Heft 4
Architekturbeispiele

- Grundlagen der Rechnerarchitektur, Kapitel 9 -

9. Architekturbeispiele

Im folgenden werden zwei typische Architekturen überblicksmäßig beschrieben:

1. IA-32 - die Grundlage aller modernen PCs,
2. die Sparc-Architektur als Beispiel einer typischen RISC-Auslegung - und zwar (der Überschaubarkeit wegen) in ihrer ursprünglichen Form (als 32-Bit-Architektur).

Hierbei wollen wir uns auf jene Merkmale beschränken, die ausreichen, um einen ersten Eindruck vom jeweiligen Programmiermodell zu gewinnen. Des weiteren werden anhand von Tabellen und Abbildungen typische Merkmale weiterer Architekturen veranschaulicht.

Hinweis:
Nochmals gesagt - dieses Kapitel dient wirklich nur dem Überblick. Bitte nicht versuchen, auf Grundlage der folgenden Darstellungen in Einzelheiten eindringen zu wollen.

9.1. IA-32

Einzelheiten der Architektur haben wir immer wieder bevorzugt als Beispiele herangezogen und dabei näher erläutert. Somit können wir uns im folgenden auf einen knappen Überblick beschränken.

9.1.1. Prozessorstrukturen

Abbildung 9.1 zeigt ein Blockschaltbild des 486DX, Abbildung 9.2 eines des Pentium (P5). Beide Prozessortypen eignen sich gut als Lehrbeipiele, da sie in Aufbau und Wirkungsweise durchaus noch überschaubar sind[*)].

*): diese Prozessoren sind auch keineswegs veraltet: sie haben nach wie vor ihre Bedeutung in Embedded Systems - und teils auch in miniaturisierten Computern, die unter Windows CE laufen.

Abbildung 9.1 Der 486DX (Intel)

Abbildung 9.2 Der Pentium (P5: Intel)

Universelle Verarbeitungseinheit (CPU)
Die universelle Verarbeitungseinheit (Central Processing Unit CPU) des 486 hat eine Verarbeitungsbreite von 32 Bits. Sie umfaßt eine Verschiebeeinheit, einen Registersatz von acht allgemein nutzbaren 32-Bit-Registern, sechs 16-Bit-Segmentregistern, einem 32-Bit-Befehlszähler und einem 32-Bit-Flagregister sowie Schaltmittel für Datenverknüpfungen und Adreßrechnung.

Die Pentium-CPU enthält zwei Verarbeitungseinheiten, die mit U und V bezeichnet werden. Beide greifen auf einen gemeinsamen Registersatz zu. Jede Verarbeitungseinheit umfaßt

Schaltmittel zur Adreßrechnung sowie eine Arithmetik-Logik-Einheit (Arithmetic Logic Unit ALU) zur Datenverknüpfung. So ergeben sich zwei Befehls-Pipelines (U- und V-Pipeline). Diese Pipelines sind fünfstufig:

- Stufe 1: vorbeugendes Holen der Befehle (Instruction Prefetch); diese Stufe ist beiden Pipelines U, V gemeinsam.
- Stufe 2: Befehlsdecodierung,
- Stufe 3: Adreßrechnung,
- Stufe 4: Cache-Zugriff, Befehlsausführung,
- Stufe 5: Zurückschreiben in die CPU-Register.

Die Stufen 2...5 können in den Pipelines U, V unabhängig durchlaufen werden, so daß bis zu zwei Befehle in jedem Taktzyklus ausführbar sind (Superskalar-Organisation). Beide Pipelines sind aber nicht identisch. Vielmehr ist nur die U-Pipeline in der Lage, *alle* Maschinenbefehle auszuführen. Sie ist dazu mit einer leistungsfähigen Mikroprogrammsteuerung ausgerüstet, und die Ausführung elementarer Befehle wird direkt gesteuert (in einem einzigen Taktzyklus). Die V-Pipeline hat hingegen ausschließlich eine direkte ("fest verdrahtete") Steuerung. Sie kann deshalb *nur die elementaren* Befehle ausführen (also jene, deren Wirkung so einfach ist, daß sie ohne komplizierte Steuerabläufe erbracht werden kann).

Gleitkomma-Verarbeitungseinheit (FPU)
Die Gleitkomma-Verarbeitungseinheit (Floating Point Unit FPU) hat eine interne Verarbeitungsbreite von 80 Bits (erweitertes Gleitkommaformat gemäß Standard IEEE754-85), um die Genauigkeit der Rechenoperationen zu gewährleisten. Darüber hinaus werden folgende Datentypen unterstützt: Gleitkommazahlen von 32 und 64 Bits (einfache und doppelte Genauigkeit), ganze Binärzahlen (Integer-Zahlen) von 16, 32 und 64 Bits sowie 18-stellige vorzeichenbehaftete Dezimalzahlen (80 Bits lange gepackte BCD-Zahlen). Diese Datentypen werden beim Laden in das 80-Bit-Gleitkommaformat gewandelt. Umgekehrt können 80-Bit-Gleitkommawerte beim Speichern in die anderen Datentypen zurückgewandelt werden.

Neben den vier Grundrechenarten sind weitere häufig gebrauchte Funktionen als Maschinenbefehle vorgesehen (Quadratwurzel, Sinus, Cosinus usw.). Der Registersatz enthält acht Datenregister von 80 Bits Länge, die als Register-Stack organisiert sind.

Die FPU arbeitet parallel zur CPU; nach dem Anstoßen einer Gleitkommaoperation in der FPU beginnt die CPU bereits mit der Abarbeitung der Folgebefehle.

Die Pentium-FPU ist als 8-stufige Pipeline ausgeführt und enthält gesonderte Verarbeitungsschaltungen für Addition, Multiplikation und Division.

Segmentierungseinheit
Die Segmentierung ist die grundlegende Form der Speicherverwaltung. Sie ist nicht abschaltbar, aber es ist möglich, den Prozessor so einzurichten, daß der segmentierte Speicher als ein einziger linearer Adreßraum von 4 GBytes erscheint. Aus der Sicht der Befehls-

abarbeitung beruht die Segmentierung darauf, daß bei jedem Speicherzugriff eines der sechs Segmentregister angesprochen wird. Die Segmentierungseinheit enthält zu jedem Segmentregister ein Deskriptor-Cache-Register, in dem die erforderliche beschreibende Information bereitgehalten wird. Mit besonderen Rechen- und Auswerteschaltungen wird sowohl die jeweilige Adresse ermittelt als auch geprüft, ob der Zugriff tatsächlich stattfinden kann, d. h. ob er zulässig ist, ob Segmentgrenzen überschritten werden usw. Diese Prüfungen verlängern die Befehlsausführungszeit nicht.

Seitenverwaltungseinheit
Zusätzlich zur Segmentierung kann eine Virtualspeicherverwaltung nach dem Seitenprinzip (Paging) wirksam werden. Dazu wird der Adreßraum in feste Seiten von 4 kBytes Länge aufgeteilt. Weiterentwicklungen: (1) längere Seiten (2 oder 4 MBytes), (2) Verlängerung der physischen Adresse auf 36 Bits, (3) “feinfühlige” Beeinflußbarkeit des Caching und der Speicherzugriffsabläufe (Abschnitt 3.5.5.). Zur Seitenverwaltungshardware gehören vor allem Schnellspeicher, die die Adreßumsetzung durch assoziative Zugriffe deutlich beschleunigen (Translation Lookaside Buffers (TLBs)).

Cache(s)
Zu den Caches von 486 und Pentium siehe Heft 6.

Befehlsvorbereitungseinheit und Befehlsdecodierung
Die Befehlsvorbereitungseinheit (Instruction Prefetch Unit) stellt vorbeugend Anforderungen zum Befehlslesen. Beim 486 holt sie aus dem aktuellen Programmsegment Blöcke von 16 Bytes Länge in eine Befehlswarteschlange. Beim Pentium ist der einzelne Block 32 Bytes lang. Es sind jeweils zwei Pufferspeicher vorgesehen, die umschichtig gefüllt werden (Prinzip des Wechselpuffers). Die Befehle werden überlappend mit der Befehlsausführung decodiert (Befehlspipelining), wobei die verschiedenen Angaben eines Befehls (Vorsatzbytes, Operationscode, Modusbyte MOD-R/M usw.) gleichzeitig ausgewertet werden. Im Durchschnitt ist ein Befehl 3,2 Bytes lang, so daß die gesamte Befehlswarteschlange durchschnittlich 10 Befehle (486) bzw. 20 Befehle (Pentium) enthält.

Steuereinheit
Grundsätzlich werden die internen Abläufe im 486 von einem Mikroprogramm gesteuert, das in einem ROM gespeichert ist. Die Mikrobefehle wirken auf die CPU, auf die FPU und auf die Segmentierungseinheit. Viele Befehle haben nur einen einzigen Mikrobefehl, so daß deren Ausführung nur einen Taktzyklus erfordert. Die Pentium-Prozessoren haben eine Kombination von Mikroprogramm- und direkter Steuerung. Die FPU arbeitet weitgehend autonom und benötigt die Steuereinheit oft nur zu Beginn der Befehlsausführung (so erfordert es im 486 nur drei Taktzyklen, um die Addition, Multiplikation oder Division in der FPU zu starten).

Sprungzielpuffer
Beim 486 führt jedes Abweichen von der normalen Befehlsreihenfolge dazu, daß die Befehlswarteschlange vollständig gelöscht wird. Am häufigsten sind es Sprung- bzw. Verzweigungsbefehle, die eine solche Abweichung veranlassen. Die Pentium-Prozessoren enthalten deshalb einen Sprungzielpuffer (Branch Target Buffer BTB). Zur grundsätzlichen Wirkungsweise siehe Heft 5.

9.1.2. Betriebsarten

Realmodus (Real Mode)
Im Realmodus verhalten sich die Prozessoren wie 16-Bit-Mikroprozessoren der Typen 8086, 8088, 80186, 80188. Die Breite der allgemeinen Register, des Befehlszählers usw. ist auf 16 Bits begrenzt, und es werden nach einem vereinfachten Segmentierungsverfahren 20-Bit-Adressen erzeugt, wobei das einzelne Segment 64 kBytes umfaßt. Der Befehlsvorrat ist allerdings nicht auf den des ursprünglichen 8086-Prozessors beschränkt, sondern es können viele weitere Befehle genutzt werden, die in der Entwicklung der x86- Familie nach und nach hinzugefügt wurden, und zusätzliche IA-32-Register sind nutzbar. Im Realmodus gibt es keine Seitenverwaltung. Nach dem Hardware-Rücksetzen gelangt der Prozessor automatisch in den Realmodus. Das Umschalten in andere Betriebsarten erfolgt durch Setzen von Steuerbits in Registern.

Protected-Modus (Protected Virtual Address Mode)
Dies ist der Modus, in dem die 32-Bit-Architektur in vollem Umfang verfügbar ist. Die Operandenlänge gibt die tatsächlich genutzte Verarbeitungsbreite für Datentransporte und Operandenverknüpfungen an. Sinngemäß bezeichnet die Adreßlänge die für Adressierungszwecke tatsächlich genutzte Verarbeitungsbreite. Die Wahlmöglichkeiten: (1) 16-Bit-Verarbeitung/32-Bit-Verarbeitung, (2) 16-Bit-Adressierung/32-Bit-Adressierung. Moderne Software nutzt 32-Bit-Verarbeitung und 32-Bit-Adressieung (Beispiel: die Win32-API). Operanden- und Adreßlänge sind während des Programmablaufs von Befehl zu Befehl umschaltbar (über Vorsatzbytes).

V86-Modus (Virtueller 8086-Modus; Virtual-8086 Mode)
Dieser Modus ist eine Unterbetriebsart des Protected-Modus. Er ist vorgesehen, um mehrere unabhängige Programme für x86-Prozessoren (8086, 8088 usw.) in einer Umgebung des Protected-Modus ausführen zu können. Dazu werden die Multitasking-Vorkehrungen des Protected-Modus ausgenutzt. Im V86-Modus werden für die Abarbeitung von x86-Programmen virtuelle Maschinen gebildet, und zwar im Verbund von Hard- und Software (Virtual-8086 Monitor).

Systemverwaltungsmodus (System Management Mode SMM)
Der Systemverwaltungsmodus ist eine zusätzliche Betriebsart der P5- und P6-Prozessoren sowie verschiedenen Typen der 386- und 486-Klasse. Diese Betriebsart ist vorgesehen, um vollkommen transparent gegenüber allen Anwendungs- und Systemprogrammen zusätzliche Funktionen verwirklichen zu können. So ist es beispielsweise möglich, Programme auszuführen, die Funktionseinheiten je nach Bedarf ein- und ausschalten (stromsparende Betriebsweise), den Ladezustand des Akkus (beim portablen Computer) prüfen oder Sicherheitskontrollen ausführen (z. B. vom Nutzer die Eingabe eines Kennwortes anzufordern). Der Systemverwaltungsmodus wird grundsätzlich von der Hardware eingeleitet und durch einen besonderen Maschinenbefehl beendet.

9.1.3. Datentypen

Die elementaren Datentypen moderner Rechnerarchitekturen sind in Kapitel 1 eingehend beschrieben, so daß hier eine knappe Aufzählung ausreicht.

Numerische Datentypen:

- natürliche Binärzahlen (Ordinalzahlen) von 32, 16 und 8 Bits Länge,
- ganze Binärzahlen (Integer-Zahlen) von 64, 32, 16 und 8 Bits Länge,
- Gleitkommazahlen von 32, 64 und 80 Bits Länge,
- binär codierte Dezimalzahlen (ungepackte BCD-Zahlen) von 8 Bits Länge,
- gepackte binär codierte Dezimalzahlen (gepackte BCD-Zahlen) von 8 Bits und 80 Bits Länge,
- SIMD-Datenstrukturen von 64 und 128 Bits Länge (MMX, 3DNow, SSE).

Nichtnumerische Datentypen:

- einzelne Bytes, Worte und Doppelworte,
- Ketten (Strings) als fortlaufende Folgen von Bytes, Worten bzw. Doppelworten; max. Länge 4 GBytes,
- Bitketten (Bitstrings) als fortlaufende Folge von Bits; max. Länge 4 GBits.

Deskriptive Angaben
Deskriptive Angaben sind zu folgenden Zwecken vorgesehen:

- Adressierung und Segmentauswahl: Adressenzeiger, Segmentselektoren,
- Beschreibung von Segmenten: Segmentdeskriptoren,
- Beschreibung von Eintrittspunkten in Programme bzw. Tasks: Gate-Deskriptoren, Taskzustandssegmente (TSS),
- Auswahl und Verwaltung der Deskriptoren: Deskriptortabellen,
- Seitenverwaltung: Seitenverzeichnisse, Seitentabellen.

