Kapitel 11

RISC-Rechner (reduced instruction set computer, RISC)

11.1. Einleitung

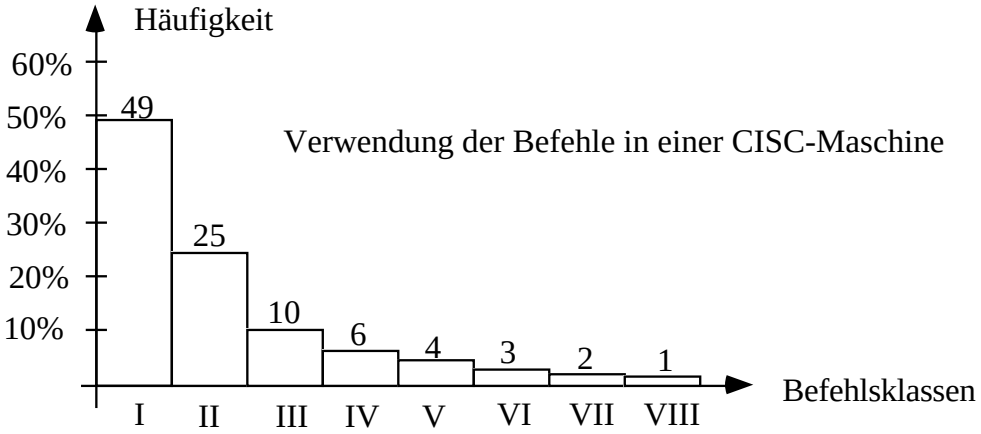
In den Achtzigerjahren änderten sich die Randbedingungen für Rechner: Hardware wurde als VLSI-Chip realisiert mit bald 10^6 Transistorfunktionen auf einem Chip. Das erlaubte die Realisierung komplexer Schaltnetze auf dem Chip als Bestandteile des Rechenwerks und ließ 32-Bit-Register zu. Speicherchips von 512 k Bit, d. h. 64 k Byte kamen auf den Markt, so daß sehr viel größere Hauptspeicher möglich wurden.

Durch die Entspannung in der Verfügbarkeit von Speicher wurde Assemblerprogrammierung völlig zurückgedrückt im Vergleich zur Programmierung in höheren Programmiersprachen, die über Compiler Code erzeugen. Dabei machte man die Entdeckung, daß die Compilerbauer und mehr noch der von Compilergeneratoren seinerseits erzeugte Code von den reichhaltigen Befehlssätzen der CISC-Maschinen nur wenig Gebrauch machten, sondern sich auf wenige Befehle konzentrierten.

Teilt man die Maschinenbefehle eines CISC-Rechners in 8 Klassen ein

I	Datentransportbefehle
II	Sprungbefehle und Unterprogrammsprünge
III	Arithmetische Befehle
IV	Vergleichsbefehle
V	Logische Befehle
VI	Schiebebefehle
VII	Bit-Manipulationsbefehle
VIII	Sonstige Befehle

dann entspricht diese Einteilung der beobachteten Häufigkeit in der Verwendung in Compiler-erzeugtem Code.



Weitaus am häufigsten sind Datentransportbefehle. Ferner treten Inkrement/Dekrementbefehle sehr häufig auf.

