Rechnerarchitektur

1. Einleitung

In welche sieben Ebenen kann man ein Rechnersystem einteilen?

1. **Anwendungsebene** *(Anwendersoftware)*
2. **Assemblerebene** *(Beschreibung von Algorithmen, Link & Bind)*
3. **Betriebssystem** *(Speichermanagment, Prozesskommunikation)*
4. **Instruction Set Architecture** *(ISA,Adressierungsarten)*
5. **Microarchitektur** *(Risc,Cisc, Branch Prediction..)*
6. **Logische Ebene** *(Register,Schieber, Latches..)*
7. **Transistorebene** *(Transistoren, MOS)*

Welche grundsätzlichen Recherarten gibt es?

- Personal Computer
- Workstations
- Mainframes
- Supercomputer
- Cluster Of Workstations
- Server
- Datenserver
- Netzcomputer

Wie lassen sich Architekturen klassifizieren?

Rechenprinzip

- Von Neumann (Steuerfluss)
- Datenfluß (Zündregel)
- Reduktion (Funktionsaufruf)
- Objektorientiert (Methodenaufruf)

Grundkonzept

- Vektorrechner (Pipeline)
- Array-Computer (Data-Array)
- Assoziativ-Rechner (Assoziativ-Speicher)

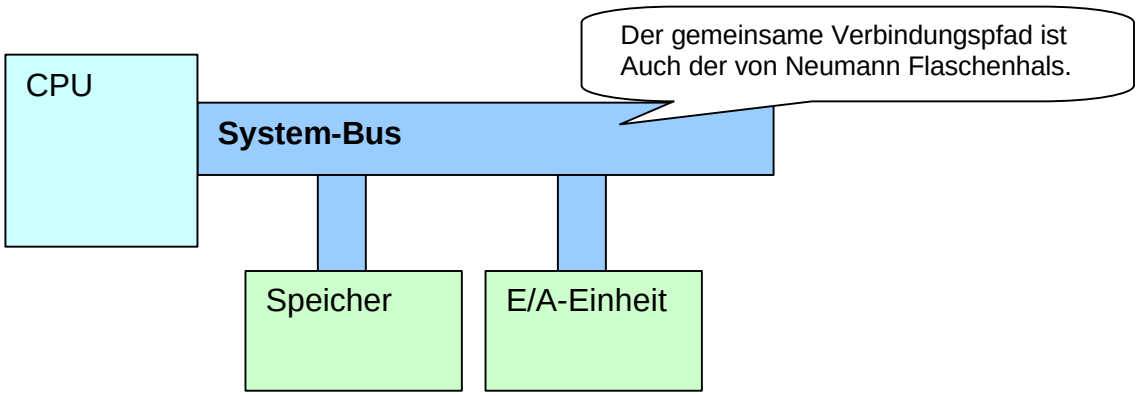
Welche drei Merkmale definieren die Leistung?

Architektur **-** Pipeline, Superskalar, Spekulative Ausführung, Caches, Busbreite

Software **-** Compileroptimierung

Siliziumbasis **-** Transistordichte und Taktraten

Was sind die vier Hauptbestandteile eines typischen Rechners?



Was unterscheidet eine Schnittstelle von einem Bus?

Ein Bus verbindet mehr als zwei Teilnehmer.

John von Neumann mit ENIAC

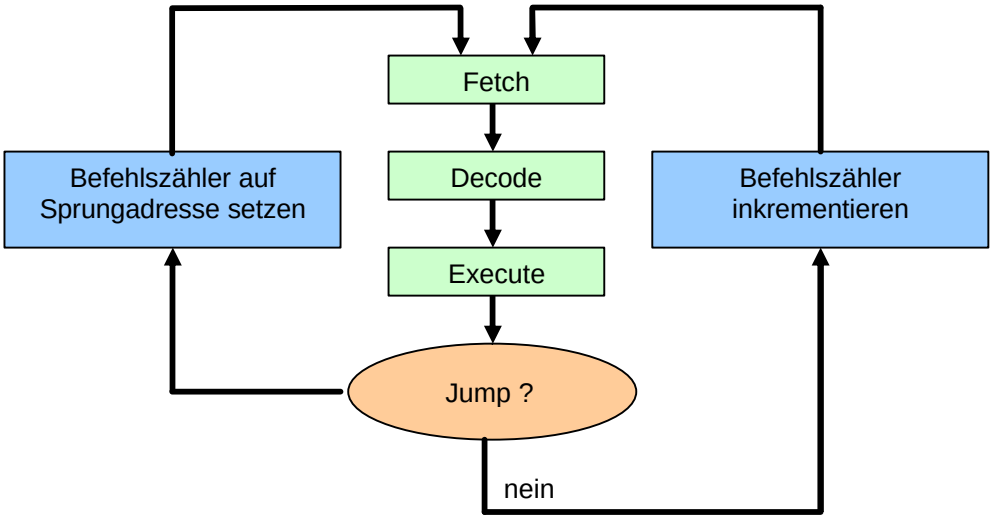
Was definiert einen Neumann-Rechner?

Der von Neumann-Rechner **arbeitet sequentiell**, Befehl für Befehl wird abgeholt, interpretiert, ausgeführt und das Resultat abgespeichert.

- *Steuerwerk (Taktgeber und Befehlszähler)*
- *Speicher*
- *Rechenwerk (CPU)*
- *I/O-Einheit*

Datenbreite, Adressierungsbreite, Registeranzahl und Befehlssatz können als Parameter verstanden werden.

Wie arbeitet die zentrale Befehlsschleife eines Von-Neumann-Rechners?



Was heißt Havard-Architektur?

Daten- und Befehlsspeicher sind getrennt. So ist es möglich Daten und Befehle Zeitgleich aus dem Speicher zu holen. Da dies aber einen extrem hohen Aufwand bedeutet, wird dies nur bei Echtzeitanwendungen implementiert.

Was ist ein Taktzyklus?

Die Interpretation und Ausführung eines Befehles erfolgt in vier Phasen.

1. Holen
2. Dekodieren (inklusive Operandenadressen berechnen)
3. Daten holen (bzw. Operanden)
4. Ausführen

Jede der vier Phasen wird in eine Anzahl von Schnittstellen bzw. Zyklen eingeteilt. Ein Taktzyklus ist die kleinstmöglich verarbeitbare Einheit. Somit benötigt ein Befehl zur Ausführung im Allgemeinen mehr als einen Taktzyklus.

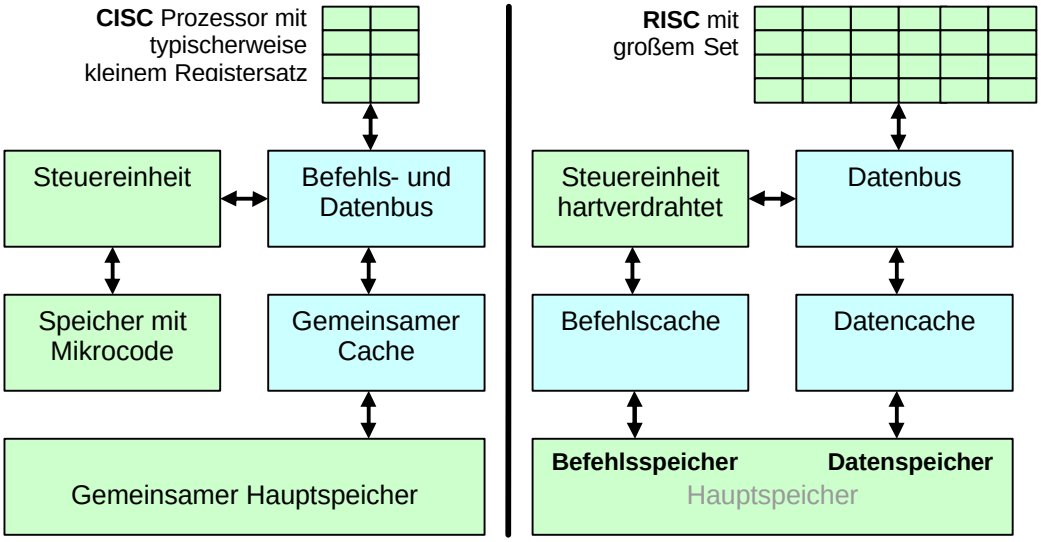
Was ist Mikroprogrammierung?

Durch Einsatz von Matrix-Speichertechnologie ist es möglich Steuersignalkombinationen in je einer Zeile dieser Speichermatrix abzulegen. Somit können Zeile für Zeile Maschinenzustande hart verdrahtet auf dem Prozessor hinterlegt werden. Das sogenannte Mikroprogramm. Die interne Logik ist eher zufällig optimiert. Daher der Begriff „Random Logic“.

Was sind Complex Instruction Set Computer (CISC)?

Durch Einführung von mnemonischen Kodierungen von Mikrobefehlen, welche von Mikrobefehls-Assemblern verarbeitet werden, sind weitaus komplexere Befehle möglich.

Gegenüberstellung der Architektur von CISC und RISC



Worin unterscheiden sich RISC und CISC besonders?

Eigenschaften	**CISC**	**RISC**
Register	Wenige (ca. 20)	Viele (bis zu 200)
Befehlssatz	Ca. 300 Befehle und mehr als 50 Befehlstypen	Nur rund 100 meist registerorientierte Befehle (außer LOAD / STORE)
Adressierungsarten	Ca. 12 verschiedene	Nur 3 bis 5 Arten Nur LOAD / STORE zum Speicher
Caches	Gemeinsame Caches Aber später auch getrennte	Getrennte Daten und Befehls-Caches nach Harvard
CPI	1 bis 20 Durchschnittlich 4	1 bei Basisoperationen im Schnitt 1,5
Befehlssteuerung	Mikrocode im Speicher Aber auch hartverdrahtet	Meistens hartverdrahtete Mikroprogramme ohne Mikroprogrammspeicher
Beispielprozessoren	Intel x86 AMD Cyrix	Sun UltraSparc PowerPC

Welche Befehlssatz-Architekturen kennen Sie?

Stack-Architektur

Diese Form benötigt keine Adressen für Operanden und ist somit eine Nulladreßmaschine. Quell und Ergebnisoperanden liegen auf einem Operanden-Stack. Vorteil dieser Architektur ist daher die Speicherplatzeinsparung durch die nicht notwendigen Adressen.

Akkumulator-Architektur

Um Verknüpfungsoperationen durchzuführen, liegt ein Operand in einem Register und ein Operand typischerweise im Hauptspeicher (Einadressmaschine) . Vorteil ist die einfache Implementierung, da nur ein internes Register benötigt wird. Nachteil ist aber die hohe Speicherlast.

Universalregister-Architektur

Ein Satz von gleichberechtigten Registern kann zum Ablegen von Daten genutzt werden. Deshalb sind im Op-Code mehrere Operanden anzugeben (Zwei-, Dreiadressmaschine etc.) Vorteil ist die freie Benutzbarkeit durch Compiler. Ausdrucksberechnungen können somit in beliebiger Reihenfolge erfolgen, was Pipelining möglich macht. Dazu kommt, daß die Speichertransferlast sinkt, die Geschwindigkeit steigt und Superskalartechniken sind effizient einsetzbar. Der Nachteil dieser Architektur sind die teilweise großen Registersets, welche bei jedem Kontextwechsel auszutauschen sind. Außerdem müssen die Operanden Adressiert werden, was zu langen Befehlen führt.

Welche Register-Architekturen gibt es?

Register-Register ohne Speicheradressen (Sparc,Mips)

Verknüpfungsoperationen verwenden nur Register. Nur in Lade- und Speicherbefehlen werden Adressen verwendet. (Load / Store – Architektur). Vorteil ist, dass die Verknüpfungen immer mit Registern geschehen und somit eine Befehlsdekodierung mit fester Länge möglich ist.

Vorteile:

Einheitliche Taktzyklen pro Befehl
Pipeline-Prinzip wird dadurch unterstützt

Nachteile:

Code wird größer, da Speichertransfers nur durch zusätzliche Befehle

Register-Speicher mit der Möglichkeit von Speicheradressen (Motorola 68000)

Vorteile:

Daten können auch im Speicher referenziert werden, ohne diese vorher Explizit laden zu müssen.

Nachteile:

Durch die variierenden Adressierungen variieren Befehlslänge und Taktzyklen pro Befehl, was äußerst negativ für Verfahren wie Pipelining ist.

Speicher-Speicher mit nur Speicheradressen (DEC-VAX)

Vorteile:

Der Programmierer braucht sich nicht um Register kümmern. Deshalb wird die Programmierung transparenter.

Nachteile:

Es entsteht ein hoher Speicherverkehr, was sich Nachteilig auf die Performance auswirkt. Falls doch Register erlaubt werden (Orthogonaler Befehlssatz / CISC), variieren auch hier Befehlslänge und Taktanzahl pro Befehl.

