

Maximilian Hütter

Altivec –  
Die SIMD-Einheit der PowerPC Prozessoren

**Studienarbeit**

an der

Universität Tübingen  
Fakultät für Informations- und Kognitionswissenschaften

Arbeitsbereich Technische Informatik

Tübingen 2005

1.	Was ist SIMD? .....	3
2.	SIMD - Erweiterungen in Mikroprozessoren .....	4
	Hewlett Packard MAX .....	4
	SPARC VIS .....	5
	Intel MMX .....	8
	AMD 3DNow! .....	9
	Intel Streaming SIMD Extension .....	10
	SSE2 .....	10
	SSE3 .....	11
	Übersicht der verschiedenen SIMD-Erweiterungen .....	12
3.	AltiVec – Die SIMD-Einheit der PowerPC Prozessoren .....	12
	Die AltiVec – Einheit und ihr Befehlssatz .....	12
	AltiVec in den verschiedenen PowerPC Prozessoren .....	16
	Überblick über den PowerPC 7400 .....	16
	PowerPC 7410 .....	18
	PowerPC 7450 .....	18
	PowerPC 970 .....	21
4.	Softwareunterstützung für AltiVec .....	24
	GNU Compiler Collection .....	26
	Apple Bibliotheken für Mac OS X .....	27
	Freescale Bibliothek für Linux .....	27
	Open Source Bibliotheken .....	28
5.	Zusammenfassung und Ausblick .....	29
6.	Literaturverzeichnis .....	30
	Anhang A AltiVec Befehle .....	32
	Anhang B VIS-Befehle .....	41
	Anhang C MMX-Befehle .....	45
	Anhang D AMD 3DNow! Befehle .....	48
	Anhang E SSE Befehle .....	51

# 1. Was ist SIMD?

Der Begriff SIMD für Single Instruction Stream, Multiple Data Streams stammt aus Michael Flynns Klassifikation von Rechnerarchitekturen.<sup>1</sup> Rechner die nach dem SIMD-Prinzip arbeiten, nutzen Datenparallelität um die Verarbeitung zu beschleunigen. Es gibt auch eine eigene Klasse von Rechnern die Datenparallelität nach dem SIMD-Prinzip schon lange nutzen, die Vektorrechner. Diese Vektorrechner werden auch oft als Supercomputer bezeichnet, da sie lange Zeit die leistungsfähigsten Computer überhaupt waren und vielfach noch sind. In der Top-500-Liste<sup>2</sup> der Supercomputer sind immer noch viele Vektorrechner vertreten. Vektorrechner sind meist auf wissenschaftliches Rechnen spezialisiert, wo mit Gleitkommazahlen doppelter Genauigkeit gearbeitet wird. Auf diese Rechner geht das Prinzip der SIMD-Verarbeitung zurück. In dieser Arbeit soll aber die Anwendung dieser Technik in Mikroprozessoren untersucht werden. Diese sind Single Instruction Stream Single Data Stream (SISD) Rechner. Die man auch als Skalare Rechner bezeichnet, da im Gegensatz zu den Vektorrechnern nur skalare Werte verarbeitet werden. Jeder Befehl verarbeitet bis zu drei Werte, etwa die Addition zweier Werte und einer Konstante. Mit SIMD wird mit einem Befehl eine größere Zahl Werte verarbeitet, so dass eine erheblich höhere Leistung als skalare oder superskalare Rechner erzielt werden kann.

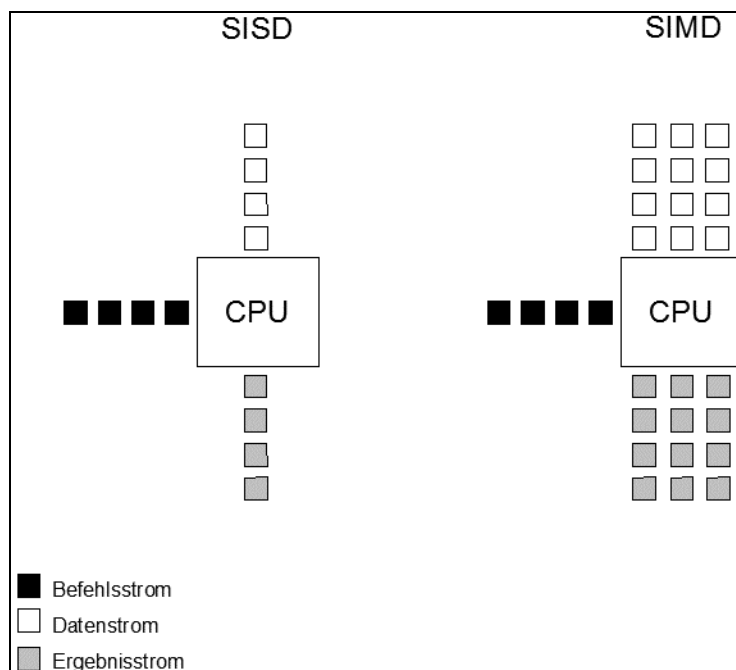


Abbildung 1 - SIMD-Prinzip

Superskalare Prozessoren sind zwar ebenfalls in der Lage im günstigen Fall mehrere Befehle gleichzeitig auszuführen und mehrere Ergebnisse zu liefern. Dies ist aber nur unter günstigen Voraussetzungen möglich. Außerdem ist bei superskalaren Architekturen großer zusätzlicher Aufwand nötig um die Abhängigkeiten der verschiedenen gleichzeitig auszuführenden Befehle zu kontrollieren. Selbst bei den VLIW-Prozessoren, die viel des Scheduling-Aufwandes an den Compiler abgeben, steigt die Komplexität und Größe des Prozessors an. Ein weiterer Vorteil der SIMD-Technik ist, dass auch weniger Adressrechnungen pro Ergebnis durchgeführt werden müssen, da immer ein ganzer Vektor geladen wird und auch

<sup>1</sup> Nach [16]

<sup>2</sup> [www.top500.org](http://www.top500.org) – Stand Mai 2005

wieder als ganzes in den Speicher zurück geschrieben wird. SIMD-Verarbeitung erfordert daher auch breite Datenpfade zwischen Speicher und Prozessor, damit die großen Datenmengen, die der Prozessor verarbeitet auch schnell genug geliefert werden können.

An einem einfachen Beispiel kann man schnell deutlich machen, wie SIMD-Technik die Verarbeitung beschleunigt. Wenn etwa ein (digitalisiertes) Bild im Computer aufgehellt werden soll, so muss für jeden Bildpunkt der Helligkeitswert angehoben werden. Das heißt die Helligkeit muss für jeden Bildpunkt um den gleichen Betrag erhöht werden. Der Befehl ist also für alle Bildpunkte gleich und ebenso der zu addierende Wert. Wenn man also ein Bild mit einer Auflösung von 320 horizontalen und 240 vertikalen Bildpunkten annimmt, bei dem jeder Bildpunkt einen 1 Byte großen Helligkeitswert besitzt, dann muss der Prozessor 76800 Byte verarbeiten. Es wird also eine Schleife für alle Bildpunkte abgearbeitet werden. Ohne SIMD-Technik muss also 76800 Mal ein Wert geladen, ein Skalar addiert und wieder gespeichert werden. Für das Laden des Wertes muss die Adresse berechnet werden und für das Speichern genauso. Jeder Wert erfordert als drei Additionen und einen Test ob die Abbruchbedingung der Schleife erfüllt ist, sowie den entsprechenden Sprung.

Ein Prozessor mit SIMD-Technik, der 64-Bit breite Register besitzt, kann dagegen in jeder Wiederholung der Schleife acht Werte auf einmal verarbeiten. Der Prozessor kann im idealen Fall für dieses Beispiel, die achtfache Leistung eines skalaren Prozessors erbringen. Dazu muss er natürlich die entsprechend breiten Datenpfade besitzen und die Daten müssen im Speicher richtig ausgerichtet sein, so dass das Laden und Speichern der 8 Werte genauso lange dauert wie beim skalaren Prozessor. Mit noch breiteren Registern lassen sich noch höhere Leistungssteigerungen erzielen. Die theoretisch möglichen Leistungssteigerungen durch SIMD-Verarbeitung sind also enorm. Es lassen sich aber nicht generell alle Algorithmen durch SIMD-Technik beschleunigen. Voraussetzung ist, dass sich die Daten auch auf diese Art verarbeiten lassen, das heißt es muss Datenparallelität vorliegen. Obwohl es viele Anwendungen gibt, die Datenparallelität aufweisen, wurde SIMD-Verarbeitung für die große Masse der Computer erst interessant, seitdem Multimediaanwendungen verbreiteter wurden. Unter Multimediaanwendungen versteht man Anwendungen, die Bilder, Musik, Videos, Animationen oder 3D-Grafik integrieren. Diese Anwendungen haben alle die Eigenschaft, vor allem auf Grund der großen Datenmengen die dabei verarbeitet werden müssen, sehr leistungshungrig zu sein. Gleichzeitig weisen Multimediadaten eine hohe Parallelität auf. Daher sind mittlerweile nahezu alle Mikroprozessorarchitekturen um SIMD-Befehle und -Einheiten erweitert worden, um die Verarbeitung digitaler Medien besser zu unterstützen. An den Bezeichnungen der Erweiterungen zeigt sich diese Zielsetzung auch deutlich.

## **2. SIMD - Erweiterungen in Mikroprozessoren**

### ***Hewlett Packard MAX***

Hewlett – Packard war die erste Firma, die 1994 ihre RISC – Prozessorreihe PA – RISC um einen SIMD – Befehlssatz erweiterte. Die Multimedia Acceleration Extension (MAX) genannte Erweiterung kam ohne zusätzliche Register und Funktionseinheiten aus. Es wurden 5 Befehle zur PA-RISC ISA<sup>3</sup> hinzugefügt. Diese nutzen die vorhandenen Register um mehrere 16-Bit Daten parallel zu bearbeiten. Die ersten PA-RISC Prozessoren mit MAX hatten noch 32-Bit Register, später wurden diese auf 64-Bit erweitert. Mit MAX-2 wurden außerdem 4 weitere SIMD-Befehle hinzugefügt. Weiterhin blieb die einzige Datenstruktur,

---

<sup>3</sup>Instruction Set Architecture

die so parallel verarbeitet werden konnte 16-Bit Integer Daten. Mit den 64-Bit breiten Register sind dann Operationen auf vier 16-Bit Elementen möglich.

Der MAX – Datentyp:

Datentyp	64-Bit Register Nutzung			
4 x 16-Bit Integer	16-Bit Integer	16-Bit Integer	16-Bit Integer	16-Bit Integer

Die MAX-1 Befehle:

Befehl	Beschreibung
HADD	Addition von 16-Bit Subwords, es gibt 3 Varianten des Befehls: <ul style="list-style-type: none"> <li>• Mit Modulo-Arithmetik: Ergebnisbereich von -32768 bis + 32767, wobei der Nachfolger von +32767 dann -32768 ist.</li> <li>• Mit vorzeichenbehafteter Sättigungsarithmetik: Bei einem Überlauf ist das Ergebnis die entsprechend kleinste bzw. größte vorzeichenbehaftete Zahl.</li> <li>• Mit Sättigungsarithmetik ohne Vorzeichen: Der Ergebnisbereich ist auf die nicht – negativen Zahlen beschränkt.</li> </ul>
HSUB	Subtraktionen von 16-Bit Subwords, mit den entsprechenden 3 Varianten wie bei HADD.
HSHLADD	Links verschieben und Addieren von 16-Bit Subwords, in 3 Varianten um 1, 2 oder 3 Bit.
HSHRADD	Rechts verschieben und Addieren von 16-Bit Subwords, ebenfalls in 3 Varianten um 1, 2 oder 3 Bit.
HAVG	Berechnung des Arithmetischen Mittels von 16-Bit Subwords

Die mit MAX-2 neu hinzugekommenen Befehle sind:

Befehl	Beschreibung
HSHR	Rechts verschieben von 16-Bit Subwords.
HSHL	Links verschieben von 16-Bit Subwords.
PERMH	Erzeugt eine Permutation der 16-Bit Subwords eines Registers. Dabei wird ein Permutationsindex verwendet, so dass es 256 Varianten des Befehls gibt (für jede mögliche Permutation).
MIXH / MIXW	Permutation von Subwords aus 2 Registern in ein neues Register. Bei jedem Befehl gibt es noch eine L und eine R Variante.

### Zusammenfassung:

Hewlett-Packards Erweiterung ist eine reine Befehlssatzerweiterung, die ohne neue Funktionseinheiten oder Register auskommen muss, die die Abarbeitung der Befehle zu beschleunigen. Die geringe Zahl der Befehle der PA RISC-Prozessoren blieb dem RISC-Anspruch treu, relativ wenige, sehr flexibel einsetzbare Befehle zu haben. Etwas problematisch ist vor allem die Beschränkung auf die 16-Bit Partitionierung der Register, die einem Einsatz in anderen Bereichen als der 3D-Grafik etwas behindern dürfte. Trotzdem bringen die Befehle für viele Anwendungen eine Geschwindigkeitssteigerung, vor allem da sie meist in nur einem Takt ausgeführt werden können.

## SPARC VIS

Sun Microsystems stellte 1995 mit der 64-Bit Prozessor UltraSPARC I ihre eigene SIMD-Erweiterung vor. Diese wurde Visual Instruction Set (VIS) getauft. Die SPARC V9 – ISA um entsprechende Befehle ergänzt. Bis auf ein neues Statusregister wurden keine neuen Register hinzugefügt. Mit der UltraSPARC – Architektur wurde aber die allgemeine Register Breite auf 64-Bit angehoben. Die bisherigen Gleitkommaregister wurden außerdem in ihrer Funktionalität dahingehend erweitert, dass sie jetzt die VIS – Integerdaten aufnehmen können

ohne eine Konvertierung durchführen zu müssen. Ähnlich dem Ansatz von Hewlett Packard wurde ein Partitionierungssystem in mehrere Elemente kleiner als 64-Bit gewählt. Das dann die parallele Verarbeitung dieser Elemente ermöglicht. Anders als bei der PA – RISC Architektur sind aber nicht nur 16-Bit Elemente möglich. Die Möglichkeit der Partitionierung in 8, 16 oder 32-Bit große Elemente ist deutlich flexibler als bei PA – RISC. Wenn auch die Register nicht vollkommen frei unterteilt werden können. Es sind aber wie bei der PA – RISC Architektur nur Integervariablen möglich, eine Unterteilung in mehrere Gleitkommazahlen ist nicht vorgesehen.

Die Abarbeitung der Befehle erledigt die Gleitkommaeinheit des Prozessors, die dafür um 2 Funktionseinheiten erweitert wurde. Der Name der Gleitkommaeinheit wurde auf Floating Point / Graphics Unit (FGU) geändert um die Erweiterung deutlich zu machen. Anders als aber bei der Intel SIMD – Erweiterung ist ein Mischen der von Gleitkomma und SIMD – Befehlen möglich.

Insgesamt sind 80 neue Befehle zum UltraSPARC Befehlssatz hinzugekommen, die sich aber noch einmal in zwei Versionen aufteilen. 1995 wurde VIS 1.0 eingeführt, das bereits den Großteil aller VIS – Befehle umfasst. Mit dem UltraSPARC III wurde 2000 VIS 2.0 eingeführt, das 3 neue Permutations- Befehle hinzufügte.

Die VIS– Befehle der UltraSPARC nennt Sun, beziehungsweise SPARC Inc. *Graphics Instructions*. Obwohl sie von der FGU ausgeführt werden, arbeiten die Befehle auf Integerdaten, die in Gleitkommaeinstellen gehalten werden. Das hat den Vorteil, dass die Integereinheiten für andere Berechnungen frei bleiben. Die Partitionierung der Register ist an den Erfordernissen der Bildverarbeitung ausgerichtet. Daher stehen nur bestimmte Partitionierung in 8- oder 16-Bit Daten zur Verfügung.

<b>Datentyp</b>	<b>64-Bit FP - Register Nutzung</b>			
8 x 8-Bit Pixel Data Format	4 x 8-Bit: Pixel Farbdaten α, G, B, R 63 - 32 Bit		4 x 8-Bit: Pixel Farbdaten α, G, B, R 31 - 0 Bit	
2 x 32-Bit Fixed32 Format	32Bit Fixed32 Format für Zwischenergebnisse 63 - 32 Bit		32Bit Fixed32 Format für Zwischenergebnisse	
2 x 16-Bit Fixed32 Single Format	Leer 63 - 32 Bit		16 Bit Integer 31 – 16 Bit	16 Bit Integer 15 - 0 Bit
4 x 16-Bit Fixed16 Format	16 Bit Integer 63 – 48 Bit	16 Bit Integer 47 - 32 Bit	16 Bit Integer 31 – 16 Bit	16 Bit Integer 15 - 0 Bit

Die Fixed Formate enthalten nur Integerwerte. Sie sollen eine ausreichende Genauigkeit für einfache Bildverarbeitung haben, wo sie zum Speichern von Zwischenergebnissen verwendet werden können. Die Multiplikationsbefehle sind insbesondere dazu da um von den Fixed Formaten wieder in das 32-Bit Pixel Data Format zu konvertieren.

Die arithmetischen Befehle erlauben, Addition, Subtraktion und Multiplikation von bis zu 4 Elementen mit einem Befehl. Die Addition und Subtraktion stehen für 16- und 32-Bit Partitionen bereit, die Multiplikation für 8- und 16-Bit Elemente. Wobei die Spezialisierung

der Partitionierung auf Bildbearbeitung die Nutzung für andere Anwendungen etwas erschweren dürfte.

Um mit SIMD-Programmierung einen hohen Datendurchsatz zu erreichen, sollten so wenige Lade-Operationen wie möglich nötig werden. Die Daten müssen also im Speicher schon so angeordnet sein, dass sie mit einer einzigen 64-Bit Lade-Befehl geholt werden können. Dazu gehört, dass die Daten die in ein 64-Bit Element geholt werden auch im Speicher nebeneinander angeordnet sind. Außerdem ist eine korrekte Ausrichtung auf die 8-Byte Grenze nötig. Da dies nicht immer der Fall ist, gibt es 3 Alignment – Befehle um die Daten korrekt auszurichten. Da die Load-Store-Einheit bisher nicht darauf ausgerichtet war 8- und 16-Bit Daten laden und speichern zu können, wurden neue Befehle zu Laden und Speichern der neuen Datenformate notwendig. Dazu wurden die beiden Load- und Store- Befehle LDDFA und STDFA um entsprechende Address Space Identifier (ASI) erweitert. Die neuen Load / Store ASI – Erweiterungen machen das Laden von entsprechenden 8-, 16-Bit Daten möglich. Die neue bedingte Store ASI – Erweiterungen erlauben ein Speichern von 8-, 16- und 32-Bit Daten, mit Maskierungsdaten die in einem weiteren Register abgelegt werden.

Neue Block-Load und -Store Befehle erlauben es größere Datenmengen schnell aus dem Speicher zu holen. Dabei kann Cache – Trashing durch besondere Befehle vermieden werden, die die Daten sofort als ungültig im Cache markieren. Die Load-Befehle laden die 64-Byte in acht Gleitkommaregister, deren Nummern in einem angegebenen Register abgelegt sind.

