Aktuelle Techniken zur Leistungssteigerung von Mikroprozessoren

Eidgenössische
Technische Hochschule
Zürich

Departement Informatik
Institut für
Computersysteme

Aktuelle Techniken zur Leistungssteigerung von Mikroprozessoren

Hans Eberle

Technical Report #237, 1995

July 1995

ETH Zürich
Departement Informatik
Institut für Computersysteme
Prof. Dr. N. Wirth

Author's address:

H. Eberle, Institut für Computersysteme, ETH Zentrum, CH-8092 Zurich, Switzerland
e-mail: eberle@inf.ethz.ch

This report is available via <http://www.inf.ethz.ch/publications/tr.html> or <ftp://ftp.inf.ethz.ch/> as pub/publications/tech-reports/237.ps

Aktuelle Techniken zur Leistungssteigerung von Mikroprozessoren

Hans Eberle

Institut für Computersysteme
ETH Zürich

Zusammenfassung

Die Rechenleistung von Mikroprozessoren wird jährlich um rund 50 % verbessert. Diese Leistungsverbesserung ist im wesentlichen den Fortschritten der Halbleiterindustrie zu verdanken, welche den Entwicklern von integrierten Schaltungen einerseits immer schnellere Komponenten und andererseits eine immer grössere Anzahl von Komponenten auf einem Chip zur Verfügung stellt.

Die Verfügbarkeit schnellerer Komponenten erklärt die erzielten Leistungsverbesserungen nur zum Teil. Die beiden wichtigsten Techniken, um die Arbeitsgeschwindigkeit von Mikroprozessoren weiter zu erhöhen, sind die Pipelineverarbeitung und die Verwendung von Cachespeicher [3, 6]. Die Pipelineverarbeitung erlaubt es, mehrere Operationen parallel auszuführen, während Cachespeicher es ermöglichen, schneller auf Operationen und Operanden zuzugreifen.

In diesem Artikel werden die erwähnten Techniken besprochen und deren Anwendung und Ausführung in modernen RISC-Mikroprozessoren vorgestellt.

1. Technologische Fortschritte

Die Fortschritte in der Computerindustrie sind vor allem auf technologische Fortschritte der Halbleiterindustrie zurückzuführen. Der mit Abstand wichtigste Technologietrend ist die kontinuierliche Verkleinerung der Strukturgrössen von integrierten Schaltungen. Kleinere Strukturen sind ein Grund, weshalb immer mehr Komponenten auf einem Chip integriert werden können. Weiter gelingt es den Halbleiterherstellern, die Fehler beim Herstellen und Verarbeiten der Wafer ebenfalls kontinuierlich zu reduzieren und damit die verfügbare Chipfläche zu vergrössern. Die Verkleinerung der Strukturen und Vergrösserung der Chipfläche tragen multiplikativ zu der jährlich fünfzigprozentigen Erhöhung der Anzahl Komponenten bei, welche auf einem Chip integriert werden können.

Wie Figur 1 zeigt, wiesen die ersten integrierten Schaltungen vor etwas mehr als 30 Jahren Strukturgrössen von 50 µm und Chipflächen von 4 mm^2 auf. Heute können rund eine Million dichtere Chips realisiert werden mit Strukturgrössen von 0.5 µm und Chipflächen von 290 mm^2. Der Graph zeigt ebenfalls, wie sich die Anzahl Transistoren auf einem Mikroprozessorchip von 2'200 im Jahre 1971 auf 9.3 Millionen heute, im Jahre 1994 erhöht hat.

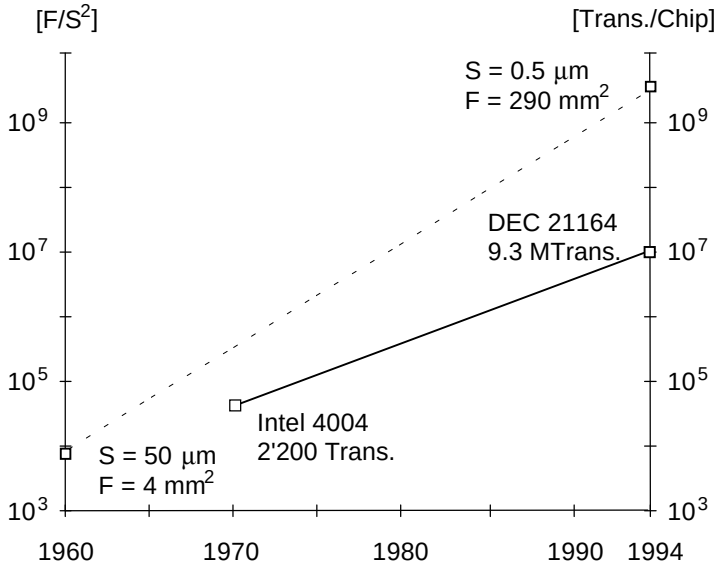
Die Auswirkungen dieses Entwicklungstrends auf die Schlüsselkomponenten von Rechnern, d.h. Speicher- und Prozessorchips lässt sich wie folgt als 50-Prozent-Regel zusammenfassen:

- Die Kapazität von Speicherchips wird jedes Jahr um 50 Prozent erhöht.
- Die Leistung von Prozessoren wird jedes Jahr um 50 Prozent erhöht.

Wie ist nun die Leistungsverbesserung von Prozessorchips zu erklären? Ein wesentlicher Faktor ist der Umstand, dass sich proportional zur Verkleinerung der Strukturgrösse auch die Schaltgeschwindigkeit der Transistoren erhöht. Somit kann die Leistungsfähigkeit von Prozessoren verbessert werden, indem man lediglich einen dichteren Herstellungsprozess wählt und zu diesem Zweck die zur Chipherstellung benötigten Masken verkleinert.