9.1.4. Registersatz

Abbildung 9.3 zeigt die herkömmliche Registerstruktur, wie sie sich für den Programmierer darstellt, Abbildung 9.4 veranschaulicht die Register der SIMD-Erweiterungen (MMX, 3DNow, SSE).

Abbildung 9.3 Die Registerstruktur aus der Sicht des Programmierers (am Beispiel der 486-Prozessoren)

Abbildung 9.4 Die Registerstrukturen der SIMD-Erweiterungen

Allgemeine Register
Acht allgemeine Register zu 32 Bits können sowohl für arithmetische und logische Operationen als auch für Adressierungszwecke verwendet werden.

Unterschiede gegenüber anderen Architekturen:

- **#** es sind keine "echten" Universalregister (wie sie beispielsweise in den typischen RISC-Architekturen vorgesehen sind). Alle Register sind zwar zu Adressierungszwecken nutzbar und können die Resultate der meisten arithmetischen und logischen Operationen aufnehmen, aber bestimmte Befehle greifen unmittelbar auf bestimmte Register zu (implizite Registernutzung). Beispielsweise verwenden String-Befehle die Register ECX (für die String-Länge), ESI (für die Adressierung des Quell-String; Source Index) und EDI (für die Adressierung des Resultat-String; Destination Index).
- **#** die Register werden nicht ausschließlich mit der vollen Breite von 32 Bits genutzt. Vielmehr werden sie gemäß der jeweiligen Operanden- bzw. Adreßlänge als 16- bzw. 32-Bit-Register verwendet. In vier Registern können die beiden Bytes des niederwertigen Wortes von entsprechenden Befehlen einzeln angesprochen werden.

Diese Besonderheiten erklären sich aus der Abwärtskompatibilität zur x86-Architektur. Vom 8086 an sind die allgemeinen Register 16 Bits lang (AX, BX, CX, DX, SI, DI, BP, SP). Um die Kompatibilität zu wahren, lassen sich die niederen 16 Bits der 32-Bit-Register als 16-Bit-Register ansprechen. Des weiteren sind vier 16-Bit-Register nochmals byteweise zugänglich, so daß praktisch acht einzelne Byteregister verfügbar sind (AL, AH, BL, BH, CL, CH, DL, DH).

Flagregister
Bedingungen (wie z. B. Übertrag, Resultat = 0 usw.) und Modusbits werden in einem 32-Bit-Flagregister EFLAGS gehalten. (In der Intel-Terminologie werden sowohl Bedingungsanzeigen als auch Steuerbits als Flags bezeichnet.)

Segmentregister
Es sind sechs Segmentregister vorgesehen (CS, SS, DS, ES, FS, GS). Jedes dieser Register ist 16 Bits lang und nimmt einen Segmentselektor auf. Zur Nutzung:

- **#** CS: aktuelles Programmsegment (Code Segment),
- **#** SS: Stacksegment,
- **#** DS, ES, FS, GS: Datensegmente.

Welches Segmentregister für welchen Zugriff verwendet wird, ist in der Architektur definiert (z. B. CS beim Befehlslesen, SS bei Stackzugriffen, DS bei den meisten Datenzugriffen). Diese implizite Segmentzuordnung kann aber, was Datenzugriffe angeht, programmseitig übergangen werden. Dazu sind besondere Vorsatzbytes (Segment Override Prefixes) vorgesehen, die den betreffenden Befehlen vorangestellt werden. So ist es möglich, (1) mit mehreren Segmenten gleichzeitig zu arbeiten und (2) auch Datenzugriffe zum Programmsegment bzw. zum Stacksegment zu führen (Lesen von Konstanten aus dem

aktuellen Programm; relative Adressierung im Stack, z. B. um Parameter zu holen oder Ergebnisse zurückzugeben).

Register der Gleitkomma-Verarbeitungseinheit (FPU-Register)
Die Gleitkomma-Verarbeitungseinheit (FPU) enthält acht Datenregister von 80 Bits Länge (R0 bis R7). Dieser Registersatz ist als Stack organisiert, wobei die einzelnen Register wahlfrei zugänglich sind (relative Registeradressierung im Stack). Jedes Datenregister hat zwei zusätzliche Kennzeichnungsbits (TAG-Bits), in denen der Benutzungszustand (Belegt/Frei usw.) codiert ist. Diese Bits sind zwecks programmseitiger Zugänglichkeit in einem TAG-Wort zusammengefaßt, das als besonderes Register ansprechbar ist. Weiterhin hat die FPU ein 16-Bit-Steuerregister (das unter anderem den Stackzeiger für die oberste Position des Datenregister-Stack enthält), ein 16-Bit-Zustandsregister und zwei 48-Bit-Register, die vorgesehen sind, um Informationen zur Ausnahmebehandlung zu speichern (Befehlszeiger, Datenzeiger).

System-Adreßregister
Es sind zwei 48-Bit-Register vorgesehen, die jeweils eine 32-Bit-Adresse und eine 16-Bit-Längenangabe enthalten, und zwar (1) das Adreßregister für die globale Deskriptortabelle (Global Descriptor Table Register GDTR) und (2) das Adreßregister für die Interrupt-Deskriptortabelle (Interrupt Descriptor Table Register IDTR).

System-Segmentregister
Es sind zwei 16-Bit-Segmentregister vorgesehen, die jeweils einen Segmentselektor enthalten, und zwar (1) das *Segmentregister für die aktuelle Task* (Task Register TR) und (2) das *Segmentregister für die aktuelle lokale Deskriptortabelle* (Local Descriptor Table Register LDTR).

Das Register TR enthält den Segmentselektor für das Taskzustandssegment (TSS) der aktuellen Task. Das Register LDTR enthält den Segmentselektor für die aktuelle lokale Deskriptortabelle (LDT). Beide Segmentselektoren beziehen sich auf die globale Deskriptortabelle GDT.

Steuerregister
Für die Systemsteuerung wurden mit dem 386-Prozessor vier 32-Bit-Steuerregister CR0...-CR3 eingeführt. CR0 enthält allgemeine Systemsteuerbits. CR1 ist für künftige Erweiterungen reserviert. CR2 und CR3 werden für die Seitenverwaltung verwendet. Von der P5-Klase an gib es ein fünftes Steuerregister CR4, das Steuerbits für neu hinzugekommene Funktionen enthält.

Fehlersuchregister
Acht 32-Bit-Fehlersuchregister (Debugging-Register) DR0...DR7 sind vorgesehen, um das Ausprüfen von Software (Debugging) zu unterstützen. In diesen Registern können bis zu vier Vergleichsstopadressen eingestellt werden.

Maschinenspezifische Register, Testregister
Diese Register gehören an sich nicht zur Architektur. Sie wurden zunächst vorgesehen, um das Testen der TLB- und Cache-Hardware zu unterstützen (Testregister). Vom Pentium an

heißen solche Register MSRs (Model Specific Registers). In solchen Registern hat man all das untergebracht, was programmseitig zugänglich sein muß, aber über die herkömmlichen IA-32-Festlegungen hinausgeht (Beispiele: die MTRRs und die PAT).

Der Maschinenzeitzähler (Time-Stamp Counter)
Dieser Zähler wird beim Rücksetzen der Hardware auf Null gestellt und ansonsten mit jedem (internen) Prozessortakt um Eins weitergezählt. Es gibt eigens eine Maschinenbefehl, um den Zählerinhalt zu lesen. Einführung: vom Pentium an. Auslegung: Binärzähler von 64 Bits Länge (damit ist garantiert, daß der Zählwert auch dann nicht überläuft (von FF...FFH nach Null), wenn der Prozessor über Jahre hinweg nicht zurückgesetzt wird).

Die Leistungsmeßzähler (Performance Monitoring Counters)
Vom Pentium an wurden maschinenspezifische Leistungsmeßzähler vorgesehen. Die Auslegung in P5- und P6-Prozessoren: zwei Binärzähler zu 40 Bits Länge. Jeder Zähler kann entweder Ereignisse zählen oder die Dauer eines bestimmten Zustandes messen (durch Auszählen mit dem internen Prozessortakt). Betriebartensteuerung, Ereignisauswahl usw.: über weitere modellspezifische Register (MSRs). Eine kleine Auswahl von Leistungsdaten, die derart gemessen werden können (P6):

- Anzahl der Treffer im Dtencache,
- Anzahl der Befehlslesevorgänge,
- Anzahl der Lesezugriffe auf den L2-Cache,
- Anzahl der ausgeführten Gleitkommaoperationen,
- Anzahl der ausgeführten Multiplikationsbefehle,
- Anzahl der Speicherzugriffe auf nicht-integrale Adressen (Misaligned Accesses),
- Anzahl der empfangenen Interrupts,
- Zeit, in der Interrupts nicht zugelassen sind (in Prozessortakten),
- Anzahl der decodierten Befehle,
- Anzahl der erledigten (retired) Befehle.

MMX-Register
Die MMX-Erweiterung nutzt die Gleitkomma-Datenregister, und zwar deren niederwertige 64 Bits. Bezeichnung in dieser Betriebsart: MM0...MM7. Registerbelegung: mit Datenstrukturen gemäß Abschnitt 1.4. (Abbildung 9.4 deutet an, daß jedes Register z. B. mit 2 32-Bit-Worten belegt sein kann).

*): und auch die 3DNow-Erweiterung (AMD).

Der Grund für diese Lösung: man wollte die MMX-Erweiterung vollkommen transparent zur bisherigen Architektur einführen. Vor allem die Systemsoftware solle vom Vorhandensein der MMX-Erweiterung gar nichts merken (betrifft das Retten von Zuständen, die Taskumschaltung usw.). Hätte man hingegen zusätzliche Register vorgesehen, so müßte die Betriebssysteme um entsprechende Verwaltungsfunktionen erweitert werden. Allerdings muß der Programmierer, der die MMX-Vorkehrungen ausnutzen will, genau achtgeben (vor allem darauf, ob er gerade mit Gleitkommazahlen oder mit MMX-Daten arbeitet). Der

zweite Nachteil: beim Wechseln zwischen MMX- und Gleitkommaverarbeitung ist es stets notwendig, die jeweiligen Registerbelegungen in den Arbeitsspeicher zu retten bzw. die Register jedesmal neu zu laden (Leistungsverlust).

SSE-Register
Es sind 8 128-Bit-Register vorgesehen (XMM0...XMM7). Belegung:

- SSE1: vorzugsweise mit jeweils 4 Gleitkommazahlen zu 32 Bits (wie in Abbildung 9.4 angedeutet),
- SSE2: mit Datenstrukturen gemäß Abbildung 1.33.

9.1.5. Befehlsformate

Es sind verschiedene Befehlsformate und Befehlslängen vorgesehen. Abbildung 9.5 zeigt den Befehlsaufbau. Jeder Befehl wird grundsätzlich durch einen Operationscode bestimmt, der ein oder zwei Bytes lang ist. Ihm können Vorsatzbytes (Prefix Bytes) vorangestellt werden. Dem Operationscode können ein Modusbyte (MOD-R/M) und ein Skalierungs-Index-Byte (SIB) nachfolgen, um Adressierungsart und Registerauswahl zu steuern. An diese Bytes können sich Adressen- und Direktwertangaben anschließen, die jeweils 1, 2 oder 4 Bytes lang sein können.

Abbildung 9.5 Der grundsätzliche Befehlsaufbau. IA-32 ist eine wirkliche CISC-Architektur...

Die meisten Befehle, in denen kein Operand angegeben ist, belegen lediglich ein Byte. Befehle mit einem Operanden sind üblicherweise zwei Bytes lang. Darüber hinaus werden die Befehle wirklich “komplex” (vgl. anhand der Abbildungen 9.10 und 9.18 die Befehlsformate typischer RISC-Architekturen). Maximale Befehlslänge: 16 Bytes.

9.1.6. Befehlsliste

Der Befehlsvorrat umfaßt Befehle für folgende Aufgaben:

- Transporte,
- arithmetische Verknüpfungen,
- logische Verknüpfungen,
- Verschieben/Rotieren,
- Stringoperationen,
- Bitoperationen,
- Steuerung des Programmablaufs,
- Unterstützung höherer Programmiersprachen,
- Betriebssystemunterstützung,
- Steuerung des Prozessors.

Befehlsübersicht

Der Befehlsvorrat (einschließlich der Vorsatzbytes) ist nachfolgend angegeben. Stand der Technik: bis einschließlich Pentium III (wobei wir Spitzfindigkeiten - z. B. der Abwärtskompatibilität - vernachlässigen). Die Befehle werden mit mnemonischen Abkürzungen*) gemäß der Intel-Dokumentation bezeichnet. Die zugehörige deutsche Bezeichnung strebt eine knappe und treffende Charakterisierung der jeweiligen Befehlswirkung an. Erweiterungen (MMX, 3DNow, SSE) sind in den folgenden Tabellen nicht enthalten. Sie werden anschließend gesondert beschrieben.

*): das sind jene Abkürzungen, wie sie der Programmierer in Assemblerprogrammen hinschreibt. (Bedeutet anders herum: wer nicht in Assember programmiert, braucht diese Bezeichnungen gar nicht.)

Hinweis:

Befehlssätze mit mehreren hundert Befehlen - wovon viele keineswegs einfache Wirkungen haben - kann man kaum noch in Kurzform darstellen. Wir versuchen im folgenden, einen ersten Eindruck von der Grundausstattung der IA-32-Prozessoren zu vermitteln. Viele dieser Befehlswirkungen sind durchaus noch “einfach”. So wird wohl jeder eine deutliche Vorstellung davon, haben, was ein Datentransport oder eine Addition bewirken. Bei den Erweiterungen wird es aber schon etwas schwieriger. Wir beschreiben deshalb die Wirkungen der typischen MMX-Befehle etwas ausführlicher (um an noch überschaubaren Beispielen zu zeigen, was SIMD-Verarbeitung eigentlich bedeutet), beschränken uns aber des weiteren auf eine knappe Aufzählung in Tabellenform. Dabei verzichten wir auf die mnemonischen Bezeichnungen aus der Dokumentation. (Es gibt sehr viele, und sie sind auch kaum noch selbsterklärend. Was bedeutet z. B. CVTPS2PD?)

Packed Single-Precision to Packed Double-Precision Floating Point Conversion = Datenumwandlung von n gepackten Gleitkommazahlen einfacher in solche doppelter Genauigkeit. Ein Befehl der SSE2-Erweiterung.

