



Kapitel 1

Landschaft der Prozessoren

1

1	Landschaft der Prozessoren	
1.1	Prozessortypen	3
1.1.1	Personal Computer und ihre Prozessoren	3
1.1.2	Embedded Prozessoren	5
1.1.3	Signalprozessoren	5
1.1.4	Mehrprozessorsysteme und Supercomputer	6
1.1.5	Einsatzgebiete für Prozessoren	6
1.2	Technologische Randbedingungen	8
1.3	Befehlssätze	11
1.3.1	Grundlegende Eigenschaften von Befehlssätzen	11
1.3.2	Registersätze und Programmiermodell	11
1.3.3	Unterscheidung von Befehlssätzen nach Registerbreite...	12
1.3.4	Angabe von Operanden und Ergebnis	13
1.3.5	Adressierungsarten	15
1.3.6	Befehlsformate	16
1.3.7	Befehlstypen	19
1.3.8	Parallelverarbeitung auf Ebene des Befehlssatzes	23
1.3.9	Exkurs: Stackmaschinen	24

1 Landschaft der Prozessoren

Computergesteuerte Geräte haben unseren Alltag längst vollständig durchdrungen. Jedes davon enthält mindestens einen Prozessor. Bekannt sind vor allem die in den Personal Computern eingesetzten Prozessoren von Intel. Meist wird dabei aber übersehen, dass es eine sehr große Anzahl von verschiedenen Prozessoren gibt, die in den Geräten des Alltags eingesetzt werden. Dieses Kapitel soll einen kurzen Überblick darüber geben.

1.1 Prozessortypen

1.1

➤ 1.1.1 Personal Computer und ihre Prozessoren

Die Käufer eines Personal Computer (Desktop-Systeme bzw. Arbeitsplatzrechner) sind oft bereit, viel Geld für ein System auszugeben, das dem jeweils aktuellen Stand der Technik entspricht. Die Software-Industrie liefert leistungshungrige Programme, die die entsprechende Leistung der Geräte auch verlangen. Die Prozessor-Hersteller führen einen gnadenlosen Kampf, immer leistungsfähigere Prozessoren zu bauen. Im Bereich der Personal Computer wird oft die Taktfrequenz oder die Cache-Größe der Prozessoren erfolgreich als Verkaufsargument ins Feld geführt.

In einem Personal Computer stecken deutlich mehr Komponenten als nur der Prozessor. Viele Komponenten sind erforderlich, um die geforderten Aufgaben zu erfüllen, etwa Festplatten, Netzwerkanschlüsse und Bussysteme zu bedienen.

Im Einzelnen finden wir in einem modernen Personal Computer folgende Komponenten, deren Zusammenspiel in Abbildung 1.1 gezeigt ist¹:

- Der Prozessor ist über einen schnellen Bus mit den Komponenten auf der *Hauptplatine* (Main Board oder Mother Board) verbunden. Diejenigen Komponenten, die dem Prozessor zuarbeiten und ihn mit Daten versorgen bezeichnet man als *Chipsatz*.
- Zwischen Prozessor und dem Chipsatz wird seit 1998 der so genannte *Front Side Bus* (FSB) eingesetzt. Wichtig zu wissen ist, dass die Taktfrequenz, mit der dieser Bus betrieben wird, deutlich niedriger ist als die Taktfrequenz, mit welcher der Prozessor arbeitet.
- Der Chipsatz hat so viele Aufgaben zu erfüllen und so viele Geräte und Busse zu bedienen, dass er schon seit langem in zwei Teile geteilt wird: Die North Bridge und die South Bridge. Diese geographischen Bezeichnungen

¹Diese Sicht ist stark an Intel angelehnt.

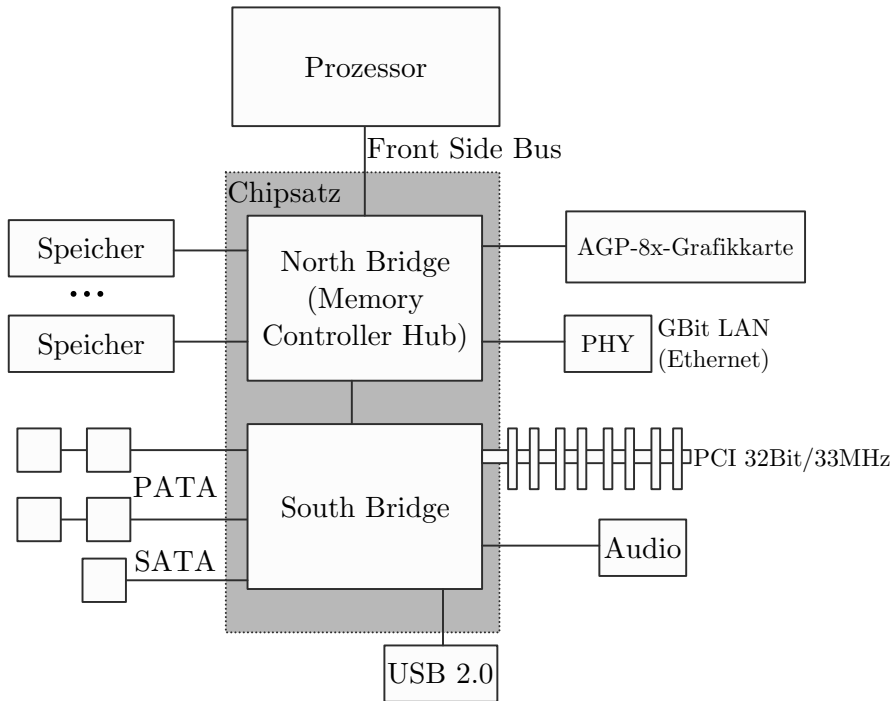


Abbildung 1.1. Komponenten aktueller Personal Computer

sehen den Prozessor im Norden (auf der Landkarte oben gezeichnet; siehe auch Abbildung 1.1).

- Die North Bridge bedient sowohl den Hauptspeicher als auch die Grafikkarte und die physische Schicht (PHY) der Netzwerkkarte. Grafikkarten kommunizieren mit dem Chipsatz über den *Advanced Graphics Port* (AGP). Beim Netzwerk ist das extrem schnelle Gigabit-Ethernet heute Standard.
- Die South Bridge bedient die langsameren Busse und peripheren Geräte; dies sind auch die historisch älteren Komponenten. Dazu gehören die Festplatten an parallelen und neuerdings auch seriellen ATA-Bussen. Ferner die *Peripheral Component Interconnect*-Busse (PCI) und der *Universal Serial Bus* (USB).

Am weitesten verbreitet sind in Personal Computern heute Prozessoren von Intel und AMD, die über den so genannten X86-Befehlssatz verfügen (er wird auch oft mit IA-32 oder i386 bezeichnet). Dies ist der Befehlssatz, den Intel mit dem 80386 eingeführt und seither nicht mehr wesentlich verändert hat. Dieser Befehlssatz ist auch in nachfolgenden Prozessoren von Intel und

AMD implementiert (486, Pentium, Athlon). Eine Ausnahme ist der Itanium-Prozessor, mit dem eine grundlegend neue Architektur und ein neuer Befehlssatz (IA-64) eingeführt wurde.

➤ 1.1.2 Embedded Prozessoren

Unter Embedded Systemen versteht man Geräte, die durch einen oder mehrere Prozessoren gesteuert werden, die man aber nicht als Computer bezeichnet. Bei diesen Embedded Systemen finden wir eine ganz andere Situation vor als im Bereich der Personal-Computer-Systeme. Die dort verwendeten Prozessoren (Embedded Prozessoren) müssen ganz anderen Anforderungen genügen als Prozessoren, die in Desktop-Systemen eingebaut werden:

- Die zur Verfügung stehende elektrische Leistung ist begrenzt. Bei mobilen Geräten wie etwa Telefonen soll die Zeit, die das Gerät mit einer Akkuladung betrieben werden kann, möglichst groß sein. Dadurch muss der Bedarf an elektrischer Leistung möglichst gering gehalten werden.
- Der verfügbare Platz ist begrenzt. Prozessoren, die in Chipkarten eingebaut werden, müssen beispielsweise sehr klein sein.
- Für Embedded Prozessoren wollen die Hersteller der Geräte möglichst wenig ausgeben, da ja die verbauten Prozessoren kein im Vordergrund stehendes Marketing- und Verkaufsargument sind wie bei den Desktop-Systemen.