Mit der VIS-Erweiterung sollen vor allem 3D-Grafikoperationen wie Texturemapping und Volumenrendering unterstützt werden. Ein Problem was dabei auftritt ist, dass wenn nach Daten zu 3D-Koordinaten gesucht wird, eine kleine Änderung der z-Koordinaten einen Zugriff auf sehr weit entfernte Speicherstellen erzeugt. Das führt dann häufig zu Cache-Misses. Um das Problem zu umgehen, werden die Daten in Blöcken angeordnet, so dass Punkte mit nahe beieinander liegenden Koordinaten auch im Speicher nebeneinander liegen. Um die Umrechnung von den 3D-Koordinaten in die entsprechenden Speicherstellen zu beschleunigen, wurden daher entsprechende Befehle eingeführt.

Neue Vergleichsoperationen erlauben Vergleiche mit mehreren Werten auf einmal. Die Partitionierungen die verglichen werden können, sind vier 16-Bit oder zwei 32-Bit Elemente. Das 4-Bit oder 2-Bit Ergebnis des Vergleichs wird in einem Integerregister abgelegt. Ebenfalls zu den Vergleichsbefehlen, gehören die Edge-Erkennungsbefehle. Mit ihnen ist es möglich, einen Bildrand der innerhalb eines 64-Bit Blocks liegt zu erkennen. Für die Daten, die als nicht mehr zum Bild gehörend erkannt werden, wird eine Maske erzeugt. Mit Hilfe der Maske können diese Daten maskiert werden und werden dann auch nicht mehr verarbeitet. Die verschiedenen Befehle beziehen sich auf die verschiedenen möglichen Datenformate für Bilddaten, die in VIS vorgesehen sind. Also 8-Bit, 16-Bit und 32-Bit Daten in einem 64-Bit Block.

Ein sehr interessanter Befehl ist der Pixel Distanz Befehl (PDIST). Er erlaubt die Berechnung der Summe der absoluten Differenzen zwischen zwei acht 8-Bit Pixel Paaren, die in zwei 64-Bit Register stehen. Zusätzlich wird diese Summe zu dem bereits im Ergebnisregister vorhandenen Wert addiert. Dieser Befehl ist sehr nützlich für Videokompression, die meistens auf Bewegungsabschätzung (motion estimation) basiert. Dabei werden die Unterschiede zwischen zwei Gruppen von Bildpunkten berechnet, um zu erkennen ob sich dieselben Bildpunkte aus dem einen Bild im anderen Bild an anderer Stelle wieder finden. Der Befehl braucht nur einen Takt und kann so die Bewegungsabschätzung stark beschleunigen und damit den Vorgang der Videokompression.

Weiter bietet Visual Instruction Set auch noch eine ganze Palette Logik- Verknüpfungen. Logische Verknüpfungen können natürlich ohne weiteres auch mit den Standard-Integereinheiten, mit bereits vorhandenen Befehlen ausgeführt werden. Die Übernahme dieser Befehle in das VIS ist nur dazu da, viele Takte kostendes Kopieren der Daten zwischen den Gleitkomma- und Integerregistern zu vermeiden.

Mit VIS 2.0 wurde der Befehlsbestand dann vor allem um einen Permutationsbefehl ergänzt, der hier Byte-Shuffle genannt wird. Damit wird jede beliebige Permutation eines 8-Byte Sets aus einem der Gleitkommaregister möglich. Gesteuert wird die Permutation über eine 32-Bit Maske, die im Graphics Status Register mit dem ebenfalls neuen Befehl Byte Mask abgelegt wird.

**Zusammenfassung:**

Das Visual Instruction Set stellt mächtige SIMD-Befehle zur Verfügung, mit denen sich für einige Anwendungen große Geschwindigkeitsgewinne erzielen lassen. Als problematisch muss man dabei aber ansehen, dass die Befehle oft auf eine sehr spezielle Anwendung beschränkt sind. Es ist insgesamt sehr deutlich für die Beschleunigung der Verarbeitung von digitalen Medien entworfen worden. VIS ist in der Partitionierung der Register etwas flexibler als Hewlett-Packards MAX-2 erlaubt aber keine völlig freie Unterteilung.

**Intel MMX**

Intel hat schon Anfang der Neunziger Jahre einen RISC Prozessor vorgestellt, der auch schon SIMD-Befehle enthielt, den Intel 860. Dieser fand aber keine große Verbreitung. 1996 wurde von Intel die SIMD-Idee wieder aufgegriffen und der populäre Pentium Prozessor um SIMD-Befehle erweitert. Die Erweiterung bekam den Namen Multi Media eXtensions, kurz MMX. Mit MMX kamen 57 neue Befehle zur Intel IA32 Befehlsarchitektur hinzu. Als Register werden die bereits vorhandenen 8 Gleitkomma – Register verwendet, allerdings mit 64-Bit Breite statt der eigentlichen 80-Bit. Die restlichen Bits werden mit Einsen aufgefüllt, so dass sie als ungültige Gleitkommawerte erscheinen. Mit MMX können nur Integer-Elemente verarbeitet werden. Die Abbildung der MMX-Register auf die Gleitkommaregister erlaubte Kompatibilität mit bereits existierenden Betriebssystemen, da diese bei einem Prozesswechsel nicht die neuen Register sichern müssen. Allerdings ist es dadurch nicht möglich MMX- und Gleitkommabefehle zu mischen.

Intel sieht eine Partitionierung der Register folgende Datentypen vor:

Datentyp	MMX – Register Nutzung							
1x 64 Bit Integer Single Quad Word	64-Bit Single Quad Word Bits 63 – 0							
2x 32 Bit Integer Packed Double Word	32 Bit Double Word 63 - 32 Bit				32 Bit Double Word 31 - 0 Bit			
4x 16 Bit Integer Packed Word	16 Bit Word 63 – 48 Bit		16 Bit Word 47 - 32 Bit		16 Bit Word 31 – 16 Bit		16 Bit Word 15 - 0 Bit	
8x 8 Bit Integer Packed Byte	Byte 63 - 56 Bit	Byte 56 - 48 Bit	Byte 47 - 40 Bit	Byte 39 - 32 Bit	Byte 31 - 24 Bit	Byte 23 - 16 Bit	Byte 15 - 8 Bit	Byte 7 - 0 Bit

Die arithmetischen MMX-Befehle gibt es in drei Varianten, nach Art der Überlaufbehandlung. Bei den Wraparound-Befehlen wird ein Überlauf ignoriert und es werden nur die unteren 8, 16 oder 32 geschrieben. Bei den saturierenden Befehlen gibt es noch die Unterscheidung zwischen der Behandlung des Vorzeichens.

Die Konvertierungsbefehle, die Daten zwischen den Formaten wandeln gibt es ebenfalls in den Varianten mit und ohne Vorzeichenbeachtung. Weiter gibt es Verschiebeoperationen und Boolesche Verknüpfungen.

Um Daten in die MMX - Register laden zu können und aus diesen wieder in den Speicher zu schreiben, wurden auch noch 2 Transferbefehle hinzugefügt. Mit dem MOVD-Befehl ist es außerdem möglich zwischen den MMX- und den Standard-Registern Daten zu bewegen.

### **Zusammenfassung:**

MMX war zur Zeit ihrer Einführung die umfangreichste Erweiterung der x86 – Architektur seit dem 80386. Die einfache Lösung des Kompatibilitätsproblems, durch die Verwendung der Gleitkommaregister, stellt aber auch eine relativ große Einschränkung dar. Es muss immer zwischen beiden Modi gewechselt werden, wenn Gleitkomma und MMX-Befehle verwendet werden sollen. MMX wird außerdem durch alte Probleme des x86 – Befehlssatzes in seiner Nutzbarkeit eingeschränkt, da dieser immer nur mit zwei Adressen arbeitet. Einer der beiden Operanden ist immer auch gleich der Zieloperand und wird überschrieben.

MMX ansonsten relativ flexibel einsetzbar, die meisten Befehle sind nicht allzu spezialisiert. Auffällig ist aber das Fehlen von Permutationsbefehlen, die alle anderen Erweiterungen unterstützen.

### ***AMD 3DNow!***

Intels Mitbewerber bei den Prozessoren mit x86-Architektur hat mit dem K6-2 eine eigene SIMD-Erweiterung eingeführt. Diese Eigenentwicklung wurde mittlerweile eingestellt. Hier werden nur die Änderungen gegenüber MMX dargestellt, da AMDs 3DNow! auch alle MMX-Befehle umfasst und die gleichen Register benutzt.

3DNow! fügt einen neuen Datentyp zu MMX hinzu. Während MMX nur mit Integerwerten rechnet, kann 3DNow! 32-Bit Gleitkommawerte einfacher Genauigkeit verwenden. Der neue Datentyp Packed Floating Point entspricht dem IEEE-754 Standard für 32-Bit Gleitkommawerte einfacher Genauigkeit. Ein 3DNow!-Register (sonst identisch mit MMX-Registern) kann zwei solche Werte enthalten. 3DNow! umfasst MMX und fügt 21 neue Befehle hinzu. Mit dem Athlon Prozessor der 1999 erschien, ergänzte AMD den 3DNow!-Befehlssatz um weitere 24 Befehle, zu Advanced 3DNow!. Damit wurden vor allem die Integer-Fähigkeiten erweitert.

Das Ziel von 3DNow! war die Berechnungen von 3D-Grafik besser zu unterstützen, wie der Name es ja auch schon sagt. Die Möglichkeiten wurden gegenüber MMX deutlich verbessert, die Spezialisierung auf 3D-Grafik und Medienverarbeitung aber immer noch sehr deutlich. Anspruchsvollere Bild- und Tonverarbeitung wird aber durch die Gleitkommazahlen besser unterstützt, als durch MMX. Mit Advanced 3DNow! ging AMD diesen Weg weiter und nahm teilweise schon die SSE- Erweiterung von Intel vorneweg. Es folgte noch eine 3DNow!, die 3DNow! Professional getauft wurde. Sie umfasste aber nur noch die Übernahme der SSE Befehle und der 8 Register von Intel, um die Kompatibilität in diesem Bereich zu wahren. Seitdem hat AMD eine eigene Entwicklung von SIMD-Technik eingestellt und übernimmt nur noch die Erweiterungen von Intel, inklusive der Namen.

### **Zusammenfassung:**

Mit 3DNow! machte AMD noch vor Intel den wichtigen Schritt zur Gleitkommaverarbeitung. Es ist daher flexibler als MMX und steht ungefähr auf einer Stufe mit Intels SSE. Allerdings

ist es durch die Verwendung der Gleitkomma- bzw. MMX-Register beschränkt. Nach 3DNow! Professional gab AMD eigene Entwicklung von SIMD-Technik auf und beschränkte sich auf die Übernahme der SSE-Erweiterungen von Intel.

### **Intel Streaming SIMD Extension**

MMX erwies sich insbesondere für Beschleunigung von 3D-Grafik als unzureichend. Ein Grund dafür war die Beschränkung auf Integerwerte. Die 3DNow!-Erweiterung von AMD setzte ja auch genau an diesem Problem an, wenn auch mit begrenztem Erfolg, da nur wenige Software 3DNow! unterstützt.

1999 machte Intel einen neuen Anlauf für eine SIMD-Erweiterung der populären x86-Architektur. Bei der mit dem Pentium III eingeführten SSE standen Gleitkommaoperationen im Vordergrund. Außerdem wurden die Möglichkeiten der Cache-Kontrolle verbessert, etwas das AMD ebenfalls mit dem PREFETCH Befehl schon begonnen hatte. Mit SSE kamen weitere 72 Befehle und acht 128-Bit Register zur x86-Architektur hinzu, sowie ein neues Statusregister MXCSR. Durch die neuen Register und Interrupts wurde eine Unterstützung durch das Betriebssystem nun zwingend notwendig. Die neuen Register und Funktionseinheiten führten außerdem zu einer deutlichen Vergrößerung der Die-Fläche um etwa 10 Prozent. Später wurde SSE selbst noch mehrmals erweitert.

Die neuen 72 Befehle teilen sich auf in:

- 50 Befehle, die auf dem neuen Gleitkomma-Datentyp arbeiten.
- 8 Befehle das Verhalten des Cache kontrollieren.
- 12 Befehle, die MMX erweitern und auf den MMX-Registern arbeiten.

Das neue Statusregister MXCSR dient dazu Numerischen Exceptions zu behandeln, das Rundungsverhalten zu steuern, den Gleitkomma-Modus zu setzen und den Status abzufragen. Es gibt bei SSE 2 neue Datentypen, der eine ist ein Packed Floating-Point und besteht aus vier 32-Bit Gleitkommazahlen nach dem IEEE-754 Standard. Der zweite Datentyp ist ein 32-Bit Gleitkomma Skalarer Wert, ebenfalls nach dem IEEE-754 Standard.

<b>Datentyp</b>	<b>Register Nutzung</b>			
4 x 32-Bit Packed Floating Point	32-Bit Floating Point 127 – 96 Bit	32-Bit Floating Point 95 - 64 Bit	32-Bit Floating Point 63 – 32 Bit	32-Bit Floating Point 31 – 0 Bit
1 x 32-Bit Floating Point				32-Bit Floating Point 31 – 0 Bit

SSE kennt zwei Betriebsarten, den IEEE-Modus der Gleitkommaverarbeitung nach dem Standard leistet und den Flush-to-Zero-Modus. Der zweite Modus ist für Echtzeitanwendungen gedacht und erlaubt das maskieren von Exceptions um die Verarbeitung nicht aufzuhalten. Ein Unterlauf liefert im Flush-to-Zero-Modus einfach eine Null zurück.

### **SSE2**

Mit dem Pentium 4 Prozessor ergänzte Intel die SIMD-Befehle der x86-Architektur abermals. Während bei SSE-1 trotz des im Namen geführten „Streaming“ wenige Befehle zur Steuerung des Cacheverhaltens gab, wurden nun vor allem dazu neue Befehle hinzugefügt. Insgesamt gibt es 144 neue Befehle, der Großteil davon sind aber Integerbefehle für die SSE-Register. Vor allem wurden aber auch neue Datentypen ergänzt, so etwa der 128-Bit Integerdatentyp. Außerdem können jetzt auch 64-Bit Gleitkommazahlen doppelter Präzision verarbeitet

werden. Die Packed Byte, Word und Double Word können jetzt also auch jeweils in doppelter Menge SIMD-Befehlen verarbeitet werden.

Die SSE-Datentypen:

Datentyp	Register Nutzung															
1 x 128-Bit Double Quad Word	128 Bit Integer															
2 x 64-Bit Packed Quad Word	64-Bit Integer								64-Bit Integer							
2 x 64-Bit Packed Double Precision Floating Point	64-Bit Floating Point								64-Bit Floating Point							
4 x 32-Bit Packed Floating Point	32-Bit Floating Point				32-Bit Floating Point				32-Bit Floating Point				32-Bit Floating Point			
1 x 32-Bit Floating Point													32-Bit Floating Point			
4 x 32-Bit Packed Double Word	32-Bit Integer				32-Bit Integer				32-Bit Integer				32-Bit Integer			
8 x 16-Bit Packed Word	16-Bit Integer		16-Bit Integer		16-Bit Integer		16-Bit Integer		16-Bit Integer		16-Bit Integer		16-Bit Integer		16-Bit Integer	
16 x 8-Bit Packed Byte	8-Bit Int	8-Bit Int	8-Bit Int	8-Bit Int	8-Bit Int	8-Bit Int	8-Bit Int	8-Bit Int	8-Bit Int	8-Bit Int	8-Bit Int	8-Bit Int	8-Bit Int	8-Bit Int	8-Bit Int	8-Bit Int

### SSE3

Intel erweiterte 2003 den Pentium 4 Prozessor um ihre Hyper-Threading Technologie, die eine bessere Auslastung des Prozessorkerns ermöglichen soll. Im Zuge der Änderungen an dem Prozessor kamen noch einmal 13 neue Befehle hinzu, die der Synchronisation von zwei Threads dienen oder Videokompression beschleunigen sollen.

#### Zusammenfassung:

SSE ist sicher die umfangreichste Änderung am Befehlssatz der x86-Prozessoren seit langem gewesen. Mit SSE2 verfügen die x86-Prozessoren über eine sehr flexibel zu nutzende SIMD-Erweiterung. Die große Zahl von Datentypen und Befehlen ermöglichen es eine breite Palette von Programmen zu beschleunigen. Einzig die x86 typisch kleine Anzahl von nur 8 Registern, schränkt die Beschleunigungsmöglichkeiten wieder etwas ein.

## Übersicht der verschiedenen SIMD-Erweiterungen

	MAX	VIS	MMX	SSE 1-3	AltiVec
Register	-	-	-	8	32
Registerbreite	64-Bit	64-Bit	64-Bit	128-Bit	128-Bit
Befehle	9	83	57	229	162
Integer Datentypen	1	4	4	5	3
Gleitkomma Datentypen	0	0	0	2	1
Gleitkomma Genauigkeit	-	-	-	64-Bit	32-Bit

### 3. AltiVec – Die SIMD-Einheit der PowerPC Prozessoren

#### Die AltiVec – Einheit und ihr Befehlssatz

Apple kündigte 1999 den ersten PowerMac mit PowerPC 7400 als „Supercomputer“<sup>4</sup> an. Die wichtigste Veränderung an dem System war der neue Prozessor und dessen wichtigste Neuerung die Vektoreinheit. Diese wurde von Motorola und IBM gemeinsam entwickelt, aber bis 2004 nur von Motorola in PowerPC Prozessoren verwendet. IBM nennt die Einheit VMX<sup>5</sup> während Motorola die Einheit als AltiVec bezeichnet. Apple hat ebenfalls einen eigenen Namen dafür kreiert und bezeichnet sie als „Velocity Engine“. Im Folgenden wird als Name AltiVec verwendet, da diese Bezeichnung am gebräuchlichsten ist.

Die AltiVec-Einheit sticht durch einige Besonderheiten, gegenüber den anderen Vektorerweiterung die bereits behandelt wurden heraus. AltiVec hat zweiunddreißig 128-Bit breite Register, die zu den zweiunddreißig Integer- und den zweiunddreißig Gleitkommaregister hinzukamen. Die Vektoreinheit teilt sich nicht wie die anderen Erweiterungen Ausführungseinheiten mit den Integer oder Gleitkommabefehlen, sondern kann parallel zu den Integer- und Gleitkommaeinheiten arbeiten. Es stehen außerdem eine Vielzahl von neuen Vektordatentypen zur Verfügung:

Datentyp	AltiVec 128-Bit Register Nutzung															
4 x 32-Bit Single Precision Floating Point	32-Bit Floating Point				32-Bit Floating Point				32-Bit Floating Point				32-Bit Floating Point			
4 x 32-Bit Integer mit und ohne Vorzeichen (Word)	32-Bit Integer				32-Bit Integer				32-Bit Integer				32-Bit Integer			
4 x 32-Bit Pixel aus je 4 x 8-Bit Farbdaten (αRGB)	4 x 8-Bit Pixel				4 x 8-Bit Pixel				4 x 8-Bit Pixel				4 x 8-Bit Pixel			
8 x 16-Bit Integer mit und ohne Vorzeichen (Halfword)	16-Bit Integer		16-Bit Integer		16-Bit Integer		16-Bit Integer		16-Bit Integer		16-Bit Integer		16-Bit Integer		16-Bit Integer	
8 x 16-Bit Pixeldaten aus je 4 x 4-Bit Farbdaten (αRGB)	4 x 4-Bit Pixel		4 x 4-Bit Pixel		4 x 4-Bit Pixel		4 x 4-Bit Pixel		4 x 4-Bit Pixel		4 x 4-Bit Pixel		4 x 4-Bit Pixel		4 x 4-Bit Pixel	
16 x 8-Bit Integer mit und ohne Vorzeichen (Byte)	8-Bit Int	8-Bit Int	8-Bit Int	8-Bit Int	8-Bit Int	8-Bit Int	8-Bit Int	8-Bit Int	8-Bit Int	8-Bit Int	8-Bit Int	8-Bit Int	8-Bit Int	8-Bit Int	8-Bit Int	8-Bit Int

<sup>4</sup>Beier, Andreas, Grauer Riese – Apples 'Supercomputer' Power Macintosh G4, C't 19 / 1999, Seite 52

<sup>5</sup>Vector Multimedia eXtension

Die Vektoreinheit teilt sich nochmals in zwei Einheiten auf, die ebenfalls parallel arbeiten können. Eine Vektor-Permutationseinheit und eine Vektor-Arithmetisch-logische Einheit. Die Vektor-Arithmetisch-logische Einheit teilt sich in drei weitere Untereinheiten auf. Eine Vektor-Gleitkommaeinheit, die Gleitkommabefehle auf den vier 32-Bit Gleitkommazahlen ausführt. Sowie eine einfache Vektor-Integereinheit, die auch die Logik- und Vergleichsoperationen durchführt und eine komplexe Vektor-Integereinheit für Integer Multiplikationen.

