Heft 2
Speicherverwaltung

- Grundlagen der Rechnerarchitektur, Kapitel 3 -

3. Speicherverwaltung

3.1. Grundlagen

3.1.1. Weshalb Speicherverwaltung?

Prozessor und Speichersubsystem - der Ablauf eines Speicherzugriffs
Vom Prozessor kommt die Adresse. Das Speichersubsystem muß den betreffenden Speicherinhalt an den Prozessor liefern oder die vom Prozessor angebotenen Daten in den Speicher schreiben.

Wenn die Adressen vom Prozessor kommen und es nur einen einzigen zusammenhängenden Adreßraum gibt: reicht es nicht aus, die Adreßleitungen einfach mit den Adreßeingängen der Speicherschaltkreise zu verbinden? - In der Tat ist diese einfachste Form der Speicherorganisation von jeher üblich, und viele einfachere Mikroprozessorsysteme sind auch heutzutage so aufgebaut (vgl. die Abbildungen 2.1 und 2.2 in Heft 6). Vorkehrungen zur Speicherverwaltung sind hingegen dann erforderlich, wenn Speicher und Prozessor - in Bezug auf die anwendungs- und systemseitigen Anforderungen - nicht mehr ohne weiteres zusammenpassen:

1. Der Speicher ist - aus der Sicht des Prozessors - zu groß
Man braucht viel Speicherkapazität, der Prozessor liefert aber eine Adresse, die nur einen Teil des gesamten Adreßraums überstreicht. Beispiel: Man will 1 MBytes adressieren, der Prozessor bietet aber nur eine 16-Bit-Adresse an (damit sind nur 64 kBytes adressierbar). Es steht also die Aufgabe, aus einer 16-Bit-Adresse eine 20-Bit-Adresse zu machen, die Adresse des Prozessors in eine (längere) physische Adresse für den (größeren) Speicher umzusetzen (Adreßverlängerung).

2. Sicherheit
Selbst wenn wir einen idealen, gigantisch großen Speicher umsonst hätten: ohne besondere Vorkehrungen ist damit zu rechnen, daß - durch Fehler jeglicher Ursache - Speicherinhalte unzulässigerweise geändert werden, daß Anwendungsprogramme in Systemprogramme eingreifen usw. Die Wahrscheinlichkeit dafür steigt mit der Komplexität des gesamten Systems. Deshalb sind - von einer gewissen Leistungs- und Komplexitätsstufe an - *Schutzvorkehrungen* unumgänglich. Die Prinzipien: (1) vor jedem Zugriff wird geprüft, ob er überhaupt zulässig ist, (2) es werden mehrere unabhängige Adreßräume bereitgestellt. Siehe weiterhin Kapitel 5.

3. Der Speicher ist zu klein

4. Der Speicher ist zu langsam

Beides können wir zusammen behandeln. Die Forderungen an einen idealen Arbeitsspeicher sind (vgl. Heft 6): (1) praktisch unbegrenzte Speicherkapazität, (2) Zugriffszeit nahe 0, (3) akzeptable Kosten.

Es gibt drei wichtige Konzepte, um diese - an sich einander widersprechenden - Forderungen näherungsweise zu erfüllen:

1. das Konzept des virtuellen Speichers,
2. das Konzept des transparenten Schnellspeichers (Caches),
3. das Konzept der Speicherhierarchie, vom Massenspeicher über Arbeitsspeicher und Cache(s) bis zum Register im Prozessor.

3.1.2. Der Weg der Adreßumsetzung

Dieser Weg ist - am Beispiel der IA-32-Architektur - in Abbildung 3.1 dargestellt. Der "eigentliche" Prozessor (die CPU) liefert eine *effektive* Adresse. Daraus wird in der Speicherverwaltung eine *logische* oder *virtuelle* Adresse gebildet. Aus dieser wird wiederum die *physische* Adresse erzeugt, die schließlich zu den Speicherschaltkreisen gelangt. Diese Adreßumsetzungsvorgänge werden durch (in der Abbildung nicht dargestellte) Kontrollschaltungen überwacht.