Orthogonale Befehlssätze sind solche, welche eine beliebige Kombination von Befehlscode, Adressierungsart und Datentyp zulassen.

Was ist Byte-Ordering und Word-Alignment?

Alle konventionellen Rechner sind Byte-Adressiert. D.h. das Worte (egal ob 8, 16 oder mehr Bit) bestehen aus einer Folge (aufsteigender) Bytes. Dabei gilt das erste Byte als die Adresse des Wortes. Nimmt die Wertigkeit mit aufsteigender Adresse zu, ist es das Litte-Endian-Format, umgekehrt das Big-Endian-Format.

Falls Worte so in den Speicher passen, das keine Verschiebungen auftreten, heißt der Speicher aligned. Prüfen kann man dies durch die Formel Adresse mod Wortlänge = 0?

2. Interrupts und DMA

Was sind Software Interrupts?

Software-Interrupts werden von Programmen mit Hilfe von speziellen Maschinenbefehlen aufgerufen. Dabei müssen diese nur eine Nummer für das benötigte Interrupt kennen. Über diese Nummer wird in der Interrupt-Vektor-Tabelle die Adresse des Interrupt-Unterprogrammes (ISR) referenziert und ausgeführt.

Was versteht man unter internen und externen Interrupts?

Externe Interrupts sind asynchron, wie nichtvektorisierte und vektorisierte Interrupts.
Interne sind synchron, wie Software Interrupts oder Exection-Traps (Reaktionen auf interne Fehler wie FPU-Errors oder Page-Faults).

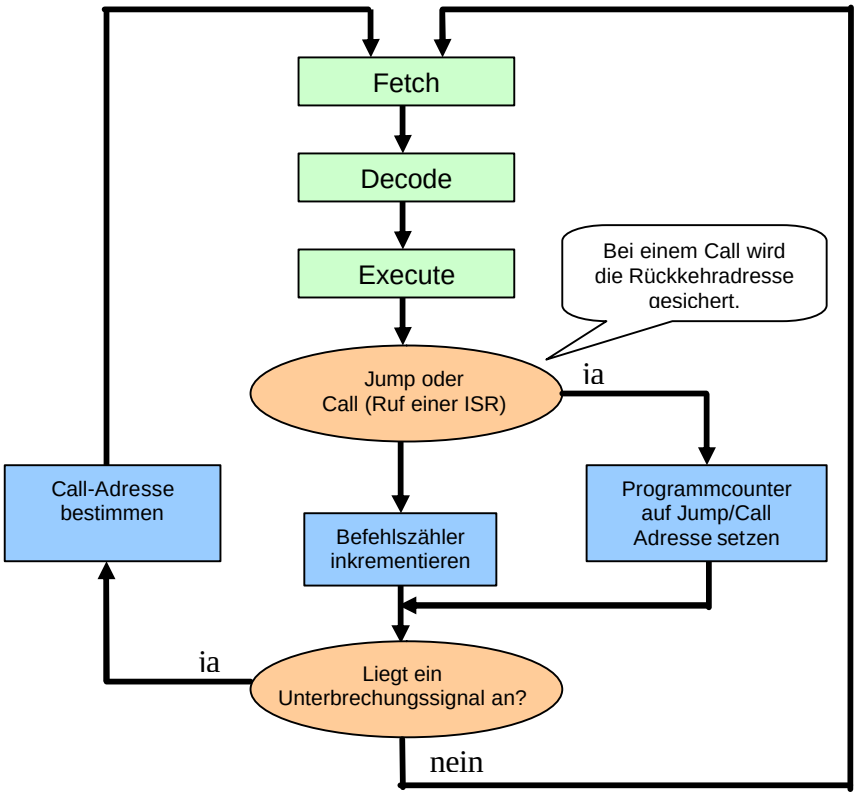
Was ist Polling?

Polling ist das zyklische Abfragen von einen oder mehreren E/A-Devices zur Feststellung der Kommunikationsbereitschaft bzw. zum Einholen von Kommunikationswünschen.

Vorteile des Pollings	Nachteile des Pollings
Einfach zu Implementieren	Hoher Programm-Overhead
Kommunikationsanforderungen erfolgen synchron zum Programmablauf	Die meisten Anfragen an die Geräte sind unnötig
	Je mehr Geräte am Bus hängen, um so mehr steigt Reaktionszeit.
	Priorisierung bei zeitgleichen Anfragen erfordert zusätzlichen Zeitaufwand

Aufgrund der vielen Nachteile sollte besser eine asynchrone Kommunikation mit den Geräten durch die Hardware unterstützt werden (Interrupts).

Erklären Sie das Interrupt Prinzip



Es kann auch über eine Art „hardware-gestütztes Polling“ über spezielle Interrupt-Signalleitungen eine Kommunikationsanforderung festgestellt werden. Dazu muss aber die Befehlsverarbeitungschleife um eine Unterbrechungsanfrage erweitert werden.

Man unterscheidet vektorisierten und nichtvektorisierten Interrupt. Bei nichtvektorisierten Interrupts wird dem Interruptsignal eine feste Adresse zugeordnet. Bei vektorisierten Interrupts wird dynamisch eine wahlfreie Adresse zugeordnet, welche durch die CPU über ein definiertes Protokoll vom Datenbus gelesen wird.

Warum wird DMA oft Interrupts vorgezogen?

Zwar befreien Interrupts die Prozessoren vom Warten auf E/A Ereignisse, aber vektorisierte Interrupts benötigen viele Taktzyklen zu ihrer Abarbeitung. Dieser Overhead steigt natürlich, um so weniger Datenmengen bei einer Interruptauslösung übertragen werden.

Interrupts werden erst nach der Befehlsabarbeitung erkannt und ausgeführt. Dies ist ein Problem bei Echtzeitanwendungen, da sich diese Verzögerung negativ auswirken kann. Außerdem kommt es durch Interrupts bei Instruction-Set-Parallismus oft zu Pipeline-Neustarts.

Die Lösung dieser Probleme wäre ein direkter Speicherzugriff eines Devices, da so der Prozessor komplett umgangen werden kann.

Wie kann DMA implementiert werden?

Zentral:
Ein zentraler DMA-Controller steht allen Geräten zu Verfügung.

Dezentral:
Jede E/A-Einheit hat ihren eigenen DMA-Controller implementiert und kann selbst Busmaster werden

Probleme bei DMA treten vor allem durch ihre Unabhängigkeit und die dadurch notwendigen Schutzmaßnahmen auf. Ein DMA-Controller wirkt wie ein weiterer Prozessor am Bus. Um Inkonsistenzen im Speicher zu vermeiden, muss ein DMA-Controller eng mit dem Speichermanagment des Systems zusammenarbeiten.

Was ist Memory-Mapped I/O?

Ein I/O Controller besteht aus einer Vielzahl von Registern, welche auf zwei Varianten adressiert werden kann:

Memory-Mapped I/O, um den konventionellen Adressraum I/O Devices zuzuordnen und Getrennten I/O Adressraum, bei dem auf einer speziellen Adressleitung die E/A-Adresse auf den Bus gelegt wird. (veraltete Variante)

3. Speicherschutz und Multitasking

Um unberechtigte Zugriffe, Datenaufrufe oder Systemprozedurecalls zu vermeiden und Task-Isolation zu gewährleisten, ist ein ausgeklügeltes Speicherschutzsystem notwendig.

Segmente zum schützen von Speicherbereichen

Segmente sind logische Speicherbereiche variabler Länge (Pages sind normalerweise gleich groß und ergeben zusammengesetzt ein Segment). In einem Segment ist wiederum eine Aufteilung in Code-, Daten- und Speichersegment zu finden. Jedes Segment definiert ein Objekt, welches eindeutig über einen Deskriptor mit Basisadresse, Zugriffsrechten und Limit beschieben wird. Auf Basis dieser Segmente arbeitet die komplette Speicherverwaltung eines Rechners-

Wie wird auf Segmente zugegriffen?

Segmente werden über eine Deskriptortabelle indiziert. Die Tabellen enthalten Pointer auf die Speicherbereiche der jeweiligen Segmente.

Was ist das besondere am segmentierten Adreßraum?

Adressen auf Basis von Segmenten sind im unterschied zu linearen Adressen zweidimensional. Sie bestehen aus Segment und Offset.

Berechnet werden sie durch einfache Addition von Segment und Offset. Vor der Addition ist das Segment um 4 Stellen nach links zu verschieben.

0002 : 000F berechnet sich also aus 0020 + 000F = 0001F

Was sind die Nachteile des Realmodes?

- Begrenzung eines Segments auf maximal 64 KB, da Offsetadresse nur 16 Bit groß ist
- Es nur das erste MByte durch das Betriebssystem adressierbar
- kein Schutz des Speichers vor anderen Programmen
- Einträge aus der Interruptvektor-Tabelle sind leicht veränderbar
- nur ein Programm kann ausgeführt werden (nur preemtives Multitasking)

Was hat Multitasking mit Protected Mode zu tun?

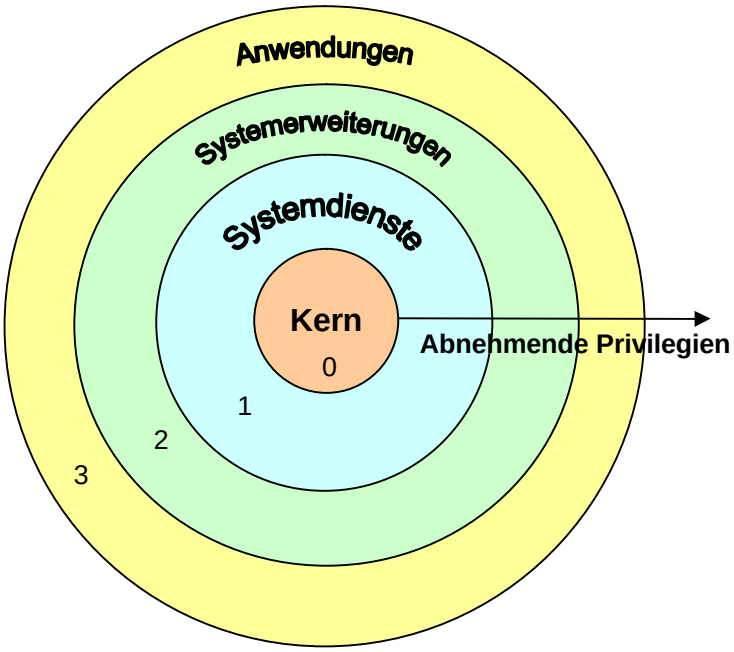
Multitasking kann nur durch Protected Mode arbeiten. Er ist sozusagen Grundlage für alle multitaskingfähigen Betriebssysteme. Insbesondere geht es um

- gegenseitigen Schutz der laufenden Tasks
- Taskwechselunterstützung durch das Betriebssystem
- Privilegierungsmechanismen
- Betriebssystemfunktionen zur Verwaltung von virtuellen Speicher
- Getrennte Stacks für Parameterübergabe
- Lösung des „Trojanischen Pferd“ Problems

Was sind Privilegebenen?

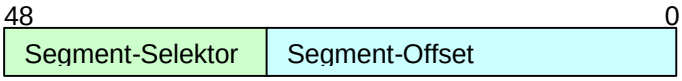
Im Protected Mode werden Anwendungen und Betriebssystem strikt getrennt. Es gibt vier Privilegstufen (null bis drei), welche über die Ausführung verschiedener Maschinensprachebefehle entscheiden.

Befehle der Ebene Null sind z.B. das Laden der globalen Deskriptorentabelle oder des Maschinenstatuswortes.



Aus welchen zwei Teilen besteht eine Virtuelle Adresse?

Eine virtuelle Adresse beinhaltet den Segmentselektor, welcher auf einen Eintrag in der Deskriptortabelle zeigt. Das Segment Offset zeigt auf die dazugehörige Adresse in dem selektierten Segment.



Aus welchen drei Teilen setzt sich ein Segmentselektor zusammen?

Aus dem Index, der den Eintrag in der Deskriptortabelle referenziert, dem Table Indicator, welcher über globalem oder lokalem Adressraum entscheidet und den Privelege Level.

TI - Table Indicator

0 = GDT (Global Deskriptor Table für den globalen Adreßraum)
1 = LDT (Local Deskriptor Table für den lokalen Adreßraum)

RPL - Requestor's Privilege Level

Privilegstufe des Segments, auf welches der Selektor verweist

Was ist ein Deskriptor?

