Computing Plattformen & Netzwerke

Prof. Dr. Alexander Paar

Duale Hochschule Schleswig-Holstein

Rechnerarchitektur

Speicherverwaltung

Speicherhierarchie

Heutige Mikroprozessoren müssen wegen ihrer hohen internen Parallelität und ihrer hohen Taktraten ihrem Befehlsbereitstellungs- und Dekodierteil genügend Befehle als auch ihrem Ausführungsteil genügend viele Daten zuführen.

Da die Daten nicht immer nur den Registern entnommen werden können, wächst mit der Erhöhung der Verarbeitungskapazität des Prozessors auch die Anforderungen an die Bandbreite, mit der dem Prozessor Daten zugeführt werden müssen.

Eine weitere Beobachtung ist, dass der Speicherbedarf eines Programms mit seinen während der Ausführung erzeugten Zwischendaten häufig sehr groß werden den kann und selbst die Kapazität des Hauptspeichers sprengt.

Ideal wäre ein einstufiges Speicherkonzept, bei dem mit jedem Prozessortakt auf jedes Speicherwort zugegriffen werden kann.

Das ist technologisch für Prozessoren hoher Leistung nicht möglich, große Speicher existieren nur mit relativ langsamem Zugriff, während Speicherbausteine mit hoher Zugriffsgeschwindigkeit in ihrer Speicherkapazität beschränkt und teuer sind.

Technologisch klafft eine größer werdende Lücke zwischen der Verarbeitungsgeschwindigkeit des Prozessors und der Zugriffsgeschwindigkeit der DRAM-Speicherchips Speicherchips des Hauptspeichers.

Die hohen Prozessortaktraten und die Fähigkeit superskalarer Mikroprozessoren, mehrere Operationen pro Takt auszuführen, erzeugen zeugen von Seiten des Prozessors einen immer größeren Bedarf nach Code und Daten aus dem Speicher.

Die Geschwindigkeit dynamischer Speicherbausteine hat über die Jahre hinweg deutlich weniger zugenommen.

Eine Ausführung der Programme aus dem Hauptspeicher hätte zur Folge, dass der Prozessor nur mit einem Bruchteil seiner maximalen Leistung arbeiten könnte.

Deshalb muss der Prozessor seine Befehle vorwiegend aus dem auf dem Chip befindlichen Code-Cache-Speicher und seine Daten aus den Registern und dem Daten-Cache-Speicher erhalten.

Speicherhierarchie

Das einer Speicherhierarchie zu Grunde liegende Prinzip ist das Lokalitätsprinzip, dem die Befehle und die Daten eines Programms weitgehend gehorchen.

Man unterscheidet die zeitliche Lokalität (engl. temporal locality) und die räumliche Lokalität (engl. spatial locality).

Zeitliche Lokalität bedeutet, dass auf dasselbe Code- oder Datenwort während einer Programmausführung mehrfach zugegriffen wird.

Räumliche Lokalität bedeutet, dass im Verlaufe einer Programmausführung auch die benachbarten Code- oder Datenwörter benötigt werden.

Bezogen auf eine Codefolge bedeutet dies, dass auf einen Befehl meist der durch den Befehlszähler adressierte nächste Befehl oder ein befehlszählerrelativ adressierter, kurzer Sprung folgt (räumliche Lokalität) und darüber hinaus die Anwendung von Schleifen zur vielfachen Ausführung derselben Codefolge führt (zeitliche Lokalität).

Die gesamte Sprungvorhersage würde ohne zeitliche Lokalität in der Programmausführung genauso sinnlos sein wie die Verwendung von Code-Cache-Speichern.

Auf Daten bezogen bedeutet zeitliche Lokalität, dass auf ein Datenwort mehrfach zugegriffen wird, und räumliche Lokalität, dass darüber hinaus auch im Speicher benachbarte Datenwörter verwendet werden.

Daten, auf die in wenigen Takten wieder zugegriffen wird, werden vom Compiler in Registern bereitgehalten (zeitliche Lokalität).

Der Daten-Cache-Speicher nutzt sowohl zeitliche Lokalität – er verdrängt einmal geholte Daten erst wieder, wenn sie durch neuere Zugriffe ersetzt setzt werden müssen – als auch räumliche Lokalität, da nach einem Cache-Fehlzugriff nicht nur das 32- oder 64-Bit-Datenwort, sondern der gesamte meist 32 Byte große Cache-Block im Cache-Speicher bereitgestellt wird.

Es macht deshalb Sinn, räumliche und zeitliche Lokalität zu nutzen, um die Befehle und Daten, auf die wahrscheinlich als nächstes zugegriffen werden muss, nahe am Prozessor zu platzieren und solche Befehle und Daten, die wahrscheinlich in nächster Zeit nicht benötigt werden, auf entfernteren Speichermedien abzulegen.

Nahe beim Prozessor bedeutet dabei auf kleinen, schnellen und häufig teuren Speichermedien.

Speicherhierarchie

Bei heutigen Rechnern existiert eine Speicherhierarchie, die sich nach *absteigenden Zugriffsgeschwindigkeiten* und *aufsteigenden Speicherkapazitäten* geordnet aus Registern, Primär-Cache-Speicher, Sekundär-Cache-Speicher, Speicher, Hauptspeicher und Sekundärspeicher zusammensetzen kann.

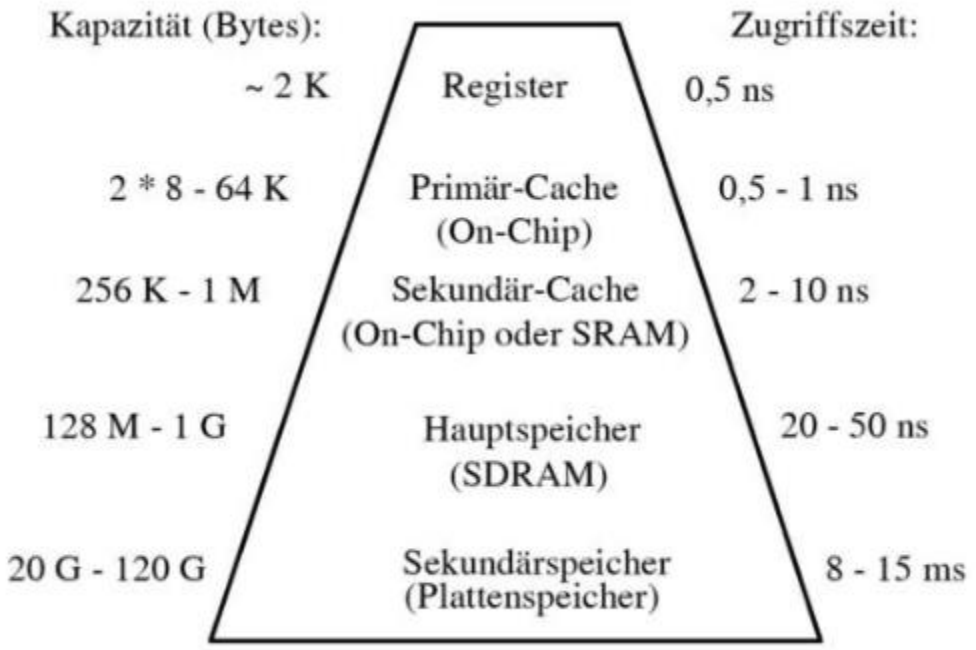
Speichermedien in höheren Ebenen der Speicherhierarchie sind kleiner, schneller, aber auch pro Byte teurer als solche in den tiefer gelegenen Ebenen.

