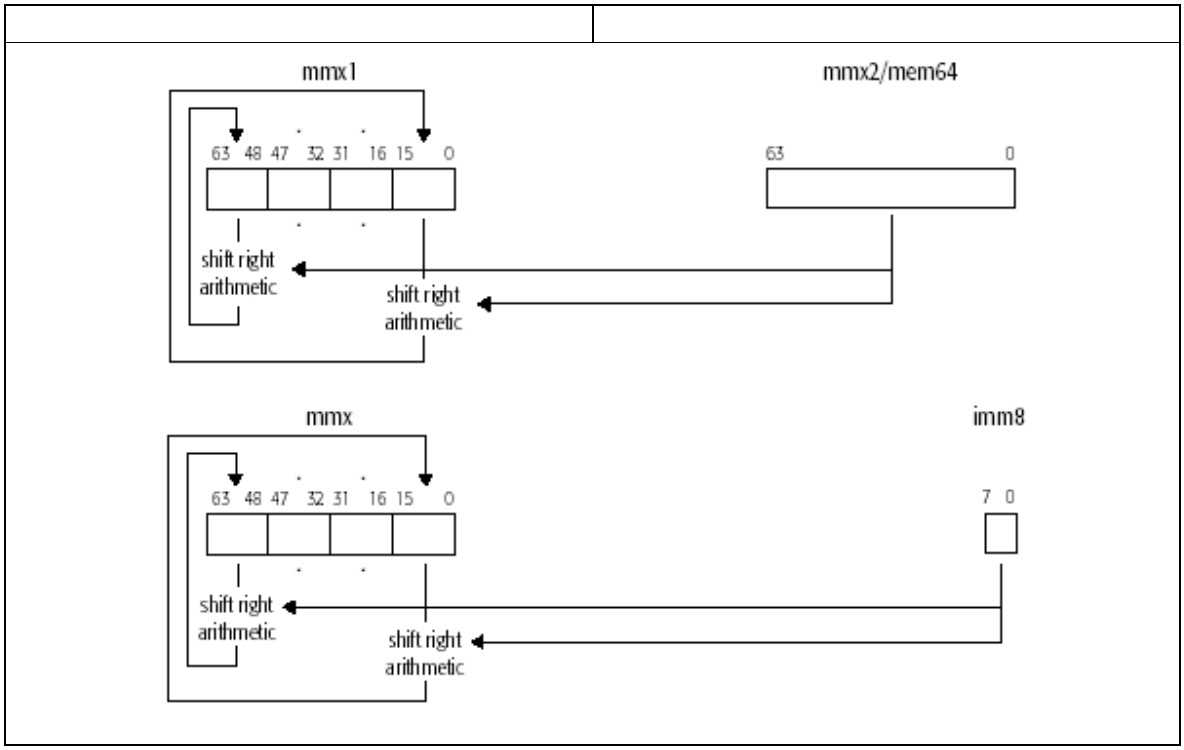
3-3-5 دستورات شیفت

MMX دارای دستوراتی برای شیفت منطقی و حسابی روی اعداد می‌باشد در جدول 5 لیست دستورات شیفت ارائه شده است

جدول5: لیست دستورات شیفت

PSLL (Packed Shift Left Logical)	PSLLW mm, mm/m64 PSLLW mm, imm8 PSLLD mm, mm/m64 PSLLD mm, imm8 PSLLQ mm, mm/m64 PSLLQ mm, imm8

<table>
<tr><td></td><td></td></tr>
<tr><td colspan="2"> 
</td></tr>
<tr><td>PSRL (Packed Shift Right Logical)</td><td>PSRLW mm, imm8
PSRLD mm, mm/m64
PSRLD mm, imm8
PSRLQ mm, mm/m64
PSRLQ mm, imm8</td></tr>
<tr><td colspan="2"> 
</td></tr>
<tr><td>PSRA (Packed Shift Right Arithmetic)</td><td>PSRAW mm, mm/m64
PSRAW mm, imm8
PSRAD mm, mm/m64
PSRAD mm, imm8</td></tr>
</table>