Diese Randbedingungen zu erfüllen ist oft nicht ganz einfach, insbesondere dann, wenn die Anforderungen an Rechenleistung und Speicherplatz groß sind.

Bei den Embedded Prozessoren ist der Integrationsgrad höher, oft umfassen sie mehr als nur den Rechnerkern. Insbesondere sind Memory Controller und Komponenten zur Busansteuerung mit auf dem Prozessor-Chip untergebracht. Prozessoren, mit denen sich die Peripherie ansteuern lässt, heißen *Controller* (oder *Mikro-Controller*). Werden gar keine externen Komponenten mehr benötigt, so spricht man von einem *System-on-a-Chip*. Als starken Kontrast zur Architektur von Personal Computern aus Abbildung 1.1 zeigt die Abbildung 1.2 die Architektur der Playstation Portable (PSP) von Sony nach [14]. Dort ist alles außer dem Hauptspeicher auf einem einzigen hochintegrierten Chip untergebracht.

➤ 1.1.3 Signalprozessoren

Signalprozessoren sind Prozessoren, deren Befehlssatz speziell für die Realisierung von Algorithmen der digitalen Signalverarbeitung geeignet ist (siehe dazu Abschnitt 1.3.7). Derartige Prozessoren sind praktisch stets in Embedded Systemen zu finden. In der Regel ist jedoch ein Signalprozessor von einem

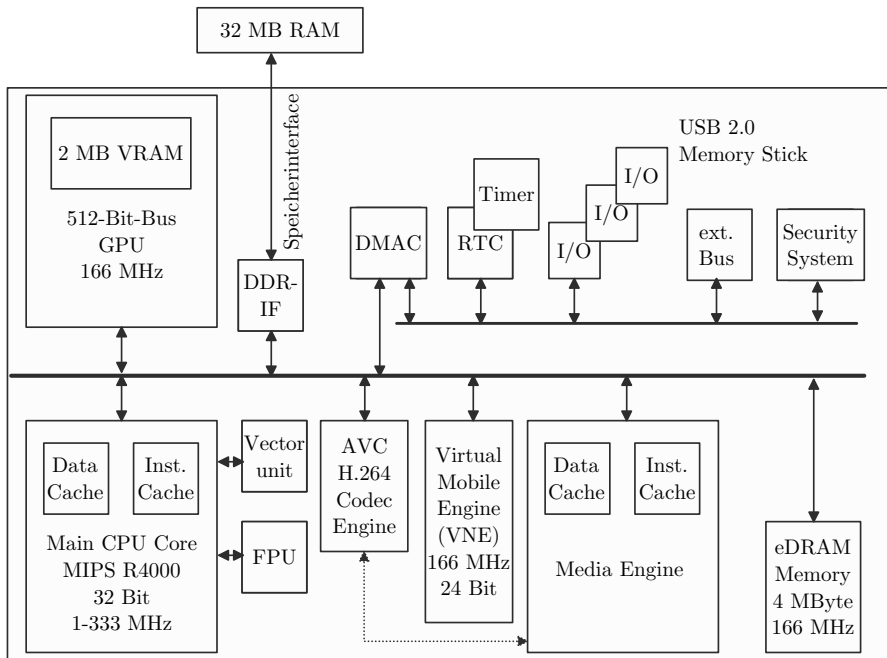


Abbildung 1.2. Architektur von Sonys Playstation Portable gemäß [14]. Mit Ausnahme der 32 MByte Speicher ist alles auf einem Chip untergebracht (System-on-a-Chip). den Kern bildet ein 32-Bit-MIPS R4000, der je nach Belastung mit einer Taktfrequenz von einem bis 333 MHz betrieben wird

Controller begleitet, der Steuerungsaufgaben übernimmt. Typische Einsatzgebiete sind Mobiltelefone und Hardware für Empfang und/oder Verarbeitung von Videosignalen.

➤ 1.1.4 Mehrprozessorsysteme und Supercomputer

Die bisher besprochenen Systeme begegnen uns praktisch überall in unserem Alltag. Daneben gibt es viele Spezialrechner und Supercomputer (Großrechner), die eine extrem hohe Rechenleistung erbringen können. Sie werden häufig von Großforschungseinrichtungen betrieben und verwendet. Diese Supercomputer sind aus vielen Einzelprozessoren der oben genannten Typen aufgebaut. Beispielsweise ist der im Herbst 2005 leistungsfähigste Rechner BlueGene/L von IBM für das Lawrence Livermore National Laboratory in Kalifornien mit 131.072 PowerPC 440-Prozessoren mit 0,7 GHz Taktfrequenz gebaut worden. Er wird für physikalische Simulationen eingesetzt.

Tabelle 1.1. Prozessoren und ihre Haupteinsatzgebiete

Prozessortyp	Einsatzgebiet	Anmerkung
Intel Pentium	Desktop	Mainstream
Intel Itanium	Server/Desktop	64-Bit
AMD Athlon	Desktop	X86-Befehlssatz
AMD Opteron	Server/Desktop	64-Bit
IBM PowerPC	z.B. (noch) Apple Macintosh, Supercomputer	
HP/DEC Alpha	Desktop, Supercomputer	
SUN SPARC	Workstations	
MIPS	Embedded (Chipkarten)	
ARM (Advanced RISC Machine)	PDAs, MP3-Player, Gameboy	Erster RISC (von Acorn) in einem Desktop (1987)
Transmeta Crusoe	Mobile (Notebooks, PDAs, Thin Clients)	Code Morphing
Atmel	MP3-Player sowie div. Embedded Systeme	kostengünstige Controller
TriCore	Siemens	Embedded Controller; synthetisierbar
TI TMS320C64xx	Signalprozessor	Video(de-)codierung
Ajile Systems aj-100	Java Prozessor	Kann Bytecode ausführen

➤ 1.1.5 Einsatzgebiete für Prozessoren

Tabelle 1.1 listet einige Prozessoren und ihre Hauptanwendungsgebiete auf. Die Angaben sind lediglich exemplarisch zu sehen. Es gibt erstens erheblich mehr Prozessoren und zweitens viel mehr Einsatzgebiete.

Übung 1.1.1 In welchen Geräten aus Ihrem Umfeld sind (wahrscheinlich) Mikrocontroller eingebaut?

1.1.1

1.2 Technologische Randbedingungen

Im Jahr 1970 hat die so genannte VLSI-Technik (*Very Large Scale Integration*) Marktreife erlangt. Die VLSI-Technik erlaubt es, viele Transistoren einschließlich deren Verdrahtung untereinander auf einem einzelnen Siliziumplättchen mit wenigen Quadratzentimetern Größe unterzubringen, dem Chip.

Bis dahin wurden Rechenwerke und Computer durch Verlöten einzelner diskreter Transistoren aufgebaut. Der erste in VLSI-Technik hergestellte Prozessor war der 4004 von Intel, ein vier-Bit-Prozessor bestehend aus 2200 Transistoren. Seither hat diese Technik eine beispiellos rasante Entwicklung durchgemacht. Die Anzahl der Transistoren, die auf einem Chip untergebracht werden können, hat sich während dieses Zeitraums etwa alle zwei Jahre verdoppelt. Der Intel-Mitbegründer Gordon Moore hatte diese Entwicklung bereits 1970 vorausgesagt. Sie ist als das *Moore'sche Gesetz* bekannt. Abbildung 1.3 verdeutlicht die Entwicklung anhand einer Grafik.

Die kleinsten Strukturen, die auf einem Chip erzeugt werden, messen weniger als 100 nm (Nanometer), also weniger als 10^{-7} Meter.