Von einer tieferen zu einer höheren und damit prozessornäheren Speicherebene werden Speicherbereiche auf Anforderung übertragen, z.B. durch einen Ladebefehl, der einen Cache-Fehlzugriff auslöst.

Meist sind die Speicherwörter in den höheren Ebenen Kopien der Speicherwörter in den unteren Ebenen, so dass sich eine hohe Redundanz ergibt.

Eine grundlegende Speicherorganisationsfrage ist, ob und wann bei einem Schreibzugriff auf einen Speicher höherer Ebene auch eine Modifikation des Speicherwortes in einem der Speicher tieferer Ebene stattfindet.

Ein sofortiges Durchschreiben (engl. write through), das die Konsistenz in allen Kopien erhält, ist nicht immer effizient implementierbar.



Speicherhierarchie eines 2 GHZ PCs (ca. 2002)

Speicherhierarchie

Eine Technik, um die Zugriffsgeschwindigkeit auf den Hauptspeicher stärker an die Verarbeitungsgeschwindigkeit der CPU anzupassen, ist, den Hauptspeicher in *n* sogenannte Speicherbänke M_0, ..., M_{n-1} zu unterteilen und jede Speicherbank mit einer eigenen Adressierlogik zu versehen.

Falls nun *k* (*k* > *n*) sequenziell hintereinander ablaufende Befehle *k* fortlaufende physische Speicherplätze mit den Adressen A_0, ..., A_{k-1} benötigen (typische Beispiele sind Vektor- und Matrixoperationen) werden die einzelnen Speicherplätze nach der folgenden Verschränkungsregel (engl. interleaving rule) auf die einzelnen Speicherbänke verteilt.

A_i wird auf die Speicherbank M_j gespeichert, genau dann wenn
$j = i \bmod n$ gilt.

Auf diese Weise werden die Adressen A_0, A_n, A_{2n}, ... der Speicherbank M_0, die Adressen A_1, A_{n+1}, A_{2n+1}, ... der Speicherbank M_1 usw. zugeteilt.

Diese Technik heißt Speicherverschränkung (engl. memory interleaving), und die Verteilung auf *n* Speicherbänke nennt man *n*-fache Verschränkung.

Der Zugriff auf die Speicherplätze kann nun ebenfalls verschränkt, das heißt zeitlich überlappt, geschehen, so dass bei der *n*-fachen Verschränkung in ähnlicher Weise wie bei einer Verarbeitungs-Pipeline nach einer gewissen Anlaufzeit in jedem Speicherzyklus *n* Speicherworte geliefert werden können.

Eine Speicherverschränkung kann auf jeder Speicherhierarchieebene angewandt werden, insbesondere neben dem Hauptspeicher auch bei Cache-Speichern.

Register und Registerfenster

Register stellen die oberste Ebene der Speicherhierarchie dar.

Ihre Inhalte können in einem Prozessortakt in die Pipeline-Register (engl. latches) der Ausführungseinheiten geladen werden.

Da die Register eines Registersatzes auf dem Prozessor-Chip untergebracht, im Befehl direkt adressiert und mit vielen Ein- und Ausgabekanälen versehen sein müssen, ist ihre Anzahl stark beschränkt.

Üblicherweise sind 8 (IA-32-Registermodell), 32 (RISC-Prozessoren) und 128 (IA-64-Registermodell) allgemeine Register und nochmals ebenso viele Gleitkommaregister sowie zusätzliche Multimedia-Register vorhanden.

Je mehr Register vorhanden sind, desto größer wird der Zeitaufwand für das Sichern der Register beim Unterprogrammaufruf, bei Unterbrechungen und beim Kontextwechsel, bzw. für das Wiederherstellen des Registerzustands bei der Rückkehr aus einem Unterprogramm, einer Unterbrechung oder einem Betriebssystem-Kontextwechsel.

Jeder Aufruf einer Prozedur oder Rücksprung aus einer Prozedur – Aktionen, die sehr häufig auftreten – ändert die lokale Umgebung eines Programmablaufs.

Bei jedem Unterprogrammaufruf muss der Registersatz gesichert werden, so dass die Daten im Anschluss an die Unterprogrammbearbeitung vom aufrufenden Unterprogramm wieder verwendet werden können.

Dazu kommt die Übergabe von Parametern.

Die Lösung dieses Problems basiert auf zwei Feststellungen

- Ein Unterprogramm besitzt typischerweise nur wenige Eingangsparameter und lokale Variablen
- Die Schachtelungstiefe der Unterprogrammaufrufe ist typischerweise relativ klein

Um diese Eigenschaften zu nutzen, werden mehrere kleine Registermengen – die Registerfenster – benötigt

Jede dieser Mengen ist in der momentanen Schachtelungshierarchie einem Unterprogramm zugeordnet

Bei einem Unterprogrammaufruf wird auf ein neues Registerfenster umgeschaltet, anstatt die Register im Speicher zu sichern

Die Fenster von aufrufendem und aufgerufenen Unterprogramm überlappen sich, um die Übergabe von Parametern zu ermöglichen

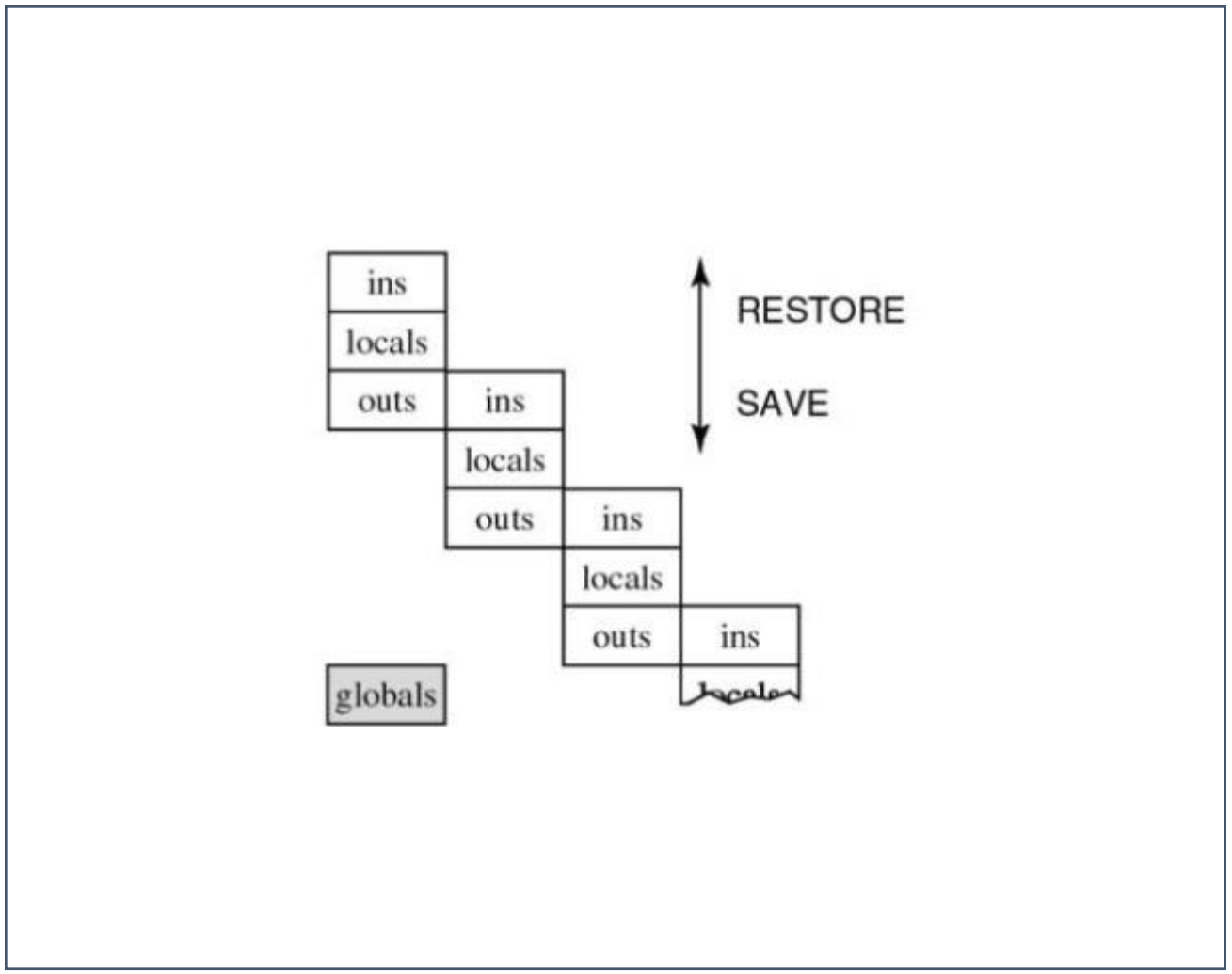
Zu jedem Zeitpunkt ist nur ein Registerfenster sichtbar und adressierbar, so als ob nur ein Registersatz vorhanden wäre