Die Vektor-Gleitkommaeinheit kennt zwei Modi, der erste arbeitet mit der Genauigkeit des IEEE 754-Standards und entspricht damit der Java-Spezifikation. Der zweite ist weniger genau, aber schneller.

Bemerkenswert an AltiVec ist auch die Menge der neuen Befehle. Es kommen 162 neue Vektor-Befehle zum PowerPC-Befehlssatz hinzu, diese Menge täuscht aber. Tatsächlich sind die meisten Befehle nur Varianten eines Befehls, die sich nach den Elementen die der Vektor enthält, sowie bei der Behandlung des Vorzeichens der Vektorelemente oder des Überlaufs unterscheiden. Diese Informationen werden über den Opcode kodiert. Zum Beispiel der Befehl `vector add unsigned byte saturated` hat den Opcode `vaddubs`. Er addiert zwei Vektoren deren Elemente Bytes ohne Vorzeichen sind und das Ergebnis wird Saturiert, das heißt bei einem Überlauf wird das Ergebnis der jeweils kleinste oder größte Wert, den der Datentyp annehmen kann. Ebenso gibt es einer Variante `vector add unsigned byte modulo` mit dem Opcode `vaddubm`.

<b>Vektor 0</b>	330	22000	606	8006	321	1567	32000	12
	+	+	+	+	+	+	+	+
<b>Vektor 1</b>	10122	21334	5541	3710	970	7401	650	21
	=	=	=	=	=	=	=	=
<b>Ergebnis</b>	10452	32767	6147	11716	1291	8968	32767	33

**Abbildung 2: vaddubs Befehl**

Die AltiVec Befehle entsprechen den Konventionen des PowerPC-Befehlssatzes. Jeder Befehl ist 4 Byte lang und hat bis zu 4 Operanden, 2 oder 3 Quellvektoren und einen Ergebnisvektor. Die Befehle arbeiten nicht deskriptiv, das heißt dass die Quelloperanden erhalten bleiben und nicht überschrieben werden. In der Befehlstabelle im Anhang A ist jeder AltiVec Befehl kurz beschrieben, dazu wird angegeben wieviel Takte er im PowerPC7400 braucht und in welchem Teil der Vektoreinheit er ausführt wird.<sup>6</sup>

Die AltiVec Befehle lassen sich in sechs Gruppen aufteilen:

- Lade- und Speicherbefehle (Load / Store Instructions)
- Cache-Prefetchbefehle
- Permutations- und Konvertierungsbefehle
- Integer Arithmetikbefehle
- Gleitkomma Arithmetikbefehle
- Vergleichs- und Logikbefehle

<sup>6</sup> Die Tabelle ist zitiert nach der Vector Instruction Cross-Reference unter: [http://developer.apple.com/hardware/ve/instruction\\_crossref.html](http://developer.apple.com/hardware/ve/instruction_crossref.html)

### **Lade- und Speicherbefehle (Load / Store Instructions)**

AltiVec folgt dem RISC-Schema der expliziten Lade- und Speicherbefehle. Diese benutzen ein System von indizierten Adressen, wobei die Addressoperanden in den Standard Registern stehen. Es könne komplette Vektoren und einzelne skalare Elemente (8-, 16- und 32-Bit) eines Vektors aus dem Speicher in die Vektorregister oder umgekehrt bewegt werden.

Wenn ganze Vektoren geladen werden, machen die Befehle keine Annahmen über den Inhalt, so dass damit auch sehr schnell große Datenmengen transportiert werden können.

Das Problem was sich beim Laden von Daten ergibt ist, dass diese nicht immer korrekt ausgerichtet im Speicher liegen. Da nur ganze Vektoren geladen werden können, muss man eventuell zwei ganze Vektoren laden und dann mit dem Vector Permute (vperm) Befehl ausrichten. Um dies zu unterstützen gibt es die Vector Load Shift Befehle (lvsl, lvsr), die aus der Adresse eines nicht ausgerichtete Vektors eine Maske für Vector Permute generieren. So können Daten schnell ausgerichtet werden.

### **Cache-Prefetchbefehle (Prefetch Instructions)**

Die Verarbeitungsgeschwindigkeit der AltiVec-Einheit ist sehr hoch. Um effektiv arbeiten zu können, ist es wichtig auch schnell genug an die Daten zu kommen. Ein Cache Miss würde viele Takte verbrauchen und sollte daher unbedingt vermieden werden. Mit Hilfe der Prefetch Befehle können Daten, bereits bevor der Prozessor sie verarbeitet in den Cache geholt werden. Die Ladebefehle können so mit anderen Befehlen überlappen und parallel arbeiten. Die vielen Takte die ein Ladebefehl braucht, werden dann genutzt ohne die Verarbeitung aufzuhalten. Die Prefetchbefehle von AltiVec erlauben dazu, Datenströme (Streams) zu definieren. Ein Datenstrom im Sinn der AltiVec-Einheit ist eine Reihe von 128-Bit Datenblöcken im Speicher. Weiter wird ein Datenstrom durch folgende Eigenschaften gekennzeichnet:

- Die Speicheradresse ab der der Datenstrom beginnt.
- Die Blockgröße, die Anzahl der 128-Bit Daten in einem Datenblock. 1 bis 32
- Der Zähler, gibt die Anzahl der Datenblöcke im Strom an. 1 bis 256
- Der Stride, die Anzahl Bytes zwischen der Adresse des eines Blocks und der Adresse des folgenden. -32768 bis 32767, ein Stride von 0 ist gleich 32768. Ein Stride kleiner als 16 Byte (ein Datenblock) ist nicht zulässig.

Der Data Stream Touch Befehle dst legt einen neuen Strom an, mit dstt wird der Strom als ein Durchgangsstrom gekennzeichnet. Das bietet sich für Daten an, die nur ein Mal gebraucht werden und dann wieder aus dem Cache gelöscht werden können. Mit dem Data Stream Store Befehl, dsts kann man einen Puffer im Cache anlegen, für Daten die später geschrieben werden sollen. Die PowerPC der 74xx – Reihe können 4 Datenströme gleichzeitig verwalten, im PowerPC 970 sind sogar 8 Ströme möglich. Die Prefetch Befehle sind eigentlich keine Prozessorbefehle, sondern müssen von der Speicherverwaltungseinheit bearbeitet werden. Diese bearbeitet die Befehle, wenn es möglich ist. Es kann auch sein, dass diese Befehle komplett ignoriert werden, wenn das Speicher- und Cachesystem diese nicht unterstützt.

### **Permutations- und Konvertierungsbefehle**

Der wichtigste Befehl aus dieser Gruppe ist der Permutationsbefehl vperm. Dieser Befehl nimmt 3 Vektoren als Eingabe, zwei sind die Operanden und er dritte steuert die Permutation. Der Ergebnisvektor wird dann als eine beliebige Anordnung von Elementen aus den zwei Eingabevektoren erzeugt.

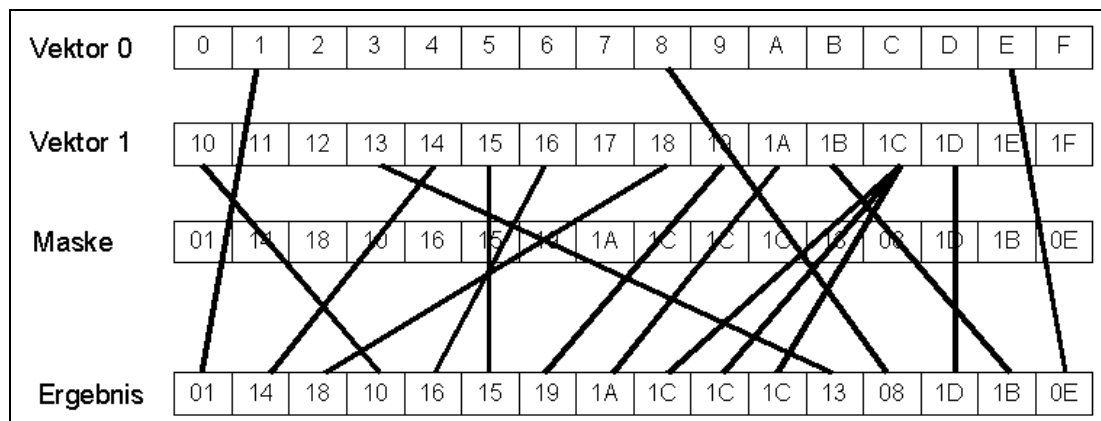


Abbildung 3: vperm Befehl

Das häufige Problem bei SIMD-Verarbeitung, dass die Organisation der Daten im Speicher nicht der vom Prozessor geforderten entspricht lässt sich damit behandeln. Durch die Permutation könne die Daten schnell entsprechend angeordnet werden, um sie schnell zu verarbeiten. Zusätzlich gibt es noch Operationen für häufige Spezialfälle der Permutation, wie den Befehl Vector merge (vmrg) das die Elemente zweier Vektoren alternierend mischt. Der Vector Splat (vsplt) vervielfältigt ein Element eines Vektors zu einem ganzen Vektor.

Die Vector Pack und Unpack Befehle vpk, vpuk konvertieren die Elemente von Vektoren in die verschiedenen Datentypen. Die interessantesten Varianten dieser Befehle sind die Vector Pixel Pack und Unpack (vpkpx, vupkpx) die zwischen den Pixeldatentypen konvertieren. Damit kann man 16-Bit Pixeldaten (jeweils 5-Bit für die Farbdaten plus 1 Bit für den Alphakanal) zu 32-Bit Pixeldaten (8-Bit Farbdaten und 8-Bit Alphakanal) konvertieren und umgekehrt. Wenn von größeren Datentypen in kleinere konvertiert wird, dann werden Elemente entsprechend saturiert.

### Integer Arithmetische Befehle

Die meisten Integer arithmetischen Befehlen arbeiten mit zwei Vektoren und erzeugen einen Ergebnisvektor. Einige Befehle arbeiten auch nur auf den Elementen eines Vektors. Neben den add, subtract und multiply Befehlen gibt es auch speziellere, die besonders auf Medienverarbeitung ausgerichtet sind. Der Vector Average vavg Befehl bildet das arithmetische Mittel der Elemente zweier Vektoren. Dies wird häufig bei MPEG-Dekodierung gebraucht. Die Vector Maximum (vmax) und Vector Minimum (vmin) erlauben es schnell den größte beziehungsweise kleinste Element eines Vektors zu finden. Damit können Filter- und Suchalgorithmen beschleunigt werden.

Viele Algorithmen arbeiten intensiv mit Matrizen, etwa bei 3D-Grafik. Matrix-Operationen kann Altivec stark beschleunigen, nicht nur weil es Vektoren verwendet. Viele Befehle sind wohl genau dafür gedacht. Vector Merge kann das Transponieren von Matrizen beschleunigen. Auch der Befehl Vector Multiply-Sum (vmsum), der zwei Vektoren multipliziert und dann zum Ergebnis einen dritten Vektor addiert und der Befehl Vector Sum (vsum) der alle Elemente eines Vektors addiert, und zu dieser Summe noch ein Element eines anderen Vektors addiert. Führt man die beiden Befehle nacheinander aus, kann man sehr schnell das Skalarprodukt zweier Vektoren berechnen. Es gibt auch Befehle die ein Konvertieren der Ergebnisse gleich miterledigen, wie Vector Multiply-Odd (vmulo), der nur die ungraden Elemente zweier Vektoren multipliziert und zu das Ergebnis als den entsprechend nächst größeren Datentyp ablegt. Als Gegenstück zu vmulo gibt es auch Vector Multiply-Even (vmule), für die graden Elemente der Vektoren. Mit den beiden Befehlen zusammen ausgeführt, können schnell Vektoren multipliziert werden, ohne auf Genauigkeit zu verzichten.

Ein Divisionsbefehl fehlt allerdings, hier hat sich Motorola gegen die große Menge zusätzlicher Hardware die ein solcher Befehl brauchen würde entschieden. Bemerkenswert

ist, das die meisten der Integer Befehle mit nur 2 Takten auskommen. Selbst die Multiplikationen benötigen nur 3 Takte.

### **Gleitkomma Arithmetische Befehle**

Der PowerPC 7400 verfügt bereits über eine skalare Gleitkommaeinheit mit 64-Bit Genauigkeit. Die Vektoreinheit arbeitet wie oben beschrieben nur mit 32-Bit Gleitkommawerten einfacher Genauigkeit. Es gibt auch hier add, subtract und multiply-add (vmuladdfp). Es gibt keinen eigenen Multiplikationsbefehl, stattdessen muss man vmuladdfp verwenden und den dritten Vektor der addiert wird auf Null setzen. Weiter gibt es Befehle um Ergebnisse zu Runden, Abschätzungen zu machen und zwischen Gleitkomma- und Integerwerten zu konvertieren. Interessant ist auch hier, dass es keinen Divisionsbefehl gibt. Es gibt allerdings den Befehl Vector Reciprocal (vrefp) um die Kehrwerte der Elemente eines Vektors zu berechnen, der dann mit einem multiply-Befehl für Divisionen benutzt werden kann, die Genauigkeit leidet dabei zwar, aber dafür ist dieses Vorgehen sehr schnell (1 Takt für vrefp und 3 Takte vmuladdfp). Zur Wurzelberechnung wird ein ähnliches Vorgehen verwendet, dazu gibt es den Befehl Vector Square Root Estimate (vrsqrtefp), der den Kehrwert der Wurzel der Elemente eines Vektors berechnet. Die Genauigkeit ist auch bei diesem Befehl deutlich schlechter, aber für schnellen Code wohl ausreichend. Für bessere Genauigkeit, müssen die Programmierer dann auf numerische Algorithmen zurückgreifen.

### **Boolesche- und Vergleichsbefehle**

AltiVec ermöglicht auch die üblichen logischen Verknüpfungen und Vergleiche auf Vektoren. Dazu stehen für die Standard booleschen Operationen bereit, wie AND, OR, XOR und NAND. Als Vergleichsoperationen gibt es für Integerwerte nur Tests auf Gleichheit oder Größer-als. Für Gleitkommawerte werden die Vergleichsoperationen Gleich, Größer-als und Größer-oder-Gleich unterstützt. Alle anderen Operationen müssen aus diesen zusammengesetzt werden. Ein Teil der Vektor – Vergleichsbefehle setzt das Standard PowerPC Condition Code Register, so daß die Standard Sprungbefehle benutzt werden können, um den Programmfluß zu kontrollieren. Um aber Sprünge möglichst zu vermeiden, gibt es Vergleichsbefehle die eine Vektormaske aus Null oder Eins erzeugen, Eins für Wahr und Null für Falsch. Diese Vektormaske kann dann zusammen mit dem Vector Conditional Select (vsel) Befehl benutzt werden, um zwischen den Elementen zweier Quelloperanden auszuwählen. Der Vector Select Befehl benutzt genau solche Vektormasken mit Nullen und Einsen um entweder das Element aus dem ersten oder zweiten Quellvektor zu kopieren. Diese Art der Auswertung der Vergleichsbefehle kann für eine bedingte Ausführung von Befehlen, ohne Sprünge zu verwenden genutzt werden. Es kann aber auch für Videoanwendungen genutzt werden, etwa wenn zwei Videobilder miteinander gemischt werden. Ebenso bei 3D-Grafik, wenn ermittelt werden muss welche Objekte von anderen verdeckt werden.

## ***AltiVec in den verschiedenen PowerPC Prozessoren***

### **Überblick über den PowerPC 7400**

Der 1999 vorgestellte PowerPC 7400 war der erste Prozessor mit AltiVec-Einheit auf dem Markt. Es handelt sich dabei um einen superskalaren Prozessor, mit einem 32-Bit Speichermodell. Er kann 4 Petabyte virtuellen und 4 Gigabyte realen Speicher adressieren. Weiter unterstützt er zwei Stufen Cache – Speicher. Der Level 1 Cache ist 64 Kilobyte groß, mit 32 Kilobyte Befehls-cache und 32 Kilobyte Datencache. Als Level 2 Cache sind drei verschiedenen Größen von 512 Kilobyte, 1 oder 2 Megabyte möglich. Der Level 2 Cache ist zwei-Wege set-assoziativ. Multiprozessorbetrieb wird durch die Fähigkeit des Level 2 Cache

den Bus abzuhören unterstützt. Außerdem kann der Level 2 Cache mehrere Anfragen gleichzeitig bearbeiten.

Die Pipeline des PowerPC 7400 hat vier Stufen:

1. Holen – Fetch
2. Dekodieren – Decode / Dispatch
3. Ausführen – Execution
4. Fertigstellen – Completion

Das superskalare Design des PowerPC 7400 erlaubt es, gleichzeitig bis zu 4 Befehle zu holen und bis zu 3 an die Ausführungseinheiten absenden. Allerdings können nur 2 der Befehle an die Ausführungseinheiten abgesendet werden. Der dritte Befehl kann immer nur ein Sprungbefehl sein. In einem Takt können sich bis zu 8 Befehle gleichzeitig in Ausführung befinden und 2 Befehle können pro Takt fertig werden. Die verschiedenen Ausführungseinheiten haben unterschiedliche Latenzen. Die meisten Befehle benötigen aber nur einen Takt.