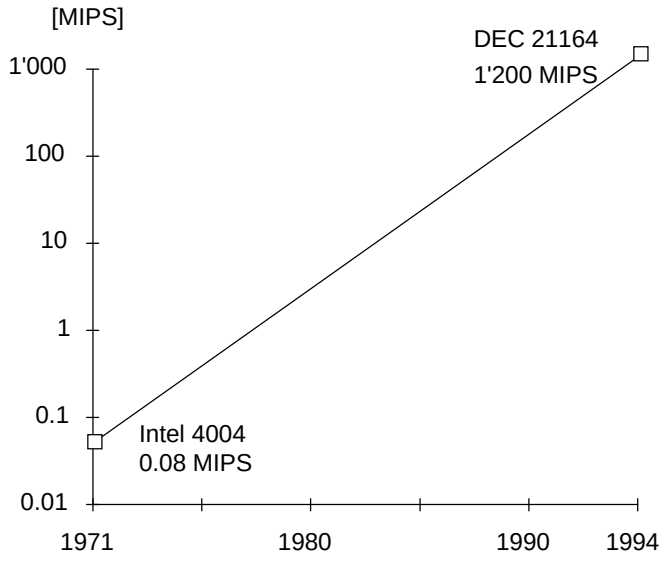
Die erzielten Leistungsverbesserungen sind aber auch das Resultat von neuartigen Prozessorarchitekturen, welche durch die erhöhte Integrationsdichte ermöglicht wurden. Die hohe Anzahl auf einem Chip verfügbarer

Komponenten lässt es zu, einerseits Speicher auf dem Prozessorchip selbst zu integrieren und damit die Speicherzugriffszeit zu verkürzen, und andererseits Instruktionen parallel auszuführen.



Figur 1: Integrationsdichte (Strukturgrösse S, Chipfläche F)

Figur 2 illustriert, wie sich die Prozessorleistung kontinuierlich seit der Markteinführung des ersten Mikroprozessors im Jahre 1971 verbessert hat. Der erste kommerziell erhältliche Mikroprozessor wurde von der Firma Intel hergestellt. Der Intel 4004, wie dieser Chip genannt wurde, realisierte eine 4-bit Architektur, das heisst, seine Funktionseinheiten konnten 4 Bits parallel verarbeiten. Dieser Mikroprozessorchip enthielt lediglich 2'200 Transistoren. Für die Ausführung einer Instruktion benötigte der Prozessor 12.5 µs. Dies entspricht einem Durchsatz von 80'000 Instruktionen pro Sekunde. Der leistungsfähigste heute verfügbare Mikroprozessorchip wurde von der Firma DEC entwickelt. Der Chip mit der Bezeichnung 21164 gehört zur Alpha-Prozessorfamilie und realisiert eine 64-bit Architektur. Auf diesem Chip befinden sich rund 9.3 Millionen Transistoren. Der Prozessor kann alle 3.3 ns bis zu 4 Instruktionen zur Ausführung bringen, was einem Durchsatz von 1'200 MIPS (*Millionen Instruktionen pro Sekunde*) entspricht.



Figur 2: Rechenleistung von Mikroprozessoren

2. Ein einfaches Leistungsmass

Die Zeit, welche ein Prozessor zur Ausführung eines Programms benötigt, lässt sich wie folgt berechnen:

$$Ausführungszeit = Anzahl\ Instruktionen \times \frac{Taktzyklen}{Instruktion} \times \frac{Zeit}{Taktzyklus}$$

oder

$$Ausführungszeit = Anzahl\ Instruktionen \times MIPS^{-1}.$$

Die drei Faktoren, welche also die Programmausführungszeit bestimmen, sind die "Zeit pro Taktzyklus", die Anzahl "Taktzyklen pro Instruktion" und die "Anzahl Instruktionen". Mit "Zeit pro Taktzyklus" ist die Taktperiode, respektive die Taktfrequenz^{-1} gemeint. Der Faktor "Taktzyklen pro Instruktion" ist ein Durchschnittswert, welcher oft mit der Einheit CPI (*Cycles per Instruction*) oder TPI (*Ticks per Instruction*) versehen wird. Mit "Anzahl Instruktionen" ist die Anzahl der tatsächlich ausgeführten Instruktionen gemeint.

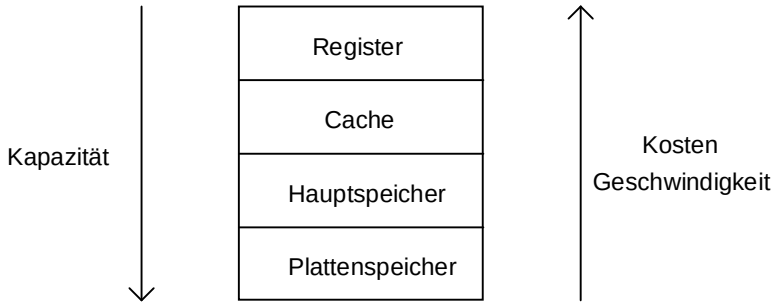
Will man die Leistung eines Prozessors erhöhen, so muss mindestens einer dieser drei Faktoren verkleinert werden. Dabei hängen diese teilweise zusammen. So hat beispielsweise der Instruktionssatz Einfluss darauf, wie viele Instruktionen zur Ausführung eines Programms benötigt werden, aber auch darauf, wie viele Taktzyklen durchschnittlich pro Instruktion gebraucht werden. Die Architektur des Prozessors wiederum hat Einfluss auf die benötigten Taktzyklen pro Instruktion und auf die Länge der Taktperiode. Die Halbleitertechnologie schliesslich bestimmt die Geschwindigkeit der Komponenten und damit die maximal erreichbare Taktrate.

Während konventionelle Prozessoren mit einer *CISC-Architektur (Complex Instruction Set Computer*) CPI-Werte von ungefähr 10 aufweisen, erreichen moderne Prozessoren mit *einer RISC-Architektur* (*Reduced Instruction Set Computer*) CPI-Werte von typischerweise 1. Diese drastische Verbesserung ist einerseits auf die Wahl des Instruktionssatzes zurückzuführen, andererseits aber auch auf die Verwendung von schnellen Cachespeicher und optimal ausgenutzten Pipelines. Diese beiden Konzepte sollen in den folgenden Abschnitten vorgestellt werden.

3. Cachespeicher

Die Anbindung des Prozessors an den Speicher wird oft als *von-Neumann-Flaschenhals* bezeichnet, da sämtliche vom Prozessor verarbeiteten Operationen und Operanden durch diesen hindurch müssen. Dieser bildet einen wesentlichen Faktor in Bezug auf die Leistungsfähigkeit eines Rechnersystems.