Register und Registerfenster

Durch eine Registerorganisation mit Registerfenstern kann die Anzahl der Speicherzugriffe und der verbundene Zeitaufwand für das Sichern des Registersatzes bei Prozeduraufrufen verringert werden.

Das Verfahren wurde zunächst eingeführt beim Berkeley RISC

Heute auch bei den superskalaren SPARC-Nachfolgern sowie in modifizierter Form beim IA-64-Registersatz des Intel Itanium



Register und Registerfenster

Beispiel SPARC-Architektur

Das Registerfenster bei der SPARC-Architektur ist in drei Bereiche aufgeteilt

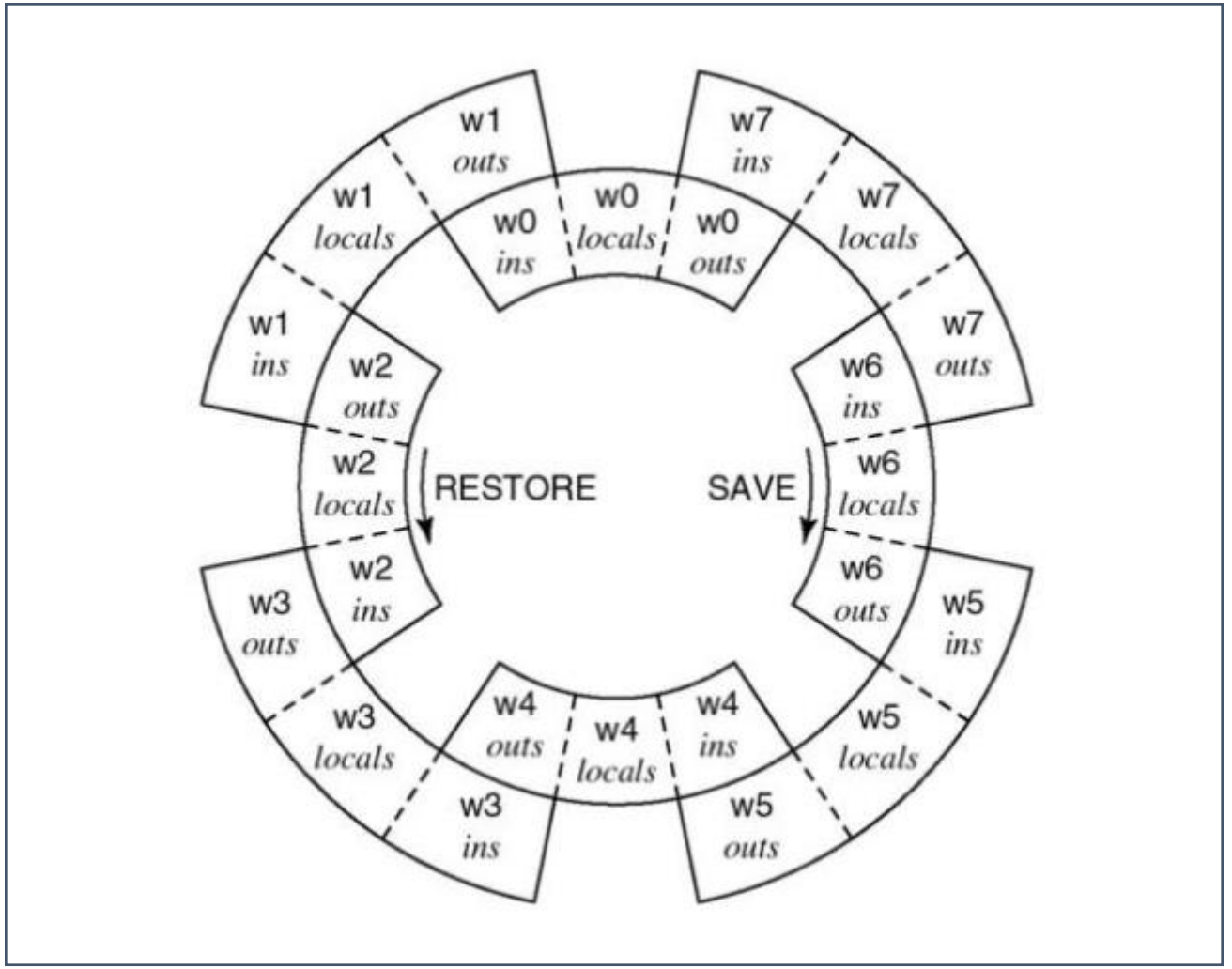
- 8 in-Register (**ins**)
- 8 lokale Register (**locals**)
- 8 out-Register (**outs**)

Die lokalen Register sind nur jeweils in einer Unterprogrammaktivierung zugänglich

Die **in**-Register enthalten sowohl die Parameter, die von dem aufrufenden Unterprogramm übergeben wurden als auch die Paramter die als Ergebnis des aufgerufenen Unterprogramms zurückgegeben werden

Die **out**-Register sind mit den in-Registern der nächsthöheren Schachtelungsebene identisch

Diese Überlappung erlaubt eine Paramterübergabe ohne Daten verschieben zu müssen



Register und Registerfenster

Um jede mögliche Schachtelungstiefe handhaben zu können, müsste die Anzahl der Registerfenster ungebrenzt sein