6-3-3 دستورات مدیریت حالات

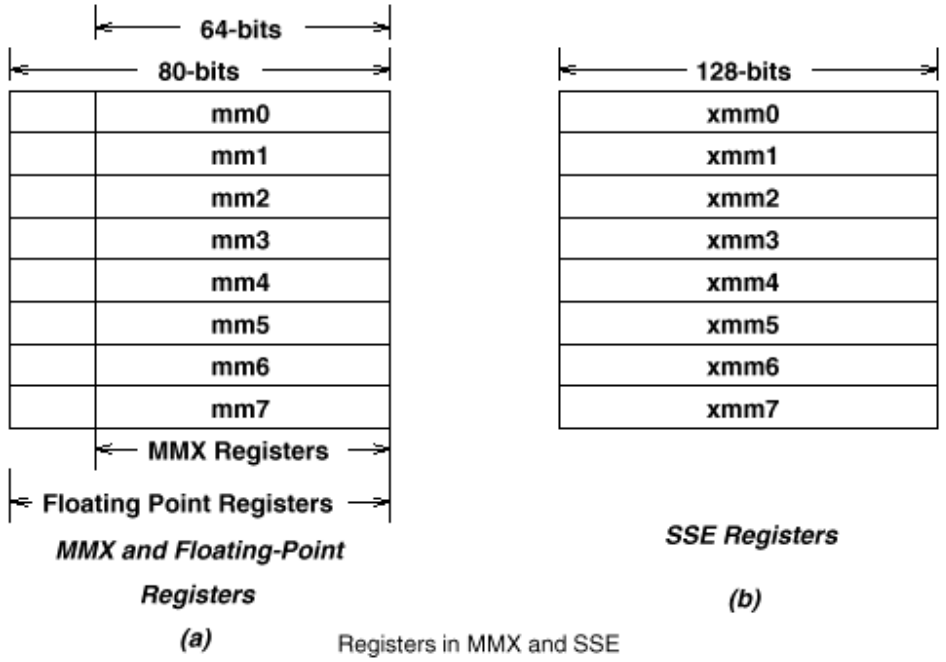
MMX دارای یک دستور به نام EMMS می‌باشد که باید حتما بعد از استفاده از دستورات MMX به کار رود تا وضعیت ثباتها و وضعیت پردازنده دوباره به حالت محاسبات Floating Point باز گردد.

4 تکنولوژی SSE

تکنولوژی SSE در پردازنده PentiumIII معرفی شد. این تکنولوژی در واقع توسعه یافته تکنولوژی MMX می باشد. مزایای تکنولوژی SSE نسبت به MMX عبارتند از:

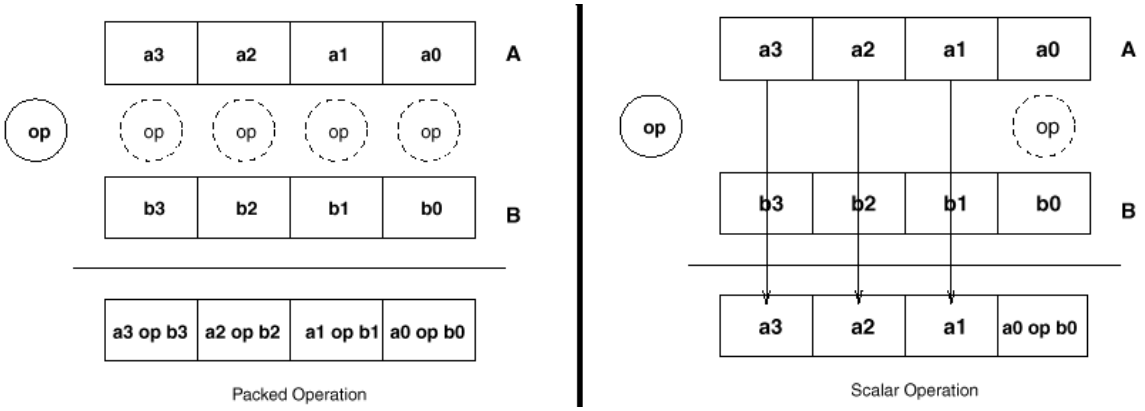
- دستورات MMX فقط روی اعداد صحیح محاسبات را انجام میدادند در حالی که در SSE محاسبات را روی اعداد اعشاری انجام می دهد.
- دستورات MMX قادر بودند به صورت همزمان محاسبات را روی دو عدد صحیح 32بیتی انجام دهند در حالی که SSE قادر است محاسبات را به صورت همزمان روی 4 عدد اعشاری 32 بیتی انجام دهد.
- تکنولوژی MMX از رجیسترهای واحد FPU پردازنده به صورت اشتراکی استفاده می کرد و رجیستر مجزا نداشت در حالی که درتکنولوژی SSE ، هشت رجیستر 128 بیتی مجزا در نظر گرفته

شده است. هر کدام از این رجیسترها قادرند چهار عدد اعشاری را نگهداری کنند. در شکل 3، رجیسترهای تکنولوژی SSE نمایش داده شده است.