Abbildung 3.1 Adreßumsetzung in der IA-32-Architektur

3.1.3. Virtuelle Speicher

Was soll ein virtueller Speicher leisten?
Er soll eine Speicherkapazität bereitstellen, die beträchtlich größer ist als die Kapazität des Arbeitsspeichers. Dies wird durch den Verbund von Arbeitsspeicher und Massenspeichern (Festplatten) erreicht - die Vorteile beider Speichertypen werden gleichsam vereinigt: die geringen Kosten je Bit des Massenspeichers mit der geringen Zugriffszeit und mit dem einfachen Zugriffsprinzip*) des Abeitsspeichers.

*): Direktzugriff über eine als Adresse dienende Binärzahl. Verglichen damit ist die Zugriffsorganisation der Dateisysteme viel komplizierter. Deshalb hat man Systemarchitekturen entwickelt, die gar kein Dateisystem im eigentlichen Sinne mehr haben - Dateien sind einfach dauernd bestehende (permanente) Objekte, die in einem riesigen virtuellen Speicher gehalten werden (Beispiel: AS/400).

Wie funktioniert ein virtueller Speicher?
Es gibt 2 Prinzipien, nach denen man eine virtuelle Speicherorganisation verwirklichen kann (Tabelle 3.1):

1. Segmentierung (Segmentation)
Gemäß ihrer funktionellen Gliederung werden Programmstücke, Datenbereiche usw. als Segmente betrachtet. Das einzelne Segment ist jeweils so groß, wie es eben nötig ist (Segmente haben eine variable Länge). Jedes Segment wird durch einen Segmentdeskriptor beschrieben, der die Segmentlänge, die Placierung des Segments (als Startadresse im RAM oder als Massenspeicher-Positionsangabe) sowie Angaben zum Segmenttyp, zu den Zugriffsrechten usw. enthält. Der Zugriff auf ein Segment ist explizit auszuprogrammieren (der Programmierer muß es also ausdrücklich hinschreiben, daß er auf ein bestimmtes Segment

zugreifen möchte). Befindet sich ein Segment, worauf zugegriffen werden soll, nicht im Arbeitsspeicher, so ist es vom Massenspeicher zu holen. Segmentierungsverfahren können ausschließlich mit Software realisiert werden. Eine hardwareseitige Unterstützung ist aber zweckmäßig (Signalisieren von Programmausnahmen, wenn ein Segment als "nicht (im RAM) anwesend" gekennzeichnet ist oder wenn Längen überschritten bzw. Zugriffsrechte verletzt werden).

2. Seitenverwaltung (Paging)
Der gesamte Adreßraum des virtuellen Speichers wird in gleich große Seiten (Pages) eingeteilt, der Arbeitsspeicher in entsprechende Seitenrahmen (Page Frames). Jene Seiten, auf die zugegriffen werden soll, werden in Seitenrahmen des Arbeitsspeichers gebracht; die verbleibenden Seiten befinden sich auf dem Massenspeicher. Die Aufteilung in Seiten bzw. Seitenrahmen nimmt keinerlei Rücksicht auf die logische Struktur bzw. Zusammengehörigkeit der dort untergebrachten Speicherinhalte. Auch hat der (Anwendungs-) Programmierer mit der Seitenverwaltung nichts zu tun (sie ist gegenüber der Anwendungssoftware völlig transparent). Position und Zustand der Seiten werden in mehrstufigen Tabellenstrukturen beschrieben. Die Zugriffe auf diese beschreibenden Angaben müssen durch Assoziativhardware (TLBs) beschleunigt werden.

Die architekturseitigen Vorkehrungen haben gleichsam werkzeughaften Charakter. Sie können verwendet werden, um, gemäß den Anforderungen des jeweiligen Anwendungsfalls, verschiedene Formen der Speicherorganisation (Speichermodelle) zu verwirklichen. Im Sinne eines Beispiels geben wir einen Einblick in die Architektur IA-32, die sowohl die Segmentierung als auch die Seitenverwaltung unterstützt. In den meisten neueren (RISC-) Architekturen gibt es keine Vorkehrungen zur Segmentierung (eine solche Form der Speicherverwaltung müßte also mit elementaren Befehlen ausprogrammiert werden). Die Seitenverwaltung ist aber nahezu immer vorgesehen, wobei sich die Prinzipien von denen der Architektur IA-32 nur in Einzelheiten unterscheiden.