11.2. Beschleunigung der Maschinen

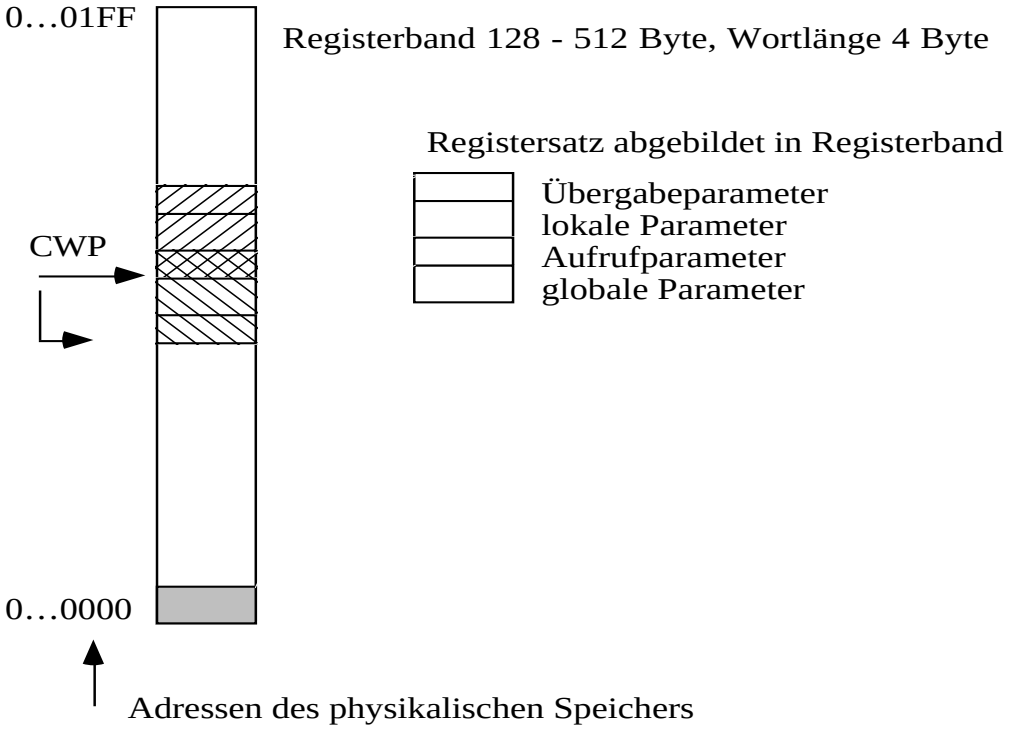
Die Beschleunigung der Maschinen sollte dann erreichbar sein durch:

a) Beschleunigung des Datentransports möglichst durch Transport auf dem Chip zwischen Registern und nicht im Speicher.
   => - Load-Store Architektur
        Rs <- M (ADR) oder M (ADR) <- Rs .
      - 3-Adress-Befehle
        Rs <- (Rx op Ry) oder Rs <- (Rx op const.)
      - Registerfile
        32 und mehr Register von 32 Bit Länge in einem File, um die laufenden Daten des Programms in Registern zu halten.

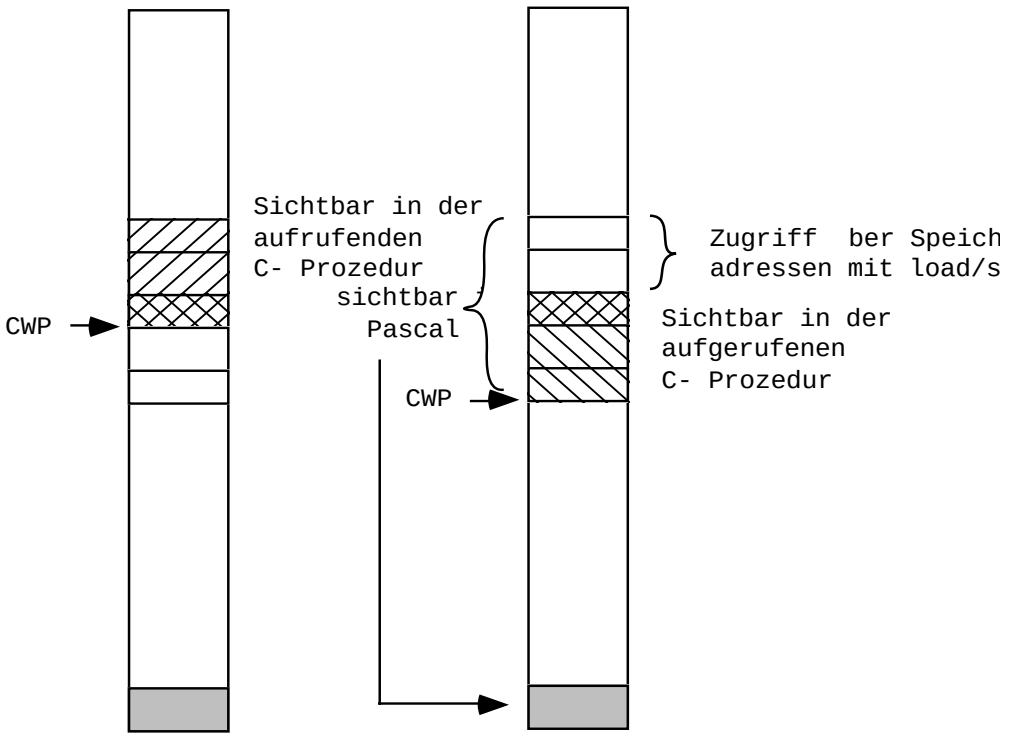
b) Um den Datentransport weiter zu beschleunigen, sollte nicht byteweise auf den Speicher zugegriffen werden:
   - Zugriff wortweise mit 32 Bit auf Daten
   - Befehle von Wortlänge: pro Speicherzugriff ein Befehl

c) Die Abarbeitung arithmetischer Befehle sollte unterstützt werden durch Schaltnetze im Rechenwerk (32-Bit-Addierer und -Multiplizierer, 32-Bit Barrelshifter).