شکل 3: **مقایسه رجیسترهای تکنولوژی MMX و SSE**

- دستورات SSE علاوه براینکه قابلیت انجام محاسبات روی تمامی اعداد یک رجیستر دارند(Packed)، می توان محاسبات را روی یک عدد نیز انجام داد(Scaler). در شکل 4، این دو قابلیت و تفاوت آنها نمایش داده شده است.



- **شکل** 4: **عملیات Scaler و Packed در تکنولوژی SSE**

4-1 تکنولوژی SSE2

تکنولوژی SSE2 نوع داده های جدیدی به شرح زیر را به SSE اضافه کرد:

- 128-bit packed double-precision floating-point value (contains two double-precision values)
- 128-bit packed byte integer value (contains 16 single-byte integer values)
- 128-bit packed word integer value (contains eight word integer values)
- 128-bit packed doubleword integer value (contains four doubleword integer values)
- 128-bit packed quadword integer value (contains two quadword integer values)

در تکنولوژی SSE2، تعدادی دستور جدید برای محاسبات اعشاری و صحیح بوجود آمد.

4-2 تکنولوژی SSE3

در تکنولوژی SSE3، نوع دادهٔ جدید معرفی نشد و فقط دستورات جدیدی به مجموعهٔ دستورات SSE2 اضافه گردید.

5 لیست دستورات SSE

به علت تعداد زیاد دستورات SSE از اوردن آنها در این گزارش خودداری میکنیم. در CD این گزارش لیست کاملی از دستورات این تکنولوژی و توضیحات آنها ارائه شده است.

6 نحوهٔ تشخیص پشتیبانی پردازنده از دستورات SIMD

در پردازنده های اینتل دستوری به نام CPUID وجود دارد که با استفاده از این دستور میتوان اطلاعات زیادی در مورد پردازنده بدست آورد. مقدار رجیستر EAX قبل از اجرای این دستور مشخص می کند که برنامه نویس به چه اطلاعاتی میخواهد دسترسی یابد.

اگر EAX=1 باشد و دستور CPUID اجرا شود، پردازنده با مقدار دهی دو رجیستر ECX و EDX مشخص می کند که آیا از تکنولوژیهای MMX، SSE، SSE2 یا SSE3 پشتیبانی می کند یا خیر. در شکل 5 بیتهایی که به این منظور استفاده شده اند را نمایش می دهد.

REGISTER	Bit	Feature
EDX	23	Supports MMX instructions
EDX	25	Supports SSE instructions
EDX	26	Supports SSE2 instructions
ECX	0	Supports SSE3 instructions

7 نحوهٔ نوشتن و اجرای برنامه با استفاده از تکنولوژی SIMD

برای نوشتن برنامه هایی که از تکنولوژی SIMD استفاده میکند، روشهای مختلفی وجود دارد.در ادامه به توضیح در مورد این روشها و آوردن یک برنامه نمونه پرداخته می شود.

7-1 استفاده از اسمبلر

این روش همان روش سنتی می باشد که برنامه به زبان اسمبلی نوشته می شود و سپس با استفاده از یک اسمبلر و linker فایل اجرایی تولید می شود. برنامه ای که به این روش نوشته می شود دارای سرعت زیادی می باشد ولی پیچیدگی زیادی دارد. برای ساخت فایل اجرایی از برنامه Microsoft Macro Assembler نسخهٔ 6.11d به بعد آن باید استفاده شود. در CD مربوط به این گزارش این نسخهٔ اسمبلر وجود دارد.