Prinzip	Vorteile	Nachteile
Segmentierung	**P** auch ohne unterstützende Hardware implementierbar, **P** Realzeitverhalten vorhersagbar, **P** umfangreiche und selektive Schutzmaßnahmen implementierbar	**P** Speicherorganisation ist beim Programmieren zu berücksichtigen, **P** komplizierte Speicherverwaltung (erfordert Garbage Collection)
Seitenverwaltung	**P** Speicherorganisation muß beim Programmieren nicht berücksichtigt werden (linearer Adreßraum), **P** einfachere Speicherverwaltung, **P** keine Garbage Collection*) notwendig	**P** TLB-Hardware erforderlich, **P** Programmlaufzeiten (Realzeitverhalten) kaum vorhersagbar, **P** System kann durch Seitenwechsel regelrecht zugestopft werden (Page Thrashing), **P** es lassen sich nur recht globale (also keine anwendungsspezifischen) Schutzmaßnahmen implementieren

Tabelle 3.1 Prinzipien der virtuellen Speicherorganisation

3.1.4. Speicherhierarchie

Die Speicherhierarchie beginnt auf der Ebene des Prozessors beim programmseitig zugänglichen Register. Darauf folgen die Caches, dann der Arbeitsspeicher und schließlich die Massenspeicher. In dieser Reihenfolge wachsen Zugriffszeit und Speicherkapazität (Tabelle 3.2). Die Nutzung der Register obliegt typischerweise dem Programmierer (oder dem Compiler). Caches werden zumeist transparent betrieben, das heißt ohne Beeinflussung seitens der Software. Massenspeicher werden teils über die Virtualspeicherverwaltung, teils über das Dateizugriffssystem genutzt (beides ist Teil der Systemsoftware).

Speicher-mittel	Speicherkapazität (Richtwerte)	Zugriffszeit	(Behälter der Information	Nutzung
Register	16...256 Register zu 32 bzw. 64 Bits (64...2048 Bytes)	wenige ns (Lesen und Schreiben in einem Maschinenzyklus)	Maschinenwort (32 oder 64 Bits)	in den Befehlen direkt adressiert, Vergabe durch Compiler (Verwaltung zur Compilierzeit)
Cache	8 kBytes...einige MBytes	einige ns (ein Zugriff = ein Maschinenzyklus)	Cache-Eintrag (Cache Line); typisch: 16, 32, 64 Bytes	transparent, keine programmseitige Steuerung (hardwaremäßige Verwaltung zur Laufzeit)
Arbeits-speicher	MBytes...GBytes	einige hundert ns bis etwa 1 µs	Adressierung aufs Byte, Zugriffsbreite 4 oder 8 Bytes, Verwaltung: Seitenrahmen (typisch: 4 kBytes)	Adressierung über Speicherverwaltung (Verwaltungsaktivitäten sowohl zur Compilierzeit wie auch zur Laufzeit)
Massen-speicher	etliche GBytes	mehrere ms	Sektoren, (typisch: 512 Bytes)	Zugriff über Systemsoftware

Tabelle 3.2 Die Speicherhierarchie im Überblick

3.1.5. Lokalität

Daß eine solche Speicherhierarchie funktioniert und daß aus dem gut organisiertem Zusammenwirken all ihrer Ebenen ein beträchtlicher Leistungsgewinn folgt, hängt von einer Erfahrungstatsache ab: das gerade laufende Programm braucht aus dem riesigen Adreßraum, den alle Speicher im System letztlich darstellen, immer nur vergleichsweise sehr wenige Angaben, und diese liegen zumeist dicht beisammen.

Beispiele:

- Befehle werden zumeist fortlaufend abgearbeitet,
- Verzweigungen innerhalb eines Programms sind viel häufiger als Verzweigungen zu anderen Programmen,

- zusammenhängende Programmstücke, z. B. Befehlsfolgen zwischen Verzweigungen, und viele Unterprogramme beziehen sich häufig auf so wenige Daten, daß diese in Universalregistersätzen gehalten werden können,
- wenn zu irgendeiner Adresse erst einmal zugegriffen wurde, ist es sehr wahrscheinlich, daß bald wieder darauf zugegriffen werden wird.

Hierzu liegen umfangreiche statistische Erhebungen vor. Computer gibt es seit über 40 Jahren, und -zigtausende von Anwendungen bilden eine beachtliche Erfahrungsbasis. Die modernen Architekturen sind auf Grundlage einschlägiger Messungen optimiert worden. Für eigene Entscheidungen beim Architektur-Entwurf bleibt aber nach wie vor noch ein weiter Ermessens-Spielraum. Auch stellen sich die verschiedenen Anbieter auf bestimmte Marktsegmente ein und streben gewisse Alleinstellungsmerkmale an. So kann es vorkommen, daß ein System, das sich in einem Anwendungsgebiet bewährt hat, bei anderen Anwendungen nur enttäuschende Leistungen erbringt.