Insgesamt 8 Ausführungseinheiten ermöglichen diese sehr hohe Parallelität bei der Befehlsausführung:

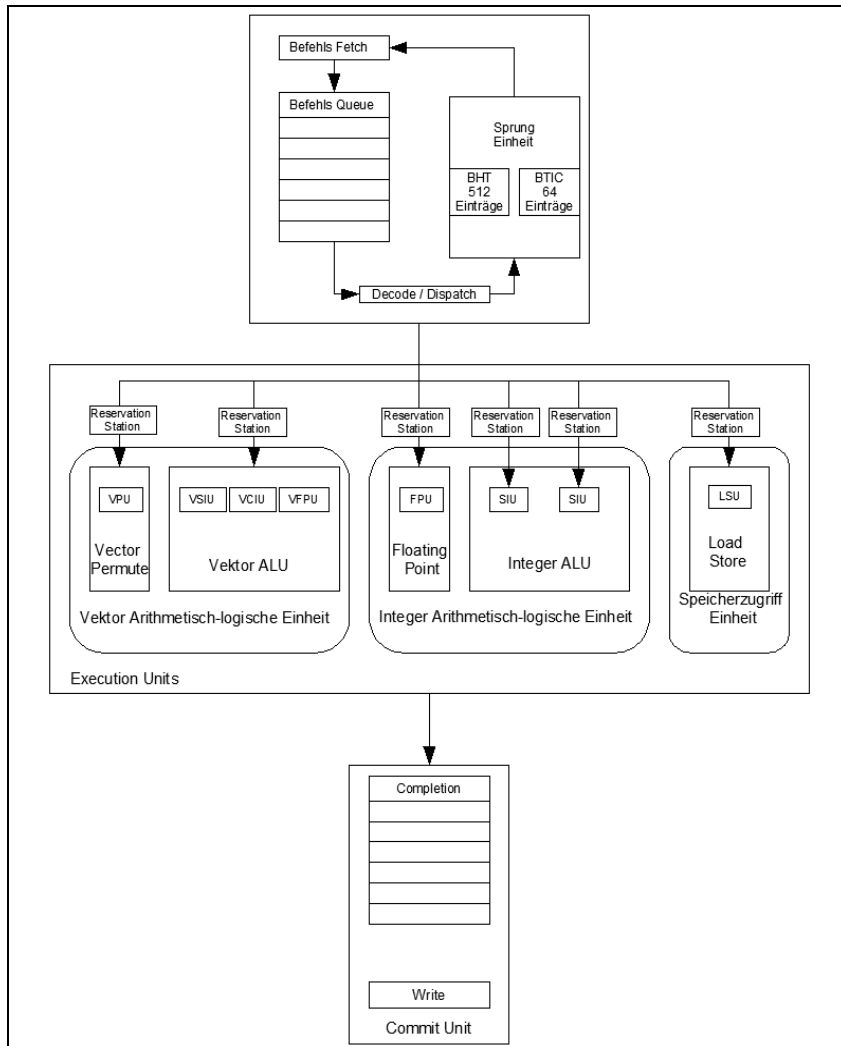
- 2 Integereinheiten
  - Eine Einheit für alle Integer – Befehle
  - Eine Einheit für alle Integer – Befehle außer Multiplikation und Division.
- 1 Gleitkommaeinheit doppelter Genauigkeit
- 2 Vektorausführungseinheiten
  - Eine Vektor – Permutationseinheit
  - Eine Vektor – Arithmetisch-logische Einheit, die sich aufteilt in drei Untereinheiten:
    - Eine Einfache Vektor – Integereinheit
    - Eine komplexe Vektor – Integereinheit
    - Eine Vektor – Gleitkommaeinheit.
- Speicherzugriffseinheit
- Sprungseinheit
- Systemeinheit

Out-of-Order-Ausführung von Befehlen wird durch jeweils 6 Rename Register für die Integer-, Gleitkomma- und Vektoreinheiten unterstützt. Reservation Stations an den jeweiligen Einheiten helfen ebenfalls Pipeline Stalls zu vermeiden, wobei nicht jede Ausführungseinheit ihre eigene Reservation Station besitzt.

Die Sprungseinheit erhält ihre Befehle unabhängig von denen anderen Ausführungseinheiten. Sie ist in der Lage unbedingte Sprünge und Sprünge bei denen die Bedingung schon entschieden ist, sofort auszuführen. Für alle anderen Sprünge ist der Prozessor in der Lage, eine Vorhersage zu machen um Verzögerung zu vermeiden. Dazu wird entweder eine statische oder eine dynamische Methode verwendet. Wenn ein Sprungbefehl keinen Eintrag im Branch History Table (BHT) hat wird die statische Methode verwendet. Dabei wird ein Sprung zurück im Befehlsstrom als genommen und Sprünge vorwärts als nicht genommen vorhergesagt.

Die dynamische Methode benutzt einen der 512 Einträge im BHT, jeder Eintrag kodiert mit 2 Bit eine Vorhersage für einen Sprung. Die 2 Bit kodieren 4 verschiedene Vorhersagen:

- sicher genommen
- genommen
- nicht genommen
- sicher nicht genommen



**Abbildung 4: Aufbau des PowerPC 7400**

Der PowerPC 7400 führt spekulativ Befehle, aus einem noch nicht entschiedenen Sprung aus. Diese Befehle könne allerdings nicht fertig gestellt werden. Ein zweiter Sprung, der nach einem noch nicht entschiedenen kommt kann ebenfalls schon spekulativ Befehle laden, allerdings werden diese Befehle noch nicht ausgeführt. Um die Zahl von, mit einem falsch vorhergesagten Sprung verschwendeten Takten gering zu halten, werden in einem Branch Target Instruction Cache (BTIC) schon geholte Befehle an Sprungzielen gehalten. Der BTIC hat 64 Einträge, die dann direkt ausgeführt werden können und nicht aus dem Befehls-cache geholt werden müssen, was einen Takt spart.

### PowerPC 7410

Im Jahr 2000 folgte der PowerPC 7410. Die wichtigste Änderung des 7410 gegenüber dem 7400 ist der verringerte Stromverbrauch. Intern wurde lediglich die Completion Queue von 6 auf 8 Stufen verlängert.

### PowerPC 7450

Der PowerPC 7450 wurde 2001 vorgestellt und brachte eine Reihe von Änderungen, auch die Altivec-Einheit wurde verändert. Vor allem wurde die Pipeline verlängert, so dass höhere Taktraten mögliche wurden.

Die Pipeline des PowerPC 7400 hat nur vier Stufen: Holen, Dekodieren, Ausführen, Fertigstellen.

Die des PowerPC 7450 hat dagegen 7 kürzere Stufen:

1. Fetch-1
2. Fetch-2
3. Decode / Dispatch
4. Issue
5. Execute
6. Complete
7. Write-back (Commit)

Die Befehls-Queue wurde ebenfalls verlängert und zwar verdoppelt 6 auf 12 Befehle. Ebenso die Completion-Queue auf 16 Einträge. Die Rate mit der die Befehle an die Ausführungseinheiten abgesendet werden erhöhte sich ebenfalls von 2 auf 3 Befehle (zusätzlich ist jeweils ein Befehl an die Sprungereinheit möglich).

Die Einführung der Issue-Queue als zusätzlicher Buffer hilft im PowerPC 7450 Pipeline-Stalls zu vermeiden. Während im PowerPC 7400, wenn keine Ausführungseinheit (oder ihre Reservation Station) frei war es zu einem Stall kam, kann der Befehl im PowerPC 7450 in den neu eingeführten Issue - Queue gepuffert werden.

Die Allgemeine Issue Queue (General Issue Queue) steht zwischen dem Decode / Dispatch und Integer-ALU und ihren Reservation Stations. Sie kann insgesamt 6 Befehle aufnehmen und pro Takt neu 3 neue. Gleichzeitig können pro Takt die 3 Befehle am Ende der Queue an die Ausführungseinheiten abgeschickt werden.

Die Vektor Issue Queue (Vector Issue Queue) kann insgesamt 4 Befehle aufnehmen und pro Takt 2 neue. Jeden Takt können die 2 untersten in der Queue an die AltiVec – Einheit abgeschickt werden. Während die Allgemeine Issue Queue allerdings bei abschicken die Reihenfolge der Befehle nicht einhalten muss, ist dies bei der Vektor Issue Queue nicht möglich. Hier müssen die Befehle in der richtigen Reihenfolge abgeschickt werden. Die Gleitkomma Issue Queue (Floating Point Issue Queue) kann nur einen Befehl aufnehmen.

Wichtig ist auch, dass der PowerPC 7450 mehr Ausführungseinheiten als sein Vorgänger besitzt. So sind 2 einfache Integereinheiten hinzugekommen, so dass insgesamt 4 Integereinheiten zur Verfügung stehen. Eine für komplexe und langsame Integerbefehle und 3 für einfache und schnelle Befehle. Die komplexe Integereinheit hat eine Reservation Station mit 2 Einträgen und alle Einheiten haben jetzt eigene Reservation Stations. Die Zahl der Rename – Register für die Integer-, Gleitkomma- und Vektoreinheit wurde ebenfalls auf 16 erhöht. Allerdings benötigen die meisten Befehle im PowerPC 7450 gegenüber dem 7400 mehr Takte.

Viele Vergleichsbefehle, die im 7400 noch von der Vektor Simple Unit ausgeführt wurden, sind im 7450 in die Vektor Floating Point Unit gewandert.

Der PowerPC 7450 stellt wohl das Ende von Motorolas bzw. Freescales PowerPC Linie für Desktopcomputer dar. Es wurden noch eine Reihe von Varianten des PowerPC 7450 entwickelt, bei denen sich aber an der Architektur nichts wesentliches mehr geändert hat. Lediglich der Level 2 Cache wurde vergrößert oder es kam Unterstützung für einen Level 3 Cache hinzu.

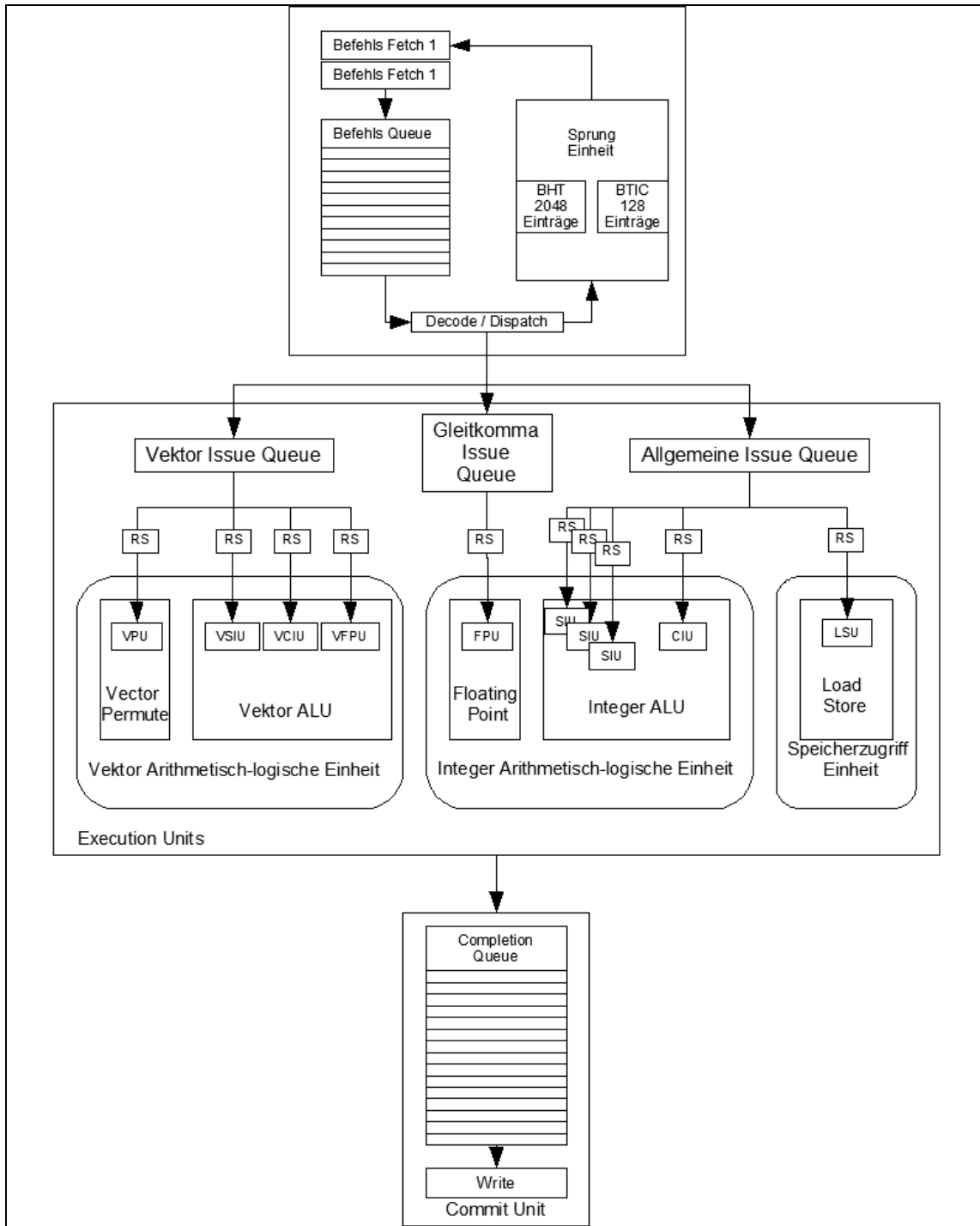


Abbildung 5: Aufbau des PowerPC 7450

Mikroprozessor Feature	PowerPC 7400	PowerPC 7450
<b>Grundlegende Pipeline Funktionen</b>		
Minimum Pipeline – Länge	4	7
Minimum Pipeline – Stufen bis zum Ausführen	3	5
Maximaler Durchsatz der Pipeline	2 + 1 Sprung	3 + 1 Sprung
<b>Pipeline Ressourcen</b>		
Größe des Befehls - Queues	6	12
Größe des Completion - Queues	8	16
Anzahl der Rename - Register	6	16

<b>Sprung - Vorhersage Ressourcen</b>		
Struktur	BHT, BTIC	BHT, BTIC, Link stack
Branch History Table	512	2048
Branch Target Instruction Cache Größe	64	128
Unentschiedene Sprünge	2	3
Sprung genommen (BTIC Hit) – Takte	0	1
Minimum Verlust - Takte für eine falsche Vorhersage	4	6
<b>Typische Ausführungszeiten</b>		
Daten Cache Hit (Integer, Vektor, Gleitkomma)	2,2,2	3,3,4
Einfache Integereinheit Befehle	1	1
Komplexe Integereinheit – Multiplikation	6	4
Komplexe Integereinheit – Division	19	23
Gleitkommaeinheit: Single Precision - Add, Multiply	3	5
Gleitkommaeinheit: Single Precision - Div	17	21
Gleitkommaeinheit: Double Precision – Add, Multiply	3	5
Gleitkommaeinheit: Double Precision – Div	31	35

## PowerPC 970

Der PowerPC 970 wurde nicht von Motorola bzw. Freescale entwickelt, sondern von IBM. Als erste PowerPC Prozessor von IBM enthält er eine AltiVec-kompatible Vektoreinheit. Der PowerPC ist im Gegensatz zu den anderen PowerPC der 74xx – Reihe, ein 64-Bit Prozessor und kann 42-Bit realen Adressraum nutzen. Die Register sind alle 64-Bit breit. Auch sonst unterscheidet sich der Aufbau des PowerPC 970 stark den 32-Bit PowerPC der 74xx – Reihe.

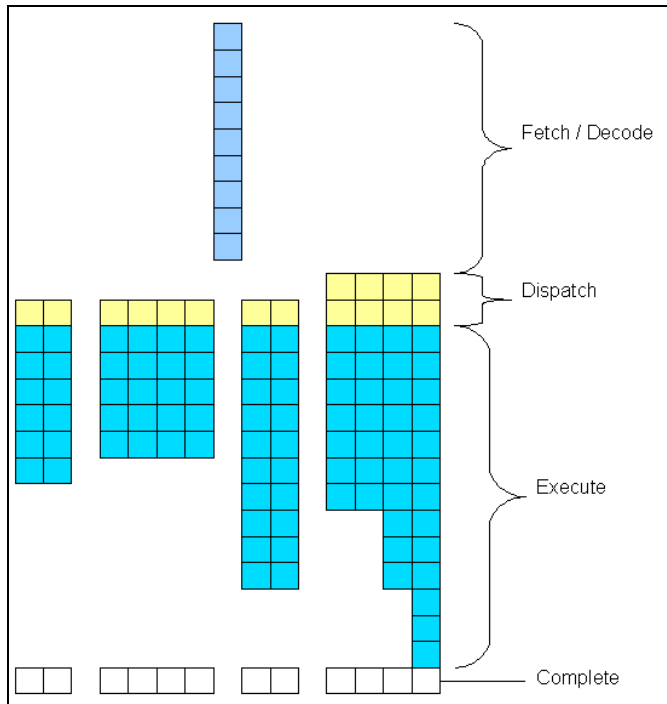
Die Pipeline ist sehr viel länger als die des PowerPC 7450. Die Zahl der Stufen variiert je nach Befehl.

Die Pipeline stellt sich dann so dar:

- 9 Pipeline Stufen - Holen und Dekodieren (Fetch and Decode)
- 2 bis 3 Stufen – Gruppieren und Abschicken (Group and Dispatch)
- 5 bis 13 Stufen – Ausführung (Execution)
- 2 bis 3 Stufen – Fertigstellen (Completion)

Jeden Takt werden acht Befehle aus dem Level 1 Cache geholt und in einen Instruction Fetch Buffer gelegt. Der Instruction Fetch Buffer hat 32 Einträge und ersetzt die bisherigen Befehls Queues. Aus dem Instruction Fetch Buffer werden dann jeweils 5 Befehle geholt und gruppiert. Wobei nur 4 Befehle tatsächlich an die Ausführungseinheiten gehen und der fünfte Befehl nur ein Sprungbefehl sein kann. Der PowerPC 970 teilt dann Befehle auf in kleinere Teile, sogenannte IOPs. Viele PowerPC Befehle entsprechen aber genau einer IOP, nur wenige werden in mehrere IOPs aufgeteilt. Für die aufgeteilten Befehle gibt es noch die Unterscheidung in cracked und microcoded.

- Ein cracked Befehl wird zu 2 IOPs
- Ein microcoded zu 5 IOPs und belegt damit eine ganze Gruppe.



**Abbildung 6: Die Pipeline des PowerPC 970, zitiert nach [23]**

**Jede Spalte steht für eine Einheit des PowerPC 970. Von Links nach Rechts: Load/Store1, Load/Store2, IntegerALU1, IntegerALU2, Branch Unit, Condition Register Unit, Floating Point ALU1, FloatingPointALU 2, Vector Permute Unit, Vector Simple Integer Unit, Vector Complex Integer, Vector FP**

Es gibt noch weitere Beschränkungen wie die IOP-Gruppen gefüllt werden können. Ein Beispiel soll dies verdeutlichen:

Eine Befehlskette beginnt mit zwei normalen Befehlen, gefolgt von einem Sprungbefehl und zwei weiteren normalen Befehlen, werden diese in zwei Gruppen aufgeteilt. Die erste Befehlsgruppe (Dispatch group) besteht aus den 2 normalen Befehlen und dem Sprungbefehl. Die nächsten zwei Befehle gehen in die nächste Befehlsgruppe.

Wenn die Befehlsgruppen dann abgeschickt werden, müssen erst die nötigen Ressourcen bereitstehen. Das sind zum einen die nötigen freien Einträge in der Store Reorder Queue, damit diese schnell fertiggestellt werden können. Ebenso die nötigen Einträge in der Load Reorder Queue. Zusätzlich muss jeder Befehl der ein Ergebnis erzeugt auch ein Register zugewiesen bekommen, in dem es abgelegt werden kann. Damit immer genug Register zur Verfügung stehen, hat der PowerPC 970 eine große Zahl von Rename-Registern. So verfügt die AltiVeceinheit über insgesamt 80 Register, wo von 32 die regulären Register sind und die restlichen 48 Rename-Register. Wenn die Ressourcen bereit stehen, werden die Befehle in die Issue Queue gelegt, wo sie auf ihre Operanden warten. Dabei gibt es noch gewisse Einschränkungen wie die Befehle in der Gruppe auf die verschiedenen Ausführungseinheiten verteilt werden, wenn es mehrere mögliche für einen Befehl gibt. So hat der PowerPC 970 ja zwei Integer- und Gleitkommaeinheiten. Die Befehle an den Position 0 und 3 der Gruppe können dann nur von Einheit 1 bearbeitet werden und die Befehle an Position 1 und 2 an die Einheit 2. An die Vektoreinheit können die Befehle aus jeder Position in der Gruppe abgegeben werden. Anders als bei den PowerPC 74xx müssen die Befehle nicht in der Reihenfolge in der sie in die Issue Queue gekommen sind, an die Ausführungseinheiten abgegeben werden. Die Issue Queues sind je nach Einheit unterschiedlich groß.