Untersuchungen zeigen jedoch, dass 8 – 10 Registerfenster ausreichen

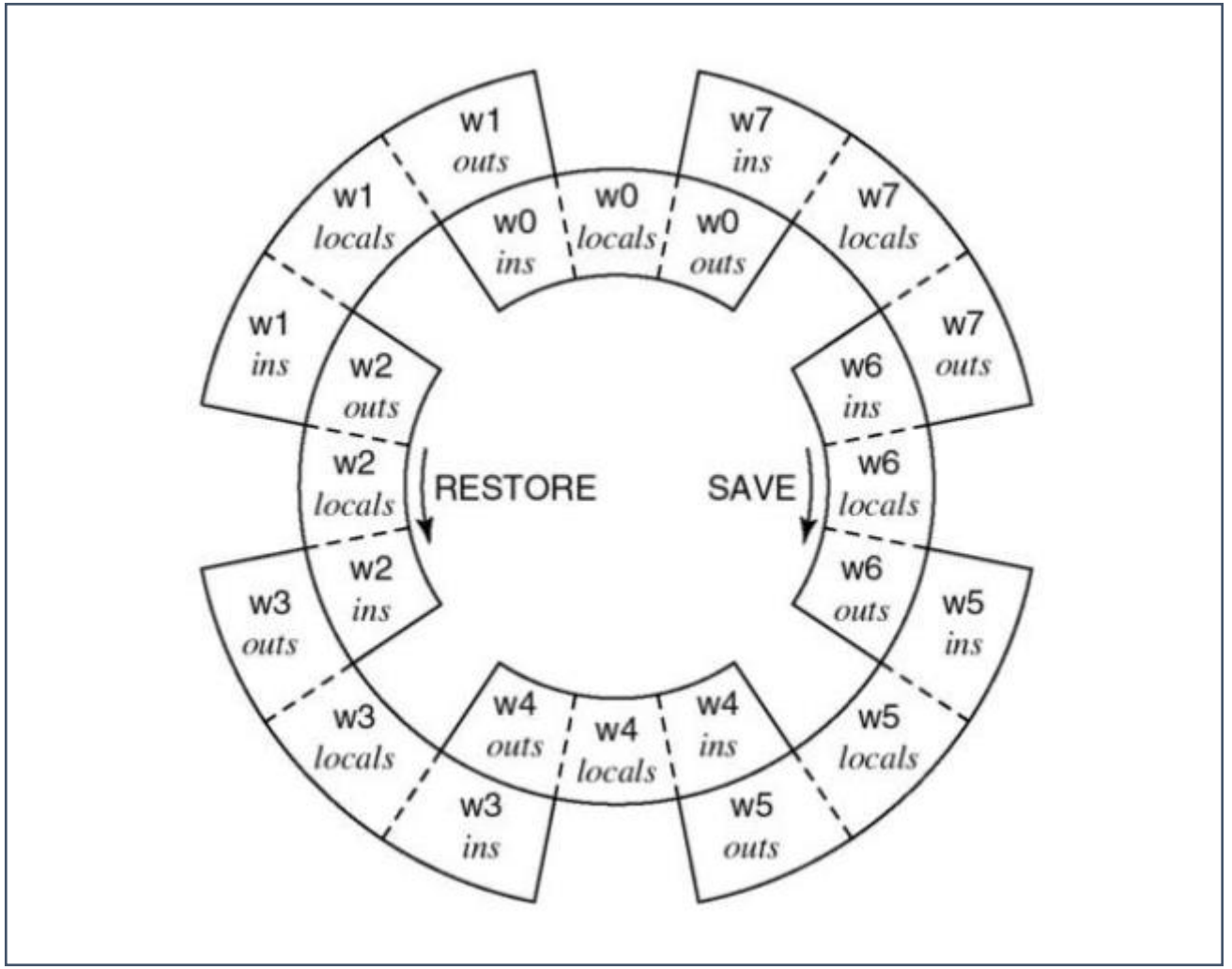
Wenn die Schachtelungstiefe die Anzahl der Registerfenster überschreitet werden die ältesten Daten zwischenzeitlich im Hauptspeicher abgelegt und die frei werdenden Register werden erneut verwendet

→ Umlaufspeicherorganisation

Die Registerfenster sind mit **w0** – **w7** bezeichnet

Das aktive Registerfenster wird über den CWP (Current Window Pointer) referenziert

Der CWP ist in einem Feld der Prozessortstatusworts untergebracht



Register und Registerfenster

Ablauf eines Unterprogrammaufrufs

Die Parameter für das Unterprogramm werden in die out-Register geschrieben

Mit einem **CALL**-Befehl wird in das Unterprogramm gesprungen

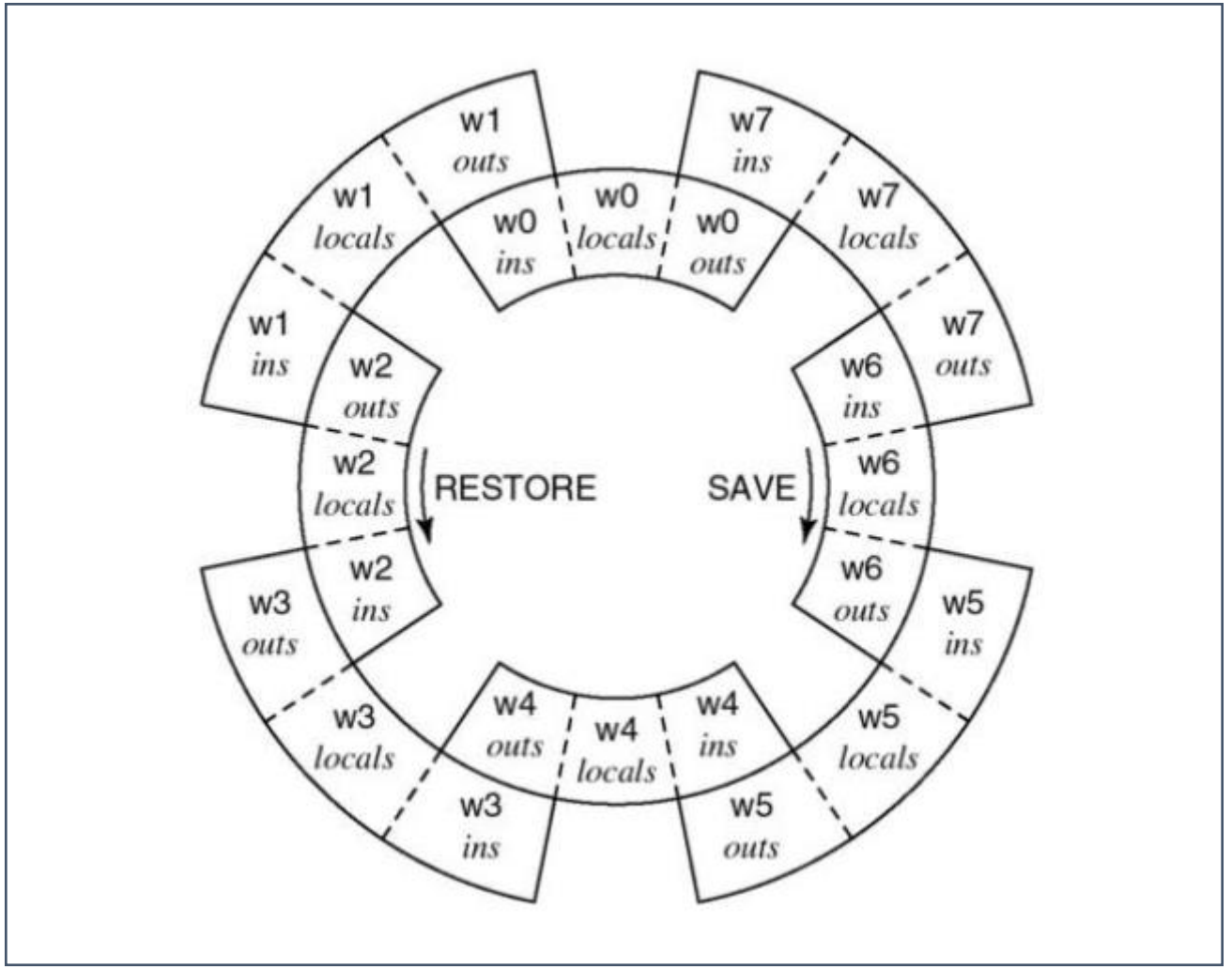
Am Anfang des Unterprogramms wird ein **SAVE**-Befehl ausgeführt und damit auf das nächster Registerfenster weitergeschaltet. Dabei wird der CWP dekrementiert.