Ein damit verbundenes Problem ist, dass die Leistungsaufnahme der Prozessoren ebenso exponentiell mitwächst. Steigende Leistungsaufnahme bedeutet höheren Stromverbrauch. Dieser ist unerwünscht, weil er die Akkulaufzeit mobiler Geräte (Telefone, Notebooks) verkürzt. Ein weiterer gravierender Nachteil ist die damit verbundene Umweltbelastung. Man kann davon ausgehen, dass etwa 10% der erzeugten elektrischen Energie von IT-Systemen verbraucht wird. Eine höhere Leistungsaufnahme ist auch mit höherer Abwärme verbunden. Diese Abwärme muss durch Kühlsysteme abgeführt werden, denn sonst würden die Prozessoren durch Überhitzung zerstört.

Die Leistungsaufnahme von Prozessoren hängt allerdings nicht nur von der Transistorzahl ab, sondern auch von der Höhe der Versorgungsspannung und der Taktfrequenz, mit der ein Prozessor betrieben wird. Sehr vereinfacht gesprochen, gilt nach [26] folgender Zusammenhang für die Leistungsaufnahme. Wir betrachten nur denjenigen Anteil, der aus dem Schalten der Transistoren resultiert²:

$$P \propto U^2 \cdot A \cdot C \cdot f$$

Diese Formel besagt, dass die Leistungsaufnahme P proportional ist zum Produkt aus der Taktfrequenz f , der Aktivität A der Transistoren (nicht alle Transistoren schalten zu jedem Taktschritt), der kapazitiven Last C an den Transistorausgängen (die durch die Anschlussleitungen gebildeten Ka-

²In die Leistungsaufnahme gehen noch andere additive Terme ein, wie etwa Leckströme, die wir hier vernachlässigen.

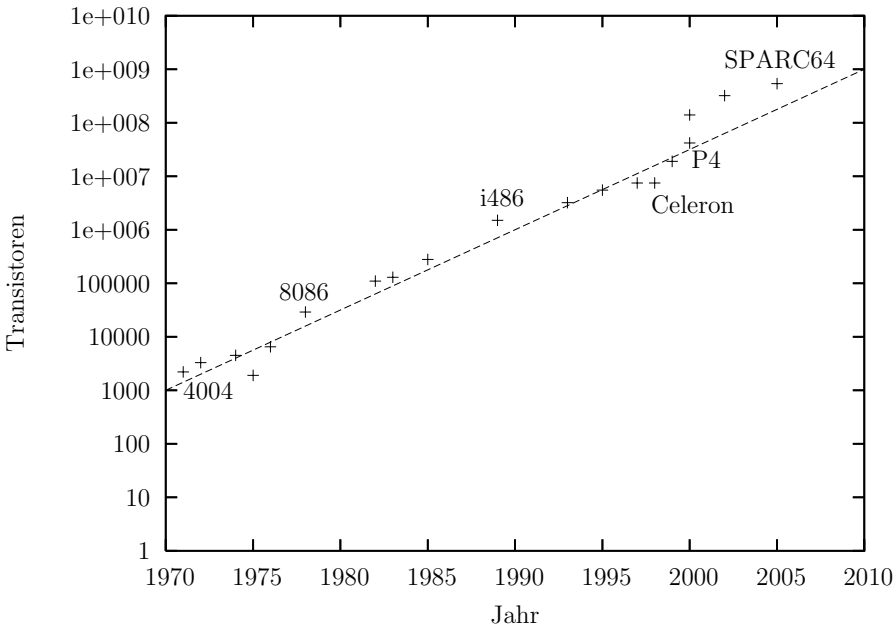


Abbildung 1.3. Entwicklung der Transistorzahlen von Prozessoren der letzten 40 Jahre. Die Ordinate ist logarithmisch skaliert. Die Kreuze entsprechen einzelnen Prozessoren. Die eingezeichnete Gerade zeigt ideales exponentielles Wachstum an. Das Wachstum der Transistorzahlen ist bislang also etwa exponentiell verlaufen (Verdoppelung alle zwei Jahre)

pazitäten/Kondensatoren müssen bei Schaltvorgängen geladen werden) und dem Quadrat der Versorgungsspannung U .

Wachsende Transistorzahlen steigern also die Leistungsaufnahme über den Term C und indirekt auch über die Aktivität A . Leider steigert auch eine Erhöhung der Taktfrequenz die Leistungsaufnahme. Die drastischen Steigerungsraten der Taktfrequenz in den vergangenen Jahren waren nur deshalb möglich, weil es gelungen ist, die Versorgungsspannung immer niedriger zu halten. Das Halbieren von U reduziert die Leistungsaufnahme auf ein Viertel. Unglücklicherweise beeinflusst die Versorgungsspannung die maximal mögliche Taktfrequenz negativ. Die Unterscheidung zwischen 0- und 1-Zuständen wird nämlich mit steigender Taktfrequenz und bei sinkender Versorgungsspannung schwieriger.

Zwei Ansätze werden in letzter Zeit immer stärker verfolgt: Einerseits versucht man, gerade nicht benötigte Teile eines Prozessors abzuschalten, was den Term A beeinflusst. Andererseits werden Strategien zur Parallelverarbeitung verfolgt. Die Idee basiert auf der theoretischen Überlegung, dass zwei

parallele Einheiten, die mit einer Taktfrequenz f betrieben werden, dieselbe Arbeit leisten können wie eine Einheit, die mit $2f$ betrieben wird. Das führt zum Konzept der Superskalarität (siehe Kapitel 4) und zu Prozessoren mit zwei oder mehr Kernen (siehe Kapitel 9). In der Praxis bewahrheitet es sich zwar nicht, dass zwei mit Taktfrequenz f betriebene Einheiten das Gleiche leisten können wie eine mit $2f$ betriebene. Dennoch ist es unausbleiblich, immer ausgefeiltere Konzepte zur Parallelarbeit zu entwickeln.

1.2.1

Übung 1.2.1 Ein moderner Pentium-Prozessor (Pentium 4 mit 3 GHz Taktfrequenz) besitzt eine Chipfläche von etwa $1,5 \text{ cm}^2$ und hat eine Leistungsaufnahme von etwa 120 Watt. Vergleichen Sie die Wärmeentwicklung des Prozessors mit der einer Herdplatte, unter der Annahme, dass $2/3$ der vom Prozessor aufgenommenen Leistung als Verlustwärme über Kühleinrichtungen abzuführen sind.

1.3 Befehlssätze

Der Befehlssatz eines Prozessors ist sein markantestes von außen wahrnehmbares Merkmal. Alle Programme müssen auf die möglichen Befehle zurückgreifen. Zur Beschreibung eines Befehlssatzes ist es wichtig, die verfügbaren Register des Prozessors zu kennen sowie die einzelnen Befehle und wie sie die Registerinhalte manipulieren. Die Gesamtheit aus Befehlssatz und verfügbaren Registern nennt man *Programmiermodell*.

► 1.3.1 Grundlegende Eigenschaften von Befehlssätzen

Register sind die schnellsten speichernden Elemente eines Prozessors. Jedes Register kann eine bestimmte Anzahl an Bits speichern, im Extremfall nur ein einzelnes. An nahezu allen Operationen eines Prozessors ist wenigstens ein Register beteiligt. Frühere Prozessoren verfügten über sehr wenige Register. Ursprünglich wurden so genannte *Akkumulatormaschinen* gebaut. Diese haben für alle arithmetischen und logischen Operationen nur ein Register zur Verfügung, den *Akkumulator*. Dieses Register ist implizit immer ein an einer Operation beteiligter Operand sowie Ziel für das produzierte Ergebnis. Der zweite Operand ist entweder ein Direktoperand oder ein Speicheroperand (siehe Abbildung 1.4). Davon ausgehend hat sich eine Vielzahl von unterschiedlichen Strukturen entwickelt. Um Befehlssätze verstehen zu können, müssen wir uns mit folgenden Punkten beschäftigen:

- Über welche Register ein Prozessor verfügt und wie diese verwendet werden.
- Woher ein Befehl seine Operanden bezieht und wohin er das Ergebnis schreibt.
- Über welche Adressierungsarten ein Prozessor verfügt, d.h. woraus sich die Adressen für Speicherzugriffe ergeben und wie diese gebildet werden.
- Wie die einzelnen Befehle aussehen.

Diese Fragen werden in den folgenden Abschnitten behandelt.