7-1-1 مثال برنامه ضرب

فرض کنید میخواهیم دو عددی که در رجیسترهای xmm1 وxmm2 قرار دارند را در هم ضرب کنیم و نتیجه را در xmm0 ذخیره کنیم . برنامه به صورت زیر میشود:

```
mov xmm0, xmm1;
mulps xmm0, xmm2;
```

7-2 نوشتن برنامه اسمبلی در یک زبان سطح بالا

در این روش با استفاده از دستور asm_ میتوان قسمتی از برنامه را به زبان اسمبلی نوشت. این روش باعث می شود که از دستورات اسمبلی مستقیما استفاده کنیم ولی درگیر استفاده از برنامههای اسمبلر و linker نشویم و همچنین برای نوشتن بقیه قسمتهای دیگر برنامه بتوان از زبان سطح بالا استفاده کرد.

در حال حاضر برنامه های Visual Stdio C++ 6 و Visual Stdio C++ 7 (.Net) برای اینکار وجود دارند. باید توجه کرد که Visual Stdio C++ 6 چون از سال 1998 میباشد از دستوراتSSE پشتیبانی نمیکند و فقط از دستورات MMX پشتیبانی میکند. بنابراین برای نوشتن برنامه های SIMD حتما باید از Visual Stdio C++ 7 یا نسخه های بالاتر آن استفاده کرد.

در نوشتن برنامه های این تحقیق از این روش استفاده شده است.

7-2-1 مثال برنامه ضرب

فرض کنید میخواهیم دو عددی که در رجیسترهای xmm1 وxmm2 قرار دارند را در هم ضرب کنیم و نتیجه را در xmm0 ذخیره کنیم . برنامه به صورت زیر میشود:

```
#include <xmmintrin.h>
...
_asm {
        push esi;
        push edi;
        ; a is loaded into xmm1
        ; b is loaded into xmm2
        mov xmm0, xmm1;
        mulps xmm0, xmm2;
        ; store result into c
        pop edi;
        pop esi;
}
...
```

3-7 استفاده از کتابخانه های Intrinsics در زبانهای برنامه نویسی

در این روش، از دستورات C که معادل دستورات SSE می باشند استفاده می شود. این دستورات در یک کتابخانه تعریف شده است و می توان از آن استفاده کرد. به عنوان مثال برای دستور addps که یکی از دستورات SSE می‌باشد، دستوری به صورت _mm_add_ps در این کتابخانه تعریف شده است. برای استفاده از این روش باید فایل xmmintrin.h در اول برنامه include گردد.

1-3-7 مثال برنامه ضرب به روش Intrinsics

فرض کنید دو عدد a و b اعداد 128 بیتی باشند و نتیجه محاسبات در عدد c ذخیره شود. برنامه ضرب به صورت زیر می‌باشد:

```
include <xmmintrin.h>
...
__m128 a, b, c;
a = _mm_set_ps(4, 3, 2, 1)
b = _mm_set_ps(4, 3, 2, 1)
c = _mm_set_ps(0, 0, 0, 0)
c = _mm_mul_ps(a, b);
...
```

4-7 استفاده از دستورات C++

در این روش از کلاس F32vec4 استفاده می‌شود. این کلاس یک لایه تجرید برای اعداد 128 بیتی معرفی می‌کند و می‌توان مشابه دستوراتی که برای محاسبات عادی وجود دارد را به کار برد. برای استفاده از این روش باید فایل fvec.h در اول برنامه include گردد.

7-4-1 مثال برنامه ضرب در C++

فرض کنید دو عدد a و b اعداد 128 بیتی باشند و نتیجه محاسبات در عدد c ذخیره شود. برنامه ضرب به صورت زیر می‌باشد:

```
#include <fvec.h>
...
F32vec4 a(4, 3, 2, 1), b(4, 3, 2, 1), c(0, 0, 0, 0);
...
c =a * b;
...
```

7-5 استفاده از کامپایلر

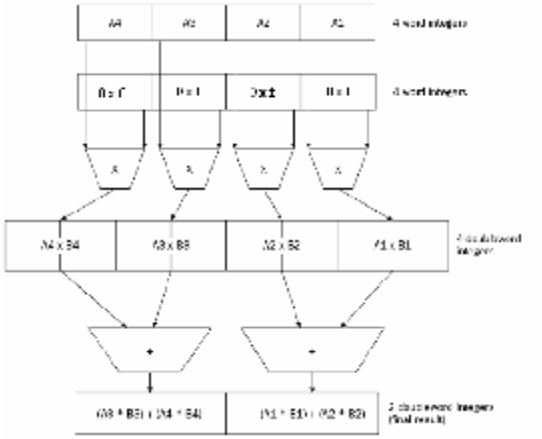
در حال حاضر فقط Intel compiler و Microsoft Macro Assembler از دستورات SSE پشتیبانی می‌کنند. از آنجا که Intel compiler با محیط Microsoft Visual Studio مجتمع شده است می‌توان با انجام تنظیماتی در محیط Microsoft Visual Studio ، کامپایلر فوق را معرفی کرد تا برنامه نوشته شده را با تبدیل به دستورات SSE کامپایل کند و کلا از دید برنامه نویس مخفی و به راحتی انجام شود.