In dem neuen Registerfenster können die Parameter aus den **in**-Registern gelesen und bearbeitet werden

Am Ende des Unterprogramms werden dessen Ergebnis wieder in die **in**-Register geschrieben

Mit einem **RESTORE**-Befehl wird auf das vorherige Fenster zurückgeschaltet (d.h. der CWP wird inkrementiert) und mit dem **RET**-Befehl in das aufrufende Programm zurückgesprungen

Dort können die Ergebnisse aus den **out**-Registern abgeholt werden



Register und Registerfenster

Wegen der begrenzten Größe des Umlaufspeichers für Registerfenster wird ein Mechanismus für das Erkennen und Bearbeiten von Fensterüberläufen (engl. window overflow) benötigt

Ein Fensterüberlauf tritt auf, wenn mit einem **SAVE**-Befehl auf ein Registerfenster, das sich mit einem bereits verwendeten Registerfenster überscheidet, umgeschaltet wird

In diesem Fall muss das zu überschreibende Fenster in den Hauptspeicher gesichert werden

Ein Fensterrücklauf (engl. window underflow) tritt auf, wenn genau ein Registerfenster belegt ist und mit einem **RESTORE**-Befehl auf das vorherige Fenster zurückgeschaltet wird

In diesem Fall muss das vorherige Fenster aus dem Hauptspeicher nachgeladen werden

Für die Erkennung eines Fensterüberlaufs bzw. eines Fensterrücklaufs wird beim SPARC das **WIM**-Register (window invalid mask) verwendet

Im **WIM**-Register kann jedes Registerfenster durch Setzen des entsprechenden Bits als invalid markiert werden

Wenn der **CWP** durch **SAVE**- bzw. **RESTORE**-Befehle ein Fenster, das als invalid markiert ist, erreicht, wird ein Trap ausgelöst

Die Trap-Routinen sind Teil des Betriebssystems und lagern bei einem Fensterüberlauf ein Fenster aus und holen bei einem Fensterrücklauf das vorherige Fenster wieder zurück

Dabei genügt es, nur die Registerbereiche **ins** und **locals** zwischenzuspeichern

In den Trap-Routinen wird außerdem das **WIM**-Register an die veränderte Situation angepasst

Register und Registerfenster

Ein Umlaufspeicher mit n Registerfenstern kann n-1 verschachtelte Unterprogrammaufrufe behandeln

Aber: bereits eine kleine Anzahl von Registerfenstern sind für eine effiziente Prozedurbehandlung ausreichend

Der Berkeley RISC-Prozessor verwendet 8 Fenster mit jeweils 16 Registern

Damit ist nur ein einem Prozent aller Unterprogrammaufrufe ein Hauptspeicherzugriff erforderlich

Eine zu große Registerdatei kann sich aber auch negativ auswirken

Für eine größere Registermenge ist die Adresscodierung für Register hardwareaufwändiger und kann mehr Zeit in Anspruch nehmen

In einer Multitasking-Umgebung, bei der der Programmablauf oft zwischen Prozessen wechselt, wird mehr Zeit benötigt , um eine große Registerdatei im Hauptspeicher zu sichern

→ Die Frage nach der optimalen Anzahl an Registern ist nicht trivial sondern hängt auch vom Anwendungsgebiet ab

Registerfenster ermögliche eine effiziente Behandlung von *lokalen* Variablen

Erforderlich sind aber auch *globale* Variablen, auf die mehrere Unterprogramme Zugriff haben

Bei der SPARC-Architektur gibt es dafür einen Registerbereich, auf den alle Unterprogramme jederzeit Zugriff besitzen

Die Register **R0** – **R31** sind die globalen Register

R8 – **R31** beziehen sich auf das aktuelle Fenster

Der Compiler entscheidet, welche Variablen den einzelnen Registerbereichen zugewiesen werden sollen

Virtuelle Speicherverwaltung

Mehrbenutzer-, Multitasking- und Multithreaded-Betriebssysteme stellen hohe Anforderungen an die Speicherverwaltung

Programme, die in einer Mehrprozessumgebung lauffähig sein sollen, müssen relozierbar (d.h. verschiebbar) sein

Geladene Programme und ihre Daten dürfen nicht an festgelegte physikalische Speicheradressen gebunden sein

Erforderlich ist außerdem ein großer Adressraum für die einzelnen Prozesse sowie geeignete Schutzmechanismen zwischen diesen Prozessen

Die Umsetzung von Schutzmechanismen erfordert zusätzliche Informationen über Zugriffsrechte und die Gültigkeit der Speicherwörter

Eine virtuelle Speicherverwaltung unterteilt den physisch vorhandenen Speicher in Speicherblöcke (Seiten, Segmente) und bindet diese an Prozesse

Als Schutzmechanismus ist der Zugriffsbereich eines Prozesses auf sein Speicherblöcke beschränkt

Virtuelle Speicherverwaltung

Eine virtuelle Speicherverwaltung bietet eine Reihe von Vorteilen…

- ein großer Adressraum mit einer Abbildung von z. B. 232 bis 264 virtuellen Adressen auf z. B. 228 physische Adressen für physisch vorhandene Hauptspeicherplätze
- eine einfache Relozierbarkeit (engl. relocation), die es erlaubt, ein Programm in beliebige Bereiche des Hauptspeichers zu laden oder zu verschieben
- die Verwendung von Schutzbits, die beim Zugriff geprüft werden und es ermöglichen, unerlaubte Zugriffe abzuwehren
- ein schneller Programmstart, da nicht alle Code- und Datenbereiche zu Programmbeginn im Hauptspeicher vorhanden sein müssen
- eine automatische (vom Betriebssystem organisierte) Verwaltung des Haupt- und Sekundärspeichers

Die Organisationsform der virtuellen Speicherverwaltung und der Cache-Speicherverwaltung haben einige Gemeinsamkeiten

- Cache-Blöcke korrespondieren mit den Speicherblöcken, d.h. den Seiten (engl. pages) oder Segmenten der virtuellen Speicherverwaltung
- Ein Cache-Fehlzugriff entspricht einem Seiten- oder Segmentzugriff

Aber:

- Die Cache-Speicherverwaltung wird vollständig in Hardware ausgeführt
- Die virtuelle Speicherverwaltung wird von Betriebssystem ausgeführt und wird von der Hardware mit der Speicherverwaltungseinheit (engl. memory management unit, MMU) und speziellen Maschinenbefehlen unterstützt

Virtuelle Speicherverwaltung

Die Adressbreite des Prozessors bestimmt die maximale Größe des virtuellen Adressraums und damit des virtuellen Speichers

Die Größe des Cache-Speichers ist davon unabhängig

Die Cache-Speicherverwaltung verschiebt Cache-Blöcke zwischen den Cache-Speichern verschiedener Hierarchieebenen und dem Hauptspeicher

Die virtuelle Speicherverwaltung verschiebt Speicherblöcke zwischen Hauptspeicher und Sekundärspeicher (HDD, SSD)