Da sich die Kosten, Geschwindigkeit und Kapazität von Speicherbausteinen nicht gleichzeitig optimieren lassen, wird versucht, diesen Ansprüchen zu genügen, indem der Speicher hierarchisch organisiert wird. Figur 3 illustriert eine typische Speicherhierarchie: die schnellen, teuren und geringe Kapazität aufweisenden Bausteine befinden sich nahe beim Prozessor, während sich die langsameren, kostengünstigeren und höhere Kapazität aufweisenden Bausteine weiter entfernt vom Prozessor befinden.



Figur 3: Speicherhierarchie

An der Spitze der Speicherhierarchie findet man die Prozessorregister. Typischerweise enthalten moderne RISC-Prozessoren 64 Register, 32 32-bit Integer-Register und 32 64-bit Floating-Point-Register. Das Schrei-

ben oder Lesen eines Registers dauert einen Zyklus. Die nächst tiefere Hierarchiestufe nehmen die Cachespeicher ein. Diese haben typischerweise eine Kapazität von 8 bis 256 kByte und können, falls sie sich auf dem Prozessorchip befinden, in 1 bis 4 Zyklen zugegriffen werden.

Der Grund, weshalb Speicher hierarchisch organisiert werden können, liegt in der Lokalität der Speicherzugriffe eines Prozessors begründet. Dabei unterscheidet man:

- *zeitliche Lokalität*: die Wahrscheinlichkeit ist gross, dass auf eine Speicherstelle zeitlich mehrfach hintereinander zugegriffen wird;
- *örtliche Lokalität*: die Wahrscheinlichkeit ist gross, dass auf örtlich nahe beieinander liegende Speicherstellen zugegriffen wird.

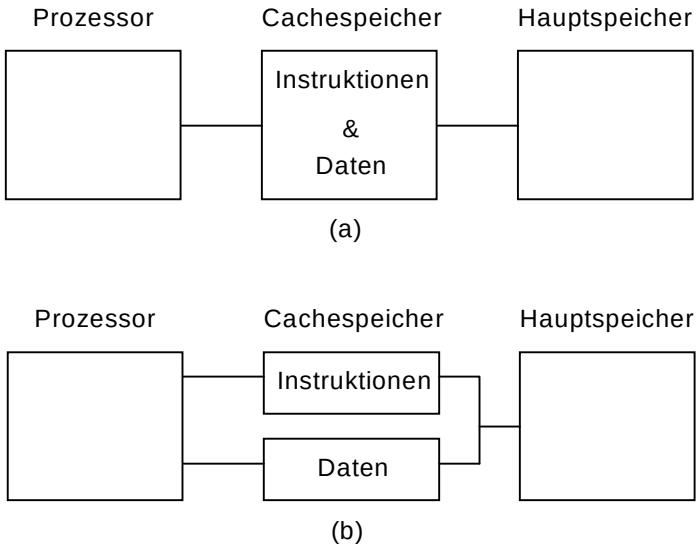
Da aus Kapazitätsgründen nicht alle Daten, welche bei der Ausführung eines Programms zugegriffen werden, sich in Prozessornähe befinden können, werden die Daten laufend von einer Hierarchiestufe zur anderen verschoben. Werden Daten zugegriffen, so müssen sie im schlimmsten Fall vom Plattenspeicher in den Hauptspeicher, von dort in den Cachespeicher und schliesslich in die Register geladen werden. Werden Daten auf diese Weise nach oben verschoben, so werden in der Regel als Folge davon Daten nach unten verschoben, um Platz zu schaffen für die neuen Daten.

Da Speicherzugriffe wie erwähnt zeitlich und örtlich lokal erfolgen, lohnt es sich, die Granularität der zugegriffenen Datenmenge zu variieren, je nachdem, auf welcher Hierarchiestufe der Zugriff stattfindet: je weiter entfernt sich der Speicher vom Prozessor befindet, desto grösser wird die Granularität gewählt. Zwischen den Registern und dem Cache werden typischerweise 32-bit Worte transferiert, zwischen dem Cache und dem Hauptspeicher *Blöcke* mit einer Grösse von 16 bis 256 Byte und zwischen dem Hauptspeicher und dem Plattenspeicher *Seiten* mit einer Grösse von 4 bis 16 kByte.

Die heute mögliche Integrationsdichte lässt sich nur mit regulären Strukturen ausnutzen, wie sie für Speicherbausteine benutzt werden. Aus diesem Grunde wird heute der grösste Anteil von Transistoren auf einem Prozessorchip für die Realisierung von Speicher, d.h. Register und Cachespeicher eingesetzt.

Während die Register mit Hilfe von Software verwaltet werden und damit für den Programmierer sichtbar sind, werden Cachespeicher von der Hardware verwaltet und sind für den Programmierer nicht sichtbar.

Im weiteren soll die Organisation von Caches betrachtet werden. Grundsätzlich lassen sich Cachespeicher in die beiden, in Figur 4 gezeigten Architekturen unterteilen. Befinden sich Instruktionen und Daten in einem gemeinsamen Cache, so spricht man von einer *Princeton-Architektur*, werden die Instruktionen und Daten hingegen in separaten Caches gespeichert, so spricht man von einer *Harvard-Architektur*. Beide Formen haben ihre Vor- und Nachteile. Der wesentliche Vorteil von separaten Caches ist die Möglichkeit, auf Daten und Instruktionen gleichzeitig zugreifen zu können. Der Vorteil eines gemeinsamen Caches ist die flexible Zuteilung von Cachespeicherplatz: benötigt ein Programm mehr Speicherplatz für die Instruktionen als für die Daten (oder umgekehrt), so wird die Speicherzuteilung automatisch entsprechend geändert.