Einheit	Issue Queue Größe
FPU1	10
FPU2	10
LSU1 / IU1	18
LSU2 / IU2	18
Sprungeinheit	12
Condition register	10
VPERM	16
VALU	20

Durch die relativ großen Issue Queues können Wartetakte auf Grund von Datenabhängigkeiten vermieden werden. Sehr kleine Schleifen werden sogar vollständig aufgerollt. Die Issue Queues verwenden allerdings Rename – Register, da den Befehlen ja bereits vor dem Abschicken in die Issue Queues ihre Ressourcen zugeteilt werden.

Bis zu 10 Befehle können pro Takt zur Ausführung abgeschickt werden. Jeweils zwei an die Load / Store Einheiten, die Integereinheiten und die Gleitkommaeinheiten. Jeweils einer an die Vektor ALU, die Vektor Permutationseinheit, die Condition Registerseinheit und ein Sprungbefehl.

Einheit	Takte pro Befehl
FPU	6
IU (+, -, logik, shift)	2 – 3
IU (Multiplikation)	5 – 7
Load / Store (Level 1 Cache Hit) im GPR, FPR, VR	3, 5, 3 – 5
Load (Level 2 Cache Hit)	11
Vector Permute Unit (VPERM)	2
Vector Simple Integer Unit (VSIU)	2
Vector Complex Integer Unit (VCIU)	5
Vector Floating Point Unit (VFPU)	8

Die Vektor ALU und die Vektor Permutationseinheit haben eigene Kopien der Vektorregister, daher wird ein zusätzlicher Takt benötigt wenn das Ergebnis einer Permutation durch eine Vektor ALU Operation benutzt wird oder umgekehrt.

Die Anzahlen der für die Vektorbefehle benötigten Takte unterscheiden sich für die verschiedenen PowerPC Prozessoren:

PowerPC Prozessor	7400 / 7410	7450 / 7455	970
Vector Load / Store unit (LSU)	3	3	4
Vector Simple Integer Unit (VSIU)	1	1	2
Vector Complex Integer Unit (VCIU)	3	4	5
Vector Floating Point Unit (VFPU)	4 (5)*	4 (5)*	8
Vector Permute Unit (VPERM)	1	2	2

\*Im IEEE – Modus wird ein Takt mehr benötigt.

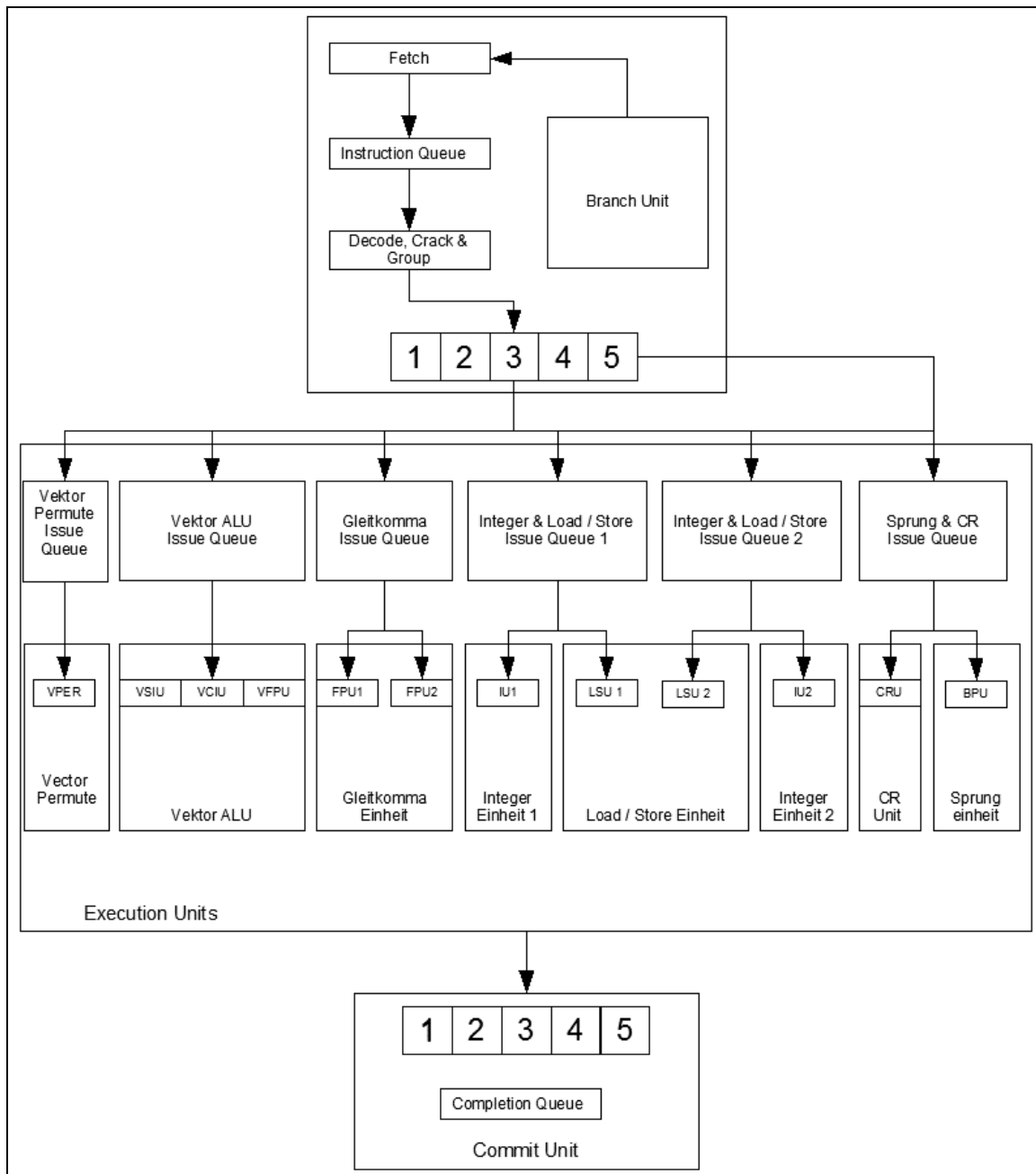


Abbildung 7: Aufbau PowerPC 970

## 4. Softwareunterstützung für Altivec

Unterstützung durch Software ist für eine SIMD-Einheit kritisch, ohne Softwareunterstützung ist eine solche Einheit nur unnötiger Ballast, die den Prozessor teuer macht und mehr Strom verbraucht. Die Programme die beschleunigt werden sollen müssen SIMD nutzen. Altivec ist schließlich eine Erweiterung des Befehlssatzes, ein Programm muss also die Befehle verwenden, es muss an die neue Technik angepasst werden. Eine SIMD-Einheit oder Befehlssatzerweiterung kann auf vier verschiedene Arten in Programmen genutzt werden:

1. Für die Performanz besonders kritische Teile werden in Assembler geschrieben, unter direkter Verwendung der entsprechenden SIMD-Befehle. Die Methode hat natürlich den Nachteil, dass sie sehr aufwendig ist. Der oder die Programmierer müssen den jeweiligen Assembler beherrschen und auf die Vorteile von Hochsprachen verzichten. Mit dieser Methode sind allerdings die größten Leistungsgewinne zu erzielen.

2. Es werden Erweiterungen von Hochsprachen verwendet, mit denen sich die SIMD-Befehle direkt einbinden lassen ohne Assembler zu verwenden. Das setzt aber voraus, dass die Compiler entsprechend angepasst sind. Bisher gibt es C, C++ und Objective C-Compiler die AltiVec-Erweiterungen unterstützen.
3. Angepasste Compiler können Hochsprachen Konstrukte automatisch vektorisieren, also in die entsprechenden SIMD-Befehle übersetzen.<sup>7</sup> So kann ein Compiler etwa kurze Schleifen komplett durch einen SIMD-Befehl ersetzen. Diese Möglichkeit ist noch sehr neu und es liegen noch keine Ergebnisse über ihre Qualität vor. Sie hat natürlich den großen Vorteil gegenüber den anderen beiden Methoden, dass sie keine Anpassungen in der Software erfordert.
4. Eine häufig genutzte Möglichkeit ist es, Bibliotheken mit SIMD optimierten Funktionen zu verwenden. Da bei Medienanwendungen häufig mit Bibliotheken oder ganzen Frameworks gearbeitet wird, profitieren alle Programme die sie nutzen automatisch, wenn diese mit SIMD-Technik optimiert wurden.

Bei alle vier Möglichkeiten ist es aber immer problematisch, dass die SIMD-Befehle prozessorspezifisch sind und die Portabilität von Programmen einschränken. Dies trifft bei den beiden ersten Möglichkeiten in hohem Maße zu. Die Möglichkeiten 3. und 4. lassen eine gewisse Portabilität zu. Das geschieht aber entweder um den Preis einer, eventuell deutlich schlechteren Leistung im Falle der automatisch optimierenden Compiler oder es ist eine Plattform übergreifende Bibliothek nötig.

Die AltiVec-Einheit wird heute von der allermeisten Software auf der Apple Macintosh Plattform unterstützt. Beinahe jede Software nutzt SIMD-Verarbeitung um die Performanz zu verbessern. Das hat einerseits den Grund, dass viele Software unter MacOS X der Bearbeitung von Medien dient und daher eine Anpassung tatsächlich einen deutlichen Geschwindigkeitsgewinn bringt. Die Apple Macintosh Plattform kann immer noch vor allem Programme aus diesem Bereich vorweisen, seien es Bildbearbeitungs-, Audioproduktions- oder Videoschnittprogramme. Viele dieser Programme werden mittlerweile auch von Apple selbst produziert.<sup>8</sup> Daher kann Apple natürlich die eigenen Technologien optimal nutzen.

Andererseits ist der Grund wohl auch in der lange Zeit nur langsamen Weiterentwicklung der PowerPC Prozessoren von Motorola zu sehen. Motorola gelang es lange nicht Prozessoren mit höheren Taktraten zu entwickeln und daher blieb als einzige Möglichkeit für Apple möglichst viel Geschwindigkeitssteigerung aus den vorhandenen Möglichkeiten herauszuholen. So stellt Apple auch umfangreiche Bibliotheken zur Verfügung, um es Entwicklern zu erleichtern die Vorteile AltiVec zu unterstützen.

Der schon obligatorischen Unterstützung von AltiVec auf der Seite der Programm für MacOS X, steht die bisher sehr eingeschränkte Unterstützung durch Programme unter Linux gegenüber. Das liegt wohl zum Teil daran, dass Linux auf der Intel Plattform entwickelt wurde und die meisten Linux Programmierer sich immer noch an dieser Plattform orientieren. Obwohl IBM schon einige Zeit Linux auch auf der, den PowerPCs verwandten POWER Plattform propagiert, gab es dort lange Zeit keine AltiVec-Einheit. So ist die Zahl der Linux Programme, die AltiVec nutzen gering geblieben.

---

<sup>7</sup> <http://gcc.gnu.org/gcc-4.0/changes.html> - Stand Mai 2005

<sup>8</sup> Apple hat so etwa Emagic, den Hersteller der Audiosoftware Logic aufgekauft. Auch ein Standard Programm für Professionellen Videoschnitt Final Cut gehört zu Apples Software Programm.

Die Darstellung der Softwareunterstützung für AltiVec beschränkt sich hier auf Gründen des Umfangs auf die Bibliotheken von Apple und Freescale<sup>9</sup>, sowie die Unterstützung durch die Gnu Compiler Collection. Die Leistungsfähigkeit der automatisch vektorisierenden Compiler konnte noch nicht untersucht werden. Es ist aber sicher eine interessante Fragestellung für spätere Arbeiten.

## **GNU Compiler Collection**

Eine relativ einfache Methode die AltiVec-Einheit zu nutzen, ist mit den C- und C++-Compilern der Gnu Compiler Collection. Apple hat diese um AltiVec-Befehle ergänzt. Dieser Modus wird mit der Option `-faltivec` aktiviert.<sup>10</sup>

Dazu wurden Vektordatentypen eingeführt, mit denen sich Daten in C leicht für AltiVec nutzbar machen lassen. Diese entsprechen den typischen skalaren Datentypen in C.

<b>8-Bit Datentypen</b>	<b>16-Bit Datentypen</b>	<b>32-Bit Datentypen</b>
vector bool char	vector bool short	vector bool int
vector signed char	vector signed short	vector signed int
vector unsigned char	vector unsigned short	vector unsigned int
	vector pixel	vector float

Jeder dieser Datentypen ist 128-Bit Array, mit den entsprechenden Daten. Zum Beispiel ist ein `vector unsigned short`, ein Array bestehend aus acht 16-Bit Integerwerten ohne Vorzeichen. Zu den Datentypen werden entsprechende C-Funktionen bereitgestellt, so dass nahezu alle AltiVec-Befehle genutzt werden können.

Ein Beispiel in C das zwei Vektoren mit der AltiVec-Einheit addiert:

```
result = vec_add( aVector, someOtherVector );
```

Der Compiler übersetzt dann, in Abhängigkeit von den verwendeten Datentypen in den entsprechenden AltiVec-Befehl:

```
vadduwm11 result, aVector, someOtherVector
```

Die AltiVec-Funktionen sind überladen, so dass der Compiler automatisch aus dem entsprechenden Datentyp erkennt, welcher AltiVec-Befehl zu benutzen ist. Beachten muss man allerdings, dass sich diese Vektordatentypen nicht einfach wie C-Datentypen verwenden lassen. Typ-Konversionen zwischen Vektordatentypen lassen sich nicht implizit machen, sonder müssen explizit geschehen. Dazu sollten die AltiVec-Funktionen zu Datentypenkonvertierung verwendet werden, da es sonst zu Fehlern kommen kann.

Mit Hilfe dieser Erweiterungen kann AltiVec relativ einfach genutzt werden, ohne dafür Assembler programmieren zu müssen. Diese Erweiterungen stehen auch in IBMs XL-C Compiler ab Version 6.0 zur Verfügung. Ebenso lassen sich Apples GCC-Erweiterungen zur Vektorverarbeitung auch unter Linux benutzen.

<sup>9</sup>Freescale ist die frühere Motorola Semiconductor Division.

<sup>10</sup> IBMs XLC und XLC++ Compiler sind kompatibel zu den GCC-Erweiterungen.

<sup>11</sup> `vadduwm` ist der AltiVec-Befehl um zwei Vektoren von 32-Bit Integerwerten zu addieren

Die Version 4.0 des GCC C-Compilers ist auch in der Lage, Programme automatisch bei der Übersetzung zu Vektorisieren, das heißt AltiVec-Befehle zu nutzen, wo sie Geschwindigkeitsvorteile bringen. Das ist vorallem bei Schleifen möglich, die dann durch entsprechende AltiVec-Schleifen ersetzt werden.

## ***Apple Bibliotheken für Mac OS X***

Apple stellt Entwicklern eine Reihe von Bibliotheken zur Verfügung, die eine einfache Nutzung der Vorteile von AltiVec ermöglichen.

### **vImage – Bibliothek zur Bildmanipulation**

Die Bibliothek enthält Funktionen zur Bildmanipulation, die direkt in Programmen verwendet werden können.

Die unterstützten Arten von Bildmanipulationen sind:

- Bildschärfe Operationen
- Geometrische Transformationen
- Histogramm Operationen
- Morph Operationen
- Alpha Compositing
- Bild Transformationen
- Format Konvertierung

### **vDSP – Bibliothek zur Signalverarbeitung**

Ist eine Bibliothek mit Funktionen zur Signalverarbeitung, die ebenfalls in eigenen Programmen verwendet werden können. Die Funktionen, die mit einfacher Genauigkeit arbeiten, nutzen die AltiVec-Einheit zur Beschleunigung. Die folgenden Gruppen von Funktionen stehen zur Verfügung:

- Vektor Konvertierungsfunktionen
- Vektor – Skalar arithmetische Funktionen
- Vektor – Vektor arithmetische Funktionen
- Funktionen zur Manipulation eines Vektors
- Ein-Dimensionale schnelle Fouriertransformation
- Zwei-Dimensionale schnelle Fouriertransformation
- Korrelation und Faltung Funktionen

### **QuickTime Framework**

Quicktime ist ein komplettes Framework um digitale Medien zu erstellen und wiederzugeben. Dazu gehören auch eine Reihe von Audio- und Videocodecs, die ab Version 5 die AltiVec-Einheit nutzen, um kodieren und dekodieren zu beschleunigen.

## ***Freescale Bibliothek für Linux***

Freescale stellt einige AltiVec optimierte Teile der libc für Linux zur Verfügung.

Aus string.s wurden folgende Funktionen durch optimierte Versionen in string\_vec.s ersetzt:

memcpy	memcpy_vec
bcopy	bcopy_vec
memmove	memmove_vec

memcpy	memcpy_vec
backwards_memcpy	backwards_memcpy_vec
memset	memset_vec
memcmp	memcmp_vec
cacheable_memcpy	cacheable_memcpy_vec
cacheable_memzero	cacheable_memzero_vec
strcpy	strcpy_vec
strcmp	strcmp_vec
strlen	strlen_vec
__copy_tofrom_user*	__copy_tofrom_user_vec*
__clear_user*	__clear_user_vec*

In checksum.s wurden in checksum\_vec.s diese Funktionen ersetzt:

csum_partial	csum_partial_vec
csum_partial_copy_generic*	csum_partial_copy_generic_vec

### **Open Source Bibliotheken**

Die meisten dieser Open Source Bibliotheken am CEPBA-IBM Research Institute (CIRI) um Altivec Unterstützung erweitert. Dabei wurden für Linux auf dem IBM eServer JS20 optimiert.

#### **BLAS (Basic Linear Algebra Subprograms)**

Eine optimierte Bibliothek für Vektor- und Matrizenrechnung, die unter C oder Fortran 77 genutzt werden kann. LAPACK baut auf BLAS auf.

#### **FFTW**

FFTW ist eine C-Bibliothek für mehrdimensionale, diskrete Fourier-Transformationen. Es existieren Schnittstellen für C und Fortran 77. Es sind aber nur die Transformationen einfacher Genauigkeit für Altivec optimiert.

#### **METIS**

METIS ist eine Sammlung von Programmen zur Partitionierung von großen, unstrukturierten Graphen und Netzen. Nur kleine Teile sind für Altivec optimiert worden.

#### **BLAST (Basic Local Alignment Search Tool)**

BLAST ist eine Sammlung von Programmen zur Ähnlichkeitssuche in Gen-Datenbanken.

#### **HMMER**

HMMER ist eine Implementierung von Methoden des Verborgenen Markov Modells, mit deren Hilfe Suchen in Datenbanken durchgeführt werden.

#### **POVRAY**

Der beliebte freie Raytracer POV-Ray ist ebenfalls für Altivec optimiert worden.