Deskriptoren sind Abbildungen zwischen der virtuellen bzw. logischen Adresse (Segmentselektor:Offset) und der linearen Adresse (Basisadresse und Offset). Aus der linearen Adresse wird dann die physikalische Adresse berechnet.(bei i286 war die lineare Adresse noch gleich der physikalischen Adresse, da es noch keine Paging-Einheit gab)

Was steht alles in so einem Eintrag in der Deskriptortabelle?

Die „normalen“ Deskriptoren, welche einen normalen Adressraum (Daten-, Code- oder Stacksegment) beschreiben, enthalten

- die Basisadresse des Segmentes im Speicher
- die Zugriffsrechte
- die Länge des Segmentes

Eine andere Klasse von Deskriptoren sind System-Segment-Deskriptoren und zur Ablaufsteuerung notwendige Deskriptoren. Erstere definieren Einsprungpunkte in spezielle System-Unterroutinen oder Gates. Letztere sind Deskriptoren für Task-State-Segmente oder Local-Deskriptor-Table.

Aktiv sind aber immer nur eine globale, eine lokale und eine Interrupt-Beschreibertabelle

Was ist ein Gate?

Gates sind spezielle Eintritts-Deskriptoren in Segmente höherer Privilegstufe. (Interrupt- oder Trap-Gate-Deskriptoren)

Worin unterscheiden sich GDT und LDT?

Der Global Descriptor Table einhält Segmente des globalen Adressraums, welcher für alle Tasks zur Verfügung steht. Dagegen sind mit Local Descriptor Table allokierte Segmente nur von den Host-Tasks selbst adressierbar. (privater Adressraum)

Was unterscheidet Real-Mode und Protected-Mode?

Im Real-Mode gibt es keine Deskriptoren und somit ist auch kein Segmentschutz möglich.
Die Basisadresse berechnet sich einfach aus dem Segment-Register, welches maximal 1 MByte adressieren kann, da es nur 20 Bit breit ist. Im Protected-Mode werden die Basisadressen mittels Deskriptoren bestimmt.

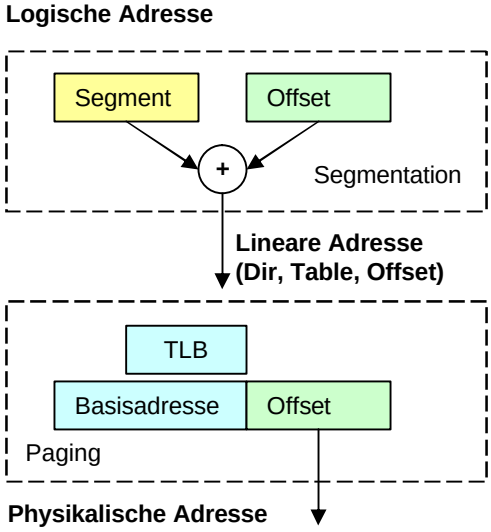
Auf Grund dieser Unterschiede sind folgende Merkmale für den Protected-Mode Signifikant:

- Virtuelle Speicherverwaltung
- Speicherschutzmechanismen durch Segmentation
- Paging möglich
- Echtes Multitasking
- I/O-Privilegierung und privilegierte Befehle

Was ist Paging und wie funktioniert es?

Paging wird ab i386 vom Prozessor unterstützt und ist nichts weiter als eine Einteilung des Speichers in gleichgroße Seiten. Vorteil des virtuellen Speichers, welcher durch Mapping / Paging erst möglich ist, sind für Anwendungen theoretisch unendlich große Arbeitsspeicher. Grund dafür ist, dass der Tertiärspeicher als Zwischenspeicher für schlafende oder temporär nicht notwendige Seiten ausgenutzt wird. Es gibt ausgeklügelte Seitenerstetzungsalgorithmen, welche das Austauschen von Seiten übernehmen. Ein weiteres Problem was beim Paging gelöst werden muss, ist die eventuell entstehende Inkonsistenz. Dieses Problem wird wie bekanntermaßen üblich durch Dirty-Bits in den Pages gelöst.

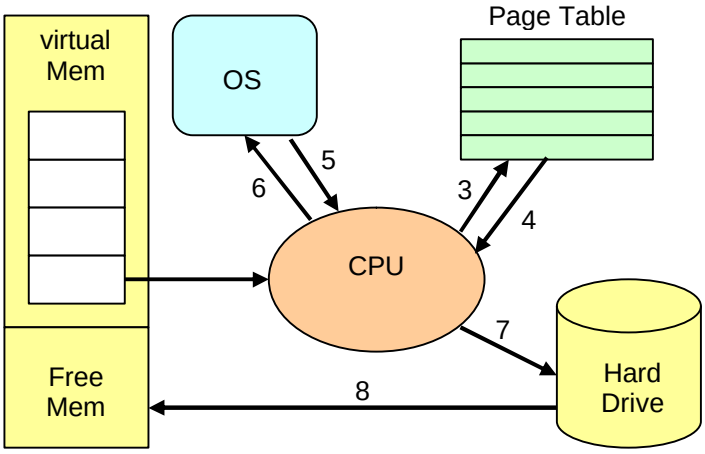
Verwirrend ist anfangs der Zusammenhang von Segmentierung und Paging. Letztendlich laufen beide Technologien gleichzeitig auf einem modernen System und ergänzen sich gegenseitig. Paging ist hinter den Segmentierungsvorgang geschalten, um Transparenz zu gewährleisten. Die durch die Segmentierung berechnete bzw. übergebene lineare Adresse entspricht ohne Paging der physikalischen. Falls Paging aktiv ist, muss noch etwas mehr getan werden….



Die Umsetzung von Linearer in Physikalischer Adresse hängt vom verwendeten Paging ab. Normalerweise wird über die ersten Bits die Page-Table referenziert und über die folgenden der Pagetable-Eintrag, aus dem die Basisadresse geholt wird. Der Offset wird normalerweise beibehalten.

Beschreiben Sie was bei einem Page-Fault intern alles abläuft?

1. Während Abarbeitung einer Befehlssequenz erfolgen mehrere Seitenzugriffe
2. Es erfolgt ein Zugriff auf eine Seite.
3. Prozessor prüft die Seite (ist sie im Speicher?).
4. Seite gibt Page Not Present State zurück (d.h. Seite nicht im Speicher)
5. CPU löst Page Fault Exception aus (Siehe System-Aufruf-Deskriptoren)
6. Betriebssystem gibt in Auftrag die Seite von Platte zu holen
7. Prozessor aktiviert Festplattenhardware und positioniert Leseköpfe
8. Seite wird über DMA-Transfer von Disk-To-free Memory übertragen
9. Betriebssystem aktualisiert Pagetable einschließlich des TLB (flush TLB)
10. Betriebssystem startet den unterbrochenen Befehl neu



Nennen Sie Vorteile und Nachteile des Pagings gegenüber Segmentation-Only!

- Performanceerhöhung eines Multitasking-Betriebssystems
- Verwaltung der Swap-Datei wird durch die Verwendung konstanter Speicherblöcke einfacher
- nur die 4-KByte werden eingelagert, die tatsächlich benötigt werden und nicht das gesamte Segment

Nachteile:

- Ausführung verzögert sich, weil die Adresse erst dekodiert werden muß
- bei Zugriff auf eine Seite/Page evtl. erst Einlagerung dieser vom Sekundärspeicher notwendig (Present-Bit)

Wie kann man die Adressdekodierung beim Paging umgehen?

Durch Translation Lookaside Buffer. Ein TLB ist ein assoziativer Vierwege-Cache, welcher die 32 Page-Table-Einträge aufnimmt, auf die der Prozessor zuletzt zugegriffen hat (LRU-Strategie). Ein TLB Eintrag besteht aus drei logischen Blöcken:

- Datenblock mit Page-Attributen und physikalische Basisadresse einer Page
- Tagblock enthält die oberen 17 Bit einer linearen Adresse und Schutz-Bits
- LRU-Block (Least Recently Used) zeigt letzten Zugriff an

Page- und Segmentschutz

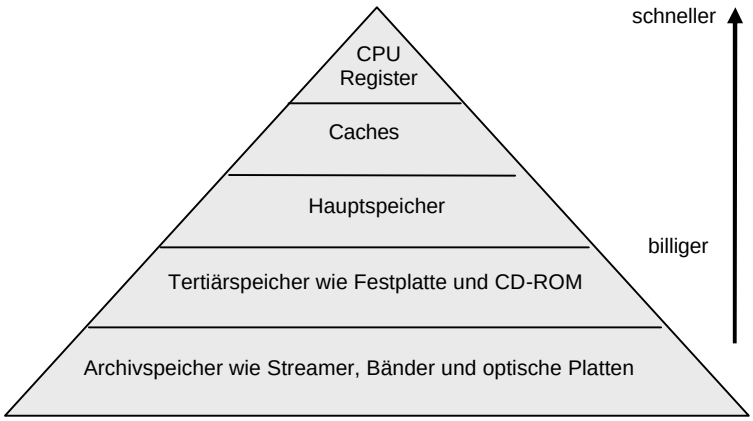
Zuerst wirkt der Segmentschutz und danach Pageschutz. Pageschutz ist nur 4-stufig. Die inneren drei Privilegebenen sind beim Paging als Supervisor-Code geschützt. Die äußere Ebene ist User-Code. Ein Zugriffsversuch einer User-Page auf eine Supervisor-Page löst eine Exception aus.

Was unterscheidet kooperatives und preemtives Multitasking?

Beim kooperativen Multitasking entscheiden die Tasks selbst über die Umschaltung der Prozessorleistung. (Naives und Gutgläubiges Verfahren, daß an die Vernunft aller Tasks und somit aller Programmierer glaubt ☺)

Preemptives Multitasking ist echtes Multitasking. Ein externer Timer steuert die Umschaltung der Tasks. Die Tasks können somit keinen Einfluss auf die Betriebsmittelumschaltung nehmen.

4. Speicherhierarchie und Caches



Was bedeutet die Eigenschaft Lokalität?

Aus programmtechnischer Sicht wiederholen sich oft Befehle und ganze Programmteile. Somit werden Daten oft wiederholt angefordert. Es gibt nun zwei Arten von Lokalität:

Zeitliche Lokalität:

Auf ein gerade zugegriffenes Datum wird sicher bald wieder zugegriffen.

Räumliche Lokalität:

Auf Daten, deren Adressen benachbart sind, wird mit hoher Wahrscheinlichkeit auch zugegriffen.

Anzumerken ist, daß Datenzugriffe eine geringere Lokalität zeigen als Befehlszugriffe.

Welche Cache-Arten kennen Sie?

Ein Cache-Eintrag besteht aus einem Tag (Identifikator) und den Daten. Die Implementierung unterscheidet sich auf verschiedene Arten.

Vollassoziativer Cache

Das Tag Feld ist hier die assoziierende Adresse des Datums im Speicher. Die Hardware ist bei vollassoziativen Caches aufwendig, da diese bei einem Cache Zugriff alle Tags gleichzeitig mit der anliegenden Adresse vergleicht. Dies ist zwar extrem schnell, aber sehr teuer. Außerdem wird er sehr langsam wenn die Anzahl der Cachezeilen hinreichend groß wird.

Direct-Mapped-Cache (einfach assoziativer Cache)

Beim Direct-Mapped-Cache entscheidet eine Map-Funktion, welche Zeile im Cache mit der anliegenden Adresse referenziert wird. Oft wird eine Funktion wie (A mod Cachesize / Zeilengröße) zur Berechnung der Cachezeile aus der anliegenden Adresse benutzt, da bei diesem Verfahren dann nur (A / Cachesize) als Tag in jeder Cachezeile gespeichert werden muss. Vorteil dieser Variante ist die einfache, kostengünstige Integration und die hohe Geschwindigkeit. Leider neigt ein Direct-Mapped-Cache zu vielen Konflikten (ähnlich den Kollisionen bei Hash-Tables), welche zusätzliche Cache-Misses bildet, da mehrere Adressen auf die gleiche Cachezeile verweisen.

n-Wege-Satz Cache (Satzassoziativer Cache)

Diese Variante ist nichts anderes als eine Implementation mehrerer parallel verknüpfter Direct-Mapped-Caches. Die Arbeitsweise ist die gleiche, nur das die Map-Funktion nicht nur auf eine Zeile im Speicher zeigt, sondern auf die n. Die Hardware des Caches vergleicht alle n Tags gleichzeitig, mit dem anliegenden Index. Ist eine der Tags gleich dem Index, ist dies ein Cache-Hit. Diese Technik reduziert die hohe Anfälligkeit von Direct-Mapped-Caches für Konflikte, benötigt aber mehr Chipfläche.

Welche Schreibstrategien für Caches gibt es?

Concurrent Write-Back

Bei einfachen Write-Back-Caches muss die CPU im Falle eines Cache-Misses warten, bis die neue Cache-Line aus dem Speicher geholt wurde. Um diese Wartezeit im Mittel zu eliminieren, wird die alte Zeile zunächst in einen Writebuffer zwischengespeichert und später, parallel zu nachfolgenden Cache-Referenzen in den Hauptspeicher übernommen.