d) Die Unterstützung der Prozeduraufrufe höherer Sprachen kann durch ein Registerband geschehen:
   Statt des im Programm zugreifbaren Satzes von 32 Registern gibt es ein Registerband von z. B. 128 Registern. Innerhalb einer Prozedur ist nur ein Fenster von 32 Registern sichtbar, gekennzeichnet durch einen "current window pointer", CWP.



Der im Programm ansprechbare Registersatz besteht aus 4 Teilen: einem Satz globaler Register abgebildet auf die gleiche Stelle im Registerband für alle Prozeduren, den Übergabeparameter für einen Prozeduraufruf, den lokalen Variablen der Prozedur und den Aufrufparametern der nächsten Prozedur.



Bei einem Prozeduraufruf wird dann der CWP nur weitergeschaltet, so daß sich der Bereich der Übergabe- und Aufrufparameter der beiden Prozeduren überlappen. Das erspart einen Stack für den Prozeduraufruf (bis auf das PSW, das nach wie vor dort gerettet werden muß) und beschleunigt die Parameterübergabe.

Dieses Verfahren ist geeignet für Sprachen wie "C". In Sprachen vom Typ "Pascal" sind in der aufgerufenen Prozedur noch die Variablen der aufrufenden Prozedur sichtbar, die hier in Registern des Registerbandes stehen, die nicht mehr unmittelbar an sprechbar sind.
Man hilft sich durch Einblenden des Registerbandes in den untersten Bereich des Adressraumes: Jedes Register kann dann auch durch Load-Store-Befehle erreicht werden.

Hardware muß den Überlauf des CWP überwachen: Wenn das Registerband bei einem Prozeduraufruf droht verlassen zu werden, muß ein interner Trap ausgelöst und durch eine Folge von Load-Store-Befehlen das Registerband in einen Stack im Speicher gerettet werden. Ein Registerband ist ungünstig bei Sprachen, die tiefe Rekursionen favorisieren wie Lisp oder Smalltalk.

e) Reduzierung der Zahl der Befehle im Vergleich zu CISC-Rechnern, dafür Abarbeitung der Befehle in der gleichen Anzahl von Takten. Das favorisiert Befehlspipelining, da das Rechenwerk für alle Befehle die gleiche Zeit braucht.

   Da Compiler eh nur einen reduzierten Befehlssatz verwenden, stört die Beschränkung nicht.

11.3. Kennzeichen der RISC-Architektur

Der Name RISC leitet sich her von dem reduzierten Befehlssatz: reduced instruction set computer, RISC.

Kennzeichen der Architektur sind

- Befehle gleicher Länge (meist 32 Bit)
- Abarbeiten mit gleicher Taktzahl: erlaubt Befehlspipelines
- Eingeschränkter Befehlssatz (32 - 128 Befehle)
- Explizite Lade/Speicher-Befehle
- Registerband mit Registerfenstern
- 3-Adress-Befehle .

Die in vorherigen Kapiteln vorgestellte DLX-Maschine realisiert eine RISC-Architektur.

11.4. Beispiel SUN-SPARC-Processor

Der SPARC-Rechner von Fa. Sun Microsystems war 1988 ein erster RISC-Rechner, der das Konzept auf dem Markt einführte und durchsetzte.