## 5. Zusammenfassung und Ausblick

Eigentlich enthalten alle verbreiteten Mikroprozessoren SIMD-Technik. Die AltiVec-Einheit ist ein Beispiel für eine Erweiterung, durch die Mikroprozessoren schon beinahe zu SIMD-Rechnern werden. Sie hat auch gezeigt, dass mit dieser Technik, sehr effizient beachtliche Leistungssteigerungen zu erzielen sind. Ohne AltiVec hätten sich die PowerPC Prozessoren sich nicht so lange gegen die Pentium Prozessoren, mit ihren enormen Taktratensteigerungen behaupten können. Die Methode der Leistungssteigerung durch höhere Taktraten scheint an ihre Grenzen zu stoßen.<sup>12</sup> Geht sie doch einher mit sehr hohem Stromverbrauch, so dass für viele Anwendungen diese Methode der Leistungssteigerung nicht möglich ist. So werden jetzt andere Techniken der Leistungssteigerung wichtiger. Alle großen Prozessorhersteller planen Prozessoren mit zwei Kernen auf den Markt zu bringen beziehungsweise, haben das schon getan. Neuere Prozessorentwicklungen wie der Cell-Prozessor des IBM / Sony / Toshiba-Konsortiums setzen aber auch stark auf SIMD zur Leistungssteigerung.<sup>13</sup> Für die Zukunft wird man einen verstärkten Einsatz von SIMD-Technik erwarten können. Obwohl PowerPC Prozessoren mit AltiVec in PCs, Workstations oder Servern seltener werden, sind sie vor allem in verschiedenen Embedded-Anwendungen zu finden. So wird auch AltiVec in Zukunft noch eine Rolle spielen.

---

<sup>12</sup> 4 Ghz Pentium IV abgesagt, <http://www.heise.de/newsticker/meldung/52187/>

<sup>13</sup> Neben den 8 SIMD-Einheiten, enthält auch der PowerPC-Kern des Cell-Prozessors eine AltiVec-Einheit.

## 6. Literaturverzeichnis

- [1] *3DNow! Technology Manual*, [http://www.amd.com/us-en/assets/content\\_type/white\\_papers\\_and\\_tech\\_docs/21928.pdf](http://www.amd.com/us-en/assets/content_type/white_papers_and_tech_docs/21928.pdf), Advanced Micro Devices Inc. 2000
- [2] *64Bit PowerPC G5 Prozessor*, <http://www.apple.com/de/g5processor/>, Apple Computer Inc. 2005
- [3] *AltiVec Fact Sheet*, [http://www.freescale.com/files/32bit/doc/fact\\_sheet/ALTIVECFACT.pdf](http://www.freescale.com/files/32bit/doc/fact_sheet/ALTIVECFACT.pdf), Freescale Semiconductor Inc. 2005
- [4] *AMD Extensions to the 3DNow! and MMX Instruction Sets Manual*, [http://www.amd.com/us-en/assets/content\\_type/white\\_papers\\_and\\_tech\\_docs/22466.pdf](http://www.amd.com/us-en/assets/content_type/white_papers_and_tech_docs/22466.pdf), Advanced Micro Devices Inc. 2000
- [5] *Apple Developer Connection – Velocity Engine*, <http://developer.apple.com/hardware/ve/>, Apple Computer Inc. 2005
- [6] *Application Enablement and Tuning for Linux on PowerPC 970 Project*, [http://www.ciri.upc.es/pblade\\_tuning/index.html](http://www.ciri.upc.es/pblade_tuning/index.html), CEPBA-IBM Research Institute 2005
- [7] Bennewitz, Nils, *Intel MMX eine Multimedia-Erweiterung*, <http://www2.informatik.uni-jena.de/~mau/seminare/SS99/bennewitz/IntelMMX2.pdf>, Seminararbeit 1999
- [8] Bleul Andreas, *Jonglierwettbewerb - Vektorarchitekturen aktueller Prozessoren im Vergleich*, C't 4/2000, S. 314-323
- [9] *Fact sheet MPC7410*, [http://www.freescale.com/files/32bit/doc/fact\\_sheet/MPC7410FACT.pdf](http://www.freescale.com/files/32bit/doc/fact_sheet/MPC7410FACT.pdf), Freescale Semiconductor Inc. 2005
- [10] Fuller, Sam, *Motorola's AltiVec Technology*, Motorola Inc. 1998
- [11] *Ganz auf Leistung eingestellt: PowerPC G4 Prozessor*, <http://www.apple.com/de/ibook/processor.html>, Apple Computer Inc. 2005
- [12] Gibbs, Ben and Arenburg, Robert; Bonaventure, Damien; Elkin, Bradley; Grima, Rogeli; Wang, Amy, *IBM eServer BladeCenter JS20 PowerPC 970 Programming Environment*, <http://www.redbooks.ibm.com/redpapers/pdfs/redp3890.pdf>, IBM 2005
- [13] Hennessey, John und Patterson, David, *Computer Architecture - A Quantitative Approach*, Morgan Kaufmann 2003, Appendix G – Vector Processors
- [14] Lund, Craig, *PowerPC SIMD Instructions*, Smart Networks Developer Forum, New Orleans 2002
- [15] Miaw, Wesley, *The Motorola AltiVec Technology*, <http://www.wesman.net/~wesley/papers/mpc7400.pdf>, University of California Berkley 1999
- [16] Michael Flynn, *Very high-speed computing systems*, Proceedings of the IEEE, Volume 54, 1966, S. 1901-1909
- [17] *MPC7450 RISC Microprocessor Family Product Brief*, [http://www.freescale.com/files/32bit/doc/ref\\_manual/MPC7450UM.pdf](http://www.freescale.com/files/32bit/doc/ref_manual/MPC7450UM.pdf), Freescale Semiconductor Inc. 2005
- [18] *MPC7450 RISC Microprocessor Family Reference Manual*, [http://www.freescale.com/files/32bit/doc/prod\\_brief/MPC7450TS.pdf](http://www.freescale.com/files/32bit/doc/prod_brief/MPC7450TS.pdf), Freescale Semiconductor Inc. 2005
- [19] Müller, Thomas, *HP PA-RISC / MAX*, <http://www2.informatik.uni-jena.de/~mau/seminare/SS99/Mueller/HPRISC.pdf>, Seminararbeit 1999
- [20] Pan, Jacob, *Using AltiVec Technology to Accelerate Network Software in a Linux Environment*, Smart Networks Developer Forum, Dallas 2003
- [21] *PowerPC 7410 Technical Summary*, [http://www.freescale.com/files/32bit/doc/prod\\_brief/MPC7410TS.pdf](http://www.freescale.com/files/32bit/doc/prod_brief/MPC7410TS.pdf), Freescale Semiconductor Inc. 2000

- [22] *PowerPC Microprocessor Family: AltiVec Technology Programming Environments Manual Version 2.0*, [http://www-3.ibm.com/chips/techlib/techlib.nsf/techdocs/FBFA164F824370F987256D6A006F424D/\\$file/AltiVec\\_pem.d20030710.pdf](http://www-3.ibm.com/chips/techlib/techlib.nsf/techdocs/FBFA164F824370F987256D6A006F424D/$file/AltiVec_pem.d20030710.pdf), IBM 2003
- [23] Sandon, Peter, *PowerPC 970: First in a new family of 64-bit high performance PowerPC processors*, Microprocessor Forum San Jose Kalifornien 2002
- [24] Seale, Susan, *PowerPC G4 Architecture White Paper*, Motorola Inc. 2001
- [25] Slater, Mark, *Vector Processing in the Motorola PowerPC 7400*, <http://www.cse.ucsc.edu/~mslater/papers/VectorProcessingG4.pdf>, University of California Santa Cruz 2000
- [26] Slingerland, Nathan und Smith, Alan Jay, *Performance Analysis of Instruction Set Architecture Extensions for Multimedia*, <http://www.pdcl.eng.wayne.edu/msp01/paper7.pdf>, 34th International Symposium on Microarchitecture, Austin (Texas) 2001
- [27] *SPARC Assembly Language Reference Manual*, <http://docs.sun.com/app/docs/doc/806-3774>, Sun Microsystems 2005
- [28] Stokes, Jon, *Inside the IBM PowerPC 970*, <http://arstechnica.com/articles/paedia/cpu/ppc970.ars>, Ars Technica 2002
- [29] Stokes, Jon, *PowerPC on Apple: An Architectural History*, <http://arstechnica.com/articles/paedia/cpu/ppc-1.ars/2>, Ars Technica 2004
- [30] Stokes, Jon, *SIMD architectures*, <http://arstechnica.com/articles/paedia/cpu/simd.ars>, Ars Technica 2000
- [31] Thakkar, Shreekant und Huff, Tom, *The Internet Streaming SIMD Extensions*, Intel Technology Journal Q2 1999
- [32] *The VIS Instruction Set*, [http://www.sun.com/processors/whitepapers/vis\\_wp\\_external.pdf](http://www.sun.com/processors/whitepapers/vis_wp_external.pdf), Sun Microsystems 2002
- [33] Thompson, Tom, *AltiVec Revealed*, <http://www.mactech.com/articles/mactech/Vol.15/15.07/AltiVecRevealed/>, MacTech Volume 15 Issue 7, 1999
- [34] *VIS Instruction Set User's Manual*, <http://www.sun.com/processors/manuals/805-1394.pdf>, Sun Microsystems 2001
- [35] *x86 Assembly Language Reference Manual*, <http://docs.sun.com/app/docs/doc/817-5477>, Sun Microsystems 2005

## Anhang A AltiVec Befehle

### Load / Store Instructions

Befehl	Beschreibung	Takte	Einheit
lvx	Lädt einen Vektor aus dem Speicher in ein Vektorregister.	2	LSU
lvxl	Lädt einen Vektor und setzt den Cache -Tag sofort auf LRU.	2	LSU
lvebx	Lädt einzelne Vektorelemente in die Vektorregister.	2	LSU
lvehx	lvebx – Byte, lvehx – Halfword (16-Bit), lvewx – Word (32-Bit)		
lvewx			
lvsl	Lädt einen Vektor und verschiebt gleichzeitig nach Links. (Wird verwendet um nicht ausgerichtete Daten zu laden)	2	LSU
lvslr	Lädt einen Vektor und verschiebt gleichzeitig nach Rechts. (Wird verwendet um nicht ausgerichtete Daten zu laden)	2	LSU
stvx	Speichert einen Vektor.	3	LSU
stvx1	Speichert einen Vektor und setzt den Eintrag im Cache auf LRU.	3	LSU
stvebx	Speichert einzelne Vektorelemente.	3	LSU
stvehx	stvebx – Byte, stvehx – Halfword (16-Bit), stvewx – Word (32-Bit)		
stvewx			

### Software Directed Prefetch Instructions

Befehl	Beschreibung	Takte	Einheit
dst	Erzeugt einen neuen Datenstrom.	3	LSU
dstt	Erzeugt einen transienten Datenstrom.	3	LSU
dstst	Erzeugt einen Datenstrom zum Speichern.	3	LSU
dststt	Erzeugt einen transienten Datenstrom zum Speichern.	3	LSU
dss	Stop einen Datenstrom.	1	LSU
dssall	Stop alle Datenströme.	0	LSU

## Data Manipulation and Conversion Instructions

<b>Befehl</b>	<b>Beschreibung</b>	<b>Takte</b>
vperm	Permutiert zwei Vektoren zu einem neuen dritten Vektor.	3
vsel	Durch die Maske in einem Condition Register gesteuert, werden Bits aus Vektor A oder Vektor B ausgewählt und einen neuen dritten Vektor geschrieben.	3
vsrcb	Vektor nach Rechts Verschieben, auf Byte, Halfword oder Word Daten.	2
vsrh		
vsrw		
vsrab	Vektor nach Rechts Verschieben, auf Byte, Halfword oder Word Daten. Das Vorzeichen wird erhalten, in dem das äußerste Bit vervielfacht wird.	2
vsrah		
vsraw		
vsrc	Vektor Rechts Verschieben, es wird mit „0“ - Bits aufgefüllt.	2
vsro	Vektor nach Rechts Verschieben um ganze Bytes.	2
vslb	Vektor nach Links Verschieben, auf Byte, Halfword oder Word Daten.	2
vslh		
vslw		
vsl	Vektor Links Verschieben, es wird mit „0“ - Bits aufgefüllt.	2
vslo	Vektor Links Verschieben um ganze Bytes.	2
vsldoi	Vektor A wird um eine Anzahl Bytes nach Links verschoben und mit Bytes aus Vektor B aufgefüllt.	3
vrlb	Rotiert die Elemente eines Vektors, nach Datentyp: Byte, Halfword, Word.	2
vrlh		
vrlw		
vmrghb	Mischt zwei Vektoren, in dem die oberen Elemente von Vektor A an die graden Stellen des Zielvektors kopiert werden und die oberen Elemente aus Vektor B an die ungraden. Nach Datentypen: Byte, Halfword, Word.	2
vmrghh		

vmrghw		
vmrglb	Mischt zwei Vektoren, in dem die unteren Elemente von Vektor A an die graden Stellen des Zielvektors kopiert werden und die unteren Elemente aus Vektor B an die ungraden. Nach Datentypen: Byte, Halfword, Word.	2
vmrglh		
vmrglw		
vspltb	Der Inhalt eines Element aus Vektor A, wird in jedes Element in Vektor B kopiert.	2
vsplth	Nach Datentypen Byte, Halfword, Word.	
vspltw		
vspltisb	Ein Immediate - Wert wird mit Vorzeichen in jedes Element eines Zielvektors kopiert. Nach Datentyp Byte, Halfword, Word.	1
vspltish		
vspltisw		
vspltiub	Ein Immediate - Wert wird ohne Vorzeichen in jedes Element eines Zielvektors kopiert. Nach Datentyp Byte, Halfword, Word.	1
vspltiuh		
vspltiuw		
mfvscr	Holt den Inhalt aus dem Vektor Status- und Kontrollregister in ein Vektoregister.	0
mtvscr	Kopiert den Inhalt eines Vektorregisters in das Vektor Status- und Kontrollregister.	1
vpkuhum	Die untere Hälfte jedes Halfword- oder Word-Elements aus zwei Registern wird in einem Register aneinandergehängt.	2
vpkuwum		
vpkpx	Zwei Vektoren werden als 32-Bit Pixel Daten zu je 8-Bit Farbwerten behandelt und entsprechend in einem neuen Vektor zu 16-Bit Farbdaten gemischt.	2
vpkpx	Zwei Vektoren werden als 32-Bit Pixel Daten zu je 8-Bit Farbwerten behandelt und entsprechend in einem neuen Vektor zu 16-Bit Pixeldaten gemischt.	2
vpkuwus	Die Integer Halfword oder Word Elemente ohne Vorzeichen werden zu Byte konvertiert, alle Werte über 8-Bit saturieren auf den höchsten Wert.	
vpkuhus		
vpkshus	Die Integer Halfword oder Word Elemente mit Vorzeichen werden zu Byte ohne Vorzeichen konvertiert, alle Werte über 8-Bit saturieren auf den höchsten Wert.	2
vpkswus		
vupkhsb	Die Byte bzw. die Halfword Element in der oberen Hälfte des Vektors werden zu	1

vupkhsh	Halfword bzw. Word Elementen im neuen Vektor konvertiert.	
vupkl1sb	Die Byte bzw. die Halfword Element in der unteren Hälfte des Vektors werden zu Halfword bzw. Word Elementen im neuen Vektor konvertiert.	1
vupklsh		
vupkhp1x		
vupklpx		

### Integer Arithmetic Instructions

Befehl	Beschreibung	Takte	Einheit
vmaxsb	Vergleicht die Elemente in Vektor A mit denen in Vektor B. Das größere Element wird jeweils in den Ergebnisvektor kopiert.  Nach Datentypen: Byte, Halfword, Word.	1	VSIU
vmaxsh			
vmaxsw			
vaddubm	Addiert zwei Vektoren, das Ergebnis wird entsprechend dem Datentyp Modulo gerechnet.  Nach Datentypen: Byte, Halfword, Word.	2	VSIU
vadduhm			
vadduwm			
vaddcuw	Vektor A aus Word ohne Vorzeichen wird zu Vektor B aus Word ohne Vorzeichen addiert. Die Carry-Bits werden in das Ergebnisregister geschrieben.	2	VSIU
vaddubs	Vektor Elemente werden entsprechend den Datentypen saturierend addiert. Varianten mit und ohne Vorzeichen.	2	VSIU
vaddsbs			
vadduhs			
vaddshs			
vadduws			
vaddsws			
vaddubm	Vektor Elemente werden entsprechend den Datentypen moduloaddiert. Varianten mit und ohne Vorzeichen.	2	VSIU
vadduhm			
vadduwm			

vaddsbm			
vaddshm			
vaddswm			
vsububm	Vektor Elemente werden entsprechend den Datentypen modulusubtrahiert. Varianten mit und ohne Vorzeichen.	2	VSIU
vsubuhm			
vsubuwm			
vsubcuw	Vektor B aus Word ohne Vorzeichen wird von Vektor A aus Word ohne Vorzeichen subtrahiert. Die Carry-Bits werden in das Ergebnisregister geschrieben.	2	VSIU
vsububs	Vektor Elemente werden entsprechend den Datentypen saturierend addiert. Varianten mit und ohne Vorzeichen.	2	VSIU
vsubsbs			
vsubuhs			
vsubshs			
vsubuws			
vsubsws			
vmuleub	Multipliziert die graden Elemente zweier Vektoren, die Ergebnisse werden entsprechend zu dem größeren Datentyp. Varianten mit und ohne Vorzeichen.	2	VCIU
vmulesb			
vmuleuh			
vmulesh			
vmuloub	Multipliziert die ungraden Elemente zweier Vektoren, die Ergebnisse werden entsprechend zu dem größeren Datentyp. Varianten mit und ohne Vorzeichen.	2	VCIU
vmulosb			
vmulouh			
vmulosh			
vmhaddshs	Die Integer Halfword Elemente in Vektor A werden multipliziert mit den entsprechenden Elementen in Vektor B und dann wird mit der Vektor C addiert und das Halfword - Ergebnis saturiert.	3	VCIU

vmladduhm	Die Integer Halfword Elemente in Vektor A werden multipliziert mit den entsprechenden Elementen in Vektor B und dann wird mit der Vektor C addiert. Das Ergebnis sind wieder Halfwords.	3	VCIU
vmhraddshs	Die Integer Halfword Elemente mit Vorzeichen in Vektor A werden multipliziert mit den entsprechenden Elementen in Vektor B das Ergebnis ist ein 32-Bit Word Integer mit Vorzeichen	3	VCIU
vmsumubm	Die entsprechenden Teilelemente (Byte / Halfword) der Word in Vektor A werden mit den entsprechenden Elementen in Vektor B multipliziert, und der nächst größere Datentyp wird erzeugt, die Summe (Modulo) dieser Werte wird zu dem Integer Word ohne Vorzeichen in Vektor C addiert.	3	VCIU
vmsumuhm			
vmsummbm			
vmsumshm			
vmsumuhs	Die entsprechenden Halfword-Teile der Word in Vektor A werden mit den entsprechenden Elementen in Vektor B multipliziert und ein 32-Bit Integer Word erzeugt. Die Summe der beiden Word Elemente wird zu den Integer Word Elementen in Vektor C addiert und saturiert.	3	VCIU
vmsumshs			
vsum4ubs	Die entsprechenden Teilelemente (Byte / Halfword) in jedem Word in Vektor B werden addiert und das Ergebnis wird zu dem entsprechenden Word Element in Vektor A addiert und saturiert.	2	VCIU
vsum4sbs			
vsum4shs			
vsum2sws	Die Summe der beiden Word mit Vorzeichen in Vektor A wird zu dem dritten Word in Vektor B addiert und saturiert. Das Ergebnis wird an die entsprechende Stelle im Ergebnisvektor kopiert. Der Rest des Ergebnisvektor wird gelöscht.	2	VCIU
vsumsws	Die Summe der vier Integer Words mit Vorzeichen in Vektor A wird zu dem vierten Integer Word in Vektor B addiert und das Ergebnis saturiert. Der Rest des Ergebnisvektor wird gelöscht.	2	VCIU
vavgub	Die entsprechenden Elemente (Byte / Halfword / Word) mit und ohne Vorzeichen in Vektor A werden zu denen in Vektor B addiert und dann um 1 erhöht. Die oberen 8-Bit sind das Ergebnis.	2	VSIU
vavgub			
vavguh			
vavgsh			
vavguw			
vavgsw			
vmaxub	Die entsprechenden Integer Elemente (Byte / Halfword / Word) aus Vektor A und B werden verglichen. Das größere der beiden kommt in	2	VSIU

vmaxsb	das Ergebnis.		
vmaxuh			
vmaxsh			
vmaxuw			
vmaxsw			
vminub	Die entsprechenden Integer Elemente (Byte / Halfword / Word) aus Vektor A und B werden verglichen. Das kleinere der beiden Elemente kommt in den Ergebnisvektor.	2	VSIU
vminsb			
vminuh			
vminsh			
vminuw			
vminsw			