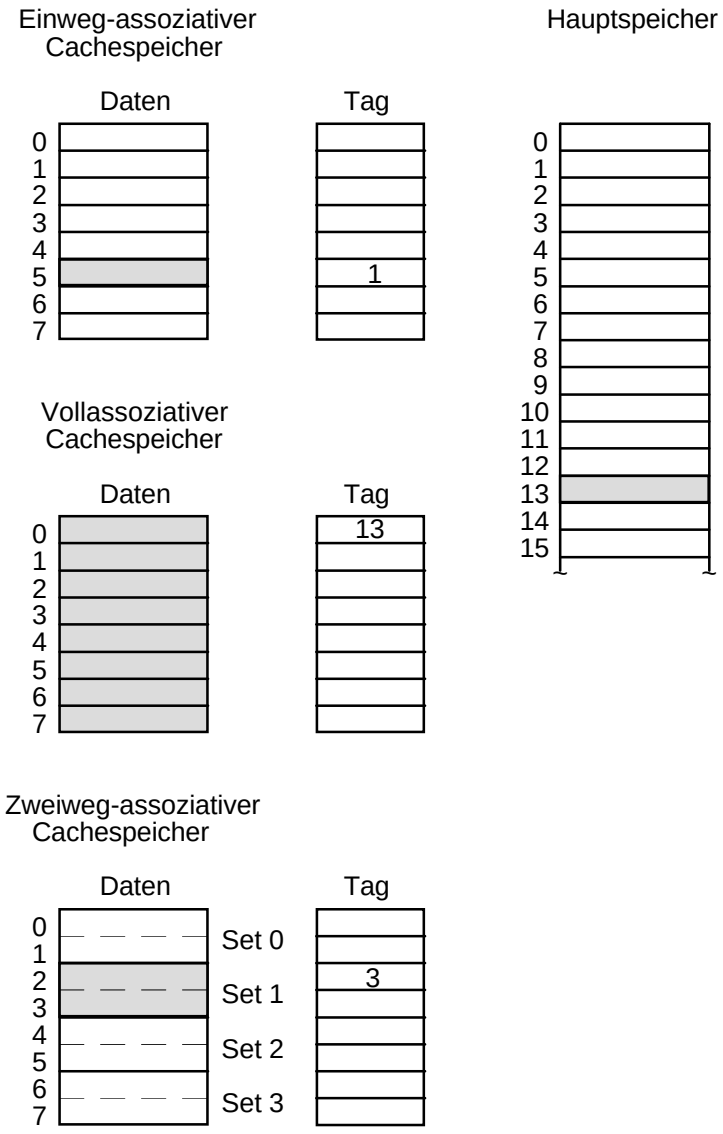


Figur 4: Cachespeicher mit einer Princeton- (a) und einer Harvard-Architektur (b).

Ein wesentliches Merkmal eines Cachespeichers ist seine *Assoziativität*. Diese gibt an, wie viele Einträge im Cache für die Speicherung eines Blocks aus dem Hauptspeicher in Frage kommen. Bei der einfachsten Organisationsform, dem *einweg-assoziativen* oder *direct-mapped* Cachespeicher gibt es für jeden Speicherblock nur einen möglichen Aufenthaltsort im Cache. Dabei geschieht die Adressumsetzung[1] wie folgt:

$$Cacheadresse = Speicheradresse\ MOD\ Anzahl\ Cacheeinträge.$$

Somit wird für das Beispiel in Figur 5 der Speicherblock mit der Adresse 13 im einweg-assoziativen Cache an der Adresse 5 (= 13 MOD 8) gespeichert.



Figur 5: Cachespeicher-Organisation

Diese einfache Adressumsetzung hat den Nachteil, dass beim Zugriff auf Speicherblöcke mit Adressen, welche auf die gleiche Adresse im Cache abgebildet werden, dieser Eintrag laufend überschrieben wird. Wür-

[1]Mit den Operatoren *DIV* und *MOD* erhält man den Quotienten *q* und den Rest *r* der ganzzahligen Division, wobei $q = \frac{x}{y}$ (ganzzahliger Anteil) und $r = x - q \times y$ ist.

den beispielsweise in Figur 5 abwechslungsweise die Speicherblöcke mit den Adressen 13 und 21 zugegriffen, so würde der Cacheeintrag mit der Adresse 5 immer wieder überschrieben und der Geschwindigkeitsvorteil von Cachezugriffen käme nicht zum tragen.

Solche Konflikte treten mit einem vollassoziativen Cache nicht auf. Bei dieser Organisationsform kommt jeder Cacheeintrag für die Speicherung von Speicherblöcken in Frage. Wie später gezeigt wird, steigt jedoch mit der Assoziativität der Realisierungsaufwand, weshalb in der Praxis günstigere Formen, wie der zweiweg-assoziative Cache gewählt werden. Allgemein gilt, dass es in einem *n*-weg-assoziativen Cache *n* mögliche Einträge für die Speicherung eines Speicherblocks gibt. Die *n* möglichen Einträge gehören zu einem *Set*. Der Index für das in Frage kommende Set wird wie folgt berechnet:

$$Cacheset = Speicheradresse\ MOD\ Anzahl\ Cachesets.$$

Im Beispiel in Figur 5 kommen also für den Speicherblock mit der Adresse 13 die beiden Einträge des Sets 1 (= 13 MOD 4), d.h. die beiden Cacheeinträge mit den Adressen 2 und 3 in Frage.

Da die Abbildung von Hauptspeicheradressen auf Cachespeicheradressen nicht eineindeutig ist, d.h. es wie erwähnt für einen Eintrag im Cache mehrere Kandidaten aus dem Hauptspeicher gibt, muss zusätzliche Information im Cache abgespeichert werden, um die Hauptspeicheradresse eines Eintrags eindeutig bestimmen zu können. Diese Information wird *Tag* genannt und besteht aus den höherwertigen Hauptspeicheradressbits. Das Tag für einen einweg-assoziativen Cache wird wie folgt berechnet:

$$Tag = Speicheradresse\ DIV\ Cacheeinträge.$$

Die Berechnung des Tags für einen mehrweg-assoziativen Cache sieht entsprechend aus:

$$Tag = Speicheradresse\ DIV\ Anzahl\ Sets.$$

Es genügt, die höherwertigen Speicheradressbits im Tag zu speichern, da ja die niederwertigen aus der Cacheadresse abgeleitet werden können.