8 نمونهٔ برنامه با تکنولوژی MMX

در این قسمت به توضیح برنامه‌ای که در محیط Microsoft Visual Studio .Net نوشته شد، پرداخته می‌شود. این برنامه علاوه بر اینکه نحوهٔ استفاده از تکنولوژی MMX در برنامه نویسی به زبان C را ارائه می‌نماید، مقایسه‌ای بین سرعت انجام یک عملیات ریاضی با استفاده از دستورات عادی پردازنده و سرعت انجام همان محاسبات با استفاده از دستورات MMX ارائه داد ه‌شده است.

در این برنامه یک آرایهٔ صدهزار عنصری از اعداد صحیح تعریف شده است. این برنامه روی هر دو عدد متوالی این آرایه فرمول روبرو را محاسبه می‌کند ($\mathrm{Re}\,sult = 15(a[i] + a[i+1])$).

محاسبهٔ فرمول فوق یک بار در تابع Compute_Without_MMX() با استفاده از دستورات ساده ریز پردازنده انجام شده است. در این برنامه از یک حلقهٔ for استفاده شده است. چون نتیجهٔ حاصلضرب دو عدد 16 بیتی، یک عدد 32 بیتی می‌شود به همین دلیل ابتدا داده‌های آرایه در متغیرهای 32 بیتی به نام Temp به صورت موقت ذخیره شده‌اند و محاسبات انجام شده است.



در روش دوم، محاسبه فرمول فوق در تابع Compute_With_MMX() با استفاده از دستورات MMX انجام شده است. در این تابع با استفاده از دستور PMADDWD که نحوهٔ عملکرد آن در شکل 4 نمایش داده شده است، محاسبات انجام شده است. عدد 64 بیتی x000f000f000f000f0 که شامل چهار عدد 15 می باشد در رجیستر mm0 قرار داده شده است و رجیستر edi به چهار عدد متوالی در آرایه اشاره می‌کند.

در برنامهٔ نوشته شده ابتدا با استفاده از دستور cpuid، از اینکه پردازنده از دستورات MMX پشتیبانی می‌کند یا خیر مطمئن می‌شویم. سپس با استفاده از تابع InitArray() آرایهٔ 100هزار عنصری با مقادیر تصادفی مقدار دهی اولیه شده است.

زمان اجرای محاسبات در دو تابع Compute_With_MMX() و Compute_ Without _MMX() توسط تابع QueryPerformanceCounter بدست آمده است. این تابع شمارندهٔ داخلی ریز پردازنده که سیکلها را شمارش می‌کند را برمی‌گرداند سپس با استفاده از فرکانس کلاک پردازنده که توسط تابع QueryPerformanceFrequency به دست می‌آید، زمان انجام محاسبات بدست آورده می‌شود.

نتیجهٔ اجرای برنامه روی یک کامپیوتر Pentium 4 3.00GHz به این صورت شد که انجام محاسبات فوق به صورت عادی حدودا 1807.701 میکروثانیه و زمان اجرا با استفاده از دستورات MMX مدت اجرا 239.881 میکرو ثانیه شد. که با استفاده از نتایج فوق مشاهده می‌شود که استفاده از دستورات MMX باعث شده که سرعت اجرا 7.5 برابر گردد.

9 نمونهٔ برنامه با تکنولوژی SSE

در این قسمت به توضیح برنامه‌ای که در محیط Microsoft Visual Studio .Net نوشته شد، پرداخته می‌شود. این برنامه علاوه بر اینکه نحوهٔ استفاده از تکنولوژی SSE در برنامه نویسی به زبان C را ارائه

می‌نماید، مقایسه‌ای بین سرعت انجام یک عملیات ریاضی با استفاده از دستورات عادی پردازنده و سرعت انجام همان محاسبات با استفاده از دستورات SSE ارائه داد ه‌شده است.