Vorsatzbytes (Prefix Bytes)

REP/ **REPE/** **REPZ**	Wiederholung von Stringbefehlen (bis Ende des String/ solange Stringelemente gleich/ solange Stringelement gleich A-Register)
REPNE/ **REPNZ**	Wiederholung von Stringbefehlen (solange Stringelemente ungleich/ solange Stringelement ungleich A-Register)
LOCK	Speicherzugriffe mit Busverriegelung
CS Segment Override	Datenzugriff über Segmentregister CS erzwingen (Zugriff zum Programmsegment)
SS Segment Override	Datenzugriff über Segmentregister SS erzwingen (Zugriff zum Stacksegment)
DS Segment Override	Datenzugriff über Segmentregister DS erzwingen
ES Segment Override	Datenzugriff über Segmentregister ES erzwingen
FS Segment Override	Datenzugriff über Segmentregister FS erzwingen
GS Segment Override	Datenzugriff über Segmentregister GS erzwingen
Operand Size Override	Operandenlänge wechseln (von 16 auf 32 Bits oder umgekehrt)
Address Size Override	Adreßlänge wechseln (von 16 auf 32 Bits oder umgekehrt)

Anwendungsbefehle der CPU

Binärarithmetik	
ADD	Addieren
ADC	Addieren mit Eingangsübertrag
INC	Erhöhen um 1
SUB	Subtrahieren
SBB	Subtrahieren mit Eingangsübertrag
DEC	Vermindern um 1
MUL	Multiplizieren vorzeichenlos
IMUL	Multiplizieren ganzzahlig
DIV	Dividieren vorzeichenlos
IDIV	Dividieren ganzzahlig
NEG	Zweierkomplement
CMP	Vergleichen arithmetisch
Dezimalarithmetik	
DAA	Dezimalausgleich nach Addition, gepackt
DAS	Dezimalausgleich nach Subtraktion, gepackt
AAA	Dezimalausgleich nach Addition, ungepackt
AAS	Dezimalausgleich nach Subtraktion, ungepackt
AAM	Entpacken Dezimalzahl nach Multiplikation
AAD	Wandeln ungepackte zweistellige Dezimalzahl vor Division

Logische Verknüpfungen	
NOT	Negation
AND	Konjunktion (UND)
OR	Disjunktion (ODER)
XOR	Antivalenz (Exklusiv-ODER)
TEST	Vergleichen logisch
Verschieben/Rotieren	
SAL/ SHL	Linksverschieben
SAR	Rechtsverschieben arithmetisch
SHR	Rechtsverschieben logisch
SHLD	Linksverschieben mit Erweiterung, logisch
SHRD	Rechtsverschieben mit Erweiterung, logisch
ROL	Linksrotieren
ROR	Rechtsrotieren
RCL	Linksrotieren über CF-Flag
RCR	Rechtsrotieren über CF-Flag
Einzelbitbefehle	
BT	Bit abfragen
BTS/BTR/BTC	Bit abfragen und setzen/löschen/wechseln
BSF	Position der niedrigstwertigen Eins
BSR	Position der höchstwertigen Eins

Transportbefehle	
MOV	Transportieren (Laden, Speichern usw.)
CMOVcc	bei erfüllter Bedingung **cc** transportieren
CPUID	Laden Prozessoridentifikation

Austausch- und Wandlungsbefehle	
BSWAP	Byteanordnung im Doppelwort wechseln
XCHG	Austauschen
CMPXCHG	Vergleichen und Austauschen
CMPXCHG8B	Vergleichen und Austauschen 8 Bytes
XADD	Austauschen und Addieren
XLAT/XLATB	Wandeln Byte über Tabelle
Elementare Stackbefehle	
PUSH	Operand auf Stack legen
PUSHA/PUSHAD	Alle allgemeinen Register auf Stack retten
POP	Operand vom Stack entnehmen
POPA/POPAD	Alle allgemeinen Register vom Stack aus einstellen
Elementare Typwandlungen	
CBW/CWDE	Wandeln Byte zu Wort/Wort zu Doppelwort
CWD/CDQ	Wandeln Wort zu Doppelwort/Doppelwort zu Quadwort
MOVZX	Laden mit Nullerweiterung
MOVSX	Laden mit Vorzeichenerweiterung
Unterstützung höherer Programmiersprachen	
ENTER	Stack-Frame für Prozedur aufbauen
LEAVE	Prozedur verlassen
BOUND	Feldgrenzen prüfen
SETcc	Byte gemäß Bedingung **cc** setzen

Stringbefehle	
LODS/LODSB/LODSW/LODSD	Laden Stringelement
STOS/STOSB/STOSW/STOSD	Speichern Stringelement
MOVS/MOVSB/MOVSW/MOVSD	Transportieren Stringelement
CMPS/CMPSB/CMPSW/CMPSD	Vergleichen zweier Stringelemente
SCAS/SCASB/SCASW/SCASD	Vergleichen Stringelement mit A-Register
REP STOS/STOSB/STOSW/STOSD	String füllen
REP MOVS/MOVSB/MOVSW/MOVSD	String transportieren
REPE/REPNE CMPS/CMPSB/CMPSW/CMPSD	Vergleichen zweier Strings auf Gleichheit/auf Ungleichheit
REPE/REPNE SCAS/SCASB/SCASW/SCASD	Suchen gleiches/ungleiches Element in String
Programmsteuerbefehle	
Jcc	Bedingte Verzweigung (gemäß Bedingung **cc**)
JCZX/JECZX	Verzweigen bei C-Register Null
JMP	Unbedingte Verzeigung (nah/fern)
CALL	Unterprogrammruf (nah/fern)
INTn/INT3/INTO	Aufruf Interruptbehandler
RET	Rückkehr aus Unterprogramm (nah/fern)
IRET/IRETD	Rückkehr aus Interruptbehandlung
LOOP/LOOPE/LOOPZ/LOOPNE/ LOOPNZ	Vermindern C-Register und Verzweigen solange gleich/ungleich Null

Adressierungsbefehle	
LEA	Laden der effektiven Adresse
LSS/LDS/LES/LFS/LGS	Laden Fernzeiger

Flagbefehle	
LAHF/SAHF	Flagbits nach AH laden/aus AH übernehmen
CLC/STC/CMC	Übertragsflag löschen/setzen/wechseln
CLD/STD	Richtungsflag löschen/setzen
PUSHF/PUSHFD	Flagregister auf Stack legen
POPF/POPFD	Flagregister vom Stack aus einstellen
Wirkungslose Befehle	
NOP	Keine Operation

Anwendungsbefehle der FPU

Transportbefehle	
FLD	Laden Gleitkommazahl
FST/FSTP	Speichern Gleitkommazahl/Speichern Gleitkommazahl und Stackfreigabe
FILD	Laden ganze Binärzahl (Wort/Doppelwort/Quadwort)
FIST/FISTP	Speichern ganze Binärzahl/Speichern ganze Binärzahl und Stackfreigabe
FBLD	Laden gepackte 80-bit-Dezimalzahl
FBSTP	Speichern gepackte 80-bit-Dezimalzahl und Stackfreigabe
FXCH	Austauschen Stackinhalte
FLD1/FLD2T/FLD2E/ FLDPI/- FLDLG2/FLDLN2/ FLDZ	Festwert laden (+1.0; ld(10); ld(e); π; lg(2); ln(2); + 0.0)

Elementare Arithmetik	
FADD/FADDP/FIADD	Addieren/Addieren und Stackfreigabe/Addieren ganze Binärzahl
FSUB/FSUBP/FISUB	Subtrahieren/Subtrahieren und Stackfreigabe/Subtrahieren ganze Binärzahl
FSUBR/FSUBPR/ FISUBR	Subtrahieren/Subtrahieren und Stackfreigabe/Subtrahieren ganze Binärzahl in umgekehrter Operandenfolge
FMUL/FMULP/ FIMUL	Multiplizieren/Multiplizieren und Stackfreigabe/Multiplizieren ganze Binärzahl
FDIV/FDIVP/FIDIV	Dividieren/Dividieren und Stackfreigabe/Dividieren durch ganze Binärzahl
FDIVR/FDIVRP/ FIDIVR	Dividieren/Dividieren und Stackfreigabe/ Dividieren ganze Binärzahl in umgekehrter Operandenfolge
Nicht-transzendente Funktionen	
FSQRT	Quadratwurzel
FSCALE	Exponentenänderung
FXTRACT	Exponent und Signifikand trennen
FPREM	Divisionsrest gemäß 8087/80287
FPREM1	Divisionsrest gemäß IEEE 754
FRNDINT	Runden ganzzahlig
FABS	Betrag
FCHS	Vorzeichenwechsel
Transzendente Funktionen	
FSIN	Sinus
FCOS	Cosinus
FSINCOS	Sinus und Cosinus
FPTAN	Tangens
FPATAN	Arcustangens
F2XM1	$2^x - 1$
FYL2X	y * ld(x)
FYL2X1	y * ld(x+1)

Vergleichsbefehle	
FCOM/FCOMP/-FCOMPP	Vergleichen/Vergleichen und Stackfreigabe/doppelte Stackfreigabe
FICOM/FICOMP	Vergleichen mit ganzer Binärzahl/Vergleichen mit ganzer Binärzahl und Stackfreigabe
FTST	Nulltest
FUCOM	Vergleichen ungeordnet
FUCOM/FUCOMP/ FUCOMPP	Vergleichen ungeordnet/Vergleichen ungeordnet und Stack-freigabe/doppelte Stackfreigabe
FXAM	Typabfrage
Steuerbefehle	
FINIT/FNINIT	Initialisieren FPU, mit/ohne Fehlerprüfung
FLDCW	Laden FPU-Steuerregister
FSTCW/FSTNCW	Speichern FPU-Steuerregister, mit/ohne Fehlerprüfung
FSTSW/FNSTSW	Speichern FPU-Zustandsregister, mit/ohne Fehlerprüfung
FSTSW AX/FNSTSW AX	Laden AX mit Zustandswort, mit/ohne Fehlerprüfung
FCLEX/FNCLEX	Löschen Ausnahmebedingungen, mit/ohne Fehlerprüfung
FLDENV	Laden FPU-Umgebung
FSTENV/FNSTENV	Speichern FPU-Umgebung, mit/ohne Fehlerprüfung
FSAVE/FNSAVE	Retten FPU-Gesamtzustand, mit/ohne Fehlerprüfung
FRSTOR	Einstellen FPU-Gesamtzustand
FINCSTP/FDECSTP	Erhöhen/Vermindern Stackzeiger
FFREE	Freigeben Datenregister
FNOP	Keine FPU-Operation
FWAIT	Prüfen auf anhängige FPU-Ausnahmebedingungen

Systembefehle

Selektorbezogene Befehle	
ARPL	Geforderte Privilegebene anpassen
LAR	Laden Zugriffsrechte
LSL	Laden Segmentlänge
VERR/VERW	Prüfen Leseberechtigung/Schreibberechtigung
Zugriffe zu Systemregistern	
LGDT/LIDT	Laden Systemadressenregister für globale Deskriptortabelle/ Interruptdeskriptortabelle (GDTR/IDTR)
SGDT/SIDT	Speichern Systemadressenregister für globale Des-kriptortabelle/Interruptdeskriptortabelle (GDTR/IDTR)
LLDT	Laden Systemsegmentregister für lokale Deskriptortabelle (LDTR)
SLDT	Speichern Systemsegmentregister für lokale Deskriptortabelle (LDTR)
LTR	Laden Taskregister (TR)
STR	Speichern Taskregister (TR)
MOV	Zugriffe zu Sonderregistern (Steuerregister, Fehlersuchregister, Testregister)
LMSW	Laden Maschinenzustandswort
SMSW	Speichern Maschinenzustandswort
RDMSR	Lesen modellspezifische Register
WRMSR	Schreiben modellspezifische Register
RDPMC	Lesen Leistungsmeßzähler
RDTSC	Lesen Maschinenzeitzähler

Ein- und Ausgabe	
IN	Eingabe von E-A-Port
OUT	Ausgabe zu E-A-Port
INS/INSB/INSW/INSD	Eingabe eines Stringelements von E-A-Port
OUTS/OUTSB/OUTSW/OUTSD	Ausgabe eines Stringelements zu E-A-Port
REP INS/INSB/INSW/INSD	Stringeingabe von E-A-Port
REP OUTS/OUTSB/OUTSW/OUTSD	Stringausgabe zu E-A-Port
Cache- und TLB-Verwaltung	
INVD	Caches leeren
WBINVD	Caches leeren mit Rückschreibanzeige
INVLPG	TLB-Eintrag ungültig machen
Systemsteuerung	
CLI/STI	Unterbrechungen verhindern/erlauben
HLT	Halt
CLTS	Löschen Taskumschaltanzeige
WAIT	Prüfen auf anhängige FPU-Fehlerbedingungen
RSM	Rückkehr aus dem Systemverwaltungszustand (SMM)
SYSENTER	Aufrufen Systemfunktion
SYSEXIT	Rückkehr aus Systemfunktion

9.1.7. SIMD-Erweiterungen (Überblick)

Die MMX-Erweiterung (Intel)

Neue Maschinenbefehle
Die MMX-Erweiterung besteht in 47 neuen Maschinenbefehlen. Das Ziel besteht darin, Elementaroperationen vorzugsweise für Anwendungen im "Multimedia"-Bereich bereitzustellen. Anwendungsgebiete (Beispiele):

- Aufbauen von graphischen Darstellungen*),
- Kompression/Dekompression von Audio- und Videodaten,
- Signalverarbeitung (Filterung, Fouriertransformation usw.),
- Verschlüsselungsalogrithmen.

*): als Einsatzgebiet werden vor allem *Spiele* mit bewegten 3D-Bildern herausgestellt (weil wir ja alle keine anderen Sorgen haben).

Man verspricht sich hiervon sowohl Leistungssteigerung als auch - über alles gesehen - Kostensenkung (da bestimmte Funktionen im Prozessor ablaufen können und keine (oder eine weniger aufwendige) Sonderhardware benötigen; so hat man z. B. die Signalverarbeitungsfunktionen eines Hochgeschwindigkeitsmodems in den MMX-Prozessor verlagert; hierdurch verringert sich der Aufwand einer Modem-Karte auf eine simple Adaptierung des Telefonanschlusses).

Überblick über den MMX-Befehlsvorrat
Da wir nicht programmieren wollen, brauchen wir keine Einzelheiten (nicht einmal die mnemonischen Bezeichner der Befehle). Im folgenden geben wir einen Überblick über jene MMX-Befehle, die auf kurze Vektoren*) gemäß Abbildung 1.30 angewendet werden können. Einzelheiten der Adressierung sowie die Transport- und Organisationsbefehle der MMX-Erweiterung haben wir nicht berücksichtigt.

*): die Intel-Bezeichnung: Packed Data (gepackte Daten).

Addieren/Subtrahieren
Die Operanden in zwei MMX-Quadworten werden parallel zueinander addiert oder voneinander subtrahiert. Die Operation betrifft entweder 8 Bytes, 4 16-Bit-Worte oder 2 32-Bit-Doppelworte in jedem der MMX-Quadworte. Varianten:

- Operanden (beide) vorzeichenlos (unsigned) oder vorzeichenbehaftet (signed),
- Ergebnisbildung gemäß herkömmlicher oder gemäß Sättigungsarithmetik.