Sonderform: Buffered Line Refill

Write-Back-Strategie

Ein zu lesendes Datum wird entweder bei einem Hit aus dem Cache gelesen oder im Falle eines Misses, aus dem Hauptspeicher geholt und parallel in den Cache eingetragen. Im Falle der Aktualisierung, muss erst das Dirty-Bit der zu überschreibenden Cache-Line geprüft werden, um diese gegebenenfalls in den Hauptspeicher zurückzuschreiben. (Write-Back)

Vorteil dieser Strategie ist das bei Hits kein Hauptspeicherverkehr oder Busbelastung auftritt. Alle Operationen können schnell innerhalb der Working-Sets mit Cache-Speed erfolgen. Somit arbeitet die CPU ungebremst. Problematisch wird dies, wenn mehrere Bus-Master am Bus hängen. Um Inkonsistenzen zu vermeiden sind dann spezielle Synchronisationsprotololle wie MESI notwendig.

Write-Throug-Strategie

Write-Through arbeitet im Prinzip wie Write-Back, nur das Schreibdaten in jedem Fall in den Cache UND in den Hauptspeicher geschrieben werden. Genau aus diesem Grund ist kein Rückschreiben eines Dirty-Datums notwendig, da es eh zu keinen Inkonsistenzen zwischen RAM und Cache kommen kann. Nachteil ist aber, dass nur bei Leseoperationen ein Geschwindigkeitsvorteil erzielt werden kann.

Sonderform: Buffered Write-Through

Zusammenfassung Caches

Write-Back wird üblicherweise mit Write-Allocate kombiniert. Beim Write Allocate (fetch-on-write): wird ein Block gelesen und in Cache gespeichert.

Beim No-write-allocate (write-around) wird der Block in der unteren Ebene der Speicherhierarchie modifiziert und nicht nicht im Cache geladen. No-write-allocate wird deshalb meist bei Write-through verwendet.

Was ist der Unterschied zwischen einen logischen und einen physischen Cache?

Physische Caches liegen vor der MMU und speichern somit nur physikalische Adressen. Ein logischer Cache liegt zwischen CPU und MMU und speichert logische Adressen. Vorteil von logischen Caches ist daher, dass die Adressumrechnung bei einem Hit entfällt. Ein großer Nachteil sind aber die Synonym-Probleme bei Multiprozessorsystemen. Des weiteren wird bei Taskwechsel ein Cache-Flush notwendig.

Multi-Level-Caches und Split-Caches

Durch Hintereinanderlegen von verschiedenen Caches kann ein gleitender Übergang zu immer größeren und langsameren Speichern erreicht werden. First Level Caches sind meist n-Wege-Satzassoziativ und folgende Direct-Mapped.

Split-Caches trennen Code und Daten und sind somit viel flexibler und besser an das Zugriffsverhalten in Bezug auf Strategie oder Assoziativität zu optimieren. Dabei unterscheidet man eine Havard-Architektur von der multiplexed Havard-Architektur (von Neumann Prinzip). Die reine Harvard trennt nicht nur Cache sondern auch den Hauptspeicher in Daten und Codebereich. Bei von Neumann liegen Daten und Code zusammen im Hauptspeicher und werden nur im Cache getrennt.

Durch Trennung von Code und Daten verdoppelt sich die Bandbreite, da zeitgleich zugegriffen werden kann.

Was geschieht wenn kein Platz mehr im Cache vorhanden ist?

Es muss eine Cache-Line ausgewählt werden, die mit den neuen benötigten Daten überschrieben werden kann. Die Auswahl erfolgt meistens mit LRU – Last Recently Used. D.h. die am längsten nicht genutzte Cache-Line fliegt raus.

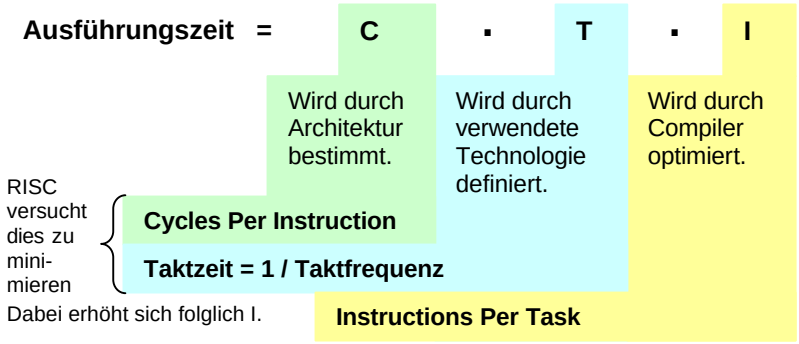
Was ist ein Burst-Cache?

Burst Caches schreiben nicht nur eine Zeile in den Speicher zurück, sondern gleich mehrere, um die Bandbreite auszunutzen und somit Zeit zu sparen.

5. Risc

Wie berechnet sich die Prozessorleistung?

Die Prozessorleistung ist umgekehrt proportional zur Ausführungszeit eines Algorithmus und wird aus folgenden drei Parametern ermittelt:



Worum geht es im Besonderen bei einer Risc-Architektur?

RISC Architekturen sind darauf aus, die Cycles Per Instruction zu minimieren. Das heißt, es wird versucht alle Befehle mit so wenig wie Möglich Takten auszuführen.

Welche architektonischen Möglichkeiten gibt es zur Veringerung der CPI?

Piplining ergibt eine CPI > 1, andere Techniken wie Superskalarität und VLIW's haben Ausführungszeiten von kleiner als eins. Kombiniert ergeben beide Ansätze eine nahezuhe Ausführung von einem Befehl pro Takt. Desweiteren ermöglicht der kleine Befehlssatz eine festverdrahtete Steuereinheit, anstatt von Mikroprogrammen, welche höhere Taktzahlen pro Befehl mit sich bringen. Desweiteren muss bei einem Risc-Befehl nicht der Op-Code dekodiert werden, um herauszufinden, wie der Befehl zu entschlüsseln ist, da alle Befehle die gleiche Struktur besitzen.

Auf welche vier Merkmale wird beim RISC-Design-Entwurf besonders geachtet?

- einfache Maschinenbefehle und Adressierungsarten mit einheitlichen Befehlsformat
- große und universelle Registersätze, für schnelle Variablenverarbeitung und größere Optimiermöglichkeiten für Compiler
- Verzahnung von Compiler und Architektur zur Bereitstellung von optimierenden Compilern
- Optimierte VLSI-Chipfläche durch platzsparende Steuerwerke schafft mehr Platz für Optimiertechniken wie Pipelining,Branch-Prediction oder Superskalarität

Techniken zur Ablaufparallelisierung für RISC-Kerne

- Einzelne Befehlsphasen durch Pipelines
- Ganze Befehle durch Superskalartechnik und VLIW
- Parallelität von Kodefäden durch Multithreading (programmierte Parallelität) oder Multiskalarität (Hardwarethreaderkennung)
- Parallelität von Befehlen unabhängiger Algorithmen (Multiprozessorsysteme)

Load / Store Architektur und Lokalhalten von Daten

Da Speicherzugriffe in Pipelines starke Konflikte hervorrufen, gibt es bei RISC-Befehlssätzen nur eine einzige Möglichkeit mit LOAD bzw. STORE auf den Speicher zuzugreifen. So werden Registerzugriffe von Speicherzugriffen getrennt. Da Speicherzugriffe bekanntermaßen immer sehr viel Zeit kosten, versucht man diese so weit wie möglich zu vermeiden. Dies Erreicht man durch Lokalhalten von Daten, bzw. das Arbeiten auf den Registern.

Was sind Registerfenster?

Registerfenster sollen das Lokalhalten von Daten unterstützen. Typische RISC Prozessoren wie die Berkeley RISC besitzen weit über 100 Register, von denen aber immer nur 32 für sichtbar sind:

- $R_0 \ldots R_9$ globale Register
- $R_{10} \ldots R_{15}$ Ausgaberegister
- $R_{16} \ldots R_{25}$ lokale Register
- $R_{26} \ldots R_{31}$ Eingaberegister

Die Idee ist nun, daß die ersten 10 Register von allen Prozeduren gesehen werden. Die Restlichen von R10 bis R31 sind jeweils nur einer Prozedur zugeordnet. Falls nun eine Prozedur eine andere aufruft, wird nur das „Fenster“ auf einen freien Registerbereich umgeschaltet. So müssen die Register nicht neu aus dem Speicher geladen werden und es wird dadurch viel Zeit gespart. Normalerweise überlappen sich die einzelnen Fenster um einige Register, um somit gleich eine effiziente Möglichkeit der Parameterweitergabe zu bieten.

Was passiert wenn alle Registerfenster voll sind?

Bei unserem Beispiel mit 138 Registern sind nach sieben Prozeduraufrufen alle Register gefüllt. Um ein Überlaufen zu vermeiden, wird das Register als Ringregister organisiert. Sind alle Registerfenster voll, wird das Älteste in den Speicher ausgelagert, was von sogenannten Trap-Routinen erledigt wird.

Was sind Superpipelines?

Superpipes vereinen Arithmetisches und Befehlspipelining. Arithmetisches Pipelining ist sogenanntes Funktionspipelining, bei dem einzelne Phasen eines Befehles in einer Pipeline-Form organisiert werden.

Bei Instruction Pipelining wird die Abarbeitung eines gesamten Befehls in einer Pipeline organisiert.

Zusammenfassung Risc

- Einfachere Befehlssätze mit ca. 40-80 Befehlstypen
- Einfachere Steuerung durch die Hardware ohne Mikroprogramme
- Effizientere Pipelines durch gleichlange, eintaktige Stufen
- Befehle können meist in einem Takt ausgeführt werden
- Datenzugriffe nur durch Load und Store um Speicherzugriffe zu vermeiden
- Mehr Register und Optimierung des Befehlssatzes durch Compiler

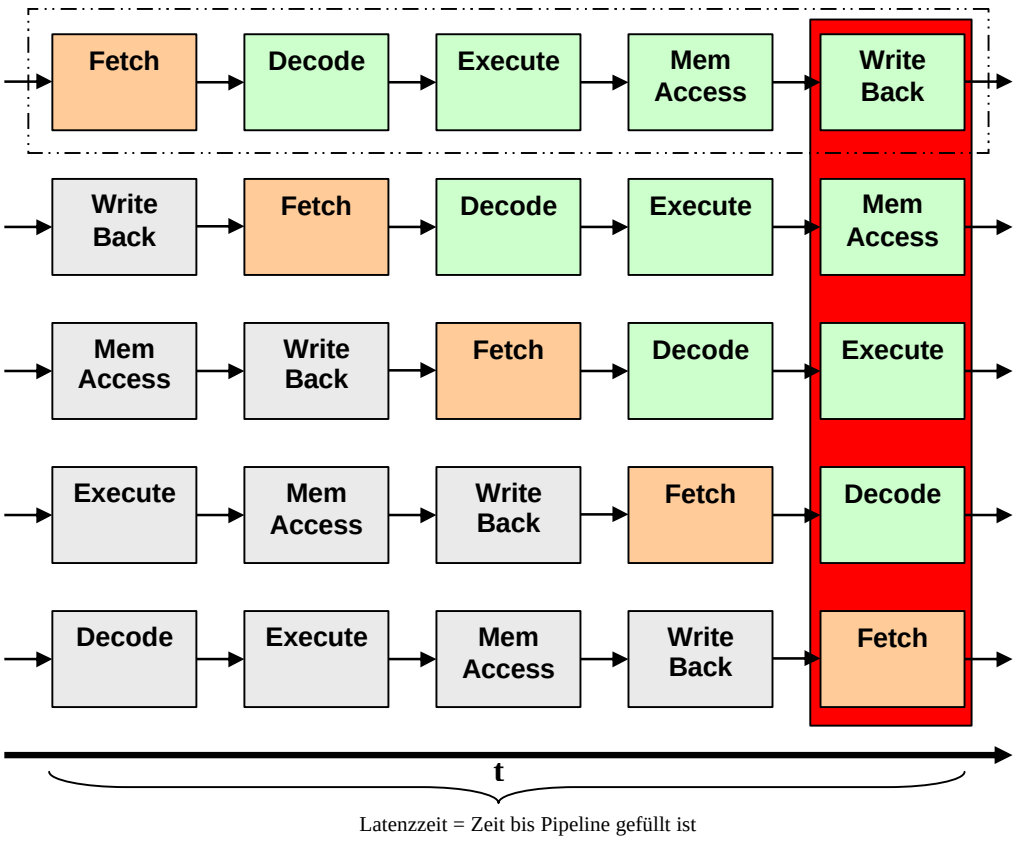
Typische Riscsysteme haben eine hartverdrahtete Steuereinheit und somit keinen Mikrocodespeicher. Der Pentium ist ein „**hybrid**"-System mit RISC Kern. Dabei werden komplexe CISC Befehle durch ein Mikroprogramm in RISC zerlegt und im Kern ausgeführt. Die einfachen Befehle werden direkt im RISC Kern in einem einzigen Datenzyklus ausgeführt. Alle wichtigen elementaren (Risc) Befehle werden direkt von Level 0 (der Hardware) ausgeführt und somit nicht via **Mikroinstruktionen** interpretiert. Dies ist ein Vorteil von reinen RISC Systemen, welche diese Interpretationsebene zwischen Hardware und ISA (Instruction Set Architecture) Ebene nicht durchlaufen müssen. Mikroinstruktionen steuern den Datenweg für einen Zyklus. Sie enthält alle notwendigen Bit-Belegungen für ALU, MEM, Register… etc. um einen Zyklus zu abarbeiten lassen zu können. Die Adresse der nächsten Mikroinstruktion wird ebenso mit codiert, wie die Art und Weise des Aufrufes. Die Mikroinstruktionen werden in einem **Steuerspeicher** gehalten, welcher das jeweilige **Mikroprogramm** enthält.
Der Steuerspeicher muss die Mikroinstruktionen nicht in geordneter oder sequentieller Folge enthalten, wie es beim Hauptspeicher der Fall ist. Es kann jede Instruktion einen Verweis auf die Nächste enthalten. Sprünge sind einfach möglich. Angewandt wird dies in Form von Opcodes, welche nichts anderes als Adressen auf Mikroinstruktionen im Steuerspeicher sind.