Größe und Zugriffsgeschwindigkeit sind für Cache- und virtuelle Speicherverwaltung sehr unterschiedlich

Parameter	Primär-Cache	virtueller Speicher
Block-/Seitengröße	16–128 Bytes	4 K–64 KBytes
Zugriffszeit	1–2 Takte	40–100 Takte
Fehlzugriffsaufwand	8-100 Takte	70000-6000 000 Takte
Fehlzugriffsrate	0.5-10%	0.00001-0.001%
Speichergröße	8 K–64 KBytes	16 M–8 GBytes

Parameterbereiche für Caches und virtuelle Speicher (1996)

Virtuelle Speicherverwaltung

Aus den Speicherreferenzen in den Maschinenbefehlen wird durch Adressrechnung eine effektive Adresse berechnet

Diese wird als sogenannte logische Adresse durch die virtuelle Speicherverwaltung zuerst in eine virtuelle Adresse und dann in eine physische Speicheradresse transformiert

Häufig wird in der Literatur nicht zwischen logischen und virtuellen Adressen unterschieden

Die virtuelle Speicherverwaltung stellt während der Programmausführung fest welche Daten gebraucht werden, transferiert die angeforderten Speicherseiten zwischen Haupt- und Sekundärspeicher und aktualisiert die Referenzen zwischen virtueller und physischer Adresse in einer Übersetzungstabelle

Wenn eine Referenz nicht im physischen Speicher gefunden werden kann wird dies als Seitenfehler (engl. page fault) bezeichnet

Die Speicherseite muss dann durch eine Betriebssystemroutine nachgeladen werden

Dies geschieht transparent für Benutzerprogramme

Aber: Die Zeit für den Umladevorgang reduziert die Ausführungsgeschwindigkeit signifikant

Die Lokalität von Code und Daten sowie eine geschickte Ersetzungsstrategie für die Seiten, die im Hauptspeicher durch Umladen überschrieben werden, gewährleistet eine hohe Wahrscheinlichkeit, dass der Code / die Daten, die durch den Prozessor angefordert werden, im physischen Hauptspeicher zu finden sind

Virtuelle Speicherverwaltung

Bei der Organisation einer virtuellen Speicherverwaltung stellen sich folgende Fragen

Wo kann der Speicherblock im Hauptspeicher platziert werden?

Wegen des außerordentlich hohen Fehlerzugriffsaufwands, der linearen Adressierung des Hauptspeichers durch physische Adressen und der festen Seitenlänge ist der Speicherort einer Seite im Hauptspeicher beliebig wählbar

Bei Segmenten sollte wegen ihrer unterschiedlichen Längen auf eine möglichst große Verdichtung geachtet werden, um keine zu kleinen Hauptspeicherbereiche zu erhalten

Welcher Speicherblock soll bei einem Fehlzugriff ersetzt werden?

Die übliche Ersetzungsstrategie ist die LRU-Strategie (least recently used)

Anhand von Zusatzbits für jeden Speicherblock wird festgestellt, auf welchen Speicherblock am längsten nicht mehr zugegriffen worden ist

Wann muss ein verdrängter Speicherblock in den Sekundärspeicher zurückgeschrieben werden?

Falls der zu verdrängende Speicherblock im Hauptspeicher unverändert ist, so kann er einfach überschrieben werden

Andernfalls muss der verdrängte Speicherblock in den Sekundärspeicher zurückgeschrieben werden

Wie wird ein Speicherblock aufgefunden, der sich in einer höheren Hierarchiestufe befindet?

Eine Seiten- oder Segmenttabelle kann sehr groß werden.

Mit einer invertierten Seitentabelle (engl. inverted page table) kann die Tabellengröße reduziert werden

Mit einem Hash-Verfahren kann der Lookup einer Seite in der invertierten Seitentabelle beschleunigt werden

Welche Seitengröße sollte gewählt werden?

Vorteile großer Seiten

- Die Größe der Seitentabelle ist umgekehrt proportional zu der gewählten Seitengröße
- Der Transport großer Seiten zwischen Haupt- und Sekundärspeicher kann effizienter organisiert werden
- Die Zahl der Einträge im Adressumsetzungspuffer ist beschränkt. Große Seiten erfassen mehr Speicherplatz und führen zu weniger TLB-Fehlzugriffen

Vorteile kleiner Seiten

- Geringere Fragmentierung des Speichers
- Schnellerer Prozessstart

Virtuelle Speicherverwaltung

Lösung

- Hybridtechnik mit mehreren unterschiedlichen Seitengrößen
- Kombination von Seiten und Segmenten

Meist erfolgt die Übersetzung einer logischen in eine physische Adresse nicht über eine sondern über mehrere Übersetzungstabellen

Oft zwei Übersetzungstabellen

- Segmenttabelle
- Seitentabelle

Bei der Speichersegmentierung wird der gesamte Speicher in ein oder mehrere Segmente variabler Länge eingeteilt

Für jedes Segment können individuell Zugriffsrechte vergeben werden

Bei der Adressübersetzung werden die Zugriffsrechte nachgeprüft

Außerdem wird beim Übersetzen einer logischen Adresse getestet, ob die erzeugte virtuelle Adresse innerhalb des vom Betriebssystem reservierten Speicherbereichs liegt

Zusammen mit prozessspezifischen Zugriffsrechten kann ein Speicherschutzmechanismus umgesetzt werden

Virtuelle Speicherverwaltung

Alternativ oder zusätzlich zur Speichersegmentierung wird die Seitenübersetzung eingesetzt

Dabei wird der Adressraum in Seiten fester Größe zerlegt (oft 4 KByte)

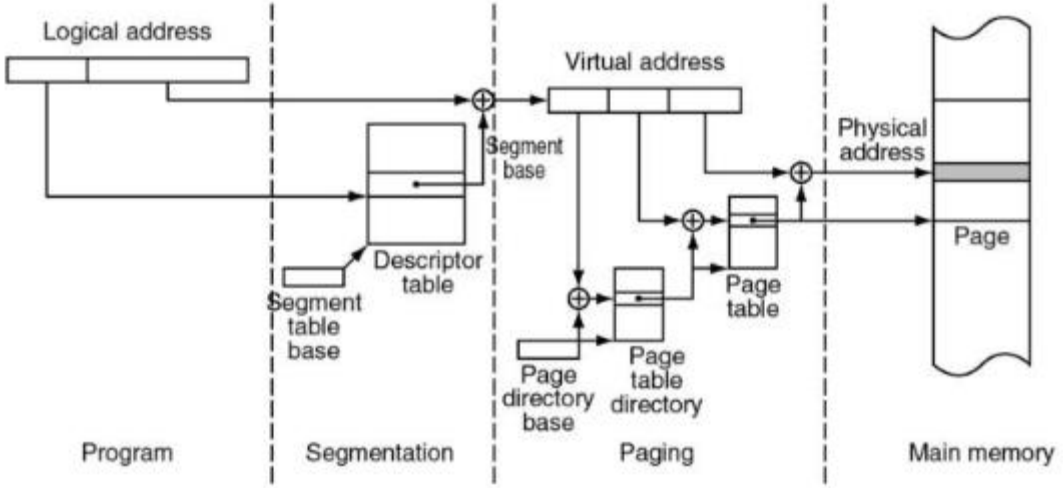
Die virtuelle Adresse wird in eine virtuelle Seitenadresse und eine Offset-Adresse aufgeteilt