► 1.3.2 Registersätze und Programmiermodell

Gegenüber den Akkulatormaschinen verfügen moderne Prozessoren über eine Vielzahl von Registern. Diese können meist auch wesentlich universeller verwendet werden als der Akkumulator.

Grundsätzlich sind allgemein verwendbare Register zu unterscheiden und solche, die einem bestimmten Zweck dienen oder bestimmten Aufgaben vorbehalten sind. Allgemein verwendbare Register heißen auch *General Purpose Register*, kurz GPR. Bei MMIX gehören dazu die Register \$0 bis \$255. Das Register \$255 wird dazu gezählt, obwohl es dazu dient, Parameter an Be-

triebssystemaufrufe zu übergeben. In der Praxis sind nicht immer alle Register, die als General Purpose Register bezeichnet werden, auch wirklich vollkommen universell verwendbar. Oft wird zwischen Registern für ganzzahlige Werte und solchen für Gleitkommawerte unterschieden, gelegentlich auch solche für Adressen. Dennoch zählen diese zu den GPR. Bei MMIX kann jedes Register sowohl für ganze Zahlen als auch für Gleitkommazahlen und für Adressen verwendet werden.

Die Menge aller möglichen Spezialregister aufzuzählen ist unmöglich und auch nicht sonderlich interessant. Es sollen hier Möglichkeiten genannt werden, die häufig vorkommen:

- Ein *Befehlszähler* (Program Counter, kurz PC) muss immer vorhanden sein. Er wird indirekt von jedem Befehl beeinflusst. Die Sprungbefehle überschreiben ihn explizit.
- Um Unterprogrammaufrufe realisieren zu können, muss immer ein Stack vorhanden sein. Häufig verfügen Prozessoren über ein oder mehrere spezielle Register, so genannte *Stackpointer*, die das aktuelle Ende des Stack anzeigen. Diese Register können oft indirekt über spezielle Push- und Pop-Operationen beeinflusst werden.
- In einem *Statusregister* gibt jedes Bit Auskunft über einen bestimmten Aspekt des aktuellen Prozessorzustands.
- Manche Prozessoren verfügen über spezielle *Indexregister* die ausschließlich für Adressrechnungen verwendet werden können.

1.3.1 Übung 1.3.1

1. Wie kann der Befehlszähler von MMIX ausgelesen werden?
2. Welche der obigen speziellen Registertypen kommen bei MMIX vor und welche nicht?

➤ 1.3.3 Unterscheidung von Befehlssätzen nach Registerbreite

Häufig findet man eine Unterscheidung von Prozessoren anhand der Breite ihrer Register vor: 32 und 64-Bit-Prozessoren (früher auch acht- und 16-Bit-Prozessoren). Allgemein besitzen n -Bit-Prozessoren Register mit je n -Bit Breite. Damit besitzen sie in der Regel auch ebenso breite Daten- und Adressbusse. Sie können damit 2^n Byte Speicher adressieren und transferieren in jedem Speicherzugriff maximal n Bits.

Der Übergang von 16 auf 32 Bit hat sich Anfang bis Mitte der 1980er Jahre vollzogen. Der Übergang von 32 zu 64 Bit läuft etwa seit Anfang der 1990er Jahre. Hauptnachteil der 32-Bit-Prozessoren ist der beschränkte Adressraum von vier Gigabyte adressierbarem Speicher.

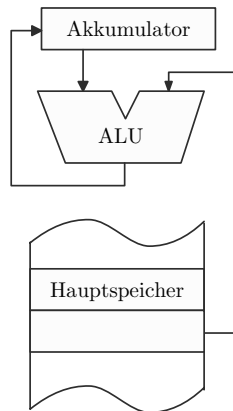


Abbildung 1.4. Akkumulatormaschine. Das Akkumulatorregister (kurz: Akkumulator) ist stets Quelle eines Operanden sowie Ziel für das Ergebnis einer Operation der ALU

Im PC-Bereich waren bislang 32-Bit-Prozessoren von Intel und AMD üblich. Für Server geht Intel neue Wege mit einem komplett neuen Befehlssatz der Itanium-Prozessoren. AMD baut in die Opteron-Prozessoren lediglich 64-Bit-Erweiterungen ein (AMD64). Um Kompatibilität mit der großen Menge vorhandener 32-Bit-Software zu bewahren, können die 64-Bit-Prozessoren den 32-Bit-Code ausführen (die Itanium-Prozessoren allerdings recht langsam, weil sie die 32-Bit-Befehle emulieren). Nach AMD hat auch Intel 64-Bit-Erweiterungen für seine 32-Bit-Prozessoren eingeführt (*Enhanced 64-Bit Memory Technology*, kurz EM64T). Als Oberbegriff hat sich mittlerweile die Bezeichnung x64 bzw. x86-64 eingebürgert [43].

➤ 1.3.4 Angabe von Operanden und Ergebnis

Als weiteres wichtiges Unterscheidungskriterium für Befehlssätze wird angegeben, woher die Befehle ihre Operanden beziehen und wohin sie ihr Ergebnis schreiben. Bei den ältesten Prozessoren, den Akkumulatormaschinen, muss in einem Befehl nur ein einziger Operand spezifiziert werden. Ein Befehl wie **ADD d**

würde etwa bewirken, dass der Inhalt der mit **d** bezeichneten Speicherstelle zum Wert des Akkumulatorregisters hinzu addiert wird. Diese Situation zeigt Abbildung 1.4. Beispielsweise handelt es sich bei den Prozessoren 6800 von Motorola und dem 6502 von MOS Technologies, der im berühmten Commodore C64 zum Einsatz gekommen ist, um Akkumulatormaschinen.

Die Akkumulatormaschinen wurden bald durch Prozessoren mit mehreren allgemein verwendbaren Registern abgelöst. Frühere Prozessoren verfügten allerdings noch über sehr wenige Register. Viele Befehle, die zwei Operan-

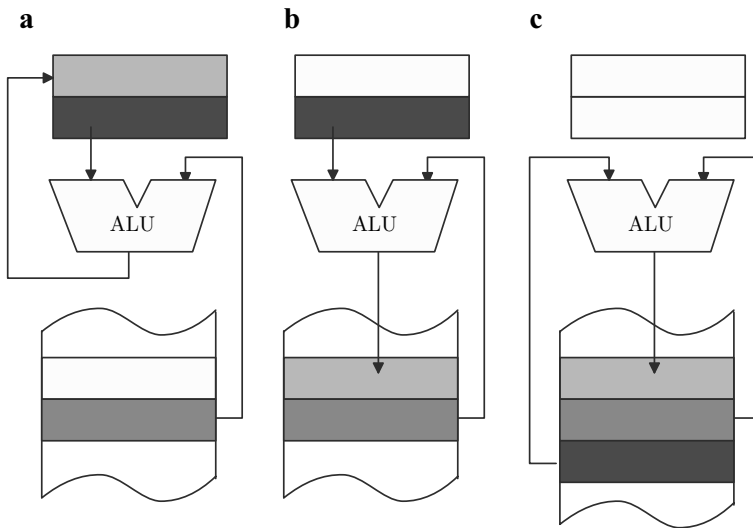


Abbildung 1.5. Register-Speicher-Befehle. **a** Je ein Operand kommt aus einem Register und dem Hauptspeicher, das Ergebnis wird in ein Register geschrieben. **b** Gegenüber Fall a wird das Ergebnis in den Hauptspeicher übertragen. **c** Beide Operanden sowie das Ergebnis liegen im Speicher. Nicht gezeigt ist die Variante, bei der beide Operanden aus Registern kommen, aber das Ergebnis in den Hauptspeicher geschrieben wird

den benötigen und ein Ergebnis erzeugen, müssen dann Operanden aus dem Hauptspeicher laden bzw. das Ergebnis in den Speicher schreiben können, oder beides. Solche Befehle heißen *Register-Speicher-Befehle* (auch *Register-Memory-Befehle*).

Diese Befehlsarten sind in Abbildung 1.5 veranschaulicht.