6. Pipelining

Pipelining soll es ermöglichen Befehle überlappt auszuführen. Dazu sind ein einheitliches Befehlsformat fester Länge Grundlage. Deshalb werden werden nur auf Register getätigt. Für Speicheroperationen wird die LOAD / STORE Philosophie verfolgt, um langsame Hauptspeicherzugriffe zu minimieren.

Der allgemeine Aufbau einer (fünfstufigen) Pipeline

Um Parallelität in der Befehlsausführungsphase zu erreichen, wird der Datenpfad so konstruiert, daß folgende (hier fünf) Phasen unabhänig voneinander arbeiten können. Nur so ist es möglich eine verzahnte Abarbeitung mehrerer Befehle zu erreichen.



Latenzzeit = Zeit bis Pipeline gefüllt ist

Somit wird nachdem eine Pipeline gefüllt ist, pro Takt ein Befehl fertig. (CPI = 1)

Welche Pipeline-Konflikte müssen behandelt werden?

Datenabhängigkeiten (Data Hazards)

Sind logische Abhängigkeiten, welche eine verzögerte Abarbeitung erfordern, weil z.B. ein Folgebefehl auf ein Ergebnis eines anderen Befehles warten muss.

Jump- / Branchverzögerungen (Control Hazards)

Bei Sprungbefehlen liegt oft das Sprungziel nach der Dekodieung noch nicht fest. Somit müssen Techniken eingesetzt werden um diese Wartezeiten zu minimieren. (Branch Prediction)

Ressourcenkonflikte (Structural Hazards)

Bei bestimmten Befehlskombinationen ist es unter Umständen möglich, daß ein Teilwerk seine Arbeit wiederholen muss. Solche Ressourcenkonflikte treten dann auf, wenn nicht jeder Teilphase völlig unabhängige Teilwerke zugeordnet sind.

Ein Beispiel ist z.B. ein zeitgleicher Lesezugriff eines LOAD/STORE Befehles, welcher sich zwangsweise mit einem eventuellen MEM ACCESS eines anderen Befehles überschneidet.

Abhilfe können hier Befehlspuffer oder getrennte Code- und Datencaches schaffen.

Data Hazards

RAW, WAW und WAR-Konflikte sind Datenabnhänigkeiten, welche in Pipelines auftreten können. Dabei ist das RAW-Problem für Pipelines typisch. WAR Konflikte treten eher bei Out-Of-Order Execution auf.

Um Read-After-Write Konflikte aufzulösen, gibt es verschiedene Ansätze wie Softwarelösungen (Compileroptimierung), Scoreboarding (zentrale Steuerlogik) und Forwarding (zusätzlicher Datenpfad).

Was ist Forwarding?

Beim Forwarding wird ein Bypass eingerichtet, welcher einer Ergebniss einer Operation schon zur Verfügung stellt, bevor es überhaupt in ein Register geschrieben wurde.

Aber trotz Load-Forwarding hat ein Ladebefehl eine Verzögerung, welche nicht gänzlich eliminiert werden kann. In diesem Fall kann die Delayed-Load Technik oder auch eine Befehlsumordnung Anhilfe schaffen.

Was ist die Delayed Load-Technik?

Bei der Delayed Load-Technik wird die Verzögerung nach einem LOAD Befehl als architektonisches Merkmal angesehn und den Compilerbauern offengelegt. Diese können nun durch Befehlsumordnungen versuchen, nach einem LOAD-Befehl einen Datenunabhängigen Befehl einzufügen, um den Slot zu nutzen.

Zusammenfassung Pipelining

Pipelines werden in allen modernen CPUs benutzt. Die UltraSparc2 hat neun und der P2 zwölf Stufen. Der Intel Pentium Itanium weißt eine 20 stufige Superpinepline (pipeline in der sich einzelne Stufen überlappen können) auf! Pipes werden heutzutage in Kombination mit der Superskalartechnik verwendet, um höchste Effizienz und Parallelverarbeitung gewährleisten zu können. Die fünf grundlegenden Stufen einer einfachen Pipeline sind **IF,ID,EX,MEM und WB.**

Takte T = Befehle + (Pipestufen – 1)

Folgende Abhängigkeiten verhindern, dass die CPI auf eins gehen.

Strucual Hazards bzw. Ressourcenkonflikte

IF und MEM wollen gleichzeitig auf Speicher lesend oder schreibend zugreifen. Geht nicht, außer bei **Dual-Port-RAM**, welche aber sehr teuer sind. Dieses Problem tritt aber bei modernen CPU's kaum noch auf, da eh intern eine **Havard**-ähnliche Architektur mit getrenntem Befehls– und Datencache gearbeitet wird.

Data Hazards bzw. Datenabhängigkeiten

Ein Folgebefehl wartet auf das **Writeback** der darüber liegenden Pipe, da er von diesem Befehl abhängig ist. Dies kann durch Nops bzw. Stalls ineffizient gelöst werden. Besser der Programmierer oder der Compiler löst diese Abhängigkeiten durch eine clevere Umordnung der Befehlsfolge auf. Es gibt aber noch eine andere Möglichkeit, welche aber hardwareseitig unterstützt werden muss.

Forwarding
Beim Forwarding werden Ergebnisse, sobald sie vorliegen an die nächste Stufe weitergereicht und nicht erst auf das Write Back gewartet. In anderen Worten: Das Ergebnis der **ALU** wird dieser sofort wieder eingespeist.

Control Hazards bzw. Sprungverzögerungen

Sprungergebnisse stehen erst in der Write Back Phase an. Moderne Prozessoren haben aber schon in der Fetch/Decode-Einheit eine Logik, welche die Zieladresse des Sprunges berechnet. Eine andere Möglichkeit ist die des spekulativen Ausführens. Hier tritt aber das Problem auf, dass viel Aufwand bei falscher Spekulation getrieben werden muss.

7. Branch Prediction

Control Hazards (Jump / Branch Problematik)

Sprungbefehle stellen einen Dorn im Auge einer jeden Pipeline dar, da diese besondere Vorkehrungen erfordern.

Da das Ziel eines Sprungbefehles oft erst festgestellt werden muss, liegt diese Adresse erst ab der MEM ACCESS Phase bereit. Somit kann das erneute Laden des Programmcounters auch erst in dieser Phase geschehen. So verzögert sich das Holen des nächsten Befehles um einige Takte.

Durch eine Optimierung der Pipeline kann zwar die stall-Phase verkleinert, aber nicht ausgeschlossen werden. (durch Verlegung des Sprungbedingungstests in die Decode-Phase)

Welche Methoden gibt es zur Reduzierung von Sprungverlusten?

- Predict Not Taken
- Predict-Taken
- Delayed-Branch
- Brach-Prediction (statisch oder dynamisch)

Wie funktioniert die Predict-Not-Taken bzw. Predict-Taken?

Hier wird nichts weiter gemacht als entweder alle Sprünge voreingestellt abzulehnen oder alle Sprünge ersteinmal ohne Gewähr duchzuführen.

Allgemeine Programmstatistiken sagen aus, dass mehr bedingte Sprünge ausgeführt als abgewiesen werden.

Wie funktioniert die Delayed-Branch Methode?

Hier wird ein sprungunabhängiger Befehl in den Delay Slot eingeschleust. Dies muss somit schon von den Compilerbauern berücksichtigt werden.

Um diese Bedingung zu Umgehen wird die „Cancelling Branches“-Technik eingesetzt. Im Mittel werden dann trotzdem die Branch-Verluste verringert. Durch ein zusätzliches Bit im Befehlscode gibt der Compiler die wahrscheinlichste Sprungrichtung an. Nun kann entsprechend dieser Annahme ein Befehl in den Delay Slot eingefügt werden, der nur gültig ist, wenn der Sprung richtig vorhergesagt war. Falls nicht wird der Delay-Slot-Befehl abgebrochen (gecancelt).

Dynamische Brach-Prediction

Um Wartezeiten durch bedingte Sprünge zu vermeiden, sollte das Sprungziel schon mit dem Ende der Fetch-Phase zur Verfügung stehen. Es gibt zwei Ansätze

- Sprungzielspeicher (branch-target-buffer = BTB)
- Sprungvorhersage-Puffer (Branch History Table = BHT)

Wie arbeitet eine Branch History Table?

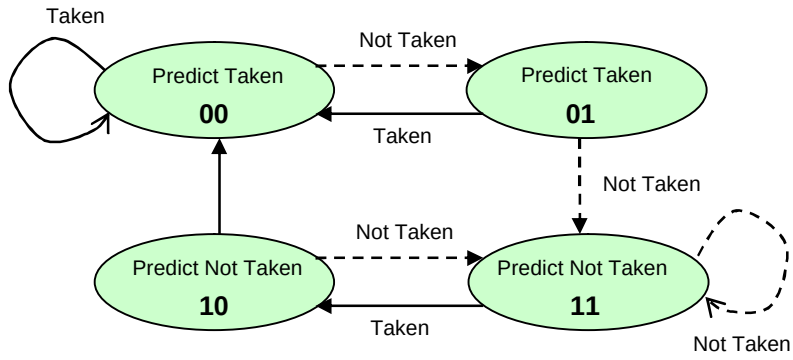
In dieser Tabelle wird im Grunde nur durch ein Bit (oder mehr) vermerkt, ob ein Sprung durchgeführt wurde oder nicht. Als Index der Tabelle dient der niederwertige Teil der Adresse des dazugehörigen Sprungbefehls. Nun kann die Pipeline in der Fetchphase nach einem eventuell vorhandenen Eintrag schauen und diesen als Entscheidungsgrundlage nehmen.

Welchen Nachteil hat die 1-Bit Sprungvorhersage?

Es wird nicht nur bei einem Schleifenaustritt der Sprung falsch vorhergesagt, sondern auch die erste Vorhersage bei erneuter Verwendung der Schleife.

Wie arbeitet die 2-Bit-Sprungvorhersage mit BHT?

Durch einen einfachen Zähler kann man den Nachteil der 1-Bit-Vorhersage minimieren. Hier wird die Vorhersage erst geändert, wenn sie zweimal falsch war. Es hat sich gezeigt, daß durch Zähler mit mehr als 2 Bit sich die Performance nicht weiter signifikant erhöhen läßt.



2-Bit Sprungvorhersage als Automat

Wie arbeitet der Branch-Target-Buffer?

Hier wird die Zieladresse eines gemachten Sprungs direkt gespeichert, um diese gegebenfalls ohne Verzögerung wiederzuverwenden. So kann bei einem Hit (Index stimmt mit Befehlsadresse überein) sofort der Instruction Counter mit der dazugehörigen Sprungadresse geladen werden).

Exeptions

Exeptions unterbrechen den Programmablauf Aufgrund verschiedenster Fehler oder Anforderungen, wie Softwareinterrupts, Page Faults oder anderen Verletzungen. Bei synchronen Exeptions treten die Fehler stehts an der gleichen Programmstelle auf. Asynchrone werden durch externe Geräte ausgelöst und können nach dem laufenden Befehl ausgeführt werden.

Was sind Precice Exeptions?