در این برنامه یک آرایهٔ صدهزار عنصری از اعداد اعشاری تعریف شده است. این برنامه روی هر عدد فرمول زیر را محاسبه می کند.

$\sqrt{2.8x}$

محاسبهٔ فرمول فوق یک بار در تابع Compute_Without_SSE () با استفاده از دستورات ساده ریز پردازنده انجام شده است. در این برنامه از یک حلقهٔ for استفاده شده است. در روش دوم، محاسبه فرمول فوق در تابع Compute_With_SSE () با استفاده از دستوراتSSEانجام شده است. در این تابع با استفاده از دستور sqrtps، محاسبات انجام شده است..

در برنامهٔ نوشته شده ابتدا با استفاده از دستور cpuid، از اینکه پردازنده از دستورات SSE پشتیبانی می کند یا خیر مطمئن می‌شویم. سپس با استفاده از تابع InitArray() آرایهٔ 100هزار عنصری مقدار دهی اولیه شده است.

زمان اجرای محاسبات در دو تابع Compute_With_SSE() و Compute_ Without _ SSE () توسط تابع QueryPerformanceCounter بدست آمده است. این تابع شمارندهٔ داخلی ریز پردازنده که سیکلها را شمارش می‌کند را برمی‌گرداند سپس با استفاده از فرکانس کلاک پردازنده که توسط تابع QueryPerformanceFrequency به دست می‌آید، زمان انجام محاسبات بدست آورده می‌شود.

نتیجهٔ اجرای برنامه روی یک کامپیوتر Pentium 4 3.00GHz به این صورت شد که انجام محاسبات فوق به صورت عادی حدودا 1666.338 میکروثانیه و زمان اجرا با استفاده از دستورات SSE مدت اجرا 343.557 میکرو ثانیه شد. که با استفاده از نتایج فوق مشاهده می‌شود که استفاده از دستورات SSE باعث شده که سرعت اجرا 48.5 برابر گردد.

10 نتیجه‌گیری

در این تحقیق ابتدا در مورد تکنولوژی‌های MMX و SSE توضیحاتی ارائه گردید، سپس در مورد انواع روشهایی که می‌توان از این تکنولوژیها برای برنامه نویسی استفاده کرد شرح داده شد و در انتها دو مثال برنامه نویسی که به زبان C نوشته شده بود، ارائه گردید.

پس از اجرای برنامهٔ فوق روی داده های بیشتر و تعداد دفعات تکرار بیشتر روی یک پردازنده به این نتیجه گیری می‌توان رسید که در کل SpeedUp تکنولوژی MMX خیلی کمتر از SSE می‌باشد.

میزان افزایش سرعت اجرای برنامه در تکنولوژی MMX از 2 تا حدودا ده برابر می‌باشد و کاملا به نوع محاسبات بستگی دارد. به عنوان مثال در جمع عادی اعداد 16 بیتی میزان افزایش سرعت 2 برابر بود، در حالی که تصور می شود باید 4 برابر باشد

میزان افزایش سرعت اجرای برنامه در تکنولوژی SSE از هشت تا حدودا صد برابر می‌باشد و قابل ملاحظه می‌باشد. سرعت انجام محاسباتی که به صورت تکراری می‌باشند و حجم داده ها کم است(دستورات و داده ها در cacheقرار داشته باشند) فوق العاده بیشتر از حالت عادی CPU می‌باشد.(حداقل 30 برابر)