Wie wir später sehen werden, lassen sich Register-Memory-Befehle schlecht fließbandartig auf Pipelines ausführen. Da Speicherzugriffe immer auch eine Adressberechnung beinhalten, sind zu deren Ausführung implizit auch mehrere arithmetische Operationen erforderlich. In modernen Befehlssätzen werden solche Befehlsarten daher gar nicht mehr vorgesehen. Die Alternative besteht nämlich darin, getrennte Befehle für Speicherzugriffe und für arithmetische oder logische Operationen vorzusehen. Man unterscheidet *Register-Register-Befehle* und *Load-Store-Befehle*. Die Load-Store-Befehle übernehmen lediglich den reinen Transport von Daten zwischen Hauptspeicher und Registern. Operationen mit den Daten werden von Register-Register-Befehlen ausgeführt und zwar ausnahmslos zwischen Registern. Diese Befehlsarten sind veranschaulicht in Abbildung 1.6.

Die Entscheidung, auf komplizierte Register-Memory-Befehle zu verzichten, ist so gravierend und beeinflusst den Prozessorentwurf so stark, dass Prozes-

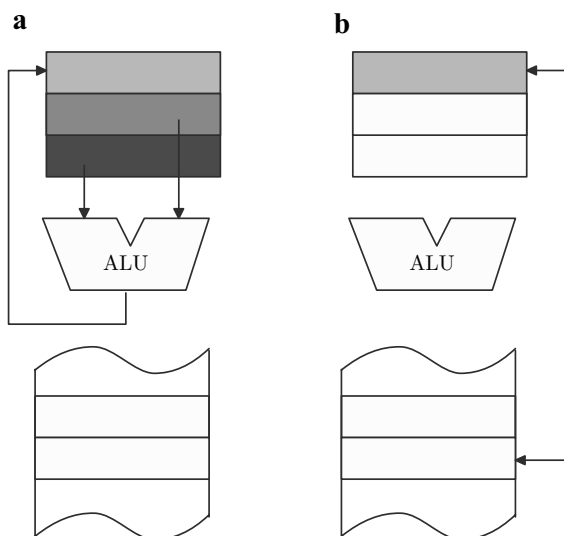


Abbildung 1.6. a: Register-Register-Befehl und b: Load- bzw. Store-Befehl

soren danach unterschieden werden, ob sie solche Befehle besitzen oder nicht. Prozessoren, die Register-Memory-Befehle ausführen können bezeichnet man als *Complex Instruction Set Computers*, kurz CISC. Im Gegensatz dazu können Prozessoren mit *Reduced Instruction Set Computers* nur Register-Register- bzw. Load-Store-Befehle ausführen.

Wir werden später noch weitere Unterscheidungsmerkmale zwischen RISC und CISC Maschinen kennen lernen. Die hier besprochenen Merkmale werden auch oft zur Unterscheidung verschiedener Architekturen herangezogen. Man unterscheidet die *Register-Memory-Architektur* und die *Load-Store-Architektur* (auch Register-Register-Architektur).

Die Beschränkung auf RISC-Befehle vereinfacht die Befehlsausführung so stark, dass keine neuen Befehlssätze mehr vorgeschlagen werden, die komplexe CISC-Befehle enthalten. Stark vertreten finden sich diese Befehle heute hauptsächlich noch bei den Prozessoren von Intel und AMD, die X86-Programme ausführen können. Dies ist aber eine rein wirtschaftliche Entscheidung und keine technische. Für diese Befehlssätze wurde mittlerweile so viel Anwendungssoftware geschrieben, dass ständig Prozessoren gefordert werden, die in der Lage sind, diese Software auch auszuführen.

Intel, AMD und Transmeta treiben erheblichen Aufwand, um die CISC-Befehle intern auf RISC-artige Befehle abzubilden, so genannte RISC86-Befehle. Diese werden dann auf einer Load-Store-Architektur ausgeführt. Darauf werden wir im Abschnitt 4.10.1 noch näher eingehen.

➤ 1.3.5 Adressierungsarten

Adressierungsarten sind in [4] sehr ausführlich und allgemein sowie in [1] MMIX-spezifisch beschrieben. Es gibt grundsätzlich drei Möglichkeiten, von wo Operanden kommen können oder wo ein Ergebnis abzulegen ist:

1. Aus dem Befehlswort. Man spricht von Konstantenadressierung oder Immediater Adressierung.
2. Aus einem Register. Das Register muss im Befehl direkt oder indirekt angegeben sein. Wir haben bereits gesehen, dass bei den Akkumulatormaschinen der Akkumulator immer implizit einen Quelloperanden liefert und Ziel einer Operation ist.
3. Aus dem Hauptspeicher. Die Speicheradresse muss spezifiziert werden. Die Speicheradresse, auf die letztlich zugegriffen wird, heißt *effektive Adresse*.

Da heute mit langen Adressen von 32 oder 64 Bit Länge gearbeitet wird, ist es unüblich, eine effektive Adresse direkt im Befehlswort anzugeben. Bei MMIX mit seinen 64-Bit-Adressen und 32-Bit-Befehlsworten ist es sogar unmöglich³. Die Adresse muss dann immer aus Werten in Registern gebildet werden. Tabelle 1.2 gibt einen Überblick über Adressierungsarten. Frühe Mikroprozessoren verfügten über viele Adressierungsarten, um die Operanden effizient aus dem Speicher zu beschaffen.

1.3.2 Übung 1.3.2 Wie lässt sich absolute Speicheradressierung und indiziert Speicher-relative Adressierung mit MMIX nachbilden?

➤ 1.3.6 Befehlsformate

Wir haben in den vorigen Abschnitten gesehen, dass in einem Befehl, der auf einer Akkumulatormaschine ausgeführt werden kann, nur ein Operand angegeben wird. Der zweite Operand ist stets implizit das Akkumulatorregister. Ebenso wird das Ergebnis eines Befehls stets in das Akkumulatorregister geschrieben. In diesem Fall spricht man von *Ein-Adress-Befehlsformat*. Dies ist die älteste Form, Befehle zu spezifizieren.

³Die so genannte Zero-Page-Adressierung wird heute praktisch nicht mehr verwendet und hier nicht weiter betrachtet. Dabei ließen sich effektive Adressen mit einer maximalen Länge von 8 oder 16 Bit im Befehlswort direkt angeben.

Tabelle 1.2. Verschiedene Adressierungsarten, die sich in Prozessoren finden. Bei **MMIX** sind nicht alle davon verfügbar

Bezeichnung	MMIX	sonstige Schreibweise	Beschreibung
Konstantenadressierung	ADD \$1,\$1,10	ADD R1,10	Direkto­perand im Befehlswort
Registerdirekte Adressierung	ADD \$0,\$1,\$2	ADD R1,R2	Verwendet Register, die explizit im Befehlswort angegeben sind
Registerindirekte Adressierung	≈ LDO \$0,\$255,0	MOV reg1,[reg2]	Effektive Adresse befindet sich in einem Register
Absolute Speicheradressierung	—	MOVE reg1,Mem	Effektive Speicheradresse ist Teil des Befehlswortes
Indiziert Speicher-relative Adressierung	—	MOVE reg1,[reg2+mem]	Effektive Adresse ist Summe aus Registerwert + aus Speicher gele­se­ner Wert
Indiziert Register-relative Adressierung	LDO \$0,\$255,\$1	MOVE R1,[R2,R3]	Summe zweier Registerinhalte lie­fert die Effektive Adresse
Indiziert Register-relative Adressierung mit Index	LDO \$0,\$255,8	MOVE R1,[R2,0ff]	Effektive Adresse=Registerinhalt + Konstante
Programmzähler relative Adressierung	JMP @+12	MOVE R1,[PC,offset]	Effektive Adresse Effektive Adres­se=PC+Offset

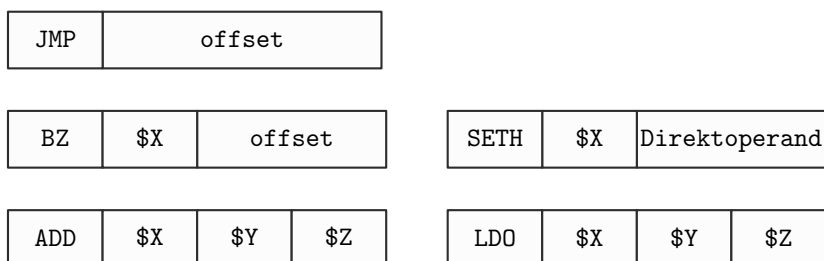


Abbildung 1.7. Beispiele für die Codierung von MMIX-Befehlen. Es gibt Befehle mit einem, zwei und drei Operanden. Allen gemeinsam ist das feste Befehlsformat mit vier Bytes, davon ein Byte für den Befehlscode.

