Tabelle 1.10: Marktallianzen im RISC-Bereich

Das HP-PRO-Lager: HP PA (Power Architecture)

- HP
- Hitachi
- Mitsubishi
- Hughes
- Oki
- Yokogawa
- Prime
- Convex
- Sequoia

Das Sparc-Lager (unter anderem): SPARC

- Sun Microsystems
- Amdahl
- Cray
- Fujitsu
- ICL
- Solbourne
- Toshiba
- Xerox
- Matsushita
- Tadpole Technology
- Goldstar
- Hyundai
- Tatung
- Force Computers
- LSI Logic
- Tulip Computers
- Meiko

Das Motorola-Lager (88-Open unter anderem): 88000 (siehe auch Power-PC)

- Motorola
- Data General
- Omron
- Encore Computer
- Dolphin Server Technology
- Sanyo/Icon International Inc.
- Cetia (Thomson CSF)
- Harris Computer Systems Division
- McDonnell Douglas Information Systems
- Micro Focus
- Philips Information Systems
- SAS Institute
- Tadpole Technology
- Unify
- Wordperfect
- Oracle
- Unix International

Das Power-Open-Lager: POWER (siehe auch Power-PC)

- IBM
- Apple
- Bull HN Information Systems Inc.
- Harris Corp.
- Motorola
- Tadpole Technology
- Thomson CSF

Das Digital-Lager: Alpha

- DEC
- Kubota
- Olivetti
- Raytheon
- Encore Computer
- Advanced Computer Research International
- Carrera Computers
- Cray Research

Das Mips-Lager (unter anderem): MIPS

- Silicon Graphics (Mips)
- DEC
- Pyramid
- SNI
- Siemens
- Olivetti
- Tandem
- NEC
- Sony
- AT&T
- Sumitomo
- Control Data
- Microsoft
- LSI Logic
- Integrated Device Technology
- Toshiba
- Acer
- Carrera Computers

Intel: (i860)

- PentiumPro (nur im Kern ?!)

Das Power-PC-Lager: POWER-PC

- IBM
- Apple
- ASI
- Bull HN Information Systems Inc.
- Canon
- Ford (für den Einsatz in Kraftfahrzeugen)
- Harris Corp.
- Motorola
- Peacock
- Tadpole Technology
- Taiwan New PC Consortium (24 Hersteller)
- Thomson CSF
- Vobis

1.3 Field replaceable units (FRU's)

Am Beispiel des kleinsten RISC-Rechners von Sun Microsystems, der ELC:

- Systemplatine, darauf:
  - NVRAM, d.h. non volatile (= nicht flüchtiger) Speicher
  - Speichermodule SIMM's (= single inline memory module)
- Bildschirm/Monitor

Und in größeren Rechnern:

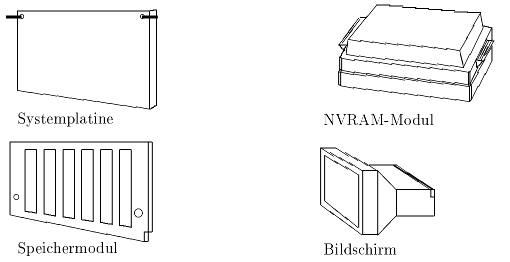
- **SBUS**-Karten: z.B. SCSI-Interface, ...

$\xrightarrow{\text{für I/O}}$ „offener“ Chip-Bus, direkt mit internem CPU-Bus verbunden, keine Treiberlogik auf Karte notwendig, postkartengroß, 96 poliger Stecker, maximal 10 Watt, 32 Bit Daten, $\leq$ 100 MB/s (aktuell 20...25 MB/s), Hardware-Adresse entspricht Steckplatz, Karte identifiziert sich bei Systemstart selbständig, kein Jumper nötig.

- **MBUS**-Karten: Prozessor (IU + FPU + Cache Controller + Cache Memory), pro Karte 1 oder 2 Prozessoren

$\xrightarrow{\text{für Memory}}$ „offener“ Memory-Bus, 40 MHz Takt, 64 Bit Daten, $\leq$ 320 MB/s für Memory oder CPU's geeignet.