11 ضمیمهٔ 1: لیست برنامه MMX

```
/*
Example of using MMX technology and compute speed up
University Of Isfahan
Author:Ali Bohlooli
Advanced Microprocessor course
Istructor:Dr Vafaei
Bahar 1386
*/#include<stdio.h>
#include<conio.h>
#include <windows.h>


#define _MMX_FEATURE_BIT        0x00800000      // bit 23
#define _SSE_FEATURE_BIT        0x02000000      // bit 25
#define ARRAY_SIZE 100000L
#define NumberOfRepeat 1L


__declspec(align(16)) __int16 InitialArray[ARRAY_SIZE];
__declspec(align(16)) __int16 ResultArray[ARRAY_SIZE];
__declspec(align(16)) __int16 ResultArray1[ARRAY_SIZE];
LARGE_INTEGER Freq;
LARGE_INTEGER BeginTime;
LARGE_INTEGER EndTime;
_int64 WithMMXTime;
_int64 WithOutMMXTime;


void InitArray()
{
    _int64 i;
    for ( i = 0; i < ARRAY_SIZE; i++ )
        InitialArray[i] =abs(2*rand());


}


void Compute_Without_MMX()
{
    _int64 i;
       __int32 temp1,temp2,temp3;
     QueryPerformanceCounter(&BeginTime);
       temp3=0xf;
       for(__int32 j=0;j<NumberOfRepeat;++j)
       for ( i = 0; i < ARRAY_SIZE;  )
       {
             temp1=InitialArray[i];
             temp2=InitialArray[i+1];
             *(__int32*)(&ResultArray[i])  = temp3*(temp1+temp2)  ;
             temp1=InitialArray[i+2];
             temp2=InitialArray[i+3];
```

```
            *(__int32*)(&ResultArray[i+2])= temp3*(temp1+temp2)  ;
            i+=4;
       }


    QueryPerformanceCounter(&EndTime);
    QueryPerformanceFrequency(&Freq);
    WithOutMMXTime = ((EndTime.QuadPart - BeginTime.QuadPart)
*1000000000)/Freq.QuadPart;
      printf("Calculation Time without MMX technology is=>
%2.3fus\n\n",(float)(WithOutMMXTime)/1000);
      return;
}


void Compute_With_MMX()
{
    __int16 * pIn = InitialArray;
    __int16 * pOut= ResultArray1;
    __int64 i = 0x000f000f000f000f;


    QueryPerformanceCounter(&BeginTime);
    _int32 nNumberOfLoops = ARRAY_SIZE / 4;
      for(__int32 j=0;j<NumberOfRepeat;++j)
      {
    __asm
    {
        emms
        mov         esi, pIn            // input pointer
        mov         edi, pOut              // output pointer
        mov         ecx, nNumberOfLoops     // loop counter
start_loop:
        movq        mm0, i                  // mm0 = 0x000f000f000f000f
        PMADDWD     mm0, [esi]
        movq        [edi], mm0              // [edi] = mm0
        add         esi, 8                  // increment input pointer
(next 4 integer Number)
        add         edi, 8                  // increment output
pointer(next 4 integer Number)
        dec         ecx
        jnz         start_loop


        emms
    }
      }
    QueryPerformanceCounter(&EndTime);
    QueryPerformanceFrequency(&Freq);
    WithMMXTime = ((EndTime.QuadPart - BeginTime.QuadPart)
*1000000000)/Freq.QuadPart;
      printf("Calculation Time with MMX technology is====>
%2.3fus\n\n",(float)(WithMMXTime)/1000);
      return;
}
void main()
{
```

```
/////////
unsigned int dwFeature = 0;


 // test processor support
    _asm
    {
        mov eax, 1
        cpuid
        mov dwFeature, edx      // this value defines support for MMX,
SSE ...
    }


    if (dwFeature & _MMX_FEATURE_BIT )  // processor supports MMX
          printf("MMX support Ok\n");
      else
      {
          printf("This CPU does'nt Support MMX \n");
          getch();
          return;
      }


    if (dwFeature & _SSE_FEATURE_BIT )  // processor supports SSE
          printf("SSE support Ok\n\n\n");
      else
      {
          printf("This CPU does'nt Support SSE \n");
          getch();
          return;
      }


      InitArray();
      printf("calculation over %ld integer
number\n\n",ARRAY_SIZE*NumberOfRepeat);
      Compute_Without_MMX();
      Compute_With_MMX();
      printf("Speed Up With MMX=%2.2f\nPress Any key
...",(float)WithOutMMXTime/WithMMXTime);
//////////////////////////////
getch();
}
```