Soll geprüft werden, ob sich ein Speicherblock im Cache befindet, so muss das Tag mit der Speicheradresse verglichen werden. Bei einem einweg-assoziativen Cache ist ein einziger Vergleich nötig, wohingegen bei einem *n*-weg-assoziativen Cache *n* Vergleiche nötig sind. Damit sind auch die hohen Kosten eines vollassoziativen Caches erklärt. Mehrweg-assoziative Caches sind einerseits aufwendig, andererseits muss eine längere Zugriffszeit als Folge der mehrfachen Vergleiche in Kauf genommen werden.

4. Pipelineverarbeitung

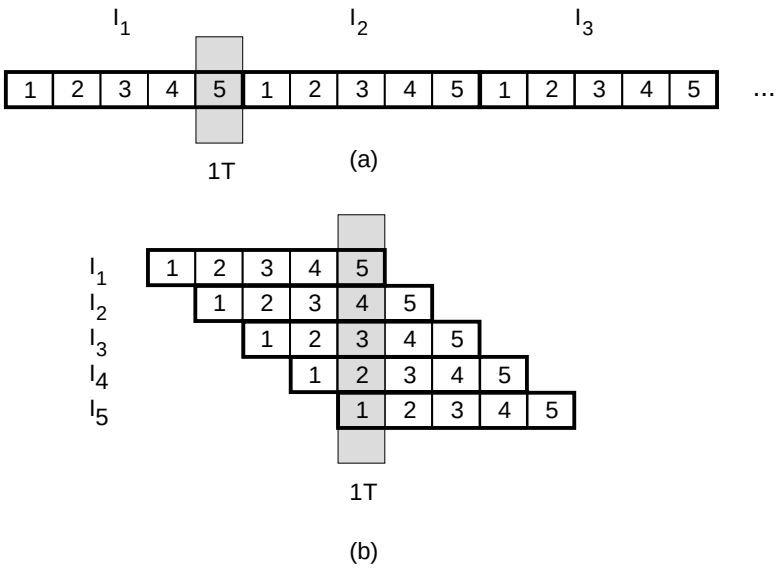
Die *Pipelineverarbeitung* ermöglicht es, mehrere Instruktionen überlappt auszuführen und damit die Arbeitsgeschwindigkeit eines Prozessors zu erhöhen. Ähnlich der Fliessbandverarbeitung wird bei der Pipelineverarbeitung die Ausführung einer Instruktion in mehrere Teilschritte zerlegt. Einen solchen Teilschritt nennt man *Pipelinestufe*. Die Pipelinestufen verarbeiten gleichzeitig mehrere Instruktionen, und zwar gleichzeitig soviele Instruktionen, wie es Pipelinestufen gibt.

Figur 6 veranschaulicht die Verarbeitung von Instruktionen mit und ohne Pipelineverarbeitung. In dem angegebenen Beispiel wird die Verarbeitung einer Instruktion in fünf Teilschritte zerlegt. Die in einem Takt geleistete Arbeit ist für beide Organisationsformen hervorgehoben. Da die Pipeline fünf Stufen enthält, befinden sich in der Pipeline bis zu fünf Instruktionen, welche gleichzeitig in Bearbeitung sind. Zu beachten ist, dass die Zeit, welche benötigt wird, um eine einzelne Instruktion zu verarbeiten, nicht kürzer wird dank Pipelineverarbeitung. Im Gegenteil, Pipelineverarbeitung erfordert normalerweise einen zusätzlichen, wenn auch geringen zeitlichen Mehraufwand. Die Verarbeitungsrate kann hingegen mittels Pipelineverarbeitung gesteigert werden. In Figur 6a wird alle 5 Zyklen die Ausführung einer Instruktion beendet, wohingegen in Figur 6b pro Zyklus die Ausführung einer Instruktion beendet wird.

Da die Pipelinestufen miteinander verbunden sind, gibt die langsamste Stufe den Arbeitstakt vor. Man nennt die Zeitdauer, welche eine Instruktion in einer Pipelinestufe verbringt, *Maschinenzyklus*. Ein Maschinenzyklus entspricht in der Regel einem *Taktzyklus*, kann aber auch, wie später gezeigt wird, durch zwei oder seltener mehrere Taktzyklen realisiert werden.

Beim Entwurf einer Pipeline wird man also bemüht sein, Pipelinestufen mit möglichst ausgeglichener Länge zu realisieren. Dank Pipelineverarbeitung kann der Durchsatz unter idealen Bedingungen um die Anzahl Pipelinestufen erhöht werden. Dieses Ideal wird in der Praxis jedoch aus mehreren Gründen nicht erreicht. Die Stufen einer Pipeline sind in der Praxis nicht perfekt ausgeglichen. Weiter ist es nicht immer mög-

lich, gleichzeitig in allen Pipelinestufen nützliche Arbeit zu verrichten. Und schliesslich wird zusätzliche Zeit benötigt, um die Daten beim Übergang von einer Pipelinestufe zur nächsten zwischenzuspeichern.



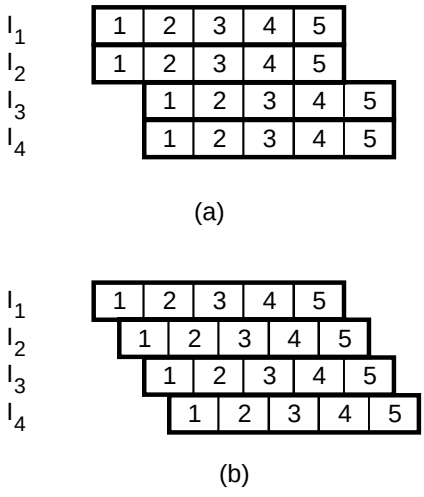
Figur 6: Instruktionsverarbeitung ohne (a) und mit (b) Pipelineverarbeitung

Es ist wichtig festzustellen, dass die parallele Verarbeitung von Instruktionen mittels Pipelineverarbeitung für den Programmierer unsichtbar bleibt. Die Instruktionen werden zwar überlappt ausgeführt, jedoch in ihrer ursprünglichen sequentiellen Ordnung abgearbeitet.