Abbildung 1.8: FRU's der SPARC-ELC

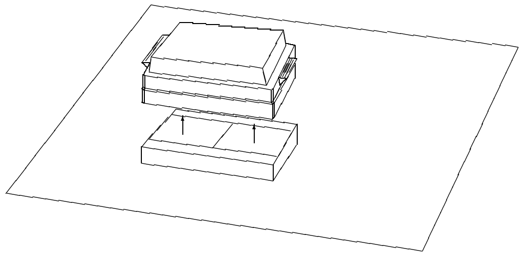


Auszüge aus Reparaturanleitungen für Feld-Techniker:

- **Entfernen des NVRAM-Moduls:**

  Fassen Sie das Gehäuse an den Griffen und ziehen Sie das NVRAM kräftig nach oben ab (siehe Abbildung):

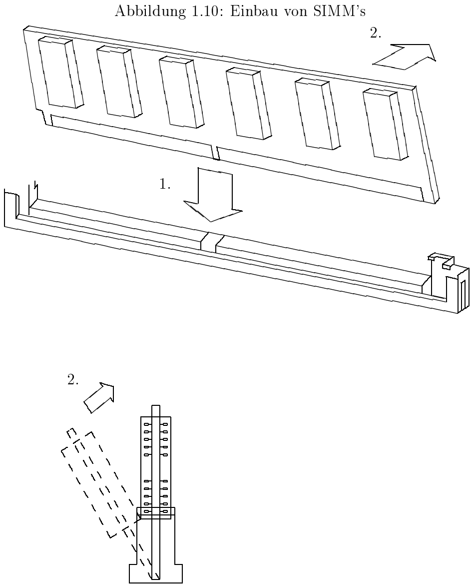
Abbildung 1.9: Entfernen des NVRAM-Moduls



Vorsicht: Die Systemplatine und das NVRAM-Modul bestehen aus hochempfindlichen elektronischen Bauteilen, die gegen statische Aufladung empfindlich sind. Vergewissern Sie sich, daß Sie beim Austausch der Komponenten alle Sicherheitsbestimmungen einhalten.

- **Einbau von Speichermodulen:**

Abbildung 1.10: Einbau von SIMM's

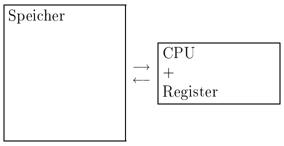


Hinweis: Es spielt keine Rolle, welches Modul Sie in welchen Steckplatz installieren. Sie können jedoch nicht ein Modul zwischen zwei anderen bereits installierten Modulen montieren, da die Sockel dafür zu eng nebeneinander liegen.

Vorsicht: Vergewissern Sie sich, daß Sie den Steckplatz MEM1 nicht frei lassen, wenn Sie auf Ihrer Systemplatine weniger als vier Module installieren. Das System arbeitet nicht richtig, wenn kein MEM1-Modul installiert ist.

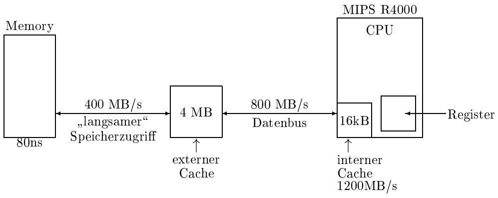
1.4 SIMD/MIMD und „data parallel programming“ (SPMD)

Abbildung 1.11: Der Flaschenhals zwischen CPU und Memory



1.4.1 Cache-Speicher

Abbildung 1.12: Der Cache als Daten-Vorratsbehälter



Bemerkungen:

- 80ns $\longleftrightarrow \frac{1}{80 \cdot 10^{-9} s} = 12.5 \cdot 10^6$ Zugriffe/s
- Bus: 50 MHz und 8 Byte Busbreite führt zu 400 MB/s
- im SIMM müssen also 32 Bit-Zugriffe „parallel“ erfolgen
- externer und interner Cache bilden einen Puffer, der mit “look ahead“ bei einer Trefferquote von $\sim 90 - 95\%$ betrieben wird.

- Es handelt sich um eine „schnelle CPU“ mit 100 MHz; also ergeben sich bei einem Durchsatz von 1200 MB/s durchschnittlich $\leq$ 12B pro Taktperiode $\leq$ 3 Worte/Eintakt-Befehl (32 Bit Befehl und 64 Bit Daten).

Hoher Durchsatz wird erreicht bei:

- sequentieller Ausführung von Befehlen,
- Zugriff auf ganze Vektoren (Daten),
- Zugriff auf 64/128 Bit-Gleitkommazahlen bei 32 Bit Datenbusbreite,
- durchschnittlich geringem Bedarf an Daten (aus dem Speicher) pro Befehl

Geringer Durchsatz wird erzielt bei:

- wilden Sprüngen, ...
- Benutzung von verketteten Listen mit „großen“ Knoten (CAD-Programme), ...