3.2. Adreßverlängerung

3.2.1. Weshalb müssen Adressen verlängert werden?

Antwort: Um programmseitige Zugriffe auf einen Adreßraum zu ermöglichen, der so groß ist, daß Adressen in der Länge der Verarbeitungsbreite dazu nicht ausreichen.

Das Problem besteht, seit es möglich ist, - bezogen auf die jeweilige Technologie - vergleichsweise große Speicherkapazitäten vorzusehen. Dabei wurden auch die optimistischsten Architekturentwickler immer wieder von der Speichertechnologie überholt. Als das legendäre System /360 entwickelt wurde, hatte man einen seinerzeit geradezu gigantischen Speicheradreßraum von 16 MBytes vorgesehen. Eine 24-Bit-Byteadresse wurde als vollkommen ausreichend und in absehbarer Zukunft kaum ausschöpfbar angesehen. In der Tat galten 256 kBytes bereits als außergewöhnlich viel. Ein solcher Speicher belegte seinerzeit wenigstens einen Einbaurahmen; für 1 MBytes war ein ganzer Schrank notwendig. Heutzutage läßt sich der gesamte Adreßraum von 16 MBytes mit einem einzigen DRAM-Speicherschaltkreis "erledigen".

Die Anbieter von Mikroprozessoren hatten Ende der 70er Jahre das gleiche Problem, aber aus einem anderen Grund: Die meisten der ersten Mikroprozessoren hatten 16-Bit-Adressen bei 8 Bits Verarbeitungsbreite, und die damit adressierbaren 64 kBytes waren seinerzeit nicht gerade wenig. Beim Übergang auf 16 Bits Verarbeitungsbreite war allerdings von vornherein mit einem wesentlichen Mehrbedarf an Speicherkapazität zu rechnen. Zwei Hindernisse standen einer Radikalkur (Übergang auf 32-Bit-Adressierung) im Weg: Abwärtskompatibilität und technologische Möglichkeiten (Integrationsgrad). Deshalb hat beispielsweise Intel beim 8086 die 16-Bit-Adressierung beibehalten und durch einen einfachen Segmentierungsmechanismus die Adresse auf 20 Bits verlängert (der ursprüngliche Grund für die Segmentierung in der x86-Architektur waren nicht die Schutzvorkehrungen und auch nicht der virtuelle Speicher, sondern eben die Adreßverlängerung unter Einhaltung gewisser Kompatibilitäts-Eigenschaften, bezogen auf die Vorgänger-Typen 8080 und 8085).

Das allgemeine Prinzip
Es ist naturgemäß völlig unmöglich, mit n Adreßbits mehr als 2^n Speicherpositionen vollkommen wahlfrei zu erreichen. So sind bei Byteadressierung mit 16-Bit-Adressen nur 2^{16} = 65 536 Bytes wahlfrei zugänglich. Deshalb kann es lediglich darum gehen, den zugänglichen Adreßraum in den größeren Speicheradreßraum steuerbar abzubilden, sei es als ein einziger Ausschnitt von beispielsweise 64 kBytes, sei es in Form mehrerer kleinerer Ausschnitte. Jede Änderung der einmal eingestellten Abbildung erfordert dann einen programmseitigen Eingriff. Die Abbildungsprinzipien wollen wir im folgenden näher betrachten. Wir wählen dazu als Beispiel eine 16-Bit-Adresse.

3.2.2. Aneinanderreihung (Konkatenation)

Der ursprünglichen 16-Bit-Adresse werden weitere Adreßbits vorangestellt (Abbildung 3.2).

Abbildung 3.2 Adreßverlängerung durch Konkatenation

Erklärung:
Es gibt zwei grundsätzliche Möglichkeiten, die zusätzlichen Adreßbits zu bilden:

1. es werden Angaben zum jeweiligen Zweck des Speicherzugriffs (Befehlszugriff, Stackzugriff, Datenzugriff usw.) hinzugefügt (mit anderen Worten:es werden spezielle Maschinenzustände in die Adreßbildung einbezogen),
2. es werden eigens programmseitig ladbare Register (Bankregister) vorgesehen.