Neuere Prozessoren weisen sogar CPI-Werte von weniger als 1 auf, was bedeutet, dass der Durchsatz mehrere Instruktionen pro Takt beträgt. Architekturen mit solchen Leistungsmerkmalen verwenden entweder eine *Superpipeline* oder eine *superskalare Pipeline*.

Ein Prozessor mit einer superskalaren Pipeline erlaubt es, mehr als eine Instruktion pro Takt zur Ausführung zu bringen. Damit dies möglich ist, müssen die zur Ausführung der Instruktionen nötigen Ressourcen, wie Funktionseinheiten und Datenpfade in mehrfacher Ausführung vorhanden sein. Vergleichbare Leistungsverbesserungen lassen sich auch mit einer Superpipeline erreichen. Dabei wird eine Pipelinestufe in weitere Stufen aufgeteilt mit dem Ziel, die Pipeline schneller zu takten und damit den Durchsatz zu erhöhen.

Figur 7 veranschaulicht diese beiden Techniken. Beide gezeigten Pipelineorganisationen weisen einen CPI-Wert von 0.5 auf.



Figur 7: Superskalare Pipeline (a) und Superpipeline (b)

5. RISC-Pipeline

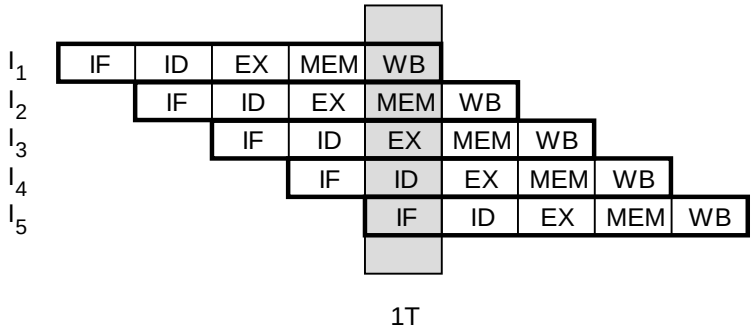
Eine nahezu optimale Pipelineverarbeitung erreichen Prozessoren mit einer RISC-Architektur. Das ursprüngliche Ziel bei der Entwicklung von RISC-Architekturen war es, eine Verarbeitungsrate von einer Instruktion pro Maschinenzyklus, d.h. 1 CPI zu erreichen. Erreicht wird dies mittels eines - wie mit der Bezeichnung RISC angedeutet wird - reduzierten, einfachen Instruktionssatzes. Die Konsequenzen des einfachen Instruktionssatzes sind eine sogenannte *Load-/Store-Architektur*, ein grosser Prozessorregistersatz und einfache Adressierungsmodi. Mit der Load-/Store-Architektur ist gemeint, dass pro Instruktion nur ein Speicherzugriff erlaubt ist. Damit wird für jeden Speicherzugriff eine separate Load- oder Store-Instruktion benötigt. Möchte man beispielsweise die Zuweisung $C := A + B$ ausführen, d.h. zwei Werte im Speicher addieren und das Resultat wieder zurück in den Speicher schreiben, so wird man typischerweise folgende Sequenz von Instruktionen erhalten:

Load Ra, A	(lade das Speicherwort *A* ins Register *a*)
Load Rb, B	(lade das Speicherwort *B* ins Register *b*)
Add Rc, Rb, Ra	(addiere Register *a* und *b* und schreibe das Resultat ins Register *c*)
Store C, Rc	(schreibe das Register *c* an die Speicherstelle *C*)

Konnte diese Sequenz mit einer CISC-Architektur mit einer einzigen komplexen Instruktion ausgedrückt werden, so werden mit einer RISC-Architektur mehrere einfache Instruktionen benötigt.

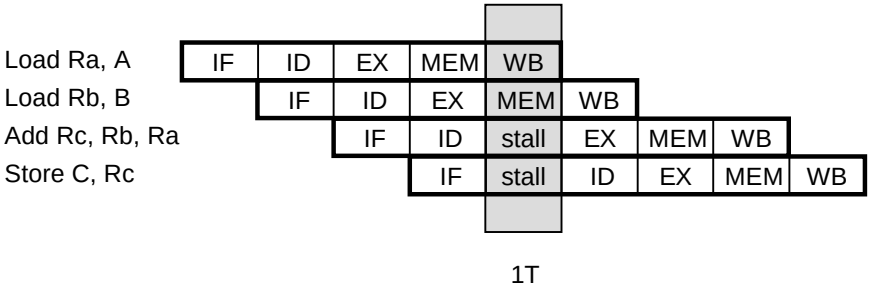
In etwas vereinfachter Form lässt sich eine typische RISC-Pipeline wie in Figur 8 beschreiben. Die Pipeline besteht aus den folgenden fünf Stufen:

- *IF* (Instruction Fetch): die Instruktion wird aus dem Speicher geholt.
- *ID* (Instruction Decode): die Instruktion wird dekodiert und die Quelloperanden werden aus den Prozessorregistern gelesen.
- *EX* (Execute): die Instruktion wird ausgeführt.
- *MEM* (Memory Reference): falls es sich um eine Load- oder Store-Instruktion handelt, wird der Speicher zugegriffen.
- *WB* (Write Back): der berechnete Zieloperand wird in ein Register geschrieben.



Figur 8: RISC-Pipeline

Wie erwähnt ist es nicht immer möglich, in allen Stufen einer Pipeline nützliche Arbeit zu verrichten. Verursacht werden diese Verzögerungen (*Pipeline Stalls*) beispielsweise durch Lade- oder Sprungbefehle. Figur 9 zeigt die Pipelineverarbeitung der vorigen Instruktionssequenz unter der Annahme, dass eine Ladeinstruktion eine Verzögerung von einem Zyklus bewirkt. Die Verzögerung tritt auf, falls das Zielregister der Ladeinstruktion als Quellregister der unmittelbar darauffolgenden Instruktion verwendet wird. Ladeinstruktionen, welche Verzögerungen bewirken, werden *Delayed Loads* genannt. Im gezeigten Beispiel wird die Add-Instruktion und damit sämtliche nachfolgenden Instruktionen um einen Zyklus verzögert, da das Quellregister *Rb* von der vorhergehenden Load-Instruktion geladen wird.