12 ضمیمهٔ2: لیست برنامه SSE

```
/*
Example of using SSE technology and compute speed up
University Of Isfahan
Author:Ali Bohlooli
Advanced Microprocessor course
Istructor:Dr Vafaei
Bahar 1386
*/
#include<stdio.h>
#include<conio.h>
#include <float.h>
#include <math.h>
#include <windows.h>
#define _MMX_FEATURE_BIT        0x00800000      // bit 23
#define _SSE_FEATURE_BIT        0x02000000      // bit 25
#define ARRAY_SIZE 100000L
#define NumberOfRepeat 1L
// Arrays processed by SSE should have 16 bytes alignment:
__declspec(align(16)) float InitialArray[ARRAY_SIZE];
__declspec(align(16)) float ResultArray[ARRAY_SIZE];

LARGE_INTEGER Freq;
LARGE_INTEGER BeginTime;
LARGE_INTEGER EndTime;

_int64 WithSSETime;
_int64 WithOutSSETime;

void InitArray()
{
    float f;
    _int64 i;
// Fill array with one sin cycle and ensure that all values are positive
    // (to use sqrt in conversion)
    for ( i = 0; i < ARRAY_SIZE; i++ )
   InitialArray[i] = (float) sin(((double)i * 6.29 / ARRAY_SIZE)) + 2.0f;
}


void Compute_Without_SSE()
{
   _int64 i;
    QueryPerformanceCounter(&BeginTime);
 for(__int32 j=0;j<NumberOfRepeat;++j)
    for ( i = 0; i < ARRAY_SIZE; i++ )
        ResultArray[i] = sqrt(InitialArray[i]  * 2.8f);
    QueryPerformanceCounter(&EndTime);
    QueryPerformanceFrequency(&Freq);
    WithOutSSETime = (EndTime.QuadPart - BeginTime.QuadPart)
*1000000000/Freq.QuadPart;
      printf("Calculation Time without SSE technology is=>
%2.3fus\n\n",(float)(WithOutSSETime)/1000);
```

```
      return;


}
void Compute_With_SSE()
{
    // xmm2 - multiplication coefficient
    float f = 2.8f;
    float* pIn = InitialArray;
    float* pOut= ResultArray;
    _int64 CalcTime;
QueryPerformanceCounter(&BeginTime);
    _int32 nNumberOfLoops = ARRAY_SIZE / 4;
      for(__int32 j=0;j<NumberOfRepeat;++j)
      {
          _asm
    {
        movss   xmm2, f                         // xmm2[0] = 2.8
        shufps  xmm2, xmm2, 0                   // xmm2[1, 2, 3] = xmm2[0]
        mov         esi, pIn                      // input pointer
        mov         edi, pOut                     // output pointer
        mov         ecx, nNumberOfLoops             // loop counter
start_loop:
        movaps      xmm1, [esi]                   // xmm1 = [esi]
        mulps       xmm1, xmm2                    // xmm1 = xmm1 * xmm2
        sqrtps      xmm1, xmm1                    // xmm1 = sqrt(xmm1)
        movaps      [edi], xmm1                   // [edi] = xmm1
        add         esi, 16
        add         edi, 16
        dec         ecx
        jnz         start_loop
}
      }
QueryPerformanceCounter(&EndTime);
QueryPerformanceFrequency(&Freq);
    WithSSETime = (EndTime.QuadPart - BeginTime.QuadPart)
*1000000000/Freq.QuadPart;
      printf("Calculation Time with SSE technology is====>
%2.3fus\n\n",(float)(WithSSETime)/1000);
      return;
}
void main()
{
/////////
unsigned int dwFeature = 0;

 // test processor support
    _asm
    {
        mov eax, 1
        cpuid
        mov dwFeature, edx      // this value defines support for MMX,
SSE ...
    }
    if (dwFeature & _MMX_FEATURE_BIT )  // processor supports MMX
            printf("MMX support Ok\n");
      else
```

```
      {
            printf("This CPU does'nt Support MMX \n");
            getch();
            return;
      }
    if (dwFeature & _SSE_FEATURE_BIT )  // processor supports SSE
            printf("SSE support Ok\n\n\n");
      else
      {
            printf("This CPU does'nt Support SSE \n");
            getch();
            return;
      }
      InitArray();
      printf("calculation over %ld integer
number\n\n",ARRAY_SIZE*NumberOfRepeat);
      Compute_Without_SSE();
      Compute_With_SSE();
      printf("Speed Up With SSE=%2.2f\nPress Any key
...",(float)WithOutSSETime/WithSSETime);
////////////////////////////////
getchar();
}
```

13 مراجع

Internet Sites:
1-http://www.tommesani.com/Tutorial.html
2-http://www.x86.org/articles/articles.htm#sse_pt1
3-http://web.cs.wpi.edu/~matt/courses/cs563/talks/powwie/p3/mmx.htm#Introduction
Ebooks:
1-Pentium® Processor Family.pdf
2-Professional Assembly Language.pdf
3-The Art of Assembly Language.chm