### Floatingpoint Arithmetic Instructions

Befehl	Beschreibung	Takte	Einheit
vmaddfp	Die Single Precision Floating Point Elemente aus Vektor A werden mit den Single Precision Floating Point Elementen aus Vektor C multipliziert und dann werden die Single Precision Floating Point Elemente aus Vektor B addiert. Das Ergebnis wird auf den nächsten Wert gerundet.	3	VFPU
vsubfp	Subtrahiert zwei Floating Point Vektoren.	2	VFPU
vnmsubfp	Alle vier Single Precision Floating Point aus Vektor A werden mit denen aus Vektor C multipliziert. Dann werden entsprechend Single Precision Floating Point aus Vektor B subtrahiert. Das Vorzeichen wird invertiert und auf die nächste Zahl gerundet.	3	VFPU
vmaxfp	Die Elemente aus Vektor A und B werden verglichen. Das größere der beiden kommt in das Ergebnis.	2	VFPU
vminfp	Die Elemente aus Vektor A und B werden verglichen. Das kleinere der beiden Elemente kommt in den Ergebnisvektor.	2	VFPU
vrfn	Die Single Precision Floating Point in Vektor B werden mit der Round to Nearest-Methode gerundet.	1	VFPU

<b>Befehl</b>	<b>Beschreibung</b>	<b>Takte</b>	<b>Einheit</b>
vrfigp	Die Single Precision Floating Point in Vektor B werden mit der Round to positiv Infinity-Methode gerundet.	1	VFPU
vrfigm	Die Single Precision Floating Point in Vektor B werden mit der "Round to negativ Infinity"-Methode gerundet.	1	VFPU
vrfigz	Die Single Precision Floating Point in Vektor B werden mit der Round to Zero-Methode gerundet.	1	VFPU
vrifp	Der Kehrwert jedes Single Precision Floating Point wird berechnet.	1	VFPU
vrqrtefp	Vector Reciprocal Square Root Estimate  Der Kehrwert der Wurzel jeden Single Precision Floating Point wird berechnet.	1	VFPU
vlogefp	Für jeden Single Precision Floating Point wird der Logarithmus zur Basis 2 berechnet.	1	VFPU
vexptefp	Für jeden Single Precision Floating Point wird der Logarithmus zur Basis 2 berechnet und mit sich selbst potenziert.	1	VFPU
vcfux	Konvertiert die Integer Word im Vektor zu Floating Point, mit und ohne Vorzeichen.	2	VFPU
vcfsx			
vctxsx	Multipliziert alle Word in Vektor B mit einem Immediate und saturiert.	2	VFPU
vctuxs	Konvertiert Single Precision Floating Point zu Integern und multipliziert mit einem Immediate und saturiert dann.	2	VFPU

### Logical Instructions

<b>Befehl</b>	<b>Beschreibung</b>	<b>Takte</b>	<b>Einheit</b>
vand	AND-Operation auf Vektoren	2	VSIU
vandc	AND- und Nicht- Operation auf Vektoren	2	VSIU
vor	OR-Operation auf Vektoren	2	VSIU

vnor	NOR-Operation auf Vektoren	2	VSIU
vxor	XOR-Operation auf Vektoren	2	VSIU

### Compare Instructions

Befehl	Beschreibung	Takte	Einheit
vcmpequb	Vektor-Vergleich auf Gleichheit, der entsprechenden Elemente (Byte, Halfword, Word).	2	VSIU
vcmpequh			
vcmpequw			
vcmpgefp	Floating Point Vektor-Vergleich auf Größer-oder-Gleich, der entsprechenden Elemente.	2	VFPU
vcmpgtub	Vektor-Vergleich auf Größer-als, der entsprechenden Elemente, mit und ohne Vorzeichen.	2	VSIU
vcmpgtub			
vcmpgtuh			
vcmpgtsh			
vcmpgtuw			
vcmpgtsw			
vcmpgtfp			VFPU
vcmpbfp	Vergleich von zwei Vektoren, ob die Floating Point Werte in Vektor A in dem Bereich der Werte in Vektor B sind.	2	VFPU
vcmpeqfp	Vergleich von zwei Floating Point Vektoren auf Gleichheit.	2	VFPU

## Anhang B VIS-Befehle

Die arithmetischen VIS - Befehle:

Befehl	Beschreibung:
FPADD16	Addiert vier 16-Bit Elemente
FPADD16S	Addiert zwei 16-Bit Elemente
FPADD32	Addiert zwei 32-Bit Elemente
FPADD32S	Addiert ein 32-Bit Element
FPSUB16	Subtrahiert vier 16-Bit Elemente
FPSUB16S	Subtrahiert zwei 16-Bit Elemente
FPSUB32	Subtrahiert zwei 32-Bit Elemente
FPSUB32S	Subtrahiert ein 32-Bit Element
FMUL8x16	8-Bit mal 16-Bit Multiplikation, wobei von den 24-Bit Ergebnissen jeweils nur die oberen 16-Bit gespeichert werden.
FMUL8x16AU	Wie FMUL8x16, aber es wird nur der obere 16-Bit Fixed Format Wert für alle Multiplikationen benutzt. Dies ist im Pixel Data Format der Alphakanal - Wert.
FMUL8x16AL	Wie FMUL8x16AU, aber es nur der untere 16-Bit Fixed Format Wert für alle Multiplikationen benutzt.
FMUL8SUx16	Multipliziert die oberen 8-Bit jedes 16-Bit Wertes des einen Registers mit den 16-Bit Werten des anderen Registers. Das 24-Bit Ergebnis wird gerundet und die oberen 16 Bit werden gespeichert.
FMUL8ULx16	Multipliziert die unteren 8-Bit jedes 16-Bit Wertes des einen Registers mit den 16-Bit Werten des anderen Registers. Das Ergebnis wird auf 32-Bit erweitert, gerundet und die oberen 16 Bit werden gespeichert.
FMULD8SUx16	Es werden die beiden oberen der unteren 4 8-Bit Werte mit den beiden unteren 16-Bit Werten multipliziert. Das Ergebnis wird um 8-Bit nach geschoben um einen 32-Bit Wert zu erhalten und so gespeichert.
FMULD8ULx16	Es werden die beiden oberen der unteren 4 8-Bit Werte mit den beiden unteren 16-Bit Werten multipliziert. Das Ergebnis wird auf einen 32-Bit Wert mit Vorzeichen erweitert und so gespeichert.

Die VIS Pack – Befehle erlauben eine Konvertierung von einem Fixed Format zu dem Pixel Data Format.

FPACK16	Konvertiert Fixed Format 16-Bit Daten zu 4 8-Bit Integer Pixel Format Daten.
FPACK32	Konvertiert Fixed Format 32-Bit Daten zu 2 8-Bit Integer Pixel Format Daten.
FPACKFIX	Konvertiert und rundet von 32-Bit Fixed Data auf 16-Bit Fixed Daten.
FEXPAND	Konvertiert 4 8-Bit Pixel Format Daten in 4 16-Bit Fixed Format Daten.

FPMERGE	Mischt 2 mal 8-Bit Integer Pixel Format Daten zu einem 64-Bit Packed Format Element.
---------	--

### Die VIS Alignment Befehle:

ALIGNADDRESS	Berechnet eine Adresse für nicht richtig ausgerichtete Daten
ALIGNADDRESS_LITTLE	Wie Alignaddress, aber mit Little-Endian Ausrichtung
FALIGNDATA	Verkettet zwei 64-Bit Register und speichert daraus 8 aufeinander folgende Bytes.

### Die ASI-Erweiterungen für die Load/Store Befehle:

ASI_FL8_P	8-Bit Load / Store aus dem primären Adressraum.
ASI_FL8_S	8-Bit Load / Store aus dem sekundären Adressraum.
ASI_FL8_PL	8-Bit Load / Store aus dem primären Adressraum, Little Endian.
ASI_FL8_SL	8-Bit Load / Store aus dem sekundären Adressraum, Little Endian.
ASI_FL16_P	16-Bit Load / Store aus dem primären Adressraum.
ASI_FL16_S	16-Bit Load / Store aus dem sekundären Adressraum.
ASI_FL16_PL	16-Bit Load / Store aus dem primären Adressraum, Little Endian.
ASI_FL16_SL	16-Bit Load / Store aus dem sekundären Adressraum, Little Endian.

### Die bedingten Load/Store – Befehle:

ASI_PST8_P	Bedingtes Speichern von acht 8-Bit Daten in primären Adressraum.
ASI_PST8_S	Bedingtes Speichern von acht 8-Bit Daten in sekundären Adressraum.
ASI_PST8_PL	Bedingtes Speichern von acht 8-Bit Daten in primären Adressraum, Little Endian.
ASI_PST8_SL	Bedingtes Speichern von acht 8-Bit Daten in sekundären Adressraum, Little Endian.
ASI_PST16_P	Bedingtes Speichern von vier 16-Bit Daten in primären Adressraum.
ASI_PST16_S	Bedingtes Speichern von vier 16-Bit Daten in sekundären Adressraum.
ASI_PST16_PL	Bedingtes Speichern von vier 16-Bit Daten in primären Adressraum, Little Endian.
ASI_PST16_SL	Bedingtes Speichern von vier 16-Bit Daten in sekundären Adressraum, Little Endian.
ASI_PST32_P	Bedingtes Speichern von zwei 32-Bit Daten in primären Adressraum.
ASI_PST32_S	Bedingtes Speichern von zwei 32-Bit Daten in sekundären Adressraum.

ASI_PST32_PL	Bedingtes Speichern von zwei 32-Bit Daten in primären Adressraum, Little Endian.
ASI_PST32_SL	Bedingtes Speichern von zwei 32-Bit Daten in sekundären Adressraum, Little Endian.

#### Die Block-Load/Store Befehle:

ASI_BLK_AIUP	64-Byte Block Load / Store aus dem oder in den primären Adressraum mit user privilege
ASI_BLK_AIUS	64-Byte Block Load / Store aus dem oder in den sekundären Adressraum mit user privilege
ASI_BLK_AIUPL	64-Byte Block Load / Store aus dem oder in den primären Adressraum mit user privilege, Little Endian
ASI_BLK_AIUSL	64-Byte Block Load / Store aus dem oder in den sekundären Adressraum mit user privilege, Little Endian
ASI_BLK_P	64-Byte Block Load / Store aus dem oder in den primären Adressraum
ASI_BLK_S	64-Byte Block Load / Store aus dem oder in den sekundären Adressraum
ASI_BLK_PL	64-Byte Block Load / Store aus dem oder in den primären Adressraum, Little Endian
ASI_BLK_SL	64-Byte Block Load / Store aus dem oder in den sekundären Adressraum, Little Endian
ASI_BLK_COMMIT_P	64-Byte Block Commit Store in den primären Adressraum, verhindert, dass die Daten im Cache bleiben
ASI_BLK_COMMIT_S	64-Byte Block Commit Store in den sekundären Adressraum, verhindert, dass die Daten im Cache bleiben

#### Konvertierungsbefehle für den Blockzugriff auf die Daten:

ARRAY8	Konvertiert 8-Bit 3D-Koordinaten in Speicheradresse im Blockformat.
ARRAY16	Konvertiert 16-Bit 3D-Koordinaten in Speicheradresse im Blockformat.
ARRAY32	Konvertiert 32-Bit 3D-Koordinaten in Speicheradresse im Blockformat.

#### Die Vergleichsbefehle:

FCMPGT16	Größer-als-Test, von zwei Registern mit 16-Bit Partitionierung. Die Bits des Ergebnisses werden für jedes Element einzeln gesetzt, wenn es in Register 1 größer als das in Register 2 ist.
FCMPGT32	Größer-als-Vergleich, von zwei Registern mit 32-Bit Partitionierung. Die Bits des Ergebnisses werden für jedes Element einzeln gesetzt, wenn es in Register 1 größer als das in Register 2 ist.

FCMPLE16	Kleiner-oder-gleich-Test mit 16-Bit Partitionierung. Die Bits des Ergebnisses werden für jedes Element einzeln gesetzt, wenn es in Register 1 kleiner oder gleich als das in Register 2 ist.
FCMPLE32	Kleiner-oder-gleich-Test mit 32-Bit Partitionierung. Die Bits des Ergebnisses werden für jedes Element einzeln gesetzt, wenn es in Register 1 kleiner oder gleich als das in Register 2 ist.
FCMPNE16	Gleichheits-Test mit 16-Bit Partitionierung. Die Bits des Ergebnisses werden für jedes Element einzeln gesetzt, wenn es in Register 1 gleich dem in Register 2 ist.
FCMPNE32	Gleichheits-Test mit 32-Bit Partitionierung. Die Bits des Ergebnisses werden für jedes Element einzeln gesetzt, wenn es in Register 1 gleich dem in Register 2 ist.
FCMPEQ16	Ungleich-Test mit 16-Bit Partitionierung. Die Bits des Ergebnisses werden für jedes Element einzeln gesetzt, wenn es in Register 1 gleich dem in Register 2 ist.
FCMPEQ32	Ungleich-Test mit 32-Bit Partitionierung. Die Bits des Ergebnisses werden für jedes Element einzeln gesetzt, wenn es in Register 1 gleich dem in Register 2 ist.

#### Edge-Erkennungsbefehle:

EDGE8	Produziert eine 8-Bit Maske.
EDGE8L	Produziert eine 8-Bit Maske, Little Endian.
EDGE16	Produziert eine 4-Bit Maske.
EDGE16L	Produziert eine 4-Bit Maske, Little Endian.
EDGE32	Produziert eine 2-Bit Maske.
EDGE32L	Produziert eine 2-Bit Maske, Little Endian.

## Anhang C MMX-Befehle

PADDB	Packed Byte Addition
PADDW	Packed Word Addition
PADDD	Packed Double Word Addition
PSUBB	Packed Byte Subtraktion
PSUBW	Packed Word Subtraktion
PSUBD	Packed Double Word Subtraktion
PMULLW	Multiplikation von Packed Word Daten mit Vorzeichen, Speicherung nur der untern 32 Bit der Lösung
PMULHW	Multiplikation von Packed Word Daten mit Vorzeichen, Speicherung nur der oberen 32 Bit der Lösung
PMULADD	Multiplikation und Addition von Word Daten zu Double Word Daten.

### Saturierende Befehle ohne Vorzeichen:

PADDUSB	Packed Byte Addition mit Saturierung ohne Vorzeichen
PADDUSW	Packed Word Addition mit Saturierung ohne Vorzeichen
PSUBUSB	Packed Byte Subtraktion mit Saturierung ohne Vorzeichen
PSUBUSW	Packed Word Subtraktion mit Saturierung ohne Vorzeichen

### Saturierende Befehle mit Vorzeichen:

PADDSB	Packed Byte Addition mit Saturierung und Vorzeichen
PADDSW	Packed Word Addition mit Saturierung und Vorzeichen
PSUBSB	Packed Byte Subtraktion mit Saturierung und Vorzeichen
PSUBSW	Packed Word Subtraktion mit Saturierung und Vorzeichen

### Die Pack – Befehle:

PACKSSWB	Packen von Double Word (32-Bit) nach Word (16-Bit) Daten, mit Vorzeichen und Saturierung.
PACKSSDW	Packen von Double Word (32-Bit) nach Byte (8-Bit) Daten , mit Vorzeichen und mit Saturierung.
PACKUSWB	Packen von Double Word (32-Bit) nach Byte (8-Bit) Daten, mit Saturierung und ohne Vorzeichen.

### Die Entpack – Befehle:

PUNPCKHBW	Entpacken von Byte (höherwertiger Anteil) in Word-Daten
PUNPCKHWD	Entpacken von Word (höherwertiger Anteil) in Doubleword-Daten
PUNPCKHDQ	Entpacken von Double Word (höherwertiger Anteil) in ein Quad Word
PUNPCKLBW	Entpacken von Byte (niederwertiger Anteil) in Word
PUNPCKLWD	Entpacken von Word (niederwertiger Anteil) in Double Word
PUNPCKLDQ	Entpacken von Double Word (niederwertiger Anteil) in ein Quad Word

MMX enthält auch eine Reihe von Befehlen für Vergleichsoperationen auf den Packed Datentypen.

PCMPEQB	Vergleichen von Byte –Daten auf Gleichheit
PCMPEQW	Vergleichen von Word – Daten auf Gleichheit
PCMPEQD	Vergleichen von Double Word –Daten auf Gleichheit
PCMPGTB	Größer-als-Vergleich auf vorzeichenbehafteten Byte - Daten
PCMPGTW	Größer-als-Vergleich auf vorzeichenbehafteten Word - Daten
PCMPGTD	Größer-als-Vergleich auf vorzeichenbehafteten Double Word - Daten

Die logischen Operationen arbeiten natürlich immer auf den vollen 64-Bit Quad Words.

POR	OR – Verknüpfung
PAND	AND – Verknüpfung
PANDN	NAND –Verknüpfung
PXOR	XOR - Verknüpfung

**Shift – Operationen:**

PSLLW	Logisches Shift-Links auf Word Daten
PSLLD	Logisches Shift-Links auf Double Word Daten
PSRLW	Logisches Shift-Rechts auf Word Daten
PSRLD	Logisches Shift-Rechts auf Double Word Daten
PSRAW	Arithmetisches Shift-Rechts auf Word Daten
PSRAD	Arithmetisches Shift-Rechts auf Double Word Daten
PSLLQ	Logisches Shift-Links auf Quad Word Daten
PSRLQ	Logisches Shift-Rechts auf Quad Word Daten

**Transferbefehle:**

MOVD	Transfer von Daten aus MMX – Registern in die Standard – Register, an Speicherstellen, so wie umgekehrt von Speicherstellen in MMX – Register oder aus Standard – Registern in MMX – Register.
MOVQ	Transfer aus dem Speicher in ein MMX – Register und umgekehrt. Sowie von einem in ein anderes MMX – Register.