Vergleichen
Die Operanden in zwei MMX-Quadworten werden parallel miteinander verglichen. Die Operation betrifft entweder 8 Bytes, 4 16-Bit-Worte oder 2 32-Bit-Doppelworte in jedem der MMX-Quadworte. Ist die Vergleichsbedingung erfüllt, besteht das das jeweilige

Ergebnis ausschließlich aus Einsen (Byte = FFH, 16-Bit-Wort = FFFFH usw.), ansonsten aus Nullen (Byte = 00H, 16-Bit-Wort = 0000H usw.). Varianten:

- **#** Vergleichen auf Gleichheit (wirkt für alle Interpretationen der Operanden),
- **P** Vergleichen auf "größer als" (1. Operand > 2. Opernad). Dabei werden beide Operanden als vorzeichenbehaftete Binärzahlen interpretiert.

Multiplizieren
Die 16-Bit-Worte in zwei MMX-Quadworten werden als ganze Binärzahlen (vorzeichengerecht) parallel miteinander multipliziert. Von den entstehenden 32-Bit-Ergebnissen wird jeweils die Hälfte gespeichert. Varianten:

- **#** Speichern der niederwertigen 16 Bits der 32-Bit-Ergebnisse,
- **#** Speichern der höherwertigen 16 Bits der 32-Bit-Ergebnisse.

Multiplizieren und Addieren (Skalarprodukt)
Die 16-Bit-Worte in zwei MMX-Quadworten werden als ganze Binärzahlen (vorzeichengerecht) parallel miteinander multipliziert. Die 32-Bit-Ergebnisse in der linken und der rechten Hälfte der 64-Bit-Struktur werden aufeinander addiert. Es entsteht ein 64-Bit-Wort, das mit 2 32-Bit-Skalarprodukten belegt ist. Rechenschema:

1. Operand = A_1 B_1 C_1 D_1, 2. Operand = A_2 B_2 C_2 D_2. Ergebnis:

- **P** rechtes 32-Bit-Wort = $(A_1 \cdot A_2) + (B_1 \cdot B_2)$,
- **P** linkes 32-Bit-Wort = $(C_1 \cdot C_2) + (D_1 \cdot D_2)$.

Verschieben
Die Operanden in einem MMX-Quadwort werden parallel verschoben. Die Operation betrifft entweder 4 16-Bit-Worte, 2 32-Bit-Doppelworte oder das gesamte 64-Bit-Quadwort. Die Bitanzahl, um die verschoben werden soll, wird wahlweise als Direktwert im Befehl angegeben oder aus einem zweiten MMX-Register entnommen (alle Operanden werden um die gleiche Bitanzahl verschoben). "Herausgeschobene" Bits werden nicht gespeichert. Varianten:

- **#** Linksverschieben (Auffüllen rechts der frei gewordenen Bitpositionen mit Nullen),
- **#** Rechtsverschieben logisch (Auffüllen der links frei gewordenen Bitpositionen mit Nullen),
- **#** Rechtsverschieben arithmetisch (Auffüllen der links frei gewordenen Bitpositionen mit dem Vorzeichen) - diese Operation gibt es aber nur für 16-Bit-Worte und für 32-Bit-Doppelworte.

Entpacken
Aus den Belegungen von 2 MMX-Quadworten wird ein Ergebnis-Quadwort gebildet. Die Operation betrifft entweder 8 Bytes, 4 16-Bit-Worte oder 2 32-Bit-Doppelworte. Dabei

entsteht jeweils aus 2 “kleineren” Operanden eine “größere” Datenstruktur (aus 2 Bytes ein 16-Bit-Wort, aus 2 Worten ein Doppelwort und aus 2 Doppelworten ein Quadwort). Varianten:

- Entpacken der linken Hälfte der Quadworte (der “höheren” 32 Bits),
- Entpacken der rechten Hälfte der Quadworte (der “niederen” 32 Bits).

Ausführungsbeispiel (mit 16-Bit-Worten):
1. Operand = A_1 B_1 C_1 D_1, 2. Operand = A_2 B_2 C_2 D_2. Es werden die “höheren 32 Bits entpackt. Im Ergebnis entstehen durch Aneinanderreihen der entsprechenden 16-Bit-Operanden die 32-Bit-Worte A_1 A_2 und B_1 B_2.

Packen
Aus den Belegungen von 2 MMX-Quadworten wird ein Ergebnis-Quadwort gebildet. Die Operation betrifft entweder 8 Bytes, 4 16-Bit-Worte oder 2 32-Bit-Doppelworte. Dabei werden jeweils aus einem “größeren” Operanden eine “kleinere” Datenstruktur gebildet (aus einem 32-Bit-Doppelwort ein 16-Bit-Wort, aus einem 16-Bit-Wort ein 8-Bit-Wort). Die Operanden der 2 MMX-Quadworte ergeben ein Ergebnis-Quadwort. Hierzu wird jede Operandenstruktur als vorzeichenbehaftete Binärzahl interpretiert. Ist deren Wert in der jeweiligen Ergebnis-Datenstruktur darstellbar, so wird er, unter Abschneiden der führenden Nullen (bzw. Vorzeichenstellen) direkt in das Ergebnis überführt. Ist der Wert zu groß oder zu klein, so wird das Ergebnis nach den Regeln der Sättigungsarithmetik gebildet. Varianten:

- die Ergebnisse werden als natürliche (vorzeichenlose) Binärzahlen gebildet,
- die Ergebnisse werden als ganze (vorzeichenbehaftete) Binärzahlen gebildet.

‚*Ausführungsbeispiel*: eine 32-Bit-Binärzahl Z soll in ein 16-Bit-Wort gepackt werden. Die Ergebnisbildung richtet sich nach der Befehlsvariante:

- vorzeichenloses Ergebnis: wenn Z im Wertebereich liegt ($0 \leq Z \leq$. 65 535), so ist das Ergebnis = Z. Ist Z größer (> 65 535), wird das Ergebnis = 65 535 (FFFFH), ist Z kleiner (negativ), wird das Ergebnis = 0.
- vorzeichenbehaftetes Ergebnis: wenn Z im Wertebereich liegt (- 32 768 $\leq$ Z$\leq$. 32 767), so ist das Ergebnis = Z. Ist Z größer (> 32 767), wird das Ergebnis = 32 767 (7FFFH), ist Z kleiner (< - 32 768), wird das Ergebnis = - 32 768 (8000H).

Logische Verknüpfungen über 64 Bits
Zwei MMX-Quadworte werden bitweise zu einem Ergebnis-Quadwort verknüpft. Folgende Operationen sind vorgesehen:

- UND-Verknüpfung,
- UND-Verknüpfung mit vorheriger bitweiser Negation des zwieten Operanden,
- ODER-Verknüpfung,
- Antivalenzverknüpfung (XOR).

Die 3DNow-Erweiterung (AMD)
In einem MMX-Quadwort werden 2 32-Bit-Gleitkommazahlen untergebracht. Der Befehlsvorrat wird mit 21 neuen Befehlen erweitert, die zusätzlich zu den MMX-Befehlen genutzt werden können (Tabelle 9.1).

Rechnen mit natürlichen bzw. ganzen Binärzahlen	
P	Mittelwertbildung über vorzeichenlose 8-Bit-Operanden,
P	16-Bit-Multiplikation (ganzzahlig) mit Rundung

Hilfsoperationen (zu Beschleunigungszwecken)	
P	schnelleres Wechseln zwischen MMX- und Gleitkommaverarbeitung,
P	Holen eines 32-Bit-Wortes in den L1-Cache

Operationen mit gepackten Gleitkommazahlen			
P	Addieren,	P	32-Bit-Ganzzahlen in Gleitkomma wandeln,
P	Akkumulieren (Quersumme über die Operanden im Quadwort),	P	Gleitkommazahlen in 32-Bit-Ganzzahlen wandeln,
P	Subtrahieren,	P	Kehrwert bilden (1/x; näherungsweise),
P	Subtrahieren in umgekehrter Operandenfolge,	P	Kehrwert der Quadratwurzel bilden ($1/\sqrt{x}$; näherungsweise),
P	Multiplizieren,	P	erster Iterationsschritt der Kehrwertbildung
P	Vergleichen auf “größer oder gleich”,	P	erster Iterationsschritt der Bildung von $1/\sqrt{x}$,
P	Vergleichen auf Gleichheit,	P	zweiter Iterationsschritt für 1/x und $1/\sqrt{x}$.
P	Vergleichen auf “größer als”		
P	Minima finden (gibt die jeweils kleineren Werte zurück),		
P	Maxima finden (gibt die jeweils größeren Werte zurück)		

Tabelle 9.1 Die Befehle der AMD 3D-Technologie (Übersicht)

Weitere Ergänzungen
Mit dem K7-Prozessor (Athlon) hat AMD die MMX-Erweiterung um 19 weitere Befehle ergänzt und die 3DNow-Erweiterung um 5 Befehle (Tabelle 9.2).

MMX-Ergänzungen (1): Zugriffbeschleunigung, besser Ausnutzung des Cache-Subsystems	
P	Datentransporte unter Umgehung der Caches*),
P	Steuerung des vorbeugenden Holens von Daten bzw. des Caching (PREFETCH)**),
P	Schreibzugriffe voneinander abgrenzen (SFENCE)
MMX-Ergänzungen (2): weitere Recheno	
P	Mittelwertbildung über vorzeichenlose Zahlen von 8 oder 16 Bits,
P	Transportieren von 16-Bit-Worten (zwischen einem MMX- und einem Universalregister),
P	Transportieren von Maskenbits (von jedem der 4 Operanden das höchstwertige Bit),
P	Maximum- und Minimumbestimmung (Datentypen 8/16 Bits, vorzeichenlos/ganzzahlig),
P	Multiplikation von vorzeichenlosen 16-Bit-Zahlen, die jeweils die höherwertigen 16 Ergebnisbits liefert
3DNow-Ergänzungen	
P	Datenformatwandlungen,
P	Akkumulationsbefehle über 32-Bit-Operanden (Addition/Subtraktion der beiden 32-Bit-Worte in jedem der beiden Operanden)

*): vermeidet "Zustopfen" der Caches beim Transportieren von Datenströmen (Audio/Video)

Tabelle 9.2 Mit dem K7 (Athlon) eingeführte Ergänzungen(Übersicht)

Die SSE1-Erweiterung (Intel)

SSE betrifft vor allem Verknüpfungen von 128 Bits langen Operandenstrukturen, die in besonderen Registern (SMM0...SMM7) gehalten werden. Diese Strukturen sind vorzugsweise mit jeweils 4 Gleitkommazahklen zu 32 Bits belegt (einfache Genauigkeit gemäß IEEE 754-85). Tabelle 9.3 gibt einen Überblick über den Befehlsvorrat.

Skalar- und Vektorverarbeitung (Scalar/Packed Operations)

Abbildung 9.6 zeigt den Unterschied.

Abbildung 9.6 Skalar- und Vektorverarbeitung mit SSE-Befehlen

Erklärung:

a) zur Wirkung von Skalarbefehlen (Scalar Operations). Es wird nur das Operandenpaar in der niedrigstwertigen Position miteinander verknüpft. Die verbleibenden 3 Positionen des Ergebnisses werden jeweils mit dem Wrt des ersten Operanden belegt.

b) zur Wirkung von Vektorbefehlen (Packed Operations). Alle 4 Operandenpaare werden miteinander verknüpft.

Datentransporte	
P	Transportieren von 32-Bit-Gleitkommazahlen (gepackt),
	• an integralen Adressen (Aligned), • an nicht-integralen Adressen (Unaligned),
P	Transportieren von 64-Bit-Worten (gepackt),
P	Transportieren von Maskenbits (von jedem der 4 Operanden das höchstwertige Bit),
P	Transportieren einer 32-Bit-Gleitkommazahl (skalar)

Datenformatwandlungen	
P	32-Bit-Ganzzahl in 32-Bit-Gleitkommazahl,
P	32-Bit-Gleitkommazahl in 32-Bit-Ganzzahl,
P	32-Bit-Gleitkommazahl in 32-Bit-Ganzzahl mit Rundung
	- alle Wandlungen skalar und gepackt -

Arithmetische Befehle mit 32-Bit-Gleitkommazahlen			
P	Addieren,	P	Quadratwurzel,
P	Subtrahieren,	P	Maximum,
P	Multiplizieren,	P	Minimum,
P	Dividieren,	P	Vergleichen
	- alle Operationen skalar und gepackt -		

Logische Verknüpfungen (bitweise, gepackt)			
P	Und,	P	Oder,
P	Und-Nicht	P	Exklusiv-Oder

weitere Befehle im Rahmen der SSE1-Erweiterung (verschiedenartige Wirkungen)	
P	Mischen von Operanden (Shuffle),
P	Mittelwertbildung,
P	Maximum/Minimum über natürliche Zahlen von 8 Bits bzw. Ganzzahlen von 16 Bits,
P	zusätzliche Transporte zwischen den Universal- und den MMX-Registern,
P	Einstellen/Retten von Zustandsangaben,
P	Steuerung des vorbeugenden Holens von Daten bzw. des Caching (PREFETCH),
P	Schreibzugriffe voneinander abgrenzen (SFENCE)

Tabelle 9.3 Die Befehle der SSE1-Erweiterung (Übersicht)

Die SSE2-Erweiterung (Intel)
Der Registersatz (Abbildung 9.4) wurde beibehalten. Es wurden aber weitere Datentypen vorgesehen. Wichtige Merkmale:

- **#** Gleitkommaverarbeitung mit je 2 64-Bit-Gleitkommazahlen (doppelte Gebnauigkeit gemäß IEEE754-85) in einem 128-Bit-Register,
- **#** Erweiterung der bisherigen MMX-Befehle von 64-Bit-Operanden in MMX-Registern auf 128-Bit-Operanden in SSE-Registern.
- **#** weitere Befehle zum Steuern des Caching und des Abgrenzens von Speicherzugriffen*).

SSE2 umfaßt insgesamt 144 neue Befehle (davon sind 76 wirklich neu; 68 sind Abwandlungen/Erweiterungen bereits vorhandener Befehle).

*): ein Abgrenzbefehl (Fence) bewirkt, daß alle anhängigen Speicherzugriffe der jeweils betreffenden Art zunächst zu Ende geführt werden. So ist z. B. nach Ausführung eines SFENCE-Befehls (vgl. Tabelle 9.3) gewährleistet, daß alle vorhergegangenen Schreibzugriffe (die ja in Schreibpuffern, Caches usw. gleichsam hängengeblieben sein könnten), tatsächlich erledigt wurden. Somit finden nachfolgende Befehle einen entsprechend aktualisierten Arbeitsspeicher vor.