Sind Exeptions, welche garantieren, dass die Exeptions direkt nach oder während des Befehles ausgeführt werden und kein Folgebefehl vorher abgearbeitet wird.

Was ist der Unterschied zwischen echten und unechten Datenabhängigkeiten?

Echte Datenabhängigkeiten sind RAW-Konflikte, bei dem ein Befehl auf die Beendigung eines Anderen warten muss, da er das Ergebnis als Operand benötigt. Unechte Datenabhängigkeit sind Abhängigkeiten, welche nur durch Namensabhängigkeit entstehen.

Es gibt zwei Arten unechter Datenabhängigkeit:

Antidependence sind WAR-Konflikte, welche entstehen, wenn ein Folgebefehl auf ein Register schreiben möchte, das noch von einem Anderen benutzt wird.

Output Dependece sind WAW-Konflikte, welche entstehen, wenn mehrere Befehle auf ein und das selbe Register schreiben. Hier muss sichergestellt werden, daß die Schreibreihenfolge der der Befehle entspricht.

Beide Abhängigkeiten können durch Register Renaming vermindert werden!

Zusammenfassung der Sprungvorhersage

Sprungvorhersage ist extrem wichtig für Pipelining und Superskalarität, um **stalls** und Verzögerungen zu minimieren. Bei statischer Vorhersage werden Rückwärtssprünge meist erst durchgeführt und Vorwärtssprünge nicht. Wurde ein Sprung falsch vorhergesagt, muss die angefangene Instruktion rückgängig gemacht werden, was aufwendig ist. Deshalb gibt es ausgeklügelte Verfahren für die Branch Prediction…

Statische Sprungvorhersage

Es werden Compiler benutzt, die **spezielle Sprungbefehle** mitführen, welche ein Bit für die Sprungvorhersage enthalten. Da der Compiler ja weiß, wie oft eine Schleife durchlaufen wird, ist das sehr effizient. Dies muss aber architektonisch von der Hardware unterstützt werden. Des Weiteren ist kein Speicher für die **History Table** notwendig, was es kostengünstiger macht. Statische Verfahren erreichen eine Trefferrate von 65 bis 85%, was für moderne CPU's mit Superpipelines zu wenig ist. Dynamische Verfahren erreichen Trefferraten bei der Vorhersage von 98% und mehr!

Dynamische Sprungvorhersage

Es gibt zwei grundlegende Methoden. BHT und BTB. Die **Branch History Table (Branch Predicion Buffer)** ist ein Cache, in der alle bedingten Sprünge protokolliert werden. (bis zu mehereren Tausend) Einfachste Version enthält ein **Valid-Bit (Branch taken oder nicht)**, welches durch den niederwertigen Teil der Sprungadresse adressiert wird. Kompliziere Implementationen arbeiten nach dem n-Wege Prinzip. Durch Second Chance kann dieses Verfahren noch verbessert werden.
Der **Branch Target Buffer** speichert nicht nur die taken-Bits, sondern auch die Sprungzieladresse, um null Verluste bei wiederholtem Aufruf zu haben. Das setzt voraus, dass nur taken branches aufgenommen werden. Bei einem Hit in der BTB kann somit während der Fetch Phase der Program Counter überschrieben werden. Werden keine History Bits mitgeführt spricht man vom BTAB.

Second Chance
Nach Beenden einer Schleife wird ein Sprung logischerweise falsch vorhergesagt. Um zu vermeiden, dass nun fälschlicherweise das **Sprungbit** falsch gesetzt wird (da ja die gleiche Schleife noch mal durchlaufen werden kann), ändert man dieses erst nach der zweiten falschen Vorhersage. Leicht zu implementieren als Finite State Machine mit vier Zuständen. Nachteil der dynamischen Vorhersage ist die notwendige teuere und komplexere Hardware.

8. Superskalare Architekturen

Dynamic Scheduling

Static Scheduling nutzt lediglich Compilertechniken zur Separierung unabhängiger Befehle und ist sehr unflexibel. Beim Dynamic Scheduling wird durch Präsenz mehrerer paralleler Execution Units versucht, Ressourcen- und Datenkonflikte zu minimieren.

Out-Of-Order Execution

In-Order-Issue Pipes müssen, falls ein Befehle gestoppt wird, alle Folgebefehle warten. Durch eine zusätzliche Hardware, welche das Umordnen der Befehlsausführung zur Laufzeit vornimmt, kann dies verhindert werden. Möglichkeiten dafür sind Scoreboards oder Tomasulo.

Superskalar-Prozessoren und VLIW-Prozessoren als Multipe-Issue CPU's

Beide Techniken senden mehr als nur einen Befehl pro Taktzyklus aus und versuchen so, die CPI unter eins zu drücken. Moderne Prozessoren kombinieren beide Techniken.

Der größte Unterschied zwischen beiden Technologien besteht darin, daß Very Long Instruction Word basierende Systeme für verschiedene Prozessoren neu compiliert werden muss. Superskalare-Prozessoren sind dagegen gleich kompatibel.

Was heißt Multiple-Issu?

Ist das Mehrfach-Aussenden von Befehlen, welchen gewissen issue criteria unterliegen müssen.

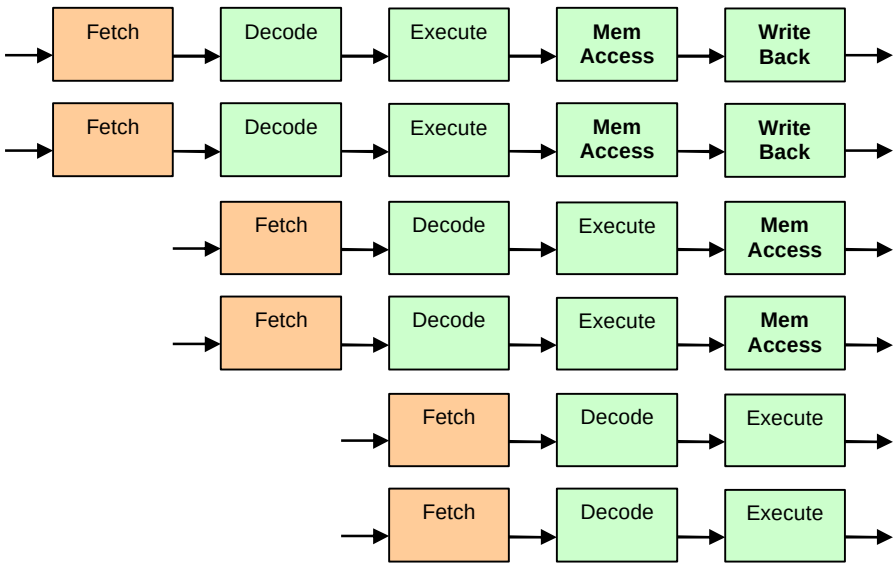


Abb: 2-fach superskalare fünfstufige Pipeline

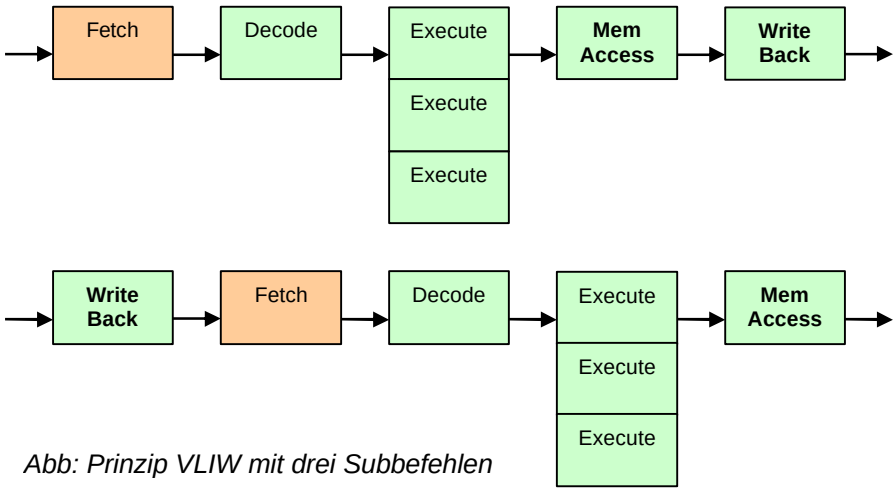


Abb: Prinzip VLIW mit drei Subbefehlen

Das Scoreboard

Ein Scoreboard ist eine zusätzliche Steuereinheit, welche die Verantwortung für das Befehlsaussenden und das Erkennen von Konflikten trägt. Das Scoreboard wählt aus einem Pool potentiel ausführbarer Befehle (Instruction Window) einen Satz von Befehlen aus. Ein Register wird als ungültig markiert, wenn die Dekodiereinheit erkannt hat, dass ein Befehl sein Ergebnis in dieses Register schreiben möchte. So wird verhindert, dass ein anderer Befehl dieses Register liest, solange es ungültig ist. Nachteil ist, daß keine Ressourcen- und auch keine Datenabhängigkeiten auftreten dürfen. Eine bessere Variante ist die Tomasulo-Methode, welche eine Auflösung von WAR- und WAW-Konflikten ermöglicht.

Die Tomasulo-Methode

Hauptidee sind hier sogenannte Reservation Stations, welche eine Art Zwischenpuffer für Operanden darstellen. Die Reservation Stations besitzen eine eindeutige ID, welche jedem Befehl mitgegeben wird. So kann die richtige Reihenfolge beibehalten werden. Wird ein Befehl ausgeführt, arbeitet dieser nicht auf den eigentlichen Registern, sondern auf den assoziierten Reservation Stations, was das Prinzip des schon erwähnten Register Renamings ist. Über einen Common Data Bus werden die Ergebnisse zu allen beteiligten Einheiten gebroadcaset.

Reservation Stations erkennen Hazards und können selbst entscheiden, wann sie den dazugehörigen Befehl ausführen. Nämlich erst dann, wenn alle Operanden vorliegen.

Um controll stalls komplett vermeiden zu können, wird die Tomasulo-Methode mit der spekulativen Befehlsausführung kombiniert.

Wie arbeitet die spekulative Befehlsausführung?

Sprungziel-Befehle werden schon ausgeführt, wenn das Ergebnis des Sprungtests noch gar nicht vorliegt. Somit muss es die Möglichkeit geben, bei falscher Vorhersage alle Änderungen zu Verwerfen.

Die Hauptsächliche Hardware-Erweiterung liegt in der Aufspaltung der Ergebnisschreibphase in eine Bereitstellungsphase mit Zwischenspeicherung im Reorder Buffer und eine Phase in der ein Befehl commited, d.h. gänzlich der Ausführung übergeben wird. Ein commit bedeutet, daß eine eventuelle Sprungvorhersage richtig war!

Somit stellt dies eine Kombination von out-of-order-execution via Tomasulo mit einem erzwungenen in-order-commit dar.

Der Reorder-Buffer stellt in der Welt des Tomasulo weitere Register zur Verfügung, welche auch die Funktion von Store Buffern übernehmen könnten, so daß dieser als Teil des Tomasulo nicht direkt mehr erkennbar wäre.

(Store Buffer enthalten Informationen darüber, welche Reservation Stations, welches Ergebnis erwartet)

Aus welchen Feldern setzt sich ein Reorder-Buffer-Eintrag zusammen?

- Befehlstyp (Branch, Store oder Registeroperaton)
- Zielfeld (Registernummer oder Speicheradresse)
- Datenfeld (Ergebnis der Operation)

Die Phasen des Tomasulo

Fetch und Decode	**Dispatch / Issue**	**Execution**	**Commit**
Holt Instruktionen in einen Befehlscache. Die **Decode-Unit** holt sich einen Teil der Befehle und versucht mehrere gleichzeitig zu **decodieren** (In-Order). Dabei wird versucht **Sprünge vorherzusagen**. Übergibt dekodierten Befehle in der richtigen Reihenfolge an die Dispatch (**Issue**) Unit	Holt dekodierte Befehle aus Befehlspuffer und übergibt sie InOrder an die **Reservation Stations der Execute Units, sobald alle Operanden verfügbar sind (Issue)**. Solange im **Reorder** Buffer Platz ist, reserviert sie ein Feld für diesen Befehl mit Hilfe des Tags und gibt dieses an die RS weiter . Wenn nicht wartet sie, bis ein Platz frei wird. (**Dispatch** Phase)	Führt Befehle auf **Schattenregistern** aus, um Data Hazards zu meiden. Nun werden Befehle in den Reorder-Buffer geschrieben und **Out-Of-Order** ausgeführt, solange es keine RAW – Konflikte gibt. Nach Beenden eines Befehls wird Ergebnis an alle RS gebroadcastet, so dass wartende Befehle fortfahren können. (**Write result**)	Die Commit/Completion oder Retire Einheit schreibt die Ergebnisse aus den Renaming Registern in die echten **ISA Register** zurück, nachdem sie geprüft hat, ob abhängige vorangehende Befehle ihre Ergebnisse geliefert haben und keine falsche Sprungvorhersage eingetreten war.