Mit Abkehr von den recht einfachen Akkumulatormaschinen hat sich zunächst das *Zwei-Adress-Befehlsformat* eingebürgert. Dabei werden im Befehl beide Operanden angegeben, aber einer der Operanden wird mit dem Ergebnis überschrieben.

ADD A, B

bedeutet $A \leftarrow A + B$. Dieses Befehlsformat verwendet der X86-Befehlssatz hauptsächlich.

Bei Maschinen mit vielen allgemein verwendbaren Registern ist es nicht mehr sinnvoll, Operanden implizit vorzugeben. Moderne Befehlssätze – wie auch der Befehlssatz von MMIX – verwenden das so genannte *Drei-Adress-Befehlsformat*. Dort können separate Register für beide Operanden sowie für das Ergebnis spezifiziert werden.

Die Befehle und ihre Operanden müssen in maschinenlesbarer Form codiert werden. MMIX hat ein festes Befehlsformat mit vier Bytes je Befehl. Dabei ist das erste Byte der Befehlscode (Opcode) und die folgenden drei Byte spezifizieren die Operanden. Es gibt Befehle mit einem, zwei oder drei Operanden. Siehe Abbildung 1.7.

Dieses feste Befehlsformat hat den Vorteil, dass die Befehle sehr leicht zu verarbeiten sind. Alle Befehlscodes sind gleich lang und Befehle beginnen im Speicher immer an durch vier teilbaren Adressen. Ferner sind die geladenen Befehle leicht decodierbar, da die Operanden immer an der gleichen Position im Befehlswort angegeben sind. Moderne Befehlssätze werden daher mit einem solchen festen Format codiert.

Im Gegensatz dazu gibt es Befehlssätze mit variablem Befehlsformat. Das prominenteste Beispiel ist wieder der X86-Befehlssatz. Die kürzesten Befehle sind nur ein Byte lang und die längsten können über 16 Bytes lang sein. Der Befehlscode kann von acht bis 16 Bit Länge variieren. Konkret bedeutet das: Falls im Opcode zuerst der hexadezimale Wert 0x0F steht, so folgt ein

weiteres Byte, das den Opcode spezifiziert. Andernfalls folgt der erste Operand. Diese komplizierten Regeln muss der Decoder berücksichtigen. Beim Laden der Befehle aus dem Speicher und beim anschließenden Decodieren müssen auch die Grenzen zwischen den Befehlen festgestellt werden. Dazu ist es erforderlich, den Befehlscode zu kennen. Da dessen Länge variabel ist, ist bereits dieser Schritt aufwändig.

Diese Komplexität des Befehlsformats ist ein weiteres Argument, das gegen den Einsatz von derartigen CISC-Befehlssätzen spricht.

Übung 1.3.3 Fassen Sie kurz die Unterschiede zwischen RISC- und CISC-Befehlssätzen zusammen.

1.3.3

➤ 1.3.7 Befehlstypen

An dieser Stelle soll keine Auflistung aller möglichen Befehle in allen bislang implementierten Befehlssätzen erfolgen. Hier gehen wir lediglich auf Besonderheiten und wichtige Spezialbefehle ein. Nicht separat abhandeln wollen wir insbesondere folgende – als bekannt vorausgesetzten – Arten von Befehlen:

- Arithmetische und logische Befehle für ganze Zahlen (bei MMIX gehören dazu Befehle wie `ADD`, `SUB`, `AND` etc.).
- Gleitkommabefehle für Arithmetik mit Gleitkommazahlen (`FADD`, `FMUL` etc.).
- Befehle für den Speicherzugriff (dies sind etwa `LDO`, `LDB` oder `STO` und `STB` bei MMIX).
- Bedingte und unbedingte Sprungbefehle (dazu gehören `JMP`, `BZ` oder `PBNZ` bei MMIX, ebenso wie dessen Befehle zum Aufruf von Unterprogrammen: `PUSHJ` und `POP`).

MMIX dient hier lediglich als Beispiel und die genannten Befehle stehen stellvertretend auch für die Befehle anderer Prozessoren.

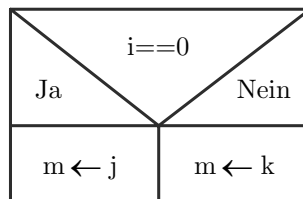
Im Folgenden listen wir einige Klassen von Befehlen auf, die in Grundlagenvorlesungen nicht unbedingt besprochen werden, denen aber dennoch eine große Bedeutung zukommt.

➤ Bedingte Befehle

In modernen Befehlssätzen finden sich zunehmend so genannte *Bedingte Befehle*. Darunter versteht man Befehle, die in Abhängigkeit von einer Bedingung ausgeführt oder nicht ausgeführt werden. Als Befehle für bedingte Sprünge (`BZ`, `BEV` etc.) sind solche Befehle selbstverständlich in jedem Befehlssatz vorhanden. Hier sind Befehle in der Art gemeint, wie: „addiere, falls

die Bedingung `xy` erfüllt ist“ Ziel der Bedingten Befehle ist stets, bedingte Sprünge zu vermeiden.

MMIX bietet die Bedingten Befehle Conditional Set `CSxx` und Zero or Set `ZSxx`, wobei `xx` für eine der von den bedingten Sprüngen her bekannten Bedingungen steht. Der Befehl `CSxx $X,$Y,$Z` setzt `$X` zu `$Z`, falls `$Y` die angegebene Bedingung erfüllt. Ansonsten lässt er `$X` unverändert. Der Befehl `ZSxx $X,$Y,$Z` setzt demgegenüber das Register `$X` andernfalls zu null. In vielen Fällen lassen sich dadurch Verzweigungen vermeiden. Wir betrachten beispielsweise folgendes Struktogramm:



Das effizienteste MMIX-Programm dazu lautet:

```

SET    m,j
CSNZ   m,i,k
  
```

In diesem Zweizeiler wird `m` mit dem Wert von `j` vorbelegt, so als würde der linke Zweig ausgeführt. Nur, falls diese Belegung falsch ist, wird `m` mit `k` überschrieben. Es ist hier also keine bedingte Verzweigung erforderlich. Ohne Unterbrechung des Programmflusses laufen immer zwei Befehle durch die Pipeline. Einer davon wird unnötigerweise ausgeführt, aber das lässt sich nicht vermeiden.

Das IA-64-Programmiermodell kennt 64 Ein-Bit-Register `p0...p63`, die so genannten *Predication Register*. Alle Vergleichsbefehle schreiben das Ergebnis des Vergleichs in eines dieser Register. Die Ausführung jedes Befehls kann von jedem Predication Register abhängig gemacht werden. Beispiel:

```

cmp.eq p6,p0=r33,r32
(p6) add    r34,r35
  
```

Hier vergleicht der erste Befehl die Register `r32` und `r33` miteinander. Das Ergebnis des Vergleichs wird in Predication Register `p6` abgelegt und das Komplement in `p0`⁴. Die folgende Addition wird nur ausgeführt, wenn `p6` den Wert eins enthält.

Wir werden in Abschnitt 3.2.3 sehen, wie bedingte Sprungbefehle die Befehlsausführung beschleunigen können. Optimierende Compiler verwenden Bedingte Befehle, wo immer möglich.

⁴Das Predication Register `p0` ist immer null und kann nicht überschrieben werden. Es müssen bei dem Vergleichsbefehl aber zwei Register für den Wert und sein Komplement angegeben werden.