9.1.8. Mikroprogrammänderungen in P6-Prozessoren

IA-32-Prozessoren haben fest gespeicherte Mikroprogramme. Änderungsvorkehrungen betreffen nicht das Erneuern der gesamten Mikroprogramm-Ausstattung, sondern lediglich das Ausbügeln von Fehlern. Dies sei anhand jener Lösung dargestellt, die Intel in den P6-Prozessoren eingeführt hat (P6 Family Microcode Update Feature).

Es sind fest formatierte Änderungsblöcke vorgesehen, die im ROM des Computers gespeichert werden (beim PC: im Rahmen des BIOS; Abbildung 9.7).

Abbildung 9.7 Prinzip der Mikroprogrammänderung (Microcode Update) eines P6-Prozessors (Intel)

Erklärung:
Die Änderungsblöcke sind im BIOS-ROM gespeichert. Je Prozessortyp gibt es nur einen gültigen Änderungsblock. Ein Ladeprogramm - das zum BIOS gehört - veranlaßt, daß dieser Block in den Prozessor transportiert wird. Ist der Änderungsblock durch einen neuen zu ersetzen, so muß dieser zunächst in den BIOS-ROM gebracht werden. Grundlage hierfür bilden die Vorkehrungen zum Ändern des BIOS (BIOS Update), wie sie in den modernen PCs vorgesehen sind (der BIOS-ROM ist ein Flash-ROM, dessen Inhalt im System mittels Software verändert werden kann).

Aufbau eines Änderungsblocks
Der Änderungsblock hat eine feste Länge von 2048 Bytes. Davon sind 2000 Bytes für die eigentlichen Änderungen vorgesehen, der Rest betrifft Angaben, die absichern sollen, daß

auch wirklich korrekt geändert wird. (Diese Blöcke werden von Intel fix und fertig geliefert - es ist unmöglich, eigene Mikroprogramme einzubringen.)

Wann ist eine Änderung korrekt?
Die Änderung muß sich auf einen bestimmten Typ und einen bestimmten Änderungsstand (Silicon Revision) des Prozessorschaltkreises beziehen (die entsprechende Angabe ist im Prozessor fest codiert und kann programmseitig gelesen werden. Der eigentliche Freigabstand (Stepping) eines P6-Schaltkreises ergibt sich gemäß Intel aus dem Zusammenhang von Änderungsstand (Silicon Revision) und der jeweiligen Mikroprogrammänderung (Intel selbst prüft die Prozessoren stets *mit* geladener Mikroprogrammänderung). Zudem muß die Änderung bis auf's Bit korrekt sein (hierfür sind im Änderungsblock entsprechende Prüfangaben vorgesehen).

Wann wird geändert?
Nach jedem Rücksetzen des Prozessors (vor allem natürlich nach jedem Einschalten), aber vor dem Zuschalten des L2-Caches.

Ablauf des Ladens
Das BIOS adressiert den aktuellen Änderungsblock (hierzu muß es ggf. feststellen, welcher Prozessortyp eigentlich installiert ist) und startet dann den Ladevorgang (durch Schreiben auf ein maschinenspezifisches Register). Das eigentliche Laden läuft selbständig ab. Anschließend kann das BIOS prüfen, ob das Laden erfolgreich war (keine Bitverfälschungen, richtige Version usw.).

Spezielle BIOS-Funktionen
Intel hat eine spezielle Programmschnittstelle über den BIOS-Interrupt 15 vorgesehen. Die einzelnen Funktionen:

- Installationsprüfung,
- Schreiben der Änderungsdaten,
- Kontrolle auf Korrektheit,
- Zurücklesen der Änderungsdaten.

9.2. Sparc

Sparc ist die Abkürzung von "Scalable Processor Architecture". "Scalable" (skalierbar) soll zum Ausdruck bringen, daß die Architektur von Anfang an daraufhin ausgelegt ist, vielfältige, in Aufwand und Leistungsvermögen abgestufte, aber softwareseitig voll kompatible Implementierungen verwirklichen zu können, beispielsweise mit immer "schnelleren" Halbleiter-Technologien, mit leistungsfähigeren Caches und Bussystemen usw., bis hin zu Superskalarverarbeitung und Multiprozessorstrukturen. Sparc ist ein typisches Beispiel für die RISC-Auslegung: fest formatierte Befehle, großer Registersatz, vergleichsweise einfache Operationen, die nur Registerinhalte betreffen, Speicherzugriffe nur über besondere Lade- und Speicherbefehle, einfache Adressierungsprinzipien. (Vergleichen Sie insbesondere Registersatz, Befehlsliste und Speicheradressierung mit den entsprechenden Merkmalen der IA-32-Architektur.)

9.2.1. Systemstruktur

Verarbeitungswerke
Ein Sparc-Prozessor hat als Kern eine universelle Verarbeitungseinheit, die als Integer Unit bezeichnet wird. Diese kann durch maximal zwei Coprozessoren erweitert werden. Dabei ist der erste Coprozessor üblicherweise eine Sparc-Gleitkommaverarbeitungseinheit (Floating Point Unit).

Erklärung:
Die Architektur definiert den Registersatz und die grundsätzlichen Befehlsstrukturen eines Coprozessors. Zum Laden und Speichern von Worten sowie Doppelworten sind besondere Befehle vorhanden, ebenso zum Auslösen von Operationen (dem einzelnen Coprozessor stehen insgesamt 1024 unterschiedliche Operationscodes zur Verfügung). Welchen Gebrauch man von diesen Möglichkeiten macht, ist an sich gleichgültig. In der Sparc-Architektur ist aber die Gleitkommaverarbeitung voll ausdefiniert. Typischerweise sind Integer- und Floating Point Unit auf einem gemeinsamen Prozessorschaltkreis untergebracht. Das Interface zum zweiten Coprozessor ist für anwendungsspezifische Zusatzhardware nutzbar.

Integer Unit
Die universelle Verarbeitungseinheit hat eine Verarbeitungsbreite von 32 Bits. Sie umfaßt eine universelle Arithmetik-Logik-Einheit (ALU), eine Verschiebeeinheit, einen Registersatz von 136 allgemein nutzbaren 32-Bit-Registern sowie 4 weitere 32-Bit-Register.

Gleitkomma-Verarbeitungseinheit (Floating Point Unit)
Die Gleitkomma-Verarbeitungseinheit hat eine interne Verarbeitungsbreite von 64 Bits (doppelt genaue Gleitkommazahlen gemäß Standard IEEE754-85). Sie kann 32-Bit- und 64-Bit-Gleitkommazahlen verarbeiten. Der Registersatz enthält 32 Datenregister zu 32 Bits (64-Bit-Gleitkommazahlen belegen zwei aufeinanderfolgende Register).

Speicherverwaltungshardware
Die Sparc-Architektur sieht Speicherverwaltungseinheiten vor, in denen das Prinzip der Seitenverwaltung verwirklicht ist.

E-A-Adressierung
Sparc kennt keine E-A-Befehle und auch keinen E-A-Adreßraum. Ein- und Ausgabehardware muß somit über Speicheradressen angesprochen werden (Memory Mapped I/O). Der wesentliche Nachteil dieses an sich eleganten Prinzips, nämlich die fehlende saubere Trennung zwischen Speicherzugriffen und den naturgemäß immer anwendungs- bzw. maschinenspezifischen E-A-Operationen, ist durch Nutzung folgender Vorkehrungen umgehbar:

- Sparc stellt bis zu 256 verschiedene Speicheradreßräume zur Verfügung (damit sollte es kein Problem sein, wenigstens einen Adreßraum eigens für E-A-Zwecke zu reservieren),
- der TLB der Speicherverwaltung läßt sich unter direkter Software-Steuerung sehr flexibel zu Adreßumwandlungen ausnutzen.

9.2.2. Datentypen

Die elementaren Datentypen moderner Rechnerarchitekturen sind in Kapitel 1 eingehend beschrieben, so daß hier eine knappe Aufzählung ausreicht.

Besonderheiten der RISC-Architektur
Die Sparc-Prozessoren verarbeiten nur Registerinhalte. Ein Register (bzw. eine Anzahl aufeinanderfolgender Register) ist in diesem Sinne ein Behälter für 32, 64 bzw. 128 Bits. "Kleinere" Datenstrukturen werden nur von Lade- oder Speicherbefehlen angesprochen. Im Register werden sie rechtsbündig angeordnet und auf die volle Registerlänge entsprechend erweitert. Abbildung 9.8 zeigt, wie die einzelnen Datenstrukturen in den Registern placiert werden.

Achtung:
Bei Sparc ist ein Wort 32 Bits lang! (16 Bits sind ein Halbwort.)

Datentypen-Übersicht:

- natürliche (vorzeichenlose) Binärzahlen (Ordinalzahlen) von 32, 16 und 8 Bits Länge bzw. beliebig interpretierbare Bytes, Halbworte, Worte und Doppelworte,
- ganze Binärzahlen (Integer-Zahlen; mit Vorzeichen) von 64, 32, 16 und 8 Bits Länge,
- Gleitkommazahlen von 32, 64 und 128*) Bits Länge als Näherungsdarstellungen reeller Zahlen (gemäß Standard IEEE 754-85),
- Seitentabellen und TLB-Einträge (siehe dazu Abschnitt 9.2.7.).

*): Verarbeitung mittels Software.

TAG-Worte
Es handelt sich um "gewöhnliche" 32-Bit-Worte, deren beide niedrigstwertigen Bits bei speziellen Additionsbefehlen TADDcc bzw. TADDccTV daraufhin überprüft werden, ob sie mit Null belegt sind. Sind sie nicht Null, wird eine Ausnahme wirksam. Zwei wichtige Anwendungen:

1. man will in den besagten beiden Bits eine Datentypkennzeichnung unterbringen. Der Wert Null bezeichnet dann ein "normales" Wort, das ohne weiteres in Additionsoperationen einbezogen wird. Ein anderer Wert kennzeichnet, daß es sich um einen Datentyp handelt, der eine Sonderbehandlung verlangt (die dann durch den Ausnahmebehandler verwirklicht wird).
2. die betreffenden Worte werden als Speicheradressen interpretiert (die irgendwelchen Rechenoperationen unterzogen oder anderweitig durch Software verarbeitet werden), und man will prüfen, ob es sich um integrale Wortadressen handelt (dann müssen die beiden niedrigstwertigen Bits Null sein) oder nicht.

Hinweis:
In der IA-32-Architektur gibt es Ähnliches: die Anordnungspüfung (Alignment Check). Sie betrifft die gleiche Datenstruktur. Eine Ausnahme wird wirksam, wenn eine 32-Bit-Struktur als Adresse für einen Doppelwortzugriff genutzt wird und die beiden niedrigstwertigen Bits ungleich Null sind.

Abbildung 9.8 Datentypen der Sparc-Architektur

9.2.3. Registersatz

Abbildung 9.9 zeigt die Registerstruktur der Sparc-Prozessoren, wie sie sich dem Programmierer darstellt.

Abbildung 9.9 Die Registerstruktur der Sparc-Architektur aus der Sicht des Programmierers

Allgemeine Register
Der einzelne Befehl "sieht" 32 32-Bit-Register (r0...r31). In der Hardware sind aber insgesamt 136 Register vorgesehen. Dem liegt folgendes Programmiermodell zugrunde: Aus der Sicht des Befehls werden die ersten acht Register r0...r7 fest adressiert (r0 ist dabei kein "richtiges" Register, sondern liefert bei Lesezugriffen den Festwert Null, während Schreibzugriffe keine Wirkung haben). Diese Register heißen "globale" Register. Die Register r8...r31 werden aus einem "Fenster" (Register Window) entnommen, das jeweils 24 Register umfaßt. Das aktuelle Fenster wird durch eine Angabe im Prozessorzustandsregister PSR ausgewählt (Fensterzeiger, Current Window Pointer CWP). Dies hat den Zweck, beim Unterprogrammaufruf Parameter übergeben zu können, ohne Registerinhalte transportieren bzw. retten zu müssen (Prinzip des Stack Frame). 8 der 24 Register sind vorgesehen, um Parameter vom rufenden Programm zu übernehmen (In-Register), weitere 8, um lokale Parameter zu halten (lokale Register), und die verbleibenden 8, um Parameter an ein weiteres zu rufendes Unterprogramm zu übergeben (Out-Register). Der entscheidende Gedanke besteht nun darin: Das aktuelle Programm will ein Unterprogramm rufen. Es hinterlegt dazu die Parameter in seinen Out-Registern. Beim Unterprogrammruf schaltet man den Fensterzeiger CWP um 16 weiter. Damit werden die Out-Register des rufenden Programms zu In-Registern des gerufenen. Das Unterprogramm findet so seine Parameter vor, ohne daß dafür irgendwelche Transportabläufe notwendig sind. Des weiteren stehen ihm 16 "eigene" Register zur Verfügung (8 lokale und 8 Out-Register). Es ist also auch nicht erforderlich, Registerinhalte auf einen Stack zu retten, nur um die Register freizubekommen. Sinngemäß können Ergebnisse in den In-Registern an das rufende Programm zurückgegeben werden. Dazu schaltet man den Fensterzeiger CWP um 16 zurück, so daß das rufende Programm die Ergebnisse in seinen Out-Registern vorfindet, wiederum ohne daß irgendeine Transportoperation erforderlich ist (siehe auch Abbildung 9.12).

Y-Register
Das Y-Register (Multiplikationsschrittregister) ist ein Hilfsregister für die Integer- Multiplikation. Es nimmt anfänglich den Multiplikator auf, der von Befehlen MULScc (Multiply Step; Multiplikationsschritt) ausgewertet und von der höchstwertigen Bitposition beginnend unter Rechtsverschiebung durch Ergebnisbits ersetzt wird, so daß schließlich das Y-Register eine Hälfte des 64-Bit-Produkts von zwei 32-Bit-Zahlen enthält.

Prozessorzustandsregister PSR
Im Prozessorzustandsregister PSR (Processor State Register) sind die wichtigsten Angaben zum Systemzustand, zum Prozessortyp sowie die Integer-Bedingungscodes (bei Intel: Flagbits) zusammengefaßt.

Systemzustände
Sparc kennt nur zwei Zustände: Anwendungszustand (User Mode) und Supervisorzustand (Supervisor Mode). Privilegierte Befehle sind nur im Supervisorzustand lauffähig, ansonsten wird eine Ausnahmebedingung wirksam.

Fenstermaskenregister WIM
Das Register WIM (Window Invalid Mask) enthält für jedes Register-Fenster ein Bit. Eine Eins kennzeichnet das betreffende Fenster als ungültig. Wird es in die Fensterumschaltung einbezogen, so wird eine Ausnahmebedingung wirksam.