Figur 9: Verzögerungen in Pipelines

6. PowerPC, Alpha und MIPS im Vergleich

In diesem Abschnitt soll die Anwendung der besprochenen Techniken in modernen RISC-Mikroprozessoren vorgestellt werden. Als deren Vertreter wurden der PowerPC 601 [4, 7, 8], der Alpha 21064 [1, 5, 8] und der MIPS R4400 [2, 8] gewählt.

In Tabelle 1 werden die Implementierungen dieser Prozessoren verglichen. Die Dimensionen des 601 sind vergleichsweise klein, da der verwendete Herstellungsprozess nicht nur über die kleinsten Strukturen verfügt, sondern auch mehr Metall-, das heisst Verbindungslagen zulässt. Auffallend sind die unterschiedlichen Werte für die Taktfrequenz und den maximalen Stromverbrauch. CMOS-Schaltungen haben die Eigenschaft, dass ihr Stromverbrauch praktisch linear mit der Taktfrequenz ansteigt. Dies bedeutet, dass ein Chip desto heisser wird, je schneller er getaktet wird.

	PowerPC 601	Alpha 21064	MIPS R4400
Technologie	0.6 µm CMOS	0.75 µm CMOS	0.65 µm CMOS
Chipgrösse	1.09 cm^2	2.33 cm^2	1.86 cm^2
Anzahl Transistoren	2.8 x 10^6	1.68 x 10^6	2.3 x 10^6
Taktfrequenz	50 MHz	200 MHz	150 MHz
Stromverbrauch	9 W	30 W	15 W

Tabelle 1: Implementierungscharakteristika

Wie aus Tabelle 2 ersichtlich ist, sind die Leistungsdaten der Prozessoren erwartungsgemäss mit der Taktfrequenz korreliert. Die angegebenen Leistungszahlen wurden mit Hilfe der *SPEC-Bewertungsprogramme* (*System Performance Evaluation Cooperative*) erhalten. Diese Bewertungsprogramme lassen sich in zwei Gruppen aufteilen: *int92* enthält Programme, welche vor allem mit ganzen Zahlen operieren, und *fp92* enthält Programme, welche ausschliesslich mit Gleitkommazahlen operieren.

	PowerPC 601	Alpha 21064	MIPS R4400
SPEC int92	40	133	88
SPEC fp92	60	200	97

Tabelle 2: Leistungszahlen

Alle drei Prozessoren besitzen Cachespeicher auf dem Prozessorchip, doch weisen diese unterschiedliche Kapazitäten und Organisationen auf, wie dies Tabelle 3 belegt. Da der 601 eine wesentlich niedrigere Taktrate als der 21064 und R4400 besitzt, verfügt er über mehr Zeit, um den Cache zuzugreifen. Dies dürfte der Grund sein, weshalb im 601 ein 8-weg-assoziativer Cache benutzt wird, für welchen, wie in Abschnitt 3 gezeigt wurde, die Zugriffszeiten länger sind verglichen mit einem einweg-assoziativen Cache.

	PowerPC 601	Alpha 21064	MIPS R4400
Harvard/Princeton	Princeton	Harvard	Harvard
Assoziativität	8	1	1
Kapazität	32 kByte	8 kByte + 8 kByte	16 kByte + 16 kByte