Bemerkung: Es wurden SparcStation10's ohne SuperCache ausgeliefert, um bei speziellen Anwenderprogrammen das „verlangsamende“ Cache-Füllen zu umgehen.

1.4.2 Vektorrechner

Statt der Verarbeitung einzelner Worte pro Taktperiode versucht man ganze Vektoren/Takt zu verarbeiten (Vervielfältigung der ALU):

Abbildung 1.13: Verarbeitungseinheit: Vektor von Worten

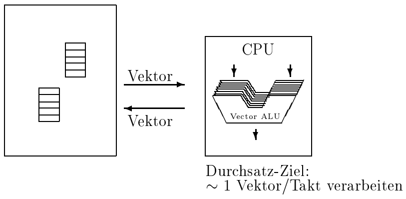
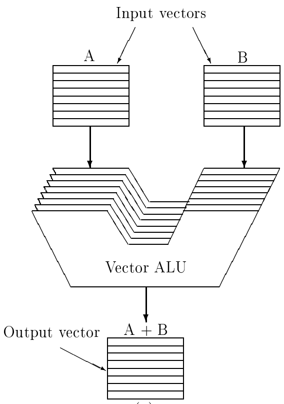


Abbildung 1.14: Vektorregeister und Vektoralu



(a)

$$\begin{array}{ccccc} A & & B & & A+B=C \\ \begin{bmatrix} 4\\6\\3\\2\\7\\8\\3\\8 \end{bmatrix} & + & \begin{bmatrix} 5\\3\\1\\3\\4\\1\\8\\8 \end{bmatrix} & = & \begin{bmatrix} 9\\9\\4\\5\\1\\9\\11\\16 \end{bmatrix} \end{array}$$

(b)

(a) Vektor-ALU.

(b) Beispiel für die Vektor-Addition.

Andere Beispiele für Vektoroperationen:

$C_i = f_1(A_i)$ mit $f_1 \in \{\cos, \sqrt{}, \ldots\}$ komponentenweise ausgewertet

$s = \sum_{i=1}^{N} A_i$ Summe, Minimum, Produkt

Minimum
Produkt
Maximum
⋮

N ALU's sind i.a.:

- zu teuer, da jede einzelne ALU sehr schnell sein soll
- unflexibel in der Vektor-Länge

Kompromiß: „eine ALU im Pipelinebetrieb statt eines Vektors von ALU's“

FPU-Pipeline (baue ALU aus „unabhängigen“ Einheiten auf):

	Beispiel: Subtraktion
1) fetch operands	$1.082 \cdot 10^{12}$, $9.212 \cdot 10^{11}$
2) adjust exponent	$1.082 \cdot 10^{12}$, $0.9212 \cdot 10^{12}$
3) execute „-“	$0.1608 \cdot 10^{12}$
4) normalize result	$1.608 \cdot 10^{11}$

$C_i = A_i - B_i \ \forall\ i = 1 \ldots N$

Zyklus	1	2	3	4	5	...
fetch	B_1, C_1	B_2, C_2	B_3, C_3	B_4, C_4	B_5, C_5	
adjust	—	B_1, C_1	B_2, C_2	B_3, C_3	B_4, C_4	
execute	—	—	$B_1 - C_1$	$B_2 - C_2$	$B_3 - C_3$	
normalize	—	—	—	$B_1 - C_1$	$B_2 - C_2$	

$\searrow B_1 - C_1 \quad \searrow B_2 - C_2$

Resümee:

- Ein Einzelbefehl (z.B. Subtraktion) braucht 4 Zyklen Zeit.
- Nach einer Startup-Zeit von 4 Zyklen (Füllen der Pipe) wird dann jedoch pro Zyklus ein Endergebnis geliefert. (Das ist allerdings nur eine Komponente des Ergebnis-Vektors.)
- Alte Vektorrechner benötigen eine Vektorlänge ≥ 100, damit sie schneller als ein Skalarrechner arbeiten; neuere Vektorrechner kommen jedoch schon mit einer Vektorlänge von $\geq 2 \ldots 4$ aus.
- **Problem:** „Datenabhängigkeit“

  Ein Wert ist noch in der Pipe, obwohl er schon als neues Argument benötigt wird.
- Jede ALU arbeitet denselben Befehl ab (SIMD = „single instruction multiple data“)
- COMPILER vektorisieren z.T. automatisch.