Befehlszähler
Zur Befehlszählung sind zwei 32-Bit-Register vorgesehen, die mit PC und nPC bezeichnet werden. Der eigentliche Befehlszähler (PC) zeigt auf den Befehl, der gerade ausgeführt wird. nPC enthält den um 4 erhöhten PC-Inhalt und zeigt damit normalerweise auf den Folgebefehl. Bei einer Unterbrechung werden sowohl PC als auch nPC gerettet. Wozu? - Es handelt sich um einen Trick, um ein typisches RISC-Prinzip zu unterstützen: das Füllen von "Lücken" in der Befehlspipeline mit eingeschobenen Befehlen.

Eingeschobene Befehle
Weicht der Programmablauf von der üblichen Befehlszählung ab, so tritt in der Befehlspipeline normalerweise eine Lücke auf. Um diese sinnvoll zu nutzen, wird der Befehl, der einem Verzweigungs- oder Unterprogrammrufbefehl folgt, normalerweise noch ausgeführt (Verzweigungsbefehle enthalten ein besonderes Bit, um dies unterdrücken zu können). Wichtig ist, daß trotz dieses Einschiebens der Ausnahmebehandler noch erkennen kann, welcher Befehl die Ausnahme wirklich verursacht hat. Deshalb wird bei Verzweigung oder Unterprogrammruf der PC mit dem Inhalt von nPC geladen, dann nPC mit der Verzweigungsadresse. Tritt nun eine Ausnahme auf, findet der Ausnahmebehandler sowohl die Adresse des Befehls, zu dem verzweigt wurde, als auch jene des eingeschobenen Befehls vor und kann diese Angaben zur genauen Bestimmung der Ursache und der Reaktion verwenden.

Register der Gleitkomma-Verarbeitungseinheit
Die Gleitkomma-Verarbeitungseinheit enthält 32 Datenregister von 32 Bits Länge, ein Zustandsregister sowie eine Warteschlange für Befehle und Adressen. Die Warteschlange wird von der Hardware geladen und kann mit Befehlen gespeichert werden. Der Zweck

besteht darin, einem Ausnahmebehandler zu ermöglichen, die in Arbeit befindlichen Befehle zu identifizieren (um sie nach der Behandlung erneut zu starten, zu emulieren usw.).

9.2.4. Befehlsformate

Alle Befehle sind einheitlich 32 Bits lang. Es handelt sich um eine "reinrassige" RISC-Befehlsgestaltung mit getrennten Speicherzugriffs- und Operationsbefehlen- Letztere wirken nur auf Registeroperanden, und zwar nach dem Dreiadreßprinzip: Bestimmung := Operand 1 **op** Operand 2. Abbildung 9.10 gibt einen Überblick über die Befehlsformate.

Abbildung 9.10 Sparc-Befehlsformate

9.2.5. Befehlsliste

Der Befehlsvorrat umfaßt Befehle für folgende Aufgaben:

- Transporte (Laden und Speichern),
- arithmetische Verknüpfungen,
- logische Verknüpfungen,
- Verschieben,
- Steuerung des Programmablaufs,
- Steuerung des Prozessors.

Befehlsübersicht

Hinweise:

1. Die Befehle werden mit mnemonischen Abkürzungen gemäß der Sparc-Dokumentation bezeichnet. Die zugehörige deutsche Bezeichnung strebt eine knappe und treffende Charakterisierung der jeweiligen Befehlswirkung an.
2. Ein P am rechen Rand kennzeichnet einen priviliegierten Befehl, der nur im Supervisorzustand ausführbar ist.

Befehle der Integer Unit

Binärarithmetik		
ADD	Addieren	
ADDcc	Addieren und Modifizieren Bedingungscode	
ADDX	Addieren mit Übertrag	
ADDXcc	Addieren mit Übertrag und Modifizieren Bedingungscode	
SUB	Subtrahieren	
SUBcc	Subtrahieren und Modifizieren Bedingungscode	
SUBX	Subtrahieren mit Übertrag	
SUBXxx	Subtrahieren mit Übertrag und Modifizieren Bedingungscode	
TADDcc	Addieren TAG-Wort und Modifizieren Bedingungscode	
TADDccTV	Addieren TAG-Wort und Modifizieren Bedingungscode; Trap bei Überlauf	
TSUBcc	Subtrahieren von TAG-Wort und Modifizieren Bedingungscode	
TSUBccTV	Subtrahieren von TAG-Wort und Modifizieren Bedingungscode; Trap bei Überlauf	
MULScc	Multiplikationsschritt und Modifizieren Bedingungscode	
Logische Verknüpfungen		
AND	Konjunktion (UND)	
ANDcc	Konjunktion (UND) und Modifizieren Bedingungscode	
OR	Disjunktion (ODER)	
ORcc	Disjunktion (ODER) und Modifizieren Bedingungscode	
ORN	Disjunktion mit negiertem 2. Operanden	
ORNcc	Disjunktion mit negiertem 2. Operanden und Modifizieren Bedingungscode	

Fortsetzung Logische Verknüpfung		
XOR	Antivalenz (Exklusiv-ODER)	
XORcc	Antivalenz (Exklusiv-ODER) und Modifizieren Bedingungscode	
XNOR	Äquivalenz (Exklusiv-NOR)	
XNORcc	Äquivalenz (Exklusiv-NOR) und Modifizieren Bedingungscode	
Verschieben		
SLL	Linksverschieben logisch	
SRL	Rechtsverschieben logisch	
SRA	Rechtsverschieben arithmetisch	
Transportbefehle		
LDSB	Laden Byte mit Vorzeichen	
LDSBA	Laden Byte mit Vorzeichen von alternativem Adreßraum	P
LDSH	Laden Halbwort mit Vorzeichen	
LDSHA	Laden Halbwort mit Vorzeichen von alternativem Adreßraum	P
LDUB	Laden Byte vorzeichenlos	
LDUBA	Laden Byte vorzeichenlos von alternativem Adreßraum	P
LDUH	Laden Halbwort vorzeichenlos	
LDUHA	Laden Halbwort vorzeichenlos von alternativem Adreßraum	P
LD	Laden Wort	
LDA	Laden Wort von alternativem Adreßraum	P
LDD	Laden Doppelwort	
LDDA	Laden Doppelwort von alternativem Adreßraum	P
SETHI	Laden der höchstwertigen 22 Bits mit Direktwert	
STB	Speichern Byte	
STSBA	Speichern Byte in alternativem Adreßraum	P

Fortsetzung Transportbefehle		
STSH	Speichern Halbwort	
STSHA	Speichern Halbwort in alternativem Adreßraum	P
ST	Speichern Wort	
STA	Speichern Wort in alternativem Adreßraum	P
STD	Speichern Doppelwort	
STDA	Speichern Doppelwort in alternativem Adreßraum	P
LDSTUB	Unteilbares Laden/Speichern eines vorzeichenlosen Bytes ("Test-and-Set"-Operation)	
LDSTUBA	Unteilbares Laden/Speichern eines vorzeichenlosen Bytes ("Test-and--Set"-Operation); alternativer Adreßraum	P
Austausch- und Wandlungsbefehle		
SWAP	Registerinhalt mit Speicherinhalt tauschen	
SWAPA	Registerinhalt mit Speicherinhalt in alternativem Adreßraum tauschen	P
Programmsteuerbefehle		
Bicc	Bedingte Verzweigung gemäß Bedingungscode der Integer Unit	
FBicc	Bedingte Verzweigung gemäß Bedingungscode der Gleit-komma-Verarbeitungseinheit	
CBccc	Bedingte Verzweigung gemäß Bedingungscode des Coprozessors	
JMPL	Unbedingte Verzeigung mit Programmverbindung (Adreßrettung)	
CALL	Unterprogrammruf	
RETT	Rückkehr aus Trap-Behandlung	
Ticc	Trap-Auslösung gemäß Bedingungscode der Integer Unit	
Umschalten des Register-Fensters		
SAVE	Retten eines Register-Fensters	
RESTORE	Einstellen eines Register-Fensters	

Zugriffe zu Sonderregistern		
RDY	Lesen Y-Register	
RDPSR	Lesen Programmzustandsregister	P
RDWIM	Lesen Fenstermaske (Window Invalid Mask)	P
RDTBR	Lesen Trap-Basisregister	P
WRY	Schreiben Y-Register	
WRPSR	Schreiben Programmzustandsregister	P
WRWIM	Schreiben Fenstermaske (Window Invalid Mask)	P
WRTBR	Schreiben Trap-Basisregister	P
Weitere Steuerbefehle		
UNIMP	Nichtimplementierter Befehl	
IFLUSH	Leeren Befehlscache	

Befehle der Gleitkomma-Verarbeitungseinheit (FPU)

Transportbefehle		
LDF	Laden Gleitkommazahl einfache Genauigkeit	
LDDF	Laden Gleitkommazahl doppelte Genauigkeit	
LDFSR	Laden FPU-Zustandsregister	
STF	Speichern Gleitkommazahl einfache Genauigkeit	
STDF	Speichern Gleitkommazahl doppelte Genauigkeit	
STFSR	Speichern FPU-Zustandsregister	
STDFQ	Speichern FPU-Warteschlange	P

Arithmetik		
FADDs	Addieren einfache Genauigkeit	
FADDd	Addieren doppelte Genauigkeit	
FSUBs	Subtrahieren einfache Genauigkeit	
FSUBd	Subtrahieren doppelte Genauigkeit	
FMULs	Multiplizieren einfache Genauigkeit	
FMULd	Multiplizieren doppelte Genauigkeit	
FDIVs	Dividieren einfache Genauigkeit	
FDIVd	Dividieren doppelte Genauigkeit	
FSQRTs	Quadratwurzel einfache Genauigkeit	
FSQRTd	Quadratwurzel doppelte Genauigkeit	
FCMPs	Vergleichen einfache Genauigkeit	
FCMPd	Vergleichen doppelte Genauigkeit	
FCMPEs	Vergleichen einfache Genauigkeit; Ausnahme, wenn ungeordnet	
FCMPd	Vergleichen doppelte Genauigkeit; Ausnahme, wenn ungeordnet	
Umwandlungsbefehle		
FABSs	Absolutbetrag (einfache Genauigkeit)	
FNEGs	Negieren	
FMOVs	Registertransport	
FsTOi	Wandeln einfache Genauigkeit in ganze Zahl	
FdTOi	Wandeln doppelte Genauigkeit in ganze Zahl	
FiTOs	Wandeln ganze Zahl in einfache Genauigkeit	
FiTOd	Wandeln ganze Zahl in doppelte Genauigkeit	
FsTOd	Wandeln einfache Genauigkeit in doppelte Genauigkeit	
FdTOs	Wandeln doppelte Genauigkeit in einfache Genauigkeit	

Coprozessorbefehle

LDC	Laden Wort	
LDDC	Laden Doppelwort	
LDCSR	Laden Coprozessor-Zustandsregister	
STC	Speichern Wort	
STDC	Speichern Doppelwort	
STCSR	Speichern Coprozessor-Zustandsregister	
STDCQ	Speichern Coprozessor-Warteschlange	P
CPop	Coprozessor-Operationsbefehle	

9.2.6. Speicheradressierung

Sparc-Prozessoren liefern 32-Bit-Byteadressen. Diese werden als virtuelle Adressen der Speicherverwaltungshardware zugeführt bzw. (in Embedded Systems) unmittelbar zur physischen Speicheradressierung verwendet.

Grundsätzlich sind nur zwei einfache Adressierungsweisen vorgesehen: (1) Registerinhalt + Registerinhalt, (2) Registerinhalt + vorzeichengerecht erweiterter Direktwert (Displacement). Je nach Befehl ist die Displacement-Angabe 13, 22 oder 30 Bits lang. Abbildung 9.11 veranschaulicht Einzelheiten der Adressenbildung im Sparc-Prozessor.

Abbildung 9.11 Die Adressierungsweisen der Sparc-Architektur

Hinweis:
Register r0 als Operandenangabe bedeutet "Operand = Null" bzw. "kein Register". Damit können Adreßanteile aus Registern auch weggelassen werden.

Unterprogrammruf

1. Mit CALL-Befehl (Umschaltung des Registerfensters)
Ein Unterprogrammruf mit Umschaltung des Registerfensters besteht aus einem CALL-Befehl mit nachfolgendem SAVE-Befehl. Die Adresse des CALL-Befehls wird in Register r15 gerettet, einem Out-Register des aktuellen Fensters. Der folgende (noch vor der Verzweigung ausgeführte) SAVE-Befehl schaltet auf das nächste Fenster um, so daß die gerettete Adresse vom Unterprogramm in dessen In-Register r31 vorgefunden wird (Abbildung 9.12).

2. Mit JMPL-Befehl
JMPL hat das Format eines Operationsbefehls: 3 Registerangaben oder 2 Registerangaben und ein Displacement. Die Befehlsadresse wird in das im Befehl angegebene Bestimmungsregister gerettet; dann wird aus zwei Registerinhalten oder aus Registerinhalt + Displacement die Verzweigungsadresse bestimmt. Auch hier wird der nachfolgende Befehl noch ausgeführt.

Rückkehr
Es wird grundsätzlich ein JMPL-Befehl verwendet, und zwar mit r0 als Bestimmungsregister (bedeutet: “Befehlsadresse verwerfen” - also: nicht retten). Die andere Registerangabe bezieht sich dann auf das Rettungsregister. Die Displacement-Angabe ist üblicherweise "+8", um aus der geretteten Adresse die erforderliche Rückkehradresse zu errechnen*). Der dem JMPL nachfolgende Befehl wird auch noch ausgeführt. Ist ein "Zurückschalten" des Register-Fensters notwendig, so ist dies zweckmäßigerweise ein RESTORE-Befehl.

*): die Adreßrettung beim Aufruf betrifft hier nicht die Adresse des Folgebefehls, sondern die des Aufrufbefehls. Da der nächste Befehl nach dem Aufruf noch ausgeführt wird, ist die Adresse also um 2 Befehlslängen = 8 Bytes zu erhöhen.

Abbildung 9.12 Registerfenster beim Unterprogrammruf

Erklärung:
1 - Registerfenster vor dem Aufruf; 2 - Registerfenster des gerufenen Unterprogramms; 3 - Registerfenster eines Unterprogramms, das im gerufenden Unterprogramm gerufen wird; 4 - globale Register; 5 - Rettung der Befehlsadresse; 6 - Parameter- und Ergebnisübergabe.