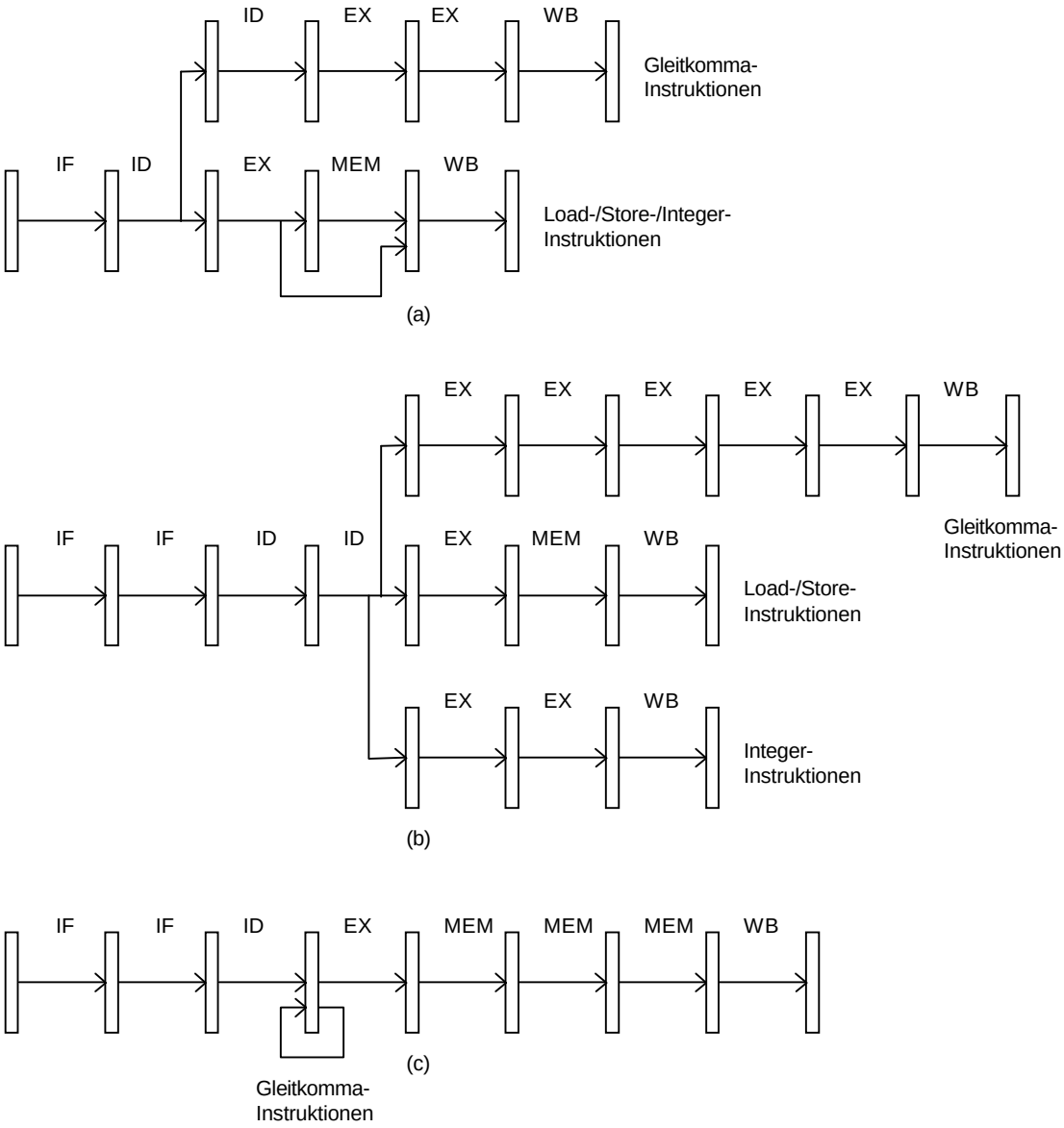
Tabelle 3: Cacheorganisation

Auch die Pipelinestruktur der betrachteten Prozessoren weist einige Unterschiede auf, wie Tabelle 4 illustriert. Während für den 601 und den 21064 eine superskalare Pipeline verwendet wird, entspricht die Struktur des R4400 einer Superpipeline. Auch die Anzahl der Pipelinestufen variiert beträchtlich. Die angegebenen Werte gelten für die meisten Instruktionen der jeweiligen Kategorie, jedoch nicht für gewisse aufwendige Instruktionen, wie Multiplikation oder Division.

	PowerPC 601	Alpha 21064	MIPS R4400
Superpipeline/ Superskalare Pipeline	Superskalare Pipeline	Superskalare Pipeline	Superpipeline
Pipelinestufen			
Floating Point	6	10	10
Load/Store	5	7	8
Integer	4	7	8

Tabelle 4: Pipelineorganisation

In Figur 10 sind die jeweiligen Pipelines im Detail angegeben. Die Aufgaben der einzelnen Pipelinestufen sind mit den aus Abschnitt 5 bekannten Bezeichnungen versehen. Die superskalaren Pipelines des 601 und 21064 erlauben es, mehr als eine Instruktion pro Takt zur Ausführung zu bringen. Im wesentlichen erlauben es beide Organisationsformen, gleichzeitig eine Gleitkommainstruktion und eine Load-/Store- oder Integer-Instruktion zur Ausführung zu bringen. Vergleicht man die Länge der beiden superskalaren Pipelines, so fällt auf, dass die Pipeline des 601 mit 4 bis 6 Stufen wesentlich kürzer ist als diejenige des 21064 mit 7 bis 10 Stufen. Die unterschiedlichen Längen liegen darin begründet, dass in einer Pipelinestufe des 601 mehr Arbeit verrichtet wird als in einer Stufe des 21064. Dies wiederum ist auch der Grund, weshalb der 21064 wesentlich schneller getaktet werden kann. Vergleicht man die Superpipeline des R4400 mit den beiden superskalaren Pipelines des 601 und 21064, so sieht man, dass die Pipeline des R4400 rund doppelt so lang ist wie diejenige des 601 und etwa gleich lang wie diejenige des 21064. Man könnte also den 21064 mit seiner hohen Anzahl Pipelinestufen auch superskalare Superpipeline nennen. Der Begriff Superpipeline ist nicht eindeutig anwendbar, da er gleichbedeutend ist mit einer tiefen Pipeline, d.h. mit einer viele Stufen enthaltenden Pipeline.



Figur 10: Pipelinestruktur des PowerPC 601 (a), Alpha 21064 (b) und MIPS R4400 (c)

7. Ausblick

Die Prozessorleistung kann mit einer Wachstumsrate von 50% pro Jahr nur dann weiterhin gesteigert werden, falls es gelingt, die Architektur der Prozessoren weiter zu verbessern. Ansonsten, wenn lediglich die technologischen Fortschritte zählen, wird die Wachstumsrate auf ca. 20% pro Jahr abfallen.

Weitere Innovationen werden nötig sein, insbesondere um noch mehr Instruktionen parallel verarbeiten zu können. Dabei bildet das Erkennen von Instruktionen, welche parallel ausgeführt werden können, das Hauptproblem, welches nicht einfach zu lösen sein wird. Da viele Programme wenig Parallelismus aufweisen und sich somit schlecht für die Verarbeitung in einer superskalaren Pipeline oder Superpipeline eignen, könnte ein Ausweg die gleichzeitige, parallele Verarbeitung von mehreren Programmen auf einem Prozessorchip sein.

Um mit der Leistungssteigerung der Prozessoren mithalten zu können, werden entsprechend leistungsfähigere Speichersysteme nötig sein. Da die Zugriffszeiten auf Hauptspeicher bisher nur unwesentlich verbessert werden konnten, wird es nötig sein, die Speicherhierarchie noch weiter zu verfeinern.

Ein bisher allzusehr vernachlässigtes Problem ist der hohe Stromverbrauch von modernen Prozessoren. Nicht nur ist die hohe Leistungsaufnahme ein ökologisches Problem, sondern auch ein Hinderungsgrund für den Einsatz von Hochleistungsprozessoren in gewissen Anwendungen, wie beispielsweise in mobilen Geräten.

Referenzen

[1] Digital Equipment Corporation: *Digital Technical Journal*. Vol. 4, no. 4, Special Issue 1992.
[2] J. Heinrich: *MIPS R4000 User's Manual*. Prentice Hall, 1993.
[3] J. Hennesy, D. Patterson: *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann, 1990.
[4] International Business Machines: *Journal of Research and Development*. Vol. 38. no. 5, September 1994.
[5] R. Sites: *Alpha Architecture Reference Manual*. Digital Press, 1992.
[6] H. Stone: *High-Performance Computer Architecture*. Addison-Wesley, 1993.
[7] S. Weiss, J. Smith: *POWER and PowerPC: Principles, Architecture, Implementation.* Morgan Kaufmann, 1994.
[8] World Wide Web: *CPU Info Center*. http://infopad.berkeley.eecs.edu/burd/~gpp/cpu.html.