➤ Nicht-unterbrechbare Befehle

Multitasking-Betriebssysteme müssen die Möglichkeit bieten, einzelne Prozesse zu synchronisieren, wenn diese auf gemeinsame Ressourcen zugreifen. Im einfachsten Fall kommunizieren zwei Prozesse über eine Variable im Speicher, zu der beide Prozesse Zugriff haben. Wir betrachten als Beispiel den Kontostand eines Kunden, der von der Software einer Bank verwaltet wird [1]. Er kann einerseits durch Auszahlungen am Bankautomaten und andererseits durch automatische Zinsgutschriften verändert werden. Bei einer Zinsgutschrift wird der Kontostand aus dem Speicher in ein Register geladen und die Zinsen werden berechnet und hinzuaddiert. Anschließend wird das Ergebnis zurück in den Speicher geschrieben. Findet dieser Vorgang parallel mit dem Abheben am Bankautomaten statt, so kann Folgendes geschehen: Der Prozess zur Zinsgutschrift wird vom Betriebssystem nach dem Holen des Kontostandes unterbrochen; der Bankautomat liest den Kontostand ebenfalls in ein Register, subtrahiert den abgehobenen Betrag und schreibt das Ergebnis zurück in den Speicher, noch bevor der andere Prozess weiter läuft und die Zinsberechnung fertig stellt. Als Nächstes läuft der erste Prozess weiter, schreibt den alten Kontostand plus Zinsen zurück in den Speicher und ersetzt dort das Ergebnis der Subtraktion.

Der erste Lösungsansatz besteht darin, eine weitere Variable zu benutzen, um den Zugriff auf den Kontostand abzusichern. Ein Prozess kann diese Variable lesen. Ist sie 0, so hat bereits ein anderer Prozess Zugriff auf den Kontostand, und der Prozess muss warten. Ist sie hingegen 1, so ist der Kontostand verfügbar. Man setzt sofort die Kontrollvariable auf 0, arbeitet mit dem Kontostand und setzt dann die Kontrollvariable wieder auf 1.

Aber diese Vorgehensweise löst das Problem nicht! Es wurde lediglich das Problem des unsynchronisierten Zugriffs auf den Kontostand verlagert: Nun muss der Zugriff auf die Kontrollvariable synchronisiert werden. Dazu könnte man eine weitere Kontrollvariable verwenden usw.

Der Kern des Problems besteht in der Möglichkeit einer Unterbrechung (Kontextwechsel) eines Prozesses durch das Betriebssystem nach dem Lesen der Kontrollvariablen und vor dem Zurückschreiben des aktualisierten Wertes. Um das Problem zu lösen, haben alle modernen Prozessoren mindestens einen Befehl, der das Lesen, Ändern und Schreiben erlaubt und nicht unterbrochen werden kann. Man spricht oft auch von *atomaren Befehlen* oder von *Test-And-Set-Befehlen* auch von *Read-Modify-Write-Befehlen*.

Beim MMIX ist ein solcher atomarer Befehl der CSWAP-Befehl (Compare and Swap), der in einer einzigen, unteilbaren und ununterbrechbaren Instruktion einen Test, einen Lesezugriff und einen Schreibzugriff implementiert. Mit dieser Instruktion kann in obigem Beispiel die Kontrollvariable für den Kon-

tostand auf den Wert 0 getestet werden und im Fall, dass sie diesen Wert hat, sofort auf 1 gesetzt werden.

Folgendes MMIX-Programm veranschaulicht den Zugriff auf eine Kontrollvariable, den so genannten *Semaphor*:

semaphor.mms		
1	LOC	Data_Segment
2 SEMA	OCTA	1
3		
4 semReg	IS	\$1
5	LOC	#100
6 Main	PUT	rP,0
7	SET	semReg,1
8	CSWAP	semReg,SEMA
9	BZ	semReg,wait

In diesem Beispiel sei vereinbart, dass der kritische Abschnitt frei ist, wenn die Semaphorvariable den Wert 1 hat. Der Befehl *CSWAP* testet dies und setzt die Semaphorvariable ggf. auf den Wert 0. Liefert der Befehl 0 zurück, so ist der kritische Abschnitt nicht frei und das Programm muss warten, bis er frei wird.

Solche Synchronisationsprobleme treten häufig auf. Java verfügt über ein eigenes Synchronisationskonzept [27]. Die Steuerung erfolgt über das Schlüsselwort *synchronized*.

➤ Vektorbefehle

Mit Intels so genannten *Multimedia Extensions* (kurz MMX – nicht zu verwechseln mit MMIX) hat 1996 eine Art von Befehlen in die Mikroprozessoren Einzug gehalten, die es vorher nur auf Großrechnern gab, nämlich die Vektorbefehle. Dies sind Befehle, mit denen eine Operation auf mehrere Operanden bzw. Paare von Operanden angewandt werden kann. Die MMX-Befehle betrachten einen 64-Bit-Wert als Vektor bestehend aus

- acht unabhängigen Bytes (Packed Bytes) oder
- vier unabhängigen zwei-Byte-Werten (Wydes; Packed Words) oder
- zwei unabhängigen vier-Byte-Werten (Tetras; Packed Double-Words) oder
- einem acht-Byte-Wert (Octa; Quad-Word)

Arithmetische Operationen wirken auf jedes Vektorelement unabhängig von den anderen Elementen. Bei einer Vektor-Addition tritt also kein Überlauf (Carry) von einem Element in das nächste auf, wie es der Fall bei einer Addition von 64-Bit-Werten wäre.

Solche Befehle hat AMD unter dem Namen 3DNow! in seine Prozessoren eingebaut. Bei MMIX gibt es solche Befehle ausschließlich für die saturierte Addition der genannten Vektoren: BDIF, WDIF, TDIF und ODIF. Dies ist zusammen mit Beispielen beschrieben in [1]. Solche Befehle sind insbesondere bei der Bildbearbeitung und Bildverarbeitung von Bedeutung, wo Bildpunkte durch einzelne Grauwerte oder durch Farbwerte nach einem Farbmodell (RGB bzw. CMYK) mit vier unabhängigen Bytes je Wert repräsentiert werden.

Vektorbefehle für Vektoren von Gleitkommazahlen werden in Mikroprozessoren seit 1999 eingebaut. Intel hat damit begonnen, seine Prozessoren mit den so genannten *Internet Streaming SIMD Extensions* (ISSE) auszustatten⁵. Bei den PowerPC-Prozessoren werden Vektorbefehle unter dem Namen *AltiVec* eingesetzt. In Anwendungen, bei denen komplexe Zahlen oder Raumkoordinaten eine Rolle spielen, können diese auf solche Vektoren abgebildet werden. Komplizierte Operationen auf diesen Vektoren können dann einfach programmiert und sehr schnell ausgeführt werden. Bei MMIX finden wir solche Befehle nicht vor.

⊗ Befehle für digitale Signalverarbeitung

Für Aufgaben der digitalen Signalverarbeitung werden oft spezielle Prozessoren eingesetzt, so genannte Digitale Signalprozessoren, kurz DSP.

Signalprozessoren verfügen meistens über spezielle komplizierte Hardware, die eine Multiplikation in nur einem Taktzyklus ausführen kann. Häufig kann diese Multiplikation auch mit einer Addition verbunden werden. Man spricht von Multiply-and-Accumulate. Damit lassen sich beispielsweise Skalarprodukte schnell berechnen, die in Algorithmen zur digitalen Signalverarbeitung eine große Rolle spielen.

Übung 1.3.4 Auf welche der hier genannten Befehle kann nicht verzichtet werden, obwohl sie dem RISC-Prinzip widersprechen?

1.3.4

➤ 1.3.8 Parallelverarbeitung auf Ebene des Befehlssatzes

Ein Ansatz, Befehlssätze zu implementieren, besteht darin, mehrere Befehle in ein Befehlswort zu kodieren. Diese Befehle können vom Prozessor parallel ausgeführt werden. Ein Programmierer oder Compiler gibt also explizit vor, welche Befehle parallel ausgeführt werden können. Da die entstehen-

⁵Die Abkürzung SIMD steht dabei für *Single Instruction Multiple Data*, also Befehle, die (als einzelner Befehl) mehrere Daten/Operanden bearbeiten können. Gewöhnliche Befehle wären demgegenüber vom Typ SISD – *Single Instruction Single Data*.