Die virtuelle Seitenadresse wird mit (mehrstufigen) Übersetzungstabellen übersetzt

Die Offset-Adresse wird unverändert übernommen

Bei Intel-Prozessoren ab dem 80386 findet vor der Seitenübersetzung noch die Segmentierung statt

Die Seitenübersetzung (engl. paging) kann über ein Bit in einem Steuerregister ausgeschaltet werden



Virtuelle Speicherverwaltung

Bei Intel-Prozessoren besteht eine logische Adresse aus einem Segmentselektor und einer Verschiebung (engl. offset)

Der Selektor steht in einem der Segment-Register **CS**, **DS**, **SS**, usw. und wird meist durch den Maschinenbefehl ausgewählt

Die Verschiebung wird explizit im Befehl spezifiziert

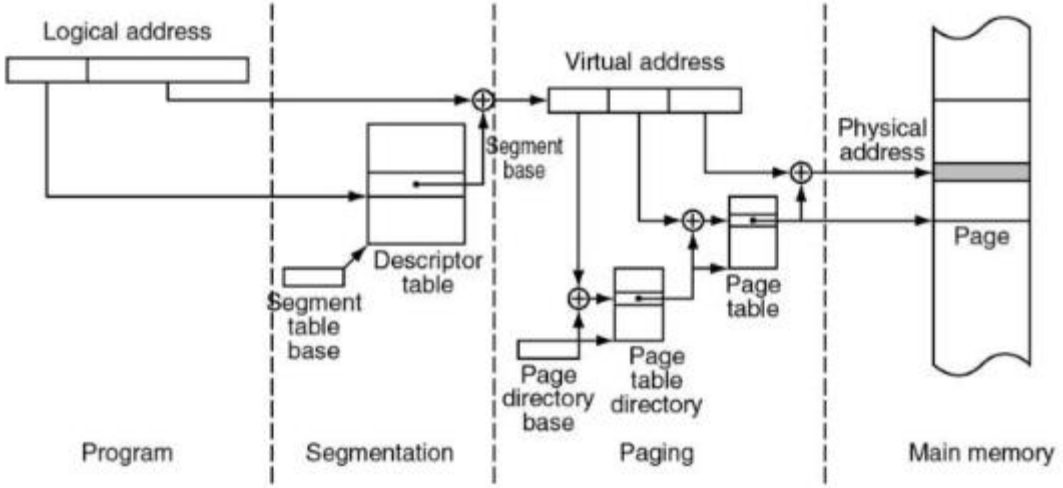
In der ersten Phase der Adressübersetzung wird die logische Adresse in eine virtuelle Adresse übersetzt

Der Segmentselektor wird als Zeiger in eine Deskriptortabelle verwendet

Dieser Deskriptor enthält Informationen über die Basisadresse und die Länge des Segments und über die erforderlichen Zugriffsrechte

Die virtuelle 32-Bit-Adresse berechnet sich durch Addition von Segment-Basisadresse und Verschiebung

Falls die Verschiebung größer oder gleich der Segmentlänge ist oder wenn der Zugriff nicht erlaubt ist wird eine Unterbrechung ausgelöst



Virtuelle Speicherverwaltung

In der zweiten Phase wird aus der virtuellen Adresse in zwei Stufen die physische Adresse erzeugt

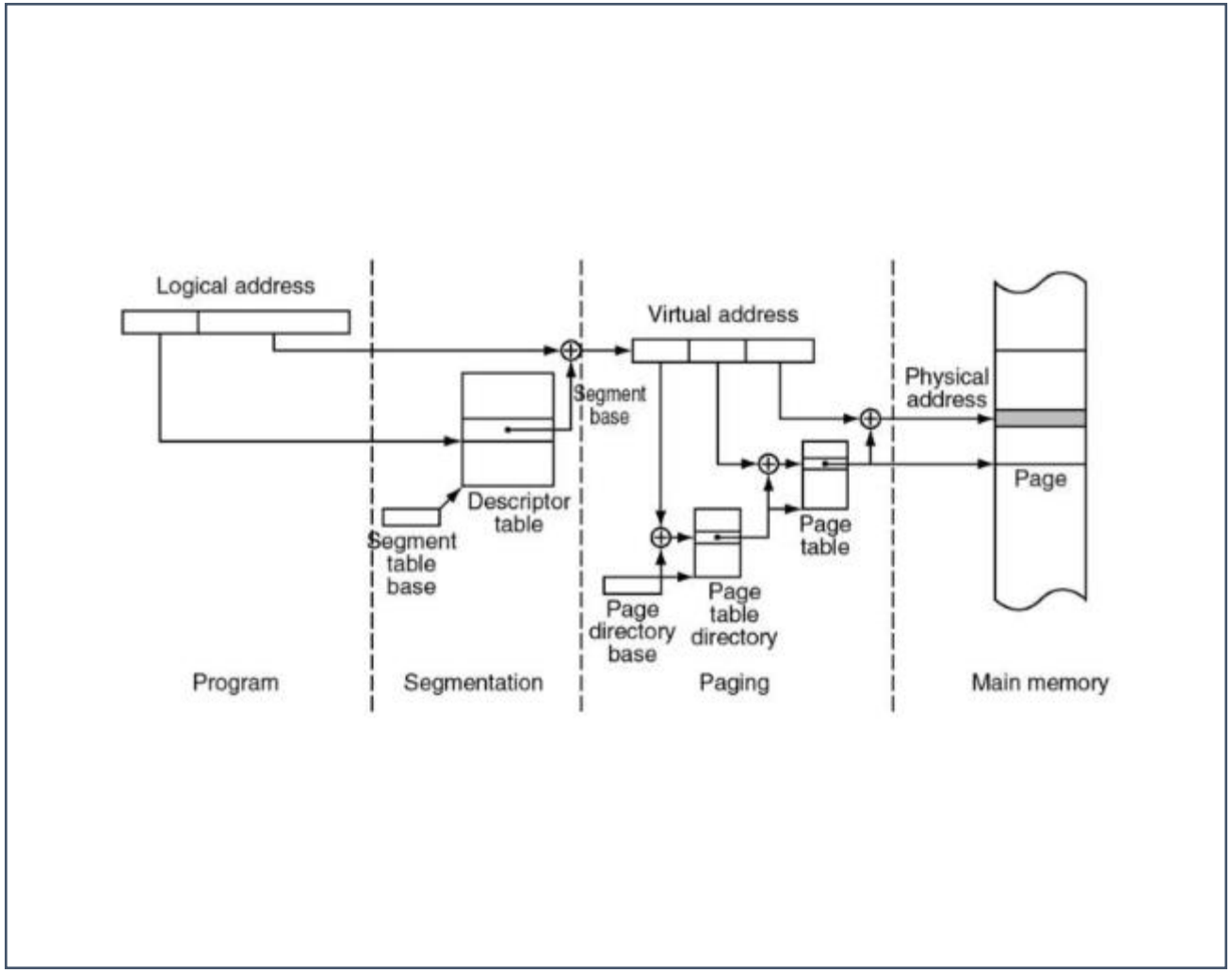
Dazu wird die virtuelle Adresse in drei Felder zerlegt

Mit den höchstwertigen 10 Bit wird ein Verzeichniseintrag aus dem Seitenverzeichnis (engl. page table directory) ausgewählt

Dieser Eintrag enthält die Adresse der zugehörigen Seitentabelle (engl. page table)…

… aus der mit den nächsten 10 Bit der Tabelleneintrag mit der endgültigen physischen Seitennummer ausgewählt wird