Er verwendet eine 4-stufige Befehlspipe

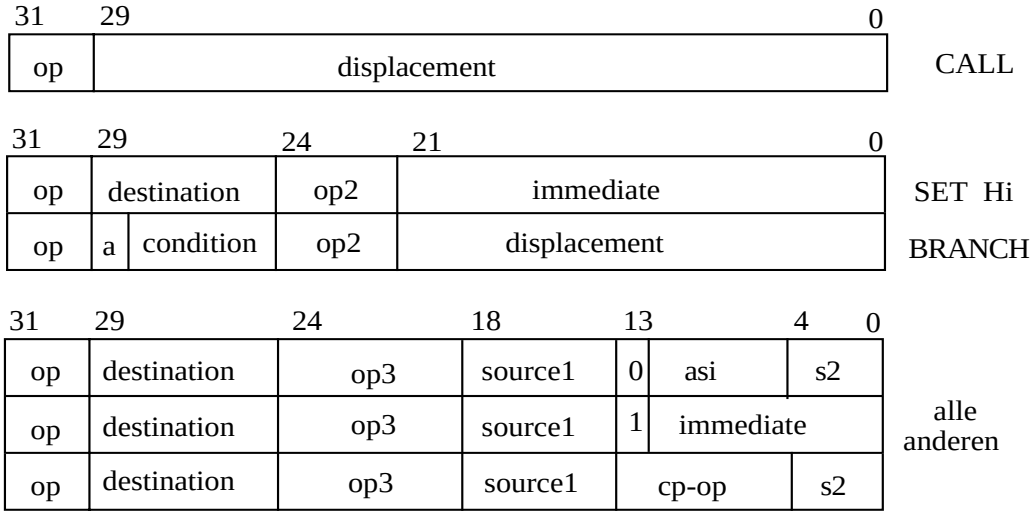
- H: Befehl holen
- D: Befehl dekodieren
- A: Befehl ausführen
- S: Resultat abspeichern .

Bei Sprüngen ist die Sprungadresse bekannt am Ende der D-Phase, der Wahrheitswert der Sprungbedingung am Ende der A-Phase.

=> Nach einem Sprungbefehl wird in die Pipeline ein "delay slot" vor dem Holen des nächsten Befehls eingefügt.

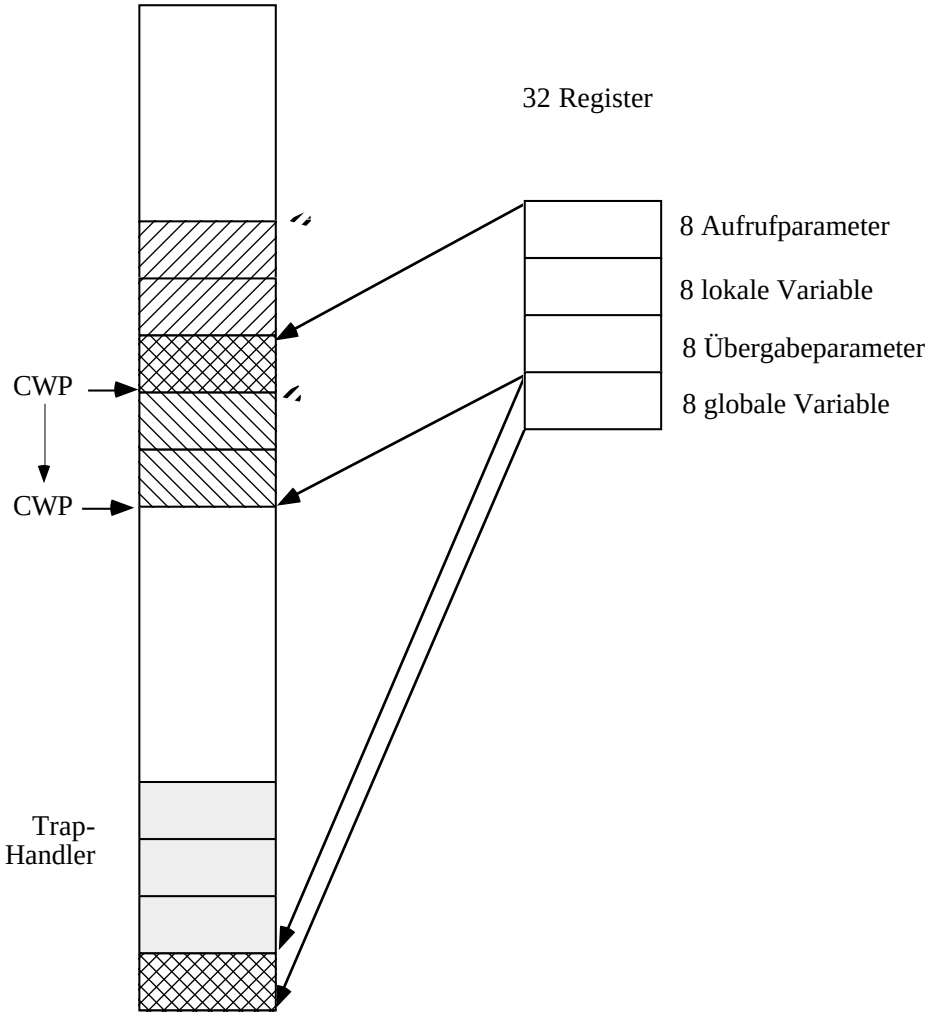
Die Befehlsformate des SPARC-Rechners zeigt das nächste Bild:

SPARC Befehlsformate



op	ist der Operationscode von 2 Bits, der den Befehlstyp festlegt.
op2	ein erweiterter Operationscode bei SET- und BRANCH-Befehlen
op3	ein erweiterter Operationscode bei allen anderen Befehlen
s2	kennzeichnet das zweite Quellregister bei 3-Adress-Befehlen
asi	kennzeichnet den Adressraum
cp-op	ist der Operationscode für einen arithmetischen Coprozessor (G-K-Befehle)
a	legt bei Sprungbefehlen fest, ob auf die erfüllte / nicht erfüllte Bedingung hin gesprungen werden soll

In den Befehlen wird ein Registersatz von 32 Registern angesprochen. Sie sind Teil eines Registerbandes.



In Prozeduren können 8 Aufrufparameter, 8 lokale und 8 globale Variable verwendet werden.

Dann sind durch die überlappende Weiterschaltung bei einem Prozeduraufruf 8 Register für Übergabeparameter vorgesehen. Die untersten 32 Register des Bandes sind reserviert für globale Parameter und den Registersatz des Trap-Handlers.

Die Sun Microsystems SPARC 10 wurde ein großer Erfolg: durch die einfachere interne Struktur konnte die Maschine erheblich schneller arbeiten als entsprechende CISC-Rechner. Allerdings wurde der vom Compiler zu erzeugende Code länger, da mehr einfache Befehle zu verarbeiten waren.

Inzwischen verwenden alle modernen Processoren im Kern eine RISC-Architektur, d.h. einen einfachen Befehlssatz mit Befehlen gleicher Länge und abgearbeitet mit der gleichen Zahl von Takten; auch Maschinen, die x86-befehlscodekompatibel sind wie der Pentium-Processor.

11.5. RISC-CISC- Architektur

Der Zwang zur Kompatibilität mit alten Systemen erzwingt, daß Rechner in der Lage sein müssen, Maschinenbefehle der alten Rechner zu verarbeiten. Wenn die alte Maschine Befehle verschiedener Länge kennt, die neue Maschine intern eine RISC-Architektur hat, dann muß intern eine Codewandlung geschehen.

Das ist der Fall bei neuen Maschinen mit x86-Befehlssatz: Dort gibt es Befehle verschiedener Länge, Speicher-Speicher-Befehle und komplexe Adressierungen. Aus einem Befehl in x-86-Code müssen dann mehrere Befehle in einem RISC-86-Code erzeugt werden:

Speicher < -- Speicher + R1 wird zu

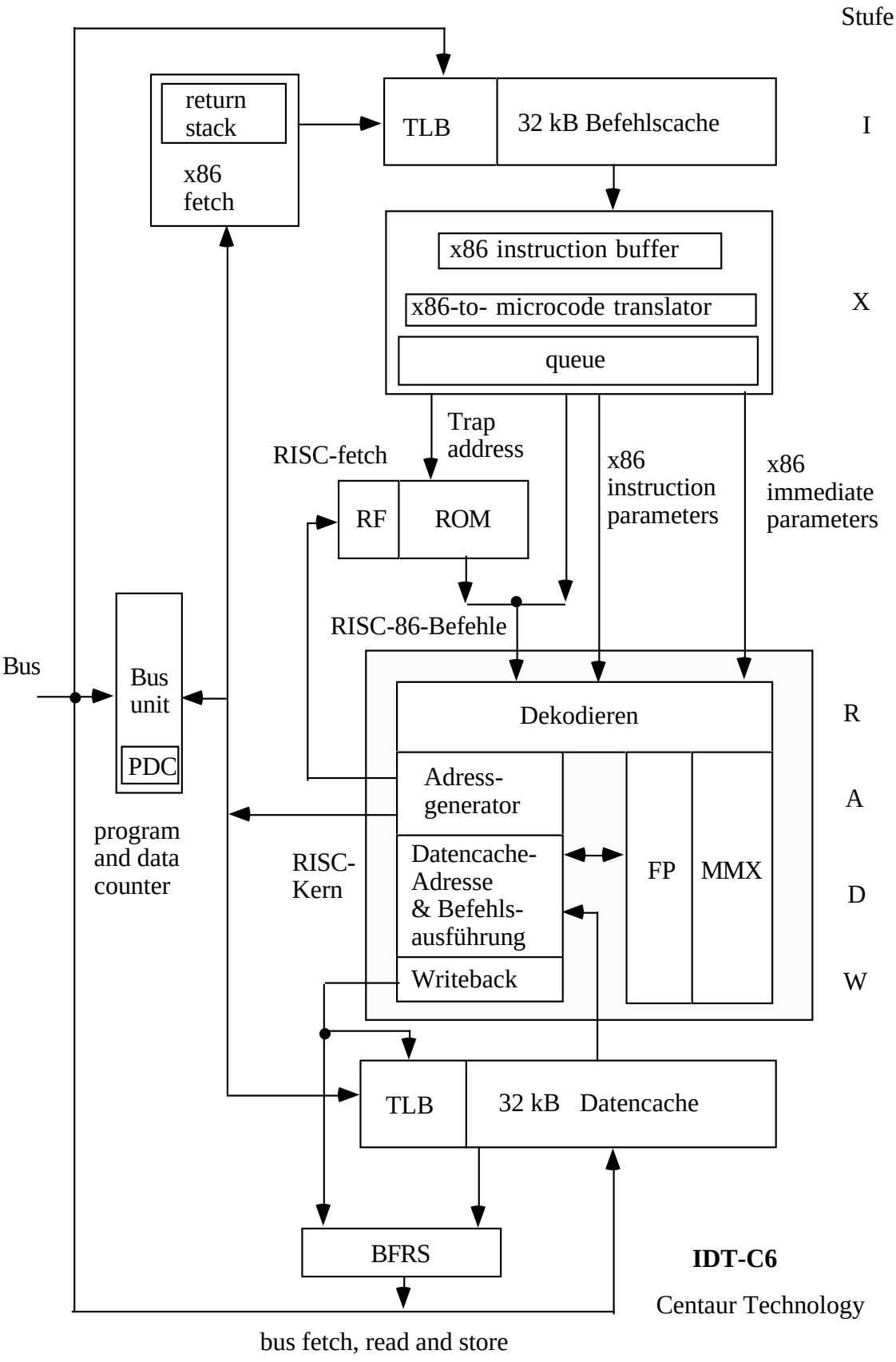
R2 <-- Speicher; R2 <-- R1 + R2; Speicher <-- R2;

Zusätzlich ist die Adresse im Befehl umzusetzen : die effektive Adresse muß durch einen oder mehrere Befehle im RISC-Code errechnet werden, die sich auf diejenigen Register des RISC-Rechners beziehen, die als die Register der x86-Architektur interpretiert werden. Die dann erzeugte Adresse ist die Speicheradresse einer RISC-Maschine, die aber aus Sicht des x86-Programms transparent ist. Nach außen hin hat der Benutzer eine verbesserte Version des 80x86 in der Hand.

Intern wird in der Risc-Maschine bei einem x86-Befehl in den meisten Fällen ein Microprogramm im RISC-Code angeworfen.

Als Beispiel soll der Rechner **IDT-C6** von Centaur Technology dienen, der für einfache Anwendungen konzipiert ist ohne den Anspruch auf Höchstleistung. Er ist vergleichbar dem Pentium MMX, aber intern lange nicht so komplex.

Er hat zwar 5.4 Millionen Transistorfunktionen auf dem Chip, doch gehen davon ab zwei Caches von je 32 kB für Daten und Programme und zwei Assoziativspeicher (translation lookaside table,TLB) für Daten und Instruktionen. Gefertigt in 0.35µm-Technik mit 4-Lagenmetallisierung in CMOS-Technik auf 88mm^2 Fläche verbraucht er ca. 10 W Leistung (Pentium 35 W). Der Kern ist ein RISC-Processor in einem Standardentwurf; die Komplexität der x-86-Maschine inklusive der neuen MMX-Befehle wird durch Microcode erbracht.


Stufe
return stack
x86 fetch
TLB
32 kB Befehlscache
I
x86 instruction buffer
x86-to- microcode translator
queue
X
Trap address
RISC-fetch
x86 instruction parameters
x86 immediate parameters
RF
ROM
RISC-86-Befehle
Bus
Bus unit
PDC
program and data counter
Dekodieren
R
Adress-generator
A
RISC-Kern
Datencache-Adresse & Befehls-ausführung
FP
MMX
D
Writeback
W
TLB
32 kB Datencache
BFRS
IDT-C6
Centaur Technology
bus fetch, read and store