Zum Ablauf:

- **#** das rufende Programm übergibt die Parameter an das Unterprogramm in seinen Registern r14...r8,
- **#** der CALL-Befehl bewirkt die Rettung der Befehlsadresse in Register 15 des Registerfensters 1. Der nachfolgend noch ausgeführte Befehl ist ein SAVE-Befehl, der auf das Registerfenster 2 umschaltet.
- **#** das Unterprogramm findet seine Parameter in den Registern r30...r24 seines Registerfensters 2 vor. Falls es seinerseits ein Unterprogramm aufruft, so stellt es dessen Parameter in seinen Registern r14...r8 bereit usw.
- **#** das Unterprogramm gibt seine Ergebnisse in den Registern r30...r24*) an das rufende Programm zurück.
- **#** der Rückkehrbefehl (JMPL) entnimmt die Verzweigungsadresse aus dem Register r31 und erhöht sie um 8. Der nachfolgend noch ausgeführte Befehl ist ein RESTORE-Befehl, der auf das Registerfenster 1 zurückschaltet.
- **#** das rufende Programm wird nun fortgesetzt. Es findet die vom Unterprogramm errechneten Ergebnisse in seinen Registern r14...r8 vor.

*): Register r31 (das ehemalige r15) enthält die gerettete Rückkehradresse.

Adreßräume
Die Architektur sieht vor, daß auf insgesamt 256 Adreßräume zu je 4 GBytes zugegriffen werden kann. Dazu liefert der Prozessor neben der 32-Bit-Adresse eine Adreßraumangabe (Address Space Identifier ASI). Befehle, die sich ausdrücklich auf einen "alternativen Adreßraum" beziehen (vgl. Abschnitt 9.2.5. - Befehlsbeispiel: LDA = Laden Wort von alternativem Areßraum), enthalten die 8-Bit-ASI-Angabe als Direktwert. Ansonsten liefert der Prozessor zu jedem Zugriff automatisch einen ASI-Wert. Dieser wird aus einem modellspezifischen Festwert gebildet, der in zwei Bitpositionen gemäß der Art des betreffenden Zugriffs modifiziert wird (das eine Bit unterscheidet zwischen Daten- und Befehlszugriff, das andere zwischen Anwender- und Supervisorzustand). Es obliegt der Speicherverwaltungs-Hardware, die ASI-Angabe auszuwerten.

9.2.7. Speicherorganisation

Die Speicherverwaltung der Sparc-Architektur beruht auf dem Prinzip der Seitenverwaltung. Die vom Prozessor gelieferte 32-Bit-Adresse wird über die Speicherverwaltungseinheit in eine physische 36-Bit-Adresse umgesetzt (somit können bis zu 64 GBytes adressiert werden).

TLB
Grundlage der Speicherverwaltungshardware (Cache Controller and Memory Management Unit CMU) bildet ein Adreßumsetzungspuffer (TLB) mit 64 Einträgen. Der TLB ist voll assoziativ organisiert. Die Seitenlänge ist umschaltbar zwischen 4 kBytes, 256 kBytes, 16 MBytes und 4 GBytes. Folgende Angaben sind in den assoziativen Vergleich einbezogen:

- die höchstwertigen 20, 14 oder 8 Adreßbits bzw. gar kein Adreßbit (je nach eingestellter Seitenlänge),
- eine 12-Bit-Kontextangabe. Damit können bis zu 4096 "logische" Adreßräume voneinander abgegrenzt werden. Ein Kontext entspricht praktisch dem einer Task zugewiesenen Adreßraum. Die Kontextangabe im TLB-Eintrag wird mit dem Inhalt eines programmseitig ladbaren Kontextregisters (in der CMU) verglichen.

Der TLB liefert die höchstwertigen 24, 18, 12 bzw. 4 Bits der physischen Adresse (je nach der eingestellten Seitenlänge).

Abbildung 9.13 veranschaulicht die Struktur eines TLB-Eintrages. (Zu näheren Einzelheiten siehe auch die nachfolgenden Abbildungen 9.14 und 9.15.)

Abbildung 9.13 Struktur eines TLB-Eintrags

Sperren von TLB-Einträgen
Es ist programmseitig ladbar, wieviele TLB-Einträge gesperrt werden können. Gesperrte TLB-Einträge werden im Fall eines TLB Miss nicht mehr automatisch geändert. Somit läßt sich der TLB nicht nur verwenden, um einen "klassischen" virtuellen Speicher zu implemen-

tieren, sondern man kann ihn auch softwareseitig direkt laden und für vielfältige andere Adreßumsetzungssaufgaben ausnutzen, z. B. für Zugriffe auf Bildspeicher, für E-A-Zugriffe usw.

Tabellenstrukturen
Die Sparc-CMUs unterstützen eine vierstufige Tabellenstruktur, um virtuelle in physische Adressen umzusetzen (Abbildung 9.14):

1. Stufe: ein Kontext-Zeigerregister in der CMU adressiert eine Kontexttabelle. Das Kontextregister wählt daraus gemäß dem aktuellen Kontext einen Eintrag. Dies ist entweder ein Seitentabellenzeiger oder ein Seitentabelleneintrag.

2. Stufe: ein Seitentabellenzeiger in der Kontexttabelle wird genutzt, um eine Level-1-Seitentabelle zu adressieren. Die höchstwertigen 8 Bits der virtuellen Adresse (Index 1) wählen darin einen Eintrag aus. Auch dieser ist entweder ein Seitentabellenzeiger oder ein Seitentabelleneintrag.

3. Stufe: ein Seitentabellenzeiger in der Level-1-Seitentabelle wird genutzt, um eine Level-2-Seitentabelle zu adressieren. Die Bits 18...23 der virtuellen Adresse (Index 2) wählen darin einen Eintrag aus. Auch dieser ist entweder ein Seitentabellenzeiger oder ein Seitentabelleneintrag.

4. Stufe: ein Seitentabellenzeiger in der Level-2-Seitentabelle wird genutzt, um eine Level-3-Seitentabelle zu adressieren. Die Bits 12...17 der virtuellen Adresse (Index 3) wählen darin einen Eintrag aus. Dieser muß ein Seitentabelleneintrag sein, ansonsten wird eine Ausnahmebedingung wirksam.

Stufenzahl und Seitenlänge
Beides hängt miteinander zusammen: eine Seitenlänge von 4 kBytes erfordert eine vierstufige Tabellenanordnung, eine Seitenlänge von 256 kBytes eine dreistufige, eine von 16 MBytes eine zweistufige, und bei 4 GBytes ist nur noch die Kontexttabelle notwendig. Die Seitenlänge ist in den Tabelleneinträgen nicht codiert. Vielmehr werden die fortlaufenden Tabellenzugriffe (Table Walk) abgebrochen, sobald kein Seitentabellenzeiger mehr gefunden wird, sondern ein Seitentabelleneintrag. Aus der Anzahl der durchlaufenen Stufen (Kontext . . . Level 3) bestimmt die CMU-Hardware die jeweilige Seitenlänge. Die Tabellenzugriffe werden automatisch ausgeführt, ohne Eingriff des Prozessors. Abbildung 9.15 veranschaulicht die Belegungen eines Tabelleneintrags.

Abbildung 9.14 Tabellenstrukturen der Adreßumsetzung

Abbildung 9.15 Tabelleninhalte der Adreßumsetzung

Zugriffsschutz
In den Seitentabelleneinträgen sind drei Bits (ACC2...0) für den Zugriffsschutz vorgesehen. Die erlaubten Zugriffe sind aus Tabelle 9.4 ersichtlich.

ACC-Belegung	Zugriff im Anwendungszustand	Zugriff im Supervisorzustand
0	nur Lesen	nur Lesen
1	Lesen und Schreiben	Lesen und Schreiben
2	Lesen und Ausführen	Lesen und Ausführen
3	Lesen und Schreiben und Ausführen	Lesen und Schreiben und Ausführen
4	nur Ausführen	nur Ausführen
5	nur Lesen	Lesen und Schreiben
6	kein Zugriff	Lesen und Ausführen
7	kein Zugriff	Lesen und Schreiben und Ausführen

Tabelle 9.4 Erlaubte Zugriffe gemäß der ACC-Angabe

9.2.8. Unterbrechungssteuerung

Das Unterbrechungssystem der Sparc-Architektur beruht auf dem Prinzip des Interruptvektors. Interruptvektoren (in der Sparc-Dokumentation als Trap Types bezeichnet) sind grundsätzlich ein Byte lang, so daß bis zu 256 verschiedenen auslösenden Ereignissen jeweils ein eigenes Behandlungsprogramm zugeordnet werden kann. Die Interruptvektoren sind von 0 bis 255 durchnumeriert. Die ersten 128 Interruptvektoren sind für die Hardware vorgesehen, die verbleibenden 128 für die softwareseitige Auslösung (über Ticc-Befehle).

Interrupttabelle
Die Interrupttabelle wird über das Trap-Basisregister TBR adressiert, das die höchstwertigen 20 Bits der Tabellenadresse liefert. Jedem Interrupt sind vier aufeinanderfolgende Worte zugeordnet (somit können in der Tabelle selbst jeweils bis zu vier Befehle der Unterbrechungsbehandlung untergebracht werden, was auf jeden Fall reicht, um zur eigentlichen Behandlungsroutine zu verzweigen).

Interruptablauf

1. nachfolgende Unterbrechungsanforderungen werden zunächst gesperrt,
2. es wird das jeweils nächste Register-Fenster (Trap Window) eingerichtet (durch Weiterschalten des Fensterzeigers),
3. der Befehlszähler (PC + nPC) wird in die Register r17, r18 des Trap Window gerettet,
4. die Interrupttabelle wird über das Register TBR und den Interruptvektor adressiert. Mit der Adresse des Tabelleneintrags wird der Befehlszähler geladen.

Externe Unterbrechungsauslösung
Interrupts aus der "Außenwelt" werden binär codiert über vier Eingänge (Interrupt Request Level IRL 3...0) an den Prozessor signalisiert. Mit der binären Codierung wird die Priorität der aktuellen Unterbrechungsanforderung ausgedrückt. Die Belegung 0 bedeutet "keine Unterbrechungsanforderung", Belegung 1 die niedrigste Priorität, Belegung 15 die höchste (nichtmaskierbare Unterbrechung). Um angenommen zu werden, muß der aktuelle Wert größer sein als die Angabe im PIL-Feld des Prozessor- Zustandsregisters.

Prioritäten
Es gibt insgesamt 32 Prioritäten. Jedem Interrupt ist fest eine bestimmte Priorität zugeordnet (Wert 1 kennzeichnet die niedrigste).

Belegung der Interruptvektoren (Trap Types)
Die Belegung der Interruptvektoren ist zusammen mit der jeweiligen Priorität aus Tabelle 9.5 ersichtlich.

Interruptvektor	Bedeutung	Priorität
0	Rücksetzen	1
1	Ausnahme bei Befehlszugriff	2
2	Unzulässiger Befehl	3
3	Privilegierter Befehl	4
4	Gleitkommaverarbeitung abgeschaltet	5
5	Register-Fenster-Überlauf	7
6	Register-Fenster-Unterlauf	8
7	Nichtintegrale Speicheradresse	9
8	Gleitkomma-Ausnahme	10
9	Ausnahme bei Datenzugriff	12
10	Überlauf bei einer TAG-Operation	13
17 .. 31	Externe Unterbrechungen	15 .. 29
36	Coprozessor abgeschaltet	6
40	Coprozessor-Ausnahme	11
128 .. 255	Software-Traps (Ticc-Befehle)	14

Tabelle 9.5 Interruptvektoren der Sparc-Architektur

9.3. Ausgewählte Merkmale weiterer Architekturbeispiele

In diesem Abschnitt sollen anhand ausgewählter Einzeldarstellungen (Tabellen 9.6...9.8, Abbildungen 9.16...9.27) typische Merkmale verschiedenartiger Rechnerarchitekturen veranschaulicht werden.

Add	Insert entry into queue
Add packed decimal string	Insert integer into variable bit field Locate character
Add with carry	Match character string
Add one and branch	Move complemented
Arithmetic shift	Move negated
Arithmetic shift and round packed decimal string	Move numerical quantities
Bit clear	Move address
Bit set	Move character string
Bit test	Move packed decimal string
Clear	Move translated characters
Compare numeric	Move zero extented numerical quantities
Compare character string	Multiply
Compare packed decimal string	Multiply packed decimal strings
Compare variable bit field to integer	Remove entry from queue
Convert data types	Rotate longword
Decrement	Scan character string
Divide	Span character string
Divide packed decimal string	Subtract
Extended divide	Subtract packed decimal strings
Extented multiply	Subtract with carry
Extract variable bit field	Subtract one and branch
Find first bit in variable bit field	Test
Increment	Exclusive Or
Index (calculate array index)	

Tabelle 9.6 Operationsbefehle einer typischen CISC-Architektur (DEC VAX)

Bezeichnung	der Operand ist gegeben durch
Short literal	N
Index	<(b + (s · <Rn>))>
Register	<Rn>
Register deferred	<<Rn>>*)
Autodecrement	<<Rn>>; zuvor <Rn> := <Rn> - s *)
Autoincrement	<<Rn>>; danach <Rn> := <Rn> + s *)
Autoincrement deferred	<<<Rn>>>; danach <Rn> := <Rn> + s *)
Displacement	<(Rn> + D)> *)
Displacement deferred	<<(Rn> + D)>> *)

- auf Befehlszähler (PC = R15) bezogene Adressierung -	
Immediate	N
Absolute	<A>
Relative	<(<PC> + D)>
Relative deferred	<(<PC> + D)>

Tabelle 9.7 Adressierungsverfahren einer typischen CISC-Architektur (DEC VAX)

Erklärung:

<...> Inhalt von...,
Rn Register,
N Direktwert (im Befehl),
A Absolutadresse (im Befehl),
s Operandenlänge (1, 2, 4, 8, 16 Bytes),
D Displacement (Byte, Wort, Langwort),
b Basisadresse des Index Mode
*) zusätzlich kann im Befehl noch ein Indexregister angegeben werden.

Abbildung 9.16 Blockschaltbild des Mikrocontrollers AT89C1051 (Atmel)

Erklärung:
8-Bit Mikrocontroller mit 8051-Architektur (diese wurde Anfang der 80er Jahre eingeführt - und noch heute werden auf dieser Grundlage neue Mikrocontroller entwickelt (sie sind schneller, brauchen weniger Strom usw.) Eine typische CISC-Einadreßmaschine in Harvard-Architektur (Heft 5). 1 - Datenspeicher (EEPROM + RAM); 2- Programmspeicher (Flash-ROM), 3 - Rechenwerk (Arithmetik-Logik-Einheit ALU); 4- Befehlsregister; 5 - Zählnetzwerk des Befehlszählers; 6 - Befehlsadreßregister; 7, 8 - Zwischenregister für die zu verknüpfenden Daten; 9 - E-A-Ports; 10 - Takterzeugung; 11 - Programmieren und Rücksetzen. Es folgen jene Funktionseinheiten, die Architekturmerkmale darstellen. A - Akkumulator; B - Hilfsregister (für Multiplikation und Division); S - Stackpointer; P - Befehlszähler; F - Flag- und Zustandsregister (Programmstatuswort PSW); D - Datenzeiger (Adreßregister zur Datenadressierung). Die weiteren in der Abbildung dargestellten Register (z. B. 4 und 6...8) haben mit der Architektur an sich nichts zu tun; sie erfüllen allein technische Aufgaben (Halten, Zwischenpuffern usw.).