Die restlichen 12 Bit der virtuellen Adresse werden unverändert an die physische Seitennummer angefügt und bilden zusammen mit dieser die physische Adresse

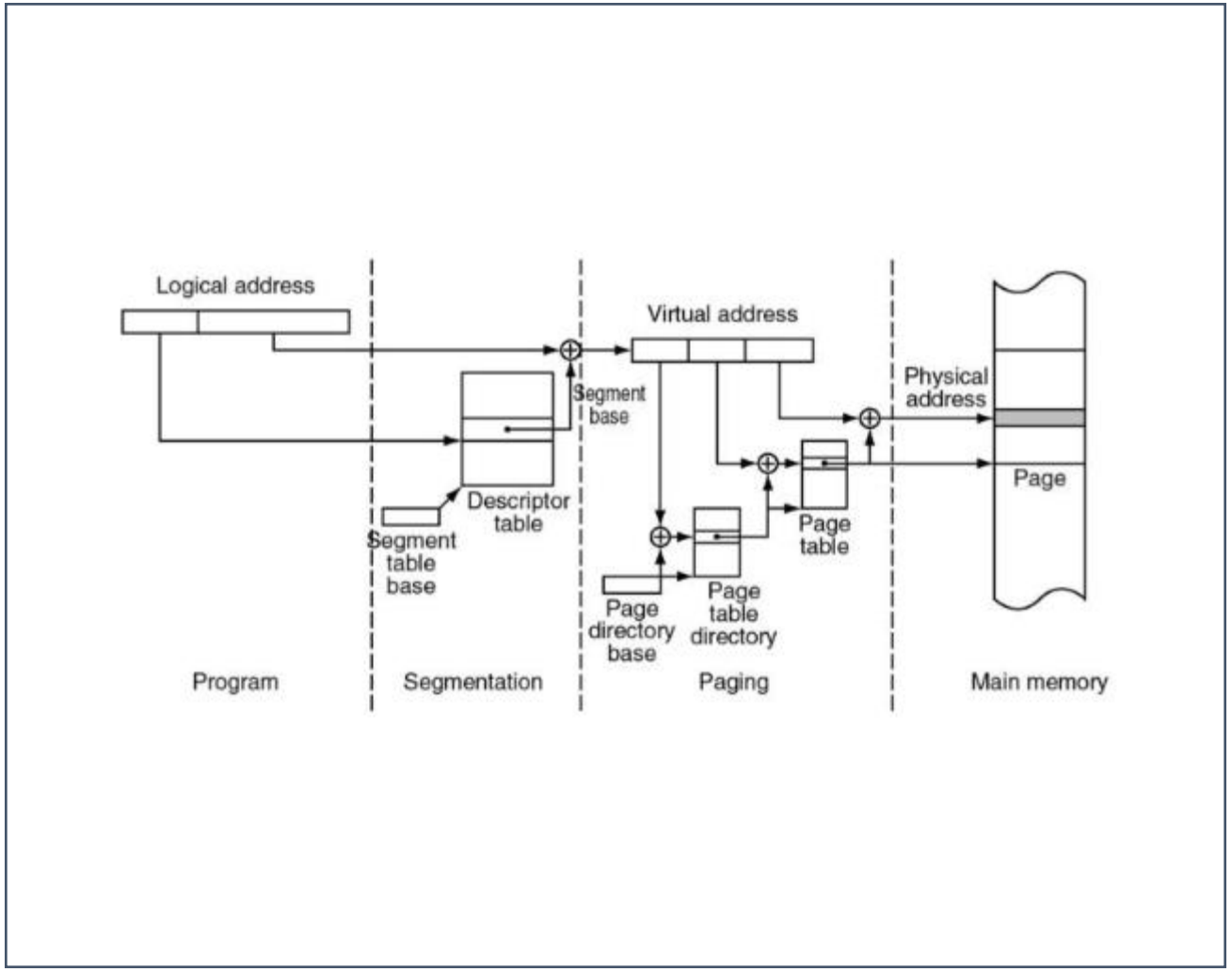


Virtuelle Speicherverwaltung

Jeder Eintrag im Seitenverzeichnis und in den Seitentabellen besteht aus einer 20 Bit breiten Seitennummer und einigen Verwaltungsbits

Insbesondere gibt es ein **PRESENT**-Bit, das anzeigt ob eine Seite im Hauptspeicher vorhanden ist

Ist dieses Bit nicht gesetzt wird ein Seitenfehlzugriff (engl. page exception) ausgelöst damit das Betriebssystem die fehlende Seite nachlädt



Virtuelle Speicherverwaltung

Das softwaremäßige Durchlaufen der Übersetzungstabellen würde verhältnismäßig lange dauern

Eine schnelle Adressumsetzung wird durch eine Hardwaretabelle erreicht

Ein Adressübersetzungspuffer (engl. translation look aside buffer, TLB) enthält die wichtigsten Adressumsetzungen

Dieser meist nur 32 – 128 Einträge großer Cache für die zuletzt durchgeführten Adressumsetzungen wird vollassoziativ verwaltet

Meist ist der TLB um zusätzliche Logik zur Seitenverwaltung und für den Zugriffsschutz erweitert

Man spricht dann von einer Speicherverwaltungseinheit (engl. memory management unit, MMU)

Ist ein gewünschter Eintrag im TLB nicht vorhanden (TLB miss) wird eine Unterbrechung ausgelöst, die die Übersetzungstabellen der virtuellen Speicherverwaltung softwaremäßig durchläuft, um die physische Adresse herauszufinden, und dann das Adresspaar in den TLB einträgt

Für den Befehls- und den Daten-Cache gibt es bei heutigen Mikroprozessoren auf dem Prozessor-Chip getrennte Speicherverwaltungseinheiten mit eigenen TLBs

Damit kann die Adressübersetzung für Code und Daten parallel zueinander durchgeführt werden

TLB-Zugriffe erfolgen im gleichen Takt, in dem auch auf den Primär-Cache-Speicher zugegriffen wird

Ein TLB-Eintrag besteht aus…

- einem Tag, der einen Teil der virtuellen Adresse enthält
- einem Datenteil mit einer physischen Seitennummer (engl. physical page frame number)
- weiteren Verwaltungs- und Schutzbits
  - die Seite befindet sich im Hauptspeicher (engl. resident)
  - der Zugriff ist nur dem Betriebssystem erlaubt (engl. supervisor)
  - Schreibzugriff ist verboten (engl. read-only)
  - die Seite wurde verändert (engl. dirty)
  - ein Zugriff auf die Seite ist erfolgt (engl. referenced)

Bei einem Tagged TLB enthält jeder TLB-Eintrag zusätzlich einen Prozesstag mit dem die Adressräume der verschiedenen Prozesse unterschieden werden können

Bei einem Prozesswechsel müssen die TLB-Einträge dann nicht gelöscht werden (TLB flush) sondern werden nach Bedarf verdrängt

DH DUALE
SH HOCHSCHULE SH

Bildnachweis

https://www.ixbt.com/mainboard/asus/asus-p5ad2-e-premium/board-big.jpg

https://en.wikipedia.org/wiki/John_L._Hennessy#/media/File:John_L_Hennessy_(cropped).jpg

https://en.wikipedia.org/wiki/David_Patterson_(computer_scientist)#/media/File:David_A_Patterson.jpg

https://en.wikipedia.org/wiki/R3000#/media/File:MIPS_R3000A_die.JPG