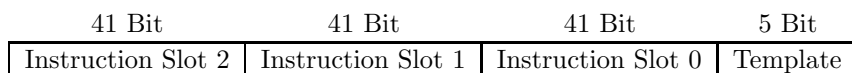


Abbildung 1.8. Das EPIC-Befehlsformat der Intel 64-Bit-Architektur (IA-64). Die drei enthaltenen RISC-Befehle sind in so genannten Instruction Slots untergebracht; zusammen bilden sie ein Bündel

den Befehlswörter dadurch sehr lang werden können, wird diese Art von Befehlssätzen als *Very Long Instruction Word* bezeichnet, kurz VLIW.

Intel und HP haben sich 1994 zusammengeschlossen, um gemeinsam eine Architektur für 64-Bit-Prozessoren zu entwerfen. Mit dem aus der Kooperation hervorgegangenen Befehlssatz verfolgen sie den VLIW-Ansatz. Weil dem Prozessor explizit vorgegeben wird, welche Befehle er parallel ausführen soll, hat man diesen Befehlssatz *Explicit Parallel Instruction Computing*, kurz EPIC, genannt.

Der von Intel und HP erstmals in den Itanium-Prozessoren realisierte Befehlssatz heißt IA-64 (Intel Architecture, 64 Bit). Jedes Befehlswort ist 128 Bit lang und besteht aus drei RISC-Befehlen zu je 41 Bit sowie einem Template, das Steuerungsinformationen enthält. Dieses Format ist in Abbildung 1.8 dargestellt. Das Template gibt darüber Auskunft, welche der Befehle tatsächlich parallel ausgeführt werden können. Ein Befehlswort wird auch als Bündel bezeichnet, die Plätze, an denen Befehle stehen können, als *Slots*.

Es gibt dabei einige Einschränkungen zu beachten:

- Bedingte Verzweigungen können immer nur den ersten Befehl eines Bündels als Ziel haben.
- Manche Befehle können nur an bestimmten Positionen innerhalb eines Bündels stehen.
- Ein Bündel kann nur maximal einen Gleitkommabefehl enthalten.
- In einem 128-Bit langen Wort werden drei Befehle untergebracht. Klassische RISC-Prozessoren bringen dort vier Befehle unter.

Nicht ausgenutzte Slots müssen mit No-Operation aufgefüllt werden. Die Größe des ausführbaren Programms wächst für diesen Befehlssatz gegenüber anderen Befehlssätzen an.

Das Template gibt an, auf welcher Ausführungseinheit ein Befehl ausgeführt werden muss. Ferner schreibt das Template vor, ob ein Befehl vor einem anderen beendet sein muss.

➤ 1.3.9 Exkurs: Stackmaschinen

Wir haben in Abschnitt 1.3.6 Ein-, Zwei- und Drei-Adress-Befehlsformate besprochen. Als etwas exotische Variante gibt es daneben auch das *Null-Adress-*

Befehlsformat. Maschinen, die damit arbeiten, heißen *Stack-Maschinen* (gelegentlich auch *Kellermaschinen*). Zentraler Bestandteil solcher Maschinen ist ein Stack, auf dem die Operanden bereit gestellt werden und auf den nach der Operation das Ergebnis abgelegt wird. Weil dadurch beide Operanden und das Ergebnis implizit angegeben sind, kommt das Null-Adress-Befehlsformat zustande. Ein Befehl

ADD

addiert beispielsweise die obersten Einträge des Stacks und legt die berechnete Summe auf den Stack zurück. Daneben sind Transportbefehle erforderlich, die Operanden (z.B. Variablenwerte) aus dem Hauptspeicher auf den Stack transportieren und umgekehrt – so genannte Push- und Pop-Operationen. Programme für Stack-Maschinen lassen sich leicht automatisch erzeugen. Compiler erzeugen bei der Übersetzung von Ausdrücken so genannte *Abstract Syntax Trees*, aus denen sich der Code durch Traversieren ergibt. Abbildung 1.9 zeigt den Abstract Syntax Tree für den einfachen Ausdruck $D = C \cdot (A - B)$. Der Code, der sich daraus für eine Stack-Maschine ergibt, ist ebenfalls mit angegeben. In Abbildung 1.10 ist veranschaulicht, wie sich der Stack bei der Ausführung des Programmstücks entwickelt. Ausgehend von einem leeren Stack werden zunächst die Operanden abgelegt. Der linke Operand wird dabei stets zuerst auf den Stack gelegt. Die arithmetischen Operationen holen danach die Operanden vom Stack und legen ihre Ergebnisse auf dem Stack ab. Nach dem Wegnehmen des letzten Ergebnisses ist der Stack wieder leer.

Die Schreibweise von Ausdrücken, bei der die Operation nach den Operanden angegeben wird, wird auch als *Postfix-Notation* bzw. *Umgekehrt Polnische Notation* bezeichnet.

Auch bedingte Verzweigungen lassen sich mit Stacks leicht realisieren. Ein entsprechender Befehl nimmt das oberste Element vom Stack und prüft es gegen eine Bedingung. Als Argument braucht dieser Befehl eine Zieladresse, die angesprungen wird, falls die Bedingung erfüllt ist. Der Befehl

IFZERO ziel

würde das Programm an der mit **ziel** bezeichneten Marke fortsetzen, falls das oberste Element auf dem Stack 0 ist.

Diese Maschinen haben durchaus praktische Bedeutung in einigen Nischen. Beispielsweise basiert die Java Virtual Machine (JVM) auf diesem Prinzip. Es gibt Ansätze, reale Prozessoren zu bauen, die in der Lage sind, Java-Bytecode auszuführen, z.B. der Prozessor aj-100 von Ajile Systems (<http://www.ajile.com/>).

Ferner baut PostScript, eine Programmiersprache mit Möglichkeiten zur Grafikausgabe, auf das Prinzip der Stack-Maschinen. Intel organisiert die Ausführung von Gleitkommabefehlen als Stack-Maschine. Das heißt, die Gleit-

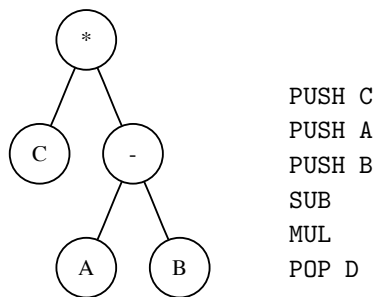


Abbildung 1.9. Zum Ausdruck $D = C \cdot (A - B)$ gehöriger Abstract Syntax Tree und erzeugter Code für eine Stack-Maschine. A, B, C und D bezeichnen dabei Speicherplätze für Variablen.

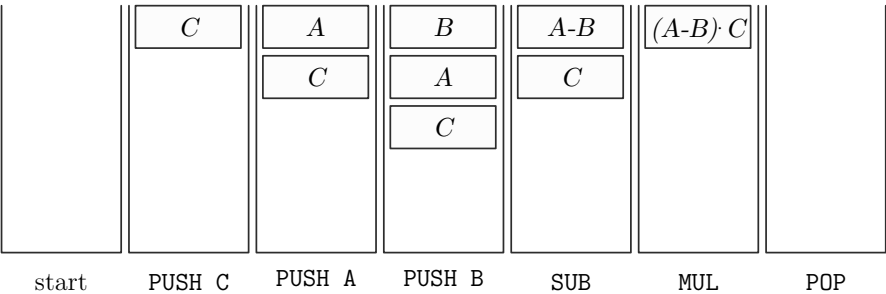


Abbildung 1.10. Entwicklung des Stacks mit dem Programm aus Abbildung 1.9. Anfangs ist kein Element auf dem Stack (links). Dann werden die Operanden auf den Stack gelegt und anschließend die arithmetischen Operationen ausgeführt.

kommaregister der x86-Prozessoren bilden einen Stack. Manche (nicht alle) Gleitkommabefehle sind Befehle mit Null-Adress-Befehlsformat.

1.3.5 Übung 1.3.5 Schreiben Sie ein Programm zur Auswertung des Ausdrucks $D = (A - B) \cdot C$. Was ändert sich dabei für den Stack gegenüber dem Ausdruck $D = C \cdot (A - B)$?



<http://www.springer.com/978-3-540-20979-9>

Rechneraufbau und Rechnerarchitektur

Böttcher, A.

2006, XI, 210 S., Softcover

ISBN: 978-3-540-20979-9