Abbildung 9.17 Blockschaltbild der Mikrocontroller-Familie PIC16C5X (Microchip)

Erklärung:
8-Bit-Mikrocontroller als RISC-Einadreßmaschine in Harvard-Architektur. Einer der einfachsten Controller-Architekturen des Marktes (es sind nur 35 verschiedene Befehle zu lernen - die wollen allerdings trickreich angewendet sein...). Die Registerstruktur und die Breite der einzelnen internen Datenwege sind erkennbar - es handelt sich schon fast um eine

richtige RTL-Darstellung. 1 - Programmspeicher; 2 - Befehlsregister; 3 - Befehlszähler; 4 - Hardware-Stack (zum Unterprogrammaufruf); 5 - Akkumulator (der hier W-Register heißt (damit's nach was Besonderem aussieht)); 6 - Flag- und Zustandsregister; 7 - Rechenwerk (Arithmetik-Logik-Einheit ALU); 8 - Datenspeicher (hier als Allzweckregistersatz ausgelegt); 9 - Bankregister zum Adresieren des Datenspeichers (verlängert die Registeradreßangabe des Befehls); 10 - E-A-Ports. Die Befehle sind 12 Bits lang (vgl. die Zugriffsbreite des Programmspeichers 1).

Abbildung 9.18 Elementare Architekturmerkmale der 64-Bit-Mips-Prozessoren im Überblick: CPU-Register und Befehlsformate (NEC)

Erklärung:
Eine der klassischen RISC-Architekturen mit großen Registersätzen und vergleichsweise einfachen Befehlen (direkte Konkurrenz zu Sparc). 1 - Universalregistersatz (von den 32 Registern sind 30 freizügig nutzbar); 2 - Hilfsregister für Multiplikation und Division; 3 - Befehlszähler; 4 - Hilfsregister (1 Bit) zur Unterstützung "unteilbarer" Speicherzugriffe; 5 - Gleitkommaregistersatz (32 Register zu 64 Bits); 6 - Steuerregister der Gleitkommaverarbeitung. Die zentrale Verarbeitungseinheit (CPU) kann durch Coprozessoren ergänzt werden. Die beiden wichtigsten: der System Control Coprocessor (CP0) und die Gleitkommaverarbeitungseinheit (FPU, CP1). CP0 enthält u. a. die Speicherverwaltung (mit TLB, die Cache-Steuerung usw.). Alle Befehle sind 32 Bits lang. Tabelle 9.7 erklärt die einzelnen Felder.

Bezeichnung	Erklärung	Bezeichnung	Erklärung
op	Operationscode (6 Bits)	immediate	Direktwert oder Displacement (16 Bits)
rs	Quellregister (5 Bits)	target	Sprungzieladresse (26 Bits)
rt	Zielregister oder 2. Quellregister oder Verzweigungsbedingung (5 Bits)	sa	Verschiebedistanz (5 Bits)
rd	Zielregister (5 Bits)	func	Funktionsauswahl (6 Bits)

Tabelle 9.8 Zu den Befehlsformaten der Mips-Architektur

Hinweise:

1. In Abbildung 9.18 fehlt das Flagregister. Die Mips-Architektur hält die entsprechenden Zustandsangaben nicht in der CPU, sondern im Coprozessor CP0.
2. Registersatz und Befehlsformate der Alpha-Architektur (DEC/Compaq) sind ähnlich ausgelegt: 32 Universalregister, 32 Gleitkommaregister zu je 64 Bits, 32-Bit-Befehle.

Abbildung 9.19 Blockschaltbild eines PowerPC-Prozessors (Motorola)

Erklärung:
Eine als RISC bezeichnete Maschine, die aber alles andere als einfach ist. 32-Bit-Ganzzahlverarbeitung, 64-Bit-Gleitkommaverarbeitung, 128-Bit-Vektorverarbeitung, 32-Bit-Befehle, Superskalarauslegung (Heft 5). 1 - voreilendes Befehlslesen; 2 - Verzweigungsbeschleuigung mit Springzielpuffer (Branch Target Cache BTC) und Branch History Table (BHT); 3 - Speicherverwaltung (Memory Management Unit MMU; mit TLB) für Befehlszugriffe; 4 - Speicherverwaltung (mit TLB) für Datenzugriffe; 5 - Befehlscache (32 kBytes); 6 - Datencache (32 kBytes); 7 - Befehlswarteschlange; 8 - Befehlsverteiler; 9 - zwei Ganzzahl-Verarbeitungseinheiten mit vorgeordneten Puffern; 10 - Vektorverarbeitungeinheit (AltiVec-Architektur)*); 11 - Universalregistersatz*); 12 - Gleitkommaregistersatz*), rechts daneben die Gleitkommaverarbeitungseinheit; 13 - Befehlserledigungseinheit (Heft 5); 14 - Steuerung für externen (L2-) Cache; 15 - Prozessorbussteuerung; 16 - Speicherzugriffsbeschleunigung (mit Puffern, Warteschlangen usw.).

*): jeweils mit 6 umbenennbaren Pufferregistern (zur Umbenennung siehe Heft 5).

Abbildung 9.20 Der Registersatz eines PowerPC-Prozessors aus Sicht des Anwendungsprogrammierers (Motorola)

Erklärung:
1 - Universalregister (32 Register zu 32 Bits); 2 - Gleitkommaregisster (32 Regiter zu 64 Bits); 3 - Bedingungs-, Zustands- und Steuerregister; 4 - Register der AltiVec-Architektur (darunter 32 Vektorregister zu 128 Bits); 5 - diverse Register zur Leistungsmessung und Fehlersuche (vgl. die Leistungsmeßzäher der Intel-Prozessoren); 6 - Linkregister (Adreßrettung beim Unterprogrammruf); 7 - Programmausnahmeregister für Ganzzahlverarbeitung; 8 - Zählregister (Schleifendurchlaufzählung in bestimmten Verzweigungsbefehlen).

Abbildung 9.21 Blockschaltbild der Signalprozessor-Familie TMS320C54x (Texas Instruments)

Erklärung:
Es handelt sich um einen noch vergleichsweise einfachen, kostenoptimierten Signalprozessor (für Audio-Anwendungen usw.). Die Abbildung soll vor allem einen Eindruck vom Aufbau der Verarbeitungshardware vermitteln. Programm- und Datenspeicher sind nicht dargestellt. Es ist aber zu erkennen, daß mehrere Bussyteme vorgesehen sind (hierüber kann gleichzeitig auf Daten und auf Befehle zugegriffen werden). 1- Adreßbildung für Befehls- und Datenadressierung; 2 - Bussysteme; 3 - Verarbeitungsschaltungen; 4 - Multiplizierer; 5 - Addierer*); 6 - Arithmetik-Logik-Einheit (ALU); 7- Verschiebeeinheit. Die einzelnen Einheiten wurde in Hinsicht auf typische Verarbeitungsabläufe der Signalverarbeitung zusammengeschaltet. Auch die Befehlsliste ist auf solche Anwendungen hin ausgelegt (beides dürfte kaum auf Anhieb zu verstehen sein - bitte nicht versuchen, allzu tief einzudringen...).

*): Multiplizierer 4 und Addierer 5 bilden zusammen die Multiplizierer-Akkumulator-Einheit (MAC Unit). Diese Verkettung dient vor allem zum Berechnen von Skalarprodukten (Sum-of-Products; Heft 5).

Abbildung 9.22 Ein Superskalar-Signalprozessor (Prozessorfamilie TMS320C6x; Texas Instruments). Blockschaltbild

Erklärung:
Die Abbildungen 9.22 bis 9.24 betreffen Superskalar-Signalprozessoren der obersten Leistungsklasse. Signalprozessoren sind typischerweise als Harvard-Maschinen ausgelegt. 1 - Programmspeicher (256 Bit Zugriffbreite); 2 - Datenspeicher (32 Bits Zugriffsbreite); 3 - Speicherbus; 4, 5 - zwei Verarbeitungspipelines A, B mit je 4 Verarbeitungswerken.

Abbildung 9.23 Die Verarbeitungspipelines des Superskalar-Signalprozessors (Texas Instruments)

Erklärung:
A, B - Verarbeitungspipelines; 1, 2 - Registersätze (je 16 Register zu 32 Bits); 3 - L-Einheiten; 4 - S-Einheiten; 5 - M-Einheiten; 6 - D-Einheiten; 7 - Schreibdatenwege; 8 - Lesedatenwege; 9 - Speicheradressen. Die Verarbeitungswerke L, S, M, D beider Pipelines A, B sind jeweils gleichartig aufgebaut. Die Verarbeitungswerke sind für unterschiedliche Aufgaben ausgelegt:

- **#** L-Einheiten: arithmetische Operationen, logische Verknüpfungen, Vergleichsoperationen, Operandenformatwandlungen,
- **#** S-Einheiten: arithmetische Operationen, Verschiebe- und Bitfeldoperationen, logische Verknüpfungen, Verzweigungen,
- **#** M-Einheiten: Multiplikation,
- **#** D-Einheiten: Adreßrechnung, Speicherzugriffe.

Abbildung 9.24 Befehlsstruktur des Superskalar-Signalprozessors (Texas Instruments)

Erklärung:
Die insgesamt 8 Verarbeitungswerke werden gleichzeitig mit jeweils einem Maschinenbefehl von 32 Bits Länge versorgt ($8 \cdot 32 = 256$ Bits = Zugriffsbreite des Programmspeichers; vgl. Abbildung 9.22). Jeder der Befehle (Instructions) A, B...H betrifft genau eines der in Abbildung 9.23 dargestellten Verarbeitungswerke. In jedem der Befehle ist ein Bit P zur Steuerung der Parallelverarbeitung vorgesehen. Ist ein P-Bit gesetzt, so kann der rechts daneben angeordnete Befehl gleichzeitig (parallel) ausgeführt werden.Das Setzen der P-Bits ist Sache der Programmentwicklung (also typischerweise des Compilers).

a) das allgemeine Format mit 8 Befehlen zu 32 Bits und einem P-Bit je Befehl,

b) alle P-Bits gelöscht. Keine Parallelausführung. Ausführungsreihenfolge A, B, C usw.

c) eine beispielhafte Belegung der P-Bits. Ausführungsreienfolge:
   1. Taktzyklus: A,
   2. Taktzyklus: B,
   3. Taktzyklus: C, D, E (Parallelausführung endet bei E, da dessen P-Bit gelöscht),
   4. Taktzyklus: F, G, H.

Abbildung 9.25 Eine weitere VLIW-Architektur: Majc (Sun)

Erklärung:
Majc soll total universell sein, also u. a. auch Aufgaben der Signalverarbeitung übernehmen und enorme Datenströme bewältigen können. Wir wollen nur die Anordnung der Verarbeitungswerke und die Besonderheiten der Befehlsgestaltung kurz vorstellen. 1...4: Verarbeitungswerke; 5 - Universalregistersatz. Der Registersatz ist wirklich universell, da er alle Datenstrukturen (ganze Binärzahlen, Gleitkommazahlen usw.) aufnehmen kann. Jede Verarbeitungeinheit hat 3 Lesedatenwege und einen Schreibdatenweg zu den Universalregistern. Anzahl der Universalregister: insgesamt 224. (Die variabel aufgeteilt werden können. Eine typische Aufteilung: 96 globale Register und je 32 Register für jede der 4 Verarbeitungswerke. Jedes Verarbeitungswerk kann maximal 128 Register adressieren.)

Abbildung 9.26 Majc: Befehlsstrukturen (Sun)

Ein Befehlspaket (Instruction Packet) ist normalerweise 128 Bits lang. Es enthält 4 Befehle zu 32 Bits. Jeder Befehl enthält 4 Registeradreßangaben zu 7 Bits (in manchen Befehlen dient das rd-Feld als Verlängerung des Operationscodes).

In den ersten beiden Bits des ersten Befehls ist codiert, wie lang das Paket tatsächlich ist. Es kann 1, 2, 3 oder 4 Befehle lang sein. Enthält das Paket nur einen Befehl, so betrifft dieser das erste Verarbeitungswerk, enthält es 2 Befehle, so betreffen diese das erste und das zweite Verarbeitungswerk usw. Die Verarbeitungswerke 2, 3, 4 sind gleichartig aufgebaut. Verarbeitungswerk 1 ist reichhaltiger ausgestattet, um wirklich alle Funktionen (einschließlich Systemsteuerung usw.) ausführen zu können. Dies ist - verglichen mit Abbildung 9.24 - ein anderes Prinzip, die Parallelausführung zu steuern:

- **#** gemäß Abbildung 9.24: es wird stets für alle (spezialisierten) Verarbeitungseinheiten jeweils ein Befehl geholt, und über die P-Bits wird die Parallelausführung gesteuert. Nicht parallel ausführbare Befehle müssen hintereinander abgearbeitet werden.
- **#** hier: Befehle, die gar nicht parallel ausführbar sind (inhärent sequentielle Befehlsfolgen) werden stets dem Verarbeitungswerk 1 übertragen. Können 2 Befehle parallel ausgeführt werden, so werden die (universellen) Verarbeitungswerke 1 und 2 genutzt usw.

Abbildung 9.27 IA-64: der Registersatz aus Sicht des Anwendungsprogrammierers (Intel)

Erklärung:
1 - Universalregister (128 Register zu 64 Bits); 2 - Belegungsflags (1 Bit je Universalregister); 3 - Gleitkommaregister (128 Register zu 82 Bits); 4 - Prädikatregister (64 Register zu einem Bit); 5 - Verzweigungsregister[1)]; 6 - Befehlszähler; 7 - Zeigerregister auf Universalregistersatz[2)]; 8 - Anwender-Maskenregister[3)]; 9 - anwendungsspezifische Register; 10 - Leistungsmeßregister; 11 - Prozessor-Identifikation.

Anmerkungen:

1) enthalten Verzweigungsadressen für indirekte Verzweigungen,

2) mit einem Teil der Universal- und der Gleitkommaregister kann man einen Register-Stack aufbauen, in dem sich z. B. Parameter an Unterprogramme praktisch “verlustlos” (= ohne Transporte) übergeben lassen. Das Prinzip haben wir bereits anhand der Sparc-Architektur kurz erläutert. Das Zeigerregister entspricht dem Fensterzeiger der Sparc-Architektur. Der Unterschied: bei Sparc handelt es sich um ein starres Weiterschalten (der Inhalt des Fensterzeigers wird stets um 16 erhöht bzw. vermindert) und somit um Stack Frames fester Größe, bei IA-64 kann hingegen die Größe jedes einzelnen Stack Frame programmseitig vorgegeben werden.

3) hierüber kann der Anwendungsprogrammierer verschiedene Zustandsangaben abfragen.