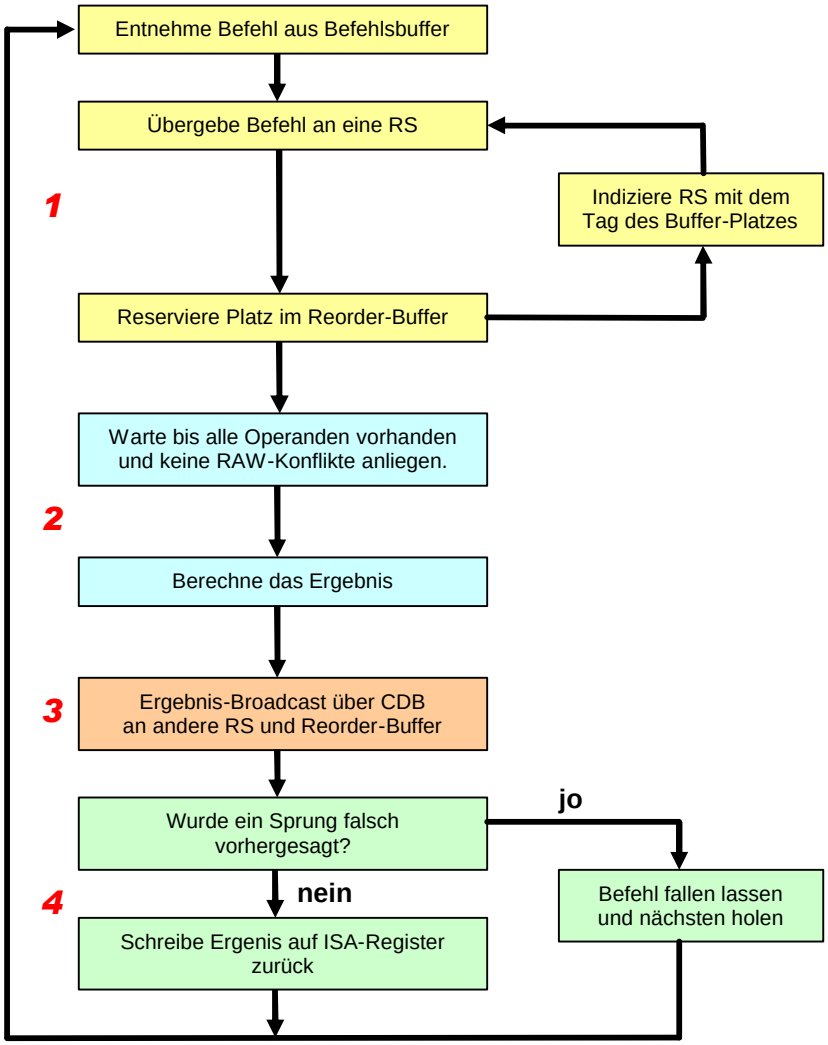
Erklären Sie die Phasen Issue, Execute, Write Result und Commit!

Die Issue-Phase entnimmt einen Befehl aus dem Befehlsbuffer und versucht diesen an eine freie Reservation Station zu übergeben. Dabei reserviert sie einen Platz im Reorder-Buffer und gibt den dazugehörigen Tag an die Reservation Station. So kann beim Ergebnis-Broadcasting das Ergebnis mit diesem Tag gekennzeichnet werden. (1)

Sind nun alle notwendigen Operanden vorhanden und keine weiteren RAW-Abhängigkeiten bestehen, kann der Befehl ausgeführt werden. (Execute-Phase). (2)

Ist das Ergebniss berechnet, wird es auf den CDB gelegt und in den Reorder-Buffer geschrieben (Write Result). (3)

Der Reorder-Buffer ist als Ringbuffer angelegt, bei dem die Reihenfolge der Befehle, durch die der erwarteten Ergebnisse (über das verliehene Tag) definiert wird. Nun werden in der Commit-Phase falsch vorhergesagte Sprünge zurückgesetzt und es wird bei dem richtigen Nachfolgebefehl fortgesetzt. Das Ergeniss wird in die ISA Register zurückgeschrieben. (4)



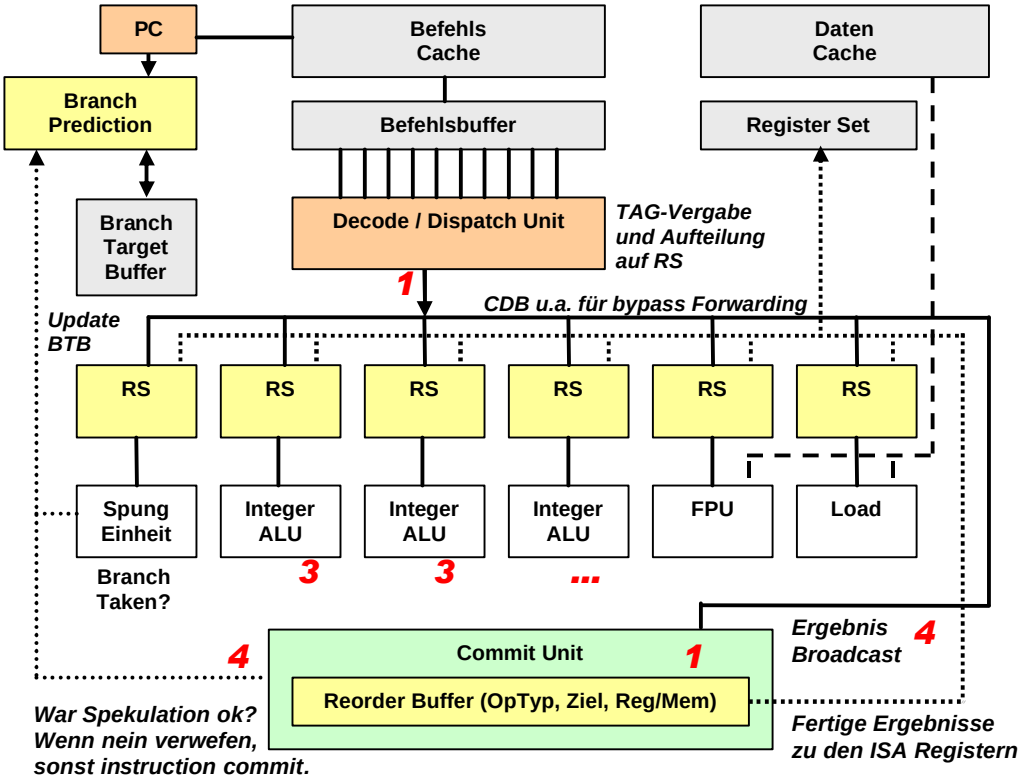


Abb.: Abstrahierter RISC-Kern eines Pentium Pro / Power PC

Muliple Issue und Pentium 2

Instruktionen werden in eine oder mehrere Mikrooperationen dekodiert und in eine Warteschlange gestellt. Sprungerkennung erfolgt erst statisch und dann dynamisch mit 4 History Bits. Dann werden Register im **Reorder-Buffer** (der P4 hat Platz für 40 µOps) zugewiesen. Dabei werden um Konflikte zu vermeiden **Renaming-Register** (Schattenregister) eingesetzt. In der Execute Phase werden Mikrooperationen aus dem Reorderbuffer entnommen und spekulativ ausgeführt.

Es wird ein komplexes **Scoreboard** verwendet, um aktive Operationen und Register zu verwalten. Falls mehrere Mikrooperationen bereit sind, wählt ein Algorithmus den nächst Wichtigsten aus (z.B. Sprünge).

Mikrooperationen deren Operanden alle verfügbar sind, werden in die **Reservation Stations** geschrieben, welche 20 solcher Einträge aufnehmen kann. Dort warten die Ops, bis eine entsprechende Ausführungseinheit frei wird.

Die **Execute-Units** haben fünf Ports, um Operationen auszuführen. Manche Ausführungseinheiten teilen sich einen Port (FPU,MMX).

Die **Retire (Completion)** Einheit sendet fertige Ergebnisse an die richtige Stelle. Der P2 unterstützt speculative execution. Falls begonnene Instruktionen nicht mehr benötigt werden, wird die Rollback Fähigkeit genutzt.

Der Pentium 4 nutzt einen Trace Cache für die letzten drei dekodierten Befehle. Im Hit Falle, bedient sich die Execution Unit direkt dieser Befehle, was vor allem bei Schleifen einen beträchtlichen Speedup bringt.

Die **UlraSparc** 2 ist dagegen eine reine Risc Maschine mit vierfachen statt 2facher Superskalarität. Durch das Risc Prinzip müssen hier Instruktionen nicht in mehrere

Mikrooperationen umgesetzt werden. Desweiteren unterstützt die meist gleich große Befehlslänge das Pipelining besser, als die z.T. komplexeren Cisc Befehle des Pentium.

Neu ist das so genannte **Folding** der **PicoJava** CPU, welche Instruktionen faltet, d.h. zusammenfasst und gleiche Instruktionen in einem Zyklus (auf verschiedenen Registern) abarbeiten kann. (5 x schneller als P2, obwohl auch CISC)

Zusammenfassung Superskalar-CPU's - dynamic scheduled pinelines

Da das Pipeline Konzept schnell an seine Grenzen gelangt ist, ist eine andere Methode notwendig. Es muss möglich sein, dass mehrere Befehle pro Takt beendet werden können und unabhängige Befehle, welche in Pipes wegen vorangehender Abhängigkeiten warten müssen, in der Ausführungsfolge **vorzuziehen**. Superskalare Prozessoren erkennen parallel verarbeitbare Befehle von selbst und benötigen somit keine speziell optimierenden Compiler, wie sie z.B. VLIW verlangen. Dafür sind sie weitaus komplexer. Um RAW, WAW und WAR Abhängigkeiten zu vermeiden, wird versucht Instruktionen anders anzuordnen. Dafür ist ein **Scoreboard** notwendig, das für jedes benutzte Register die Leseabhängigkeiten und Schreibabhängigkeiten zählt.

Der Tomasulo Algorithmus ist ein Verfahren, welches sich durchgesetzt hat, da es WAW und WAR Konflikte dynamisch auflösen kann. Dies ist so einfach möglich weil WAW und WAR keine echten Datenabhängigkeiten sind, sondern nur entstehen, wenn ein späterer Befehl ein Register wiederverwendet, ohne die darin enthaltenen Daten zu benötigen.

Hier tritt das Register Renaming oder „Arbeiten auf Schattenregistern" in Kraft. Würde es mehr Register geben, könnten diese Abhängigkeiten schon vom Compiler aufgelöst werden.

Mit **Register Renaming** werden intern versteckte Register benutzt, um Abhängigkeiten aufzulösen.

Spekulative Ausführung wird benutzt um weitere Lücken im Ablauf zu füllen. Hierbei werden Sprungziel-Befehle gegebenenfalls schon ausgeführt, wenn noch gar nicht klar ist, ob dieser überhaupt wahr werden. Dies wird meist mit dynamischen Scheduling nach Tomasulo kombiniert. Falsch vorhergesagte Befehle erreichen nie die Commit-Phase und werden verworfen. Dies ist möglich, da sich die Ereignis-Schreibphase in **Bypassing** (Zwischenspeicherung im Reorder-Buffer) und dem **Rückschreiben** auf die echten ISA Register aufteilen lässt.

Wie können RAW-Konflikte gelöst werden?

Durch Softwarelösungen, Scoreboarding und Forwarding.

Zusammenfassung VLIW

Der IA-64-Befehlssatz von Intel wird auf „**Very Long Instruction Word**s" basieren. Es werden drei Instruktionen ein einen fetten 128 Bit Befehl gepackt. Hier besteht nun die Möglichkeit, explizit festzulegen, welche Befehle parallel abgearbeitet werden sollen bzw. können. So eröffnen sich völlig neue Optimierungsmöglichkeiten im Compilerbau. Man nennt dieses Prinzip **EPIC**. Der Transmeta verwendet auch VLIW. Hier muss aber der Compiler nicht die Optimierungen vornehmen, da der Transmeta um der Core eine Morphing Softwareebene hat, welche die Aufteilung in parallel abarbeitbare Befehle ausführt. Dadurch, daß nun explizit gesagt wird, welche Instruktionen parallel ausführbar sind, ist nun nicht mehr so viel Chipfläche zum Auflösen von Hazards notwendig und kann z.B. für mehr Register verwendet werden.

9. Parallelrechner

Wie kann man Parallelrechner klassifizieren?