Befehl um den MMX – Status zu löschen:

EMMS	MMX – Status löschen, alle Register – Tags werden auf Null gesetzt.
------	---

## Anhang D AMD 3DNow! Befehle

### Integerbefehle:

PAVGUSB	Bildet den Durchschnitt von Packed Byte Daten ohne Vorzeichen
PFADD	Packed Floating-Point Addition
PFSUB	Packed Floating-Point Subtraktion
PFSUBR	Packed Floating-Point Subtraktion mit vertauschten Quell- und Zieloperanden.
PFACC	Packed Floating-Point Accumulate
PFCMPGE	Größer-oder-gleich-Vergleich von Packed Floating-Point Daten
PFCMPGT	Größer -Vergleich von Packed Floating-Point Daten
PFCMPEQ	Vergleichen von Packed Floating-Point auf Gleichheit
PFMIN	Liefert die beiden kleineren Werte von Packed Floating-Point
PFMAX	Liefert die beiden größeren Werte von zwei Packed Floating-Point Daten
PI2FD	Konvertiert Packed Double Word 32-bit Integer zu Packed Floating-Point
PF2ID	Konvertiert Packed Floating-Point zu 32-bit Packed Double Word Integer Daten
PFRCP	Befehl der den Kehrwert eines Packed Floating-Point approximiert (auf 14 Bit genau).
PFRSQRT	Befehl der die Wurzel des Kehrwerts eines Packed Floating-Point approximiert auf 15 Bit genau).
PFMUL	Packed Floating-Point Multiplikation
PFRCPIT1	Startet eine Iteration zur genauen Kehrwert - Berechnung eines Packed Floating-Point (Genauigkeit 24-Bit)
PFRSQIT1	Startet eine Iteration zur genauen Berechnung des Kehrwerts der Wurzel eines Packed Floating-Point (Genauigkeit 24-Bit)
PFRCPIT2	Schließt die Iteration zur Kehrwert / Wurzel Berechnung (Genauigkeit 24-Bit)
PMULHRW	Packed Word (Integer) Multiplikation mit Aufrunden (soll genauer sein als PMULHW)
FEMMS	Schneller Wechsel zwischen MMX und Gleitkomma Rechnung
PREFETCH / PREFETCHW	Prefetch von mindestens einer 32-byte Line in den Level1 Daten Cache

### Die neuen Befehle für den Gleitkommatentyp:

PF2IW	Konvertiert Packed Floating-Point zu Word Daten, mit Saturierung entsprechend dem Vorzeichen.
-------	---

PFNACC	Subtrahiert den oberen Floating-Point in einem MMX – Register von dem unteren und subtrahiert dann den oberen Teil eines weiteren Packed Floating-Point in einem anderen Register.
PFPNACC	Subtrahiert den oberen Floating-Point in einem MMX – Register von dem unteren und addiert dann den oberen Teil eines weiteren Packed Floating-Point in einem anderen Register.
PI2FW	Konvertiert Packed Word zu Packed Floating-Point mit Vorzeichen.
PSWAPD	Vertauscht den oberen und den unteren Teil eines Packed Floating-Point.

Die Integer SIMD - Verarbeitung wurde durch folgende Befehle erweitert:

MASKMOVQ	Speichert Packed Byte, wobei der Cache umgangen und eine Maske für bedingtes Speicher verwendet wird.
MOVNTQ	Speichert ein MMX – Register (egal welcher Datentyp) unter Umgehung des Caches.
PAVGB	Berechnet das arithmetische Mittel von Packed Byte in Registern oder im Speicher.
PAVGW	Berechnet das arithmetische Mittel von Packed Word in Registern oder im Speicher.
PEXTRW	Konvertiert durch einen Immediate maskiert ein Word aus einem Packed Word in ein 32-Bit Integerregister.
PINSRW	Konvertiert ein Word aus einem 32-Bit Integerregister an eine beliebige Position in einem Packed Word.
PMAXSW	Bestimmt das Maximum in zwei Register mit Packed Word, mit Vorzeichen.
PMAXUB	Bestimmt das Maximum in zwei Register mit Packed Byte, ohne Vorzeichen.
PMINSW	Bestimmt das Minimum in zwei Register mit Packed Word, mit Vorzeichen.
PMINUB	Bestimmt das Minimum in zwei Register mit Packed Byte, ohne Vorzeichen.
PMOVMSKB	Erstellt eine Byte – Maske aus den most significant Bits eines Packed Byte und legt sie in einem Integerregister ab.
PMULHUW	Multiplikation von Packed Word Daten, nur die oberen 16-Bit des 32-Bit Ergebnisses werden gespeichert.
PREFETCHNTA	Kopiert Daten in den Prozessocache, mit möglichst geringer Nutzung der Level 1 und 2 Caches.
PREFETCHT0	Kopiert Daten in alle Ebenen des Prozessocache.
PREFETCHT1	Kopiert Daten in den Prozessocache, ohne den Level 0 Cache zu verwenden.
PREFETCHT2	Kopiert Daten in den Prozessocache, ohne die Level 1 und Level 0 Caches zu verwenden.
PSADBW	Berechnet die Summe der absoluten Differenzen zwischen Packed Byte Daten.

PSHUFW	Permutiert Packed Word Daten entsprechend einem 8-Bit Immediate.
SFENCE	Erzwingt bei Out-of-Order Ausführung von Befehlen, die In-Order Ausführung.

## Anhang E SSE Befehle

Die Integerbefehle, die MMX erweitern:

PAVGB	Berechnet das arithmetische Mittel für Packed Byte Daten.
PAVGW	Berechnet das arithmetische Mittel für Packed Word Daten.
PMAXUB	Berechnet das Maximum der Packed Byte Daten der Quell- und Zieloperanden.
PMAXSW	Berechnet das Maximum der Packed Word Daten der Quell- und Zieloperanden.
PMINUB	Berechnet das Minimum der Packed Byte Daten der Quell- und Zieloperanden.
PMINSW	Berechnet das Minimum der Packed Word Daten der Quell- und Zieloperanden.
PMULHUW	Packed Word Multiplikation ohne Vorzeichen.
PSADBW	Berechnet die Summe der Differenzen zwischen Packed Byte Daten.
PEXTRW	Kopiert Word Daten aus einem Packed Word in den unteren Teil eines 32-Bit Integerregisters. Ausgewählt wird mit einem immediate Operand als Maske.
PINSRW	Kopiert den unteren Teil eines 32-Bit Integerregisters oder 16-Bit aus einer Speicherstelle in einen beliebigen Teil eines Packed Word.
PMOVMASKB	Berechnet einen 8-Bit Wert aus den most significant Bits jedes Wertes eines Packed Byte und legt ihn in einem 32-Bit Integerregister ab.
PSHUFW	Permutiert die Daten aus einem Packed Word, mit einem Immediate als Maske.

Die arithmetischen Befehle:

ADDPS	Addition von Packed Floating-Point Daten.
ADDSS	Addiert zwei 32-Bit Gleitkommazahlen in den SSE – Registern. Die oberen 96-Bit der Register werden einfach mit kopiert.
SUBPS	Subtraktion von Packed Floating-Point Daten.
SUBSS	Subtraktion zwei 32-Bit Gleitkommazahlen in den SSE – Registern. Die oberen 96-Bit der Register werden einfach mit kopiert.
MULPS	Multiplikation von Packed Floating-Point Daten.
MULSS	Multiplikation zwei 32-Bit Gleitkommazahlen in den SSE – Registern. Die oberen 96-Bit der Register werden einfach mit kopiert.
DIVPS	Division von Packed Floating-Point Daten.
DIVSS	Division zwei 32-Bit Gleitkommazahlen in den SSE – Registern. Die oberen 96-Bit der Register werden einfach mit kopiert.

Die Befehle für Konvertierungen ermöglichen Datenaustausch zwischen den neuen 128-Bit SSE - Registern, den 32-Bit Integerregistern und den 64-Bit MMX – Registern.

CVTPI2PS	Konvertiert Packed Double Word aus einem MMX – Register in die unteren Werte
----------	--

	eines Packed Floating Point. Der Rest des Registers wird nicht verändert.
CVTSI2SS	Konvertiert eine Double Word aus einem MMX – Register in den unteren Wert eines Packed Floating Point. Der Rest des Registers wird nicht verändert.
CVTPS2PI	Konvertiert die unteren Werte eines Packed Floating Point in eine Packed Double Word eines MMX - Registers.
CVTTPS2PI	Wie oben nur, dass bei einer ungenauen Konvertierung das Ergebnis abgeschnitten wird.
CVTSS2SI	Konvertiert die untersten Wert eines Packed Floating Point in ein 32-Bit Integerregister.
CVTTSS2SI	Wie oben nur, dass bei einer ungenauen Konvertierung das Ergebnis abgeschnitten wird.

### Die logischen Operationen:

ANDPS	AND auf Packed Floating Point.
ANDNPS	AND und Negation auf Packed Floating Point.
ORPS	OR auf Packed Floating Point.
XORPS	XOR auf Packed Floating Point.

### Die Permutationsbefehle :

SHUFPS	Ermöglicht eine beliebige Permutation eines Packed Floating Point. Ein Immediate dient als Maske.
UNPCKHPS	Mischt die oberen Hälften von zwei Packed Floating Point.
UNPCKLPS	Mischt die unteren Hälften von zwei Packed Floating Point.

Die Vergleichsbefehle erzeugen als Ergebnis einen 32-Bit Maske, die dann mit den logischen Operationen für bedingtes Kopieren verwendet werden kann.

CMPPS	Vergleicht zwei Packed Floating Point, mit einem Immediate als Prädikat. Es gibt 12 mögliche Werte für das Prädikat: <ul style="list-style-type: none"> <li>• Gleich</li> <li>• Kleiner-als</li> <li>• Kleiner-oder-gleich</li> <li>• Größer-als</li> <li>• Größer-oder-gleich</li> <li>• Ungleich</li> <li>• Nicht-kleiner-als</li> <li>• Nicht-kleiner-oder-gleich</li> <li>• Nicht-größer</li> <li>• Nicht-größer-oder-gleich</li> <li>• Geordnet</li> <li>• Ungeordnet</li> </ul>
CMPSS	Vergleicht wie CMPPS, nur mit den unteren Floating Point.
COMISS	Vergleich die beiden unteren Elemente in zwei Packed Floating Point und setzt das EFLAGS-Register.

UCOMISS	Vergleich die beiden unteren Elemente in zwei Packed Floating Point und setzt das EFLAGS-Register. Das Ergebnis wird nur ungültig, wenn der Operanden ein sNaN ist.
---------	---

### Die geometrischen Befehle von SSE:

RCPPS	Berechnet den Kehrwert eines Packed Floating Point.
RCPSS	Berechnet den Kehrwert des untersten Elements in einem Packed Floating Point.
RSQRTPS	Berechnet den approximierten Kehrwert der Wurzel eines Packed Floating Point.
RSQRTSS	Berechnet den approximierten Kehrwert der Wurzel des untersten Elements eines Packed Floating Point.

### SSE 2 – Befehle:

#### Cachesteuerungsbefehle:

CLFLUSH	Schreibt einen Operanden in den Speicher und setzt seine Cache – Einträge als ungültig, auf allen Cacheebenen.
LFENCE	Ermöglicht das Serialisieren von Ladenoperationen.
MASKMOVDQU	Schreibt über eine Maske ausgewählte Bytes aus einem SSE – Register direkt in den Speicher.
MFENCE	Ermöglicht das Serialisieren von Laden- und Speicheroperationen
MOVNTDQ	Schreibt eine Double Quad Word aus einem SSE – Register direkt in den Speicher.
MOVNTI	Schreibt ein Double Word aus einem 32-Bit Standard - Register direkt in den Speicher.
MOVNTPD	Schreibt zwei Packed Double Precision Floating Point aus einem SSE – Register direkt in den Speicher.
PAUSE	Verbessert die Performanz von Warteschleifen.

#### Datentransferbefehle:

MOVAPD	Kopiert 128-Bit Double Quad Word zwischen Speicher und einem SSE - Register, Speicherdaten müssen ausgerichtet (aligned) sein.
MOVHPD	Kopiert 64-Bit Double Precision Floating Point zwischen dem Speicher und der oberen Hälfte eines SSE – Registers.
MOVLPD	Kopiert 64-Bit Double Precision Floating Point zwischen der unteren Hälfte eines SSE – Registers.
MOVMSKPD	Erstellt eine Vorzeichen Maske aus den most-significant Bits von zwei Packed Double Precision Floating Point.
MOVSD	Kopiert 64-Bit Scalare 64-Bit Double Precision Floating Point zwischen Speicher und SSE – Register.
MOVUPD	Kopiert 128-Bit Double Quad Word zwischen Speicher und einem SSE

	- Register, Speicherdaten können unausgerichtet (unaligned) sein.
MOVDQ2Q	Kopiert Quad Word von einem SSE- in ein MMX – Register.
MOVDQA	Kopiert ausgerichtete (aligned) 128-Bit Integerdaten zwischen Speicher und SSE - Register.
MOVDQU	Kopiert unausgerichtete (unaligned) 128-Bit Integerdaten zwischen Speicher und SSE - Register.
MOVQ2DQ	Kopiert Quad Word von einem MMX- in ein SSE – Register.

### Konvertierungsbefehle:

CVTDQ2PD	Konvertiert Packed Double Word Integerwerte zu Packed Double Precision Floating Point.
CVTPD2DQ	Konvertiert Packed Double Precision Floating Point zu Packed Double Word Integerwerten.
CVTPD2PI	Konvertiert Packed Double Precision Floating Point zu Packed Double Word Integerwerten.
CVTPD2PS	Konvertiert Packed Double Precision Floating Point zu Packed Floating-Point.
CVTPI2PD	Konvertiert Packed Double Word Integerwerte zu Packed Double Precision Floating Point.
CVTPS2PD	Konvertiert Packed Floating Point zu Packed Double Precision Floating Point.
CVTSD2SI	Konvertiert Scalaren Double Precision Floating Point zu Double Word Integer.
CVTSD2SS	Konvertiert Scalar Double Precision Floating Point zu Scalar Single Precision Floating Point.
CVTSI2SD	Konvertiert Scalar Double Word Integerwerte zu Scalar Double Precision Floating Point.
CVTSS2SD	Konvertiert Scalar Single Precision Floating Point zu Scalar Double Precision Floating Point.
CVTTPD2DQ	Konvertiert mit Abschneiden Packed Double Precision Floating Point zu Packed Double Word Integerwerten.
CVTTPD2PI	Konvertiert mit Abschneiden Packed Double Precision Floating Point zu Packed Double Word Integerwerten.
CVTTSD2SI	Konvertiert mit Abschneiden Scalar Double Precision Floating Point zu Scalar Double Word Integerwerten.
CVTDQ2PS	Konvertiert Packed Double Word Integer zu Packed Single Precision Floating Point.
CVTPS2DQ	Konvertiert Packed Single Precision Floating Point zu Packed Double Word Integer.
CVTTPS2DQ	Konvertiert mit Abschneiden Packed Single Precision Floating Point

	zu Packed Double Word Integer.
--	--------------------------------

### Arithmetische Integerbefehle:

PADDQ	Addition von Packed Quad Word.
PMULUDQ	Multiplikation von Packed Double Word ohne Vorzeichen.
PSUBQ	Subtraktion von Packed Quad Word.

### Arithmetische Gleitkommabefehle:

ADDPD	Addition von Packed Double Precision Floating Point
ADDSD	Addition von Scalar Double Precision Floating Point
DIVPD	Division von Packed Double Precision Floating Point
DIVSD	Division von Scalar Double Precision Floating Point
MAXPD	Maximum ermitteln von Packed Double Precision Floating Point
MAXSD	Maximum ermitteln von Scalar Double Precision Floating Point
MINPD	Minimum ermitteln von Packed Double Precision Floating Point
MINSD	Minimum ermitteln von Scalar Double Precision Floating Point
MULPD	Multiplikation von Packed Double Precision Floating Point
MULSD	Multiplikation von Scalar Double Precision Floating Point
SQRTPD	Wurzeln errechnen von Packed Double Precision Floating Point
SQRTSD	Wurzeln errechnen von Scalar Double Precision Floating Point
SUBPD	Subtraktion von Packed Double Precision Floating Point
SUBSD	Subtraktion von Scalar Double Precision Floating Point

### Logische Verknüpfungen, Vergleichs- und Verschiebepbefehle:

ANDNPD	Bitweises logische AND NOT auf Packed Double Precision Floating Point.
ANDPD	Bitweises logische AND auf Packed Double Precision Floating Point.
ORPD	Bitweises logische OR auf Packed Double Precision Floating Point.
XORPD	Bitweises logische XOR auf Packed Double Precision Floating Point.
PSHUFD	Permutiert Packed Double Word.
PSHUFW	Permutiert Packed Word in den oberen 64-Bit eines SSE – Registers.
PSHUFLW	Permutiert Packed Word in den unteren 64-Bit eines SSE – Registers.
PSLLDQ	Logisches Verschieben von Packed Quad Word nach Links.

PSRLDQ	Logisches Verschieben von Packed Quad Word nach Rechts.
PUNPCKHQDQ	Gleichzeitiges Entpacken und Permutieren in den oberen 64-Bit eines Packed Quad Word.
PUNPCKLQDQ	Gleichzeitiges Entpacken und Permutieren in den unteren 64-Bit eines Packed Quad Word.
CMPPD	Vergleichen von Packed Double Precision Floating Point.
CMPSD	Vergleichen von Scalar Double Precision Floating Point.
COMISD	Geordneter Vergleich auf Scalar Double Precision Floating Point mit setzen der Flags im EFLAGS – Register.
UCOMISD	Ungeordneter Vergleich auf Scalar Double Precision Floating Point mit setzen der Flags im EFLAGS – Register.

### SSE 3 Befehle:

FISTTP	Konvertiert einen Standard Gleitkommawert zu einem Integer immer mit dem Rundungsmodus „chop“.
ADDSSUBPS	Addition und folgende Subtraktion auf Packed Single Precision Floating Point.
ADDSSUBPD	Addition und folgende Subtraktion auf Packed Double Precision Floating Point.
MOVSLDUP	Kopiert den 1 und 3 von Single Precision Floating Point aus dem Speicher in die oberen und unteren Teile eines SSE – Registers, in dem beide verdoppelt werden.
MOVSHDUP	Kopiert den 2 und 4 von Single Precision Floating Point aus dem Speicher in die oberen und unteren Teile eines SSE – Registers, in dem beide verdoppelt werden.
MOVDDUP	Kopiert einen Double Precision Floating Point aus dem Speicher in eine SSE – Register und verdoppelt ihn.
LDDQU	Lädt 128-Bit unausgerichtete (unaligned) Daten aus dem Speicher.
HADDPS	Addiert jeweils den ersten und zweiten, sowie den dritten und vierten Single Precision Floating Point aus zwei SSE – Registern und speichert das Ergebnis in einem SSE – Register.
HADDPD	Addiert die beiden Double Precision Floating Point aus zwei SSE – Registern und speichert das Ergebnis wieder in einem SSE – Register.
HSUBPS	Subtrahiert jeweils den ersten und zweiten, sowie den dritten und vierten Single Precision Floating Point aus zwei SSE – Registern und speichert das Ergebnis in einem SSE – Register.
HSUBPD	Subtrahiert die beiden Double Precision Floating Point aus zwei SSE – Registern und speichert das Ergebnis wieder in einem SSE – Register.
MONITOR	Setzt einen linearen Speicherbereich der überwacht werden soll und aktiviert die Monitor - Hardware.
MWAIT	Zusammen mit dem MONITOR – Befehl kann der Prozessor angehalten werden.