Einbeziehen von Maschinenzuständen
Der Vorteil: Einfachheit. Man braucht praktisch keine zusätzliche Hardware, und im normalen Betrieb sind keine zusätzlichen programmseitigen Aktivitäten erforderlich.

Die Unterscheidung zwischen Befehls- und Datenzugriffen (Abbildung 3.2a) verlängert die Adresse um 1 Bit[*)]. Dies ist vor allem bei einfacheren Mikrocontrollern gebräuchlich (Beispiel: der weit verbreitete Typ 8051). In Erweiterung des Prinzips läßt sich zwischen Befehls-, Daten-, Stackzugriffen usw. unterscheiden, darüber hinaus auch zwischen verschiedenen Zuständen (z.B. Anwenderzustand, Systemzustand, Unterbrechungsbehandlung (Abbildung 3.2b)). Dies wurde von den 60er Jahren an in verschiedenen Architekturen verwirklicht (ist aber heutzutage etwas aus der Mode gekommen)[**)]. Das Problem: die Kommunikation zwischen den Teil-Adreßräumen (die in allen komplexeren Nutzungsfällen möglich sein muß - es sind u. a. Parameter aus dem Stack zu holen, es sind sowohl Anwendungs- als auch Systemprogramme zu laden usw.). Dies erfordert besondere Befehle[**)] oder externe Zusatzhardware.

*): macht aber aus der v. Neumann-Maschine eine Harvard-Maschine (Heft 5).

**): ein vergleichsweise neueres Beispiel: die Sparc-Architektur. Die Adreßverlängerung heißt hier Address Space Identifier (ASI). Die Prozessoren liefern u. a. zwei Bits, die

den aktuellen Maschinenzustand bzw. den Zweck des Zugriffs kennzeichnen, und es gibt Speicherzugriffsbefehle, die sich auf einen - jeweils anzugebenden - alternativen Adreßraum beziehen (vgl. Abschnitt 9.2., im besonderen die Befehlsübersicht).

Bankregister
Ladbare Bankregister ermöglichen es, die Adresse praktisch beliebig zu verlängern (Abbildung 3.2c).

Kombinationslösungen
Hiervon gibt es eine große Vielfalt. Wir wollen nur 2 Ausführungen kurz vorstellen:

- einige Bitpositionen der ursprünglichen Adresse werden dazu genutzt, Bankregister auszuwählen (Abbildung 3.2d). Typische Auslegungen: (1) es gibt nur ein Bankregister, das bei bestimmten Adreßbelegungen zugeschaltet wird*), (2) n Bitpositionen der ursprünglichen Adresse wählen eines von maximal 2^n Bankregistern aus (Adreßumsetzung).
- jedem Maschinenzustand ist ein gesondertes Bankregister zugeordnet (Abbildung 3.2e).

*): bei allen anderen Adreßbelegungen wird die ursprüngliche Adresse unverändert weiterverwendet (solche Zugriffe heißen “unbanked”).

Adreßverlängerung im PC
Im PC wird vom Prinzip der Adreßverlängerung an mehreren Stellen Gebrauch gemacht. Wichtige Anwendungsfälle werden in den Abschnitten 3.2.6. bis 3.2.8. kurz beschrieben.

3.2.3. Größere adressierbare Informationseinheiten

Wenn die Adreßangabe nicht für Bytes, sondern z. B. für 32-Bit- oder 64-Bit-Worte vorgesehen wird, kann man entsprechend mehr Speicherkapazität adressieren. Mit einer 16-Bit-Adresse erreicht man beispielsweise 64 k 64-Bit-Worte, das entspricht 512 kBytes. Stehen so adressierte Worte erst einmal in Prozessorregistern, kann man zu beliebigen Bytes programmseitig zugreifen (eine solche Architektur könnte beispielsweise Registertransportbefehle für einzelne Bytes, für Halbworte usw. vorsehen). Dieses Prinzip stand zur Diskussion, als man begann, Architekturen mit vereinfachtem Befehlssatz (-RISC-Maschinen) zu entwickeln. Die Praxis hat aber gezeigt, daß es, zumindest für die Datenadressierung, weniger zweckmäßig ist. Deshalb haben die RISC-Architekturen, die