In SIMD (Single Instruction Multiple Data) wie Array-Rechner und MIMD (Muliple Instruction Multiple Data), wie

- Symetric Multiprocessor Systems
- Network Of Workstations (NOW)
- Cluster Of Workstations (COW)
- Massive Parallel Processor System (MPP)
- Metacomputer (Cluster von vernetzten homogenen Parallelrechnern)

Was ist die Klassifikation nach Flynn?

Instruktionsströme	Datenströme	Bezeichner	Typische Anwendungsfälle
1	1	SISD	Von- Neumann Rechner
1	N	SIMD	Vektorrechner
N	1	MISD	(noch) nicht existent
N	N	MIMD	Mehrprozessor und -rechner

Worin liegt der signifikante Unterschied zwischen SIMD und MIMD Systemen?

SIMD Systeme benutzen zwar getrennte Datenspeicher, bekommen ihre Befehle aber aus einem globalem Befehlsspeicher. Bei MIMD Systemen hat jede Einheit seinen eigenen Befehlsspeicher.

Shared Memory Sytem (Mehrprozessorsystem)

Alle CPU's teilen sich gemeinsamen Speicher. Deshalb ist Kommunikation hier besonders einfach. Ist jede CPU gleichberechtigt und austauschbar, wird das System SMP genannt. Nachteile sind, daß dieses Prinzip nur bei kleinen Prozessorzahlen überschaubar bleibt und das das Prinzip nicht skaliert.

Distributet Memory (Mehrrechnersystem)

Diese Systeme definieren sich dadurch, dass jede CPU ihren eigenen Speicher besitzt. Kommunikation erfolgt hier über ein Verbindungsnetz, was Mehrrechnersysteme weitaus komplizierter macht, als Multiprozessorsysteme.
Eine Möglichkeit des gemeinsamen Speichers ist Distributed Shared Memory. Hier stellt das Betriebssystem den systemweiten Adressraum als gemeinsamen Speicher zur Verfügung. Der Vorteil ist, dass LOAD und STORE Befehle auf fremde Speicher möglich sind. Es wird sozusagen die Seitenersetzung nicht von der Festplatte, sondern vom entfernten Rechner vollzogen. Kommunikation erfolgt mit Hilfe von Eingangs- und Ausgangspuffern an den jeweiligen Maschinen durch das Message Passing.

Routing von Paketen im Mehrrechnersystem

Wie die einzelnen Pakete vom Sender zum Empfänger gelangen, bestimmt das Routing. Es ist ebenso dafür verantwortlich Deadlocks zu vermeiden. Dabei sind die Prozesse die Rechner und die Betriebsmittel die Ein - und Ausgabeports der jeweiligen Stationen.

Was ist ein Vektorrechner?

Viele numerische Probleme lassen sich auf Skalarprozessoren nur in vielen Schleifendurchlaufen berechnen. Vektorcomputer bieten high level instructions, um diese Schleife mit einem Befehl zu berechnen. Pipelines bieten sich mehr als an. Da die Datenelemente unabhängig sind, treten keine Data Hazards auf und das Fehlen von Schleifensprungbefehlen lässt keine Control Hazards in der Pipeline zu. Um die Speicherbandbreite zu erhöhen, verwendet man das Prinzip der Speicherverschränkung. Caches haben wenig Effizienz, da Vektorprobleme geringe Lokalität aufweisen. Die Bandbreite wird durch Blockzugriff und zeitlich verschränkten Speicherbankzugriff maximiert. (memory interleaving und memory banking)

Was unterscheidet die Hardware eines Vektorrechners von Konventionellen?

Die Register sind nicht eindimensional sondern bestehen aus Vektoren. D.h. das ein Registersatz ein 2-Dimensional ist. Sämtliche Hardwarekomponenten sind somit auf die Verarbeitung von Vektoren, anstatt einzelner Worter ausgelegt.

Wie arbeitet Speicherverschränkung?

Der der Hauptspeicher wird in gleichgroße, voneinander unabhängige Bereiche (Module, Speicherbänke) unterteilt, die zeitlich verschränkt gelesen oder beschrieben werden können. Aufeinanderfolgende Speicherworte werden zyklisch in aufeinanderfolgenden Speicherbänken abgespeichert. Sinn macht das Ganze wegen der Diskrepanz zwischen Zugriffs- und Zykluszeit der Hauptspeichermodule. Während die Zugriffszeit jene Zeit angibt, die zum Lesen bzw. Schreiben eines Speicherwortes benötigt wird, beschreibt die Zykluszeit eines Speichers die Zeitspanne vom Beginn eines Zugriffs bis zu jenem Zeitpunkt, zu dem der nächste Zugriff beginnen kann. Daß Zugriffs- und Zykluszeit für Hauptspeichermodule nicht übereinstimmen, liegt daran, daß sich der Speicher nach einem Zugriff zunächst regenerieren muß, bevor er neue Zugriffe erlaubt. Für gängige Halbleiterbauteile ist die Zykluszeit eines Speichers etwa drei- bis viermal so groß wie dessen Zugriffszeit. Für den Durchsatz eines nichtverschränkten Speichers ist dessen Zykluszeit, nicht die Zugriffszeit, ausschlaggebend.

Bei einem verschränkten Speicher ist, wenn auf die verschiedenen Module in geeigneter Reihenfolge zugegriffen wird, für den Durchsatz die deutlich kürzere Zugriffszeit sowie die Anzahl der Module entscheidend. Die größte Beschleunigung der mittleren Zugriffszeit wird erreicht, wenn aufeinanderfolgende Speicherzugriffe auf verschiedene Module erfolgen.

Interleaved Memory bei Vektorrechnern

Da Speicher zu langsam, werden nun mehrere unabhängige Speicherbänke realisiert, von denen jeder einen bestimmten Teil des Adressraums zugesprochen bekommt. Beziehen sich aufeinander folgende Speicherzugriffe jeweils auf Adressen verschiedener Speicherbänke, können die Folgezugriffe überlappt werden. Bis 512 Speicherbänke sind durchaus normal, um diese Latenzzeiten der DRAMs zu minimieren. Um Speicherbankko nflikte zu vermeiden, müssen Programmablauf und Speicherimage aufeinander abgestimmt sein. Dabei wird der Vorteil genutzt, dass die Position von aufeinander folgenden Vektorelementen im Speicher nicht sequentiell sein muss. Wie bei Pipelines wird hier die Forwarding-Ähnliche Technik des Chainings angewandt, um Resultatelemente einer Vektoroperation sofort an den Folgebefehl weiterzuleiten, ohne auf das Fertigstellen des Befehls zu warten. Unter Nutzung mehrerer Funktionseinheiten können Vektoroperationen damit quasi parallel ausgeführt werden.

Worauf muss ein Programmierer bei Interleaved Memory achten?

Der Programmablauf muss auf das Speicherabbild abgestimmt sein, das keine Konflikte auftreten können. Konflikte treten auf, wenn bspw. durch eine ungünstige Schleifenvariante immer die gleiche Speicherbank angesprochen wird und somit eine starke Verzögerung eintritt. Die Schleife sollte so aufgebaut sein, dass die Bandbreite komplett ausgenutzt werden kann.

Was definiert im Besonderm die Leistung eines Vektorrechners?

Die Leistung berechnet sich vor allem aus der Vektorlänge. Umso größer die verarbeiteten Vektoren sind, umso höher die Leistung. Da bei Vektorrechnern die StartUp-Zeit beträchtlichen Einfluss hat, sind hohe Performance-Werte nur bei ausreichend gefüllten Vektoren zu erreichen. Pauschal kann man sagen, dass ein Vektor mindestens halb gefüllt sein muss, um parallesierbare Programme ausreichend Effizient abarbeiten zu können.

Wie arbeiten Bus-basierte Multiprozessorsysteme?

Hier arbeiten mehrere CPU's in einem System parallel. Dabei hat jeder Prozessor gleichen Zugriff zum zum Hauptspeicher über einen Systembus, welcher von allen Prozessoren genutzt wird. (Shared Memory)
Logische Konsequenz ist, daß die Speicherbandbreite proportional zu der Anzahl N der Prozessoren sinkt. (1/N) Dieser Flaschenhals wird durch Caches vermindert. Solange ein Prozessor im Mittel N Zugriffe im Cache und erst dann einen im Hauptspeicher tätigt, wirkt diese Bandbreitenbegrenzung noch nicht negativ.

Da die Lokalität von Programmen nunmal auch begrenzt ist, sind soclche Multiprozessorsysteme höchstens bis um die 30 Prozessoren skalierbar.

Was ist der Unterschied zwischen symmetrischen und asymmetrischen MPs?

Bei asymmetrischen Systemem gibt es einen Master-Prozessor, welcher an das I/O Subsystem gekoppelt ist. Restliche Prozessoren sind Slaves und sind nicht direkt mit dem I/O Subsystem verbunden. Daher können die Slaves auch nur Anwenderkode ausführen. Zur Ausführung von Systemdiensten muss der Master gerufen werden, da nur er Betriebssystemoperationen ausführen darf.

Bei symmetrischen Systemem liegt das Kernel-Image für alle Prozessoren erreichbar im Shared Memory. Das I/O Subsystem und Interrupts können parallel von mehreren Prozessoren bearbeitet werden.

Wie wird der Bus verteilt – Arbitierung?

Zentrale Arbiterlogik	Dezentral – Daisy Chained Prinzip
Die Master senden einen Bus-Request an die zentrale Abiterlogik. Diese entscheidet, welcher Prozessor das acknowlege-Signal und somit den Bus zugesprochen bekommt.	Die Priorität der einzelnen Master ist durch Ihre Reihenfolge implizit gegeben. Ein BusGrant Signal wird durchgeschleift und kann, wenn es benötigt wird, durch einen Bus-Request an die Zentrale aktiviert werden. Dieses Verfahren ist äußerst ungerecht.

Test & Set Befehle und Bus-Arbitierung

Test & Set Befehle sind atomare Befehle zum Synchronisieren von Prozessen. In Uni-Prozessorsystemem ist das Problem der „Zerissenen Befehle“ dadurch gelöst. Anders bei Multiprozessorystemen. Dort kann es nach wie vor passieren, dass der aktive Prozessor den Bus entzogen bekommt und somit eine laufende Test & Set Instruction unterbrochen würde. Deshalb ist hier ein zusätzlicher Mechanismus notwendig. Es wird ein Lock-Signal eingeführt, welches dem Bus-Arbiter ein Umschalten zwischen einer Read- und Write-Phase eines Test & Set Befehls verhindert.

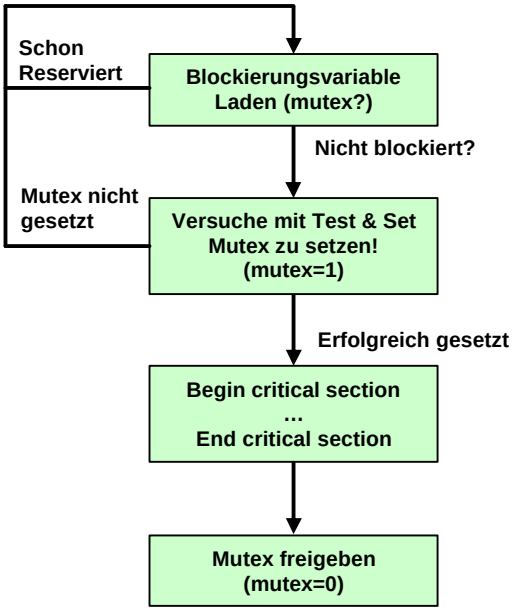


Abb.: Test & Set Protokoll für Mutual Exclusion

Cache-Kohärenz-Protokolle

Write-Through (Keine Inkonsistenz wegen Durchschreiben) und Write-Back (verringert Speicherverkehr, aber ermöglicht Inkonsistenten) sind die wichtigsten Methoden auf Singleprozessorsystemem. Write-Throung ist das einfachste und Write-Back das effizienteste, welches in Form des MESI-Protokoll z.B. im Pentium angewendet wird. Bei SMP’s dominieren **Write Invalidate** (alle anderen Kopien werden ungültig gemacht) und **Write Broadcast** (alle Kopien werden bei Schreibaktion aktualisiert) als Vertreter der Snooping-Protokolle. Ein anderes Snooping Protokoll ist das Write Once-Protokoll.

MESI-Protokoll für Multiprozessorsysteme

Ein Cache Eintrag kann einen von vier verschiedenen Zuständen besitzen. Treten Zustandswechsel auf, werden alle Prozessoren benachrichtigt.

1. **Invalid** (die Cacheline ist veraltet)
2. **Shared** (mehrere Prozessoren haben die aktuelle Cacheline)
3. **Exklusive** (Prozessor hat die korrekte Cacheline alleine)
4. **Modified** (Die Cacheline ist aktuell, aber nicht mehr mit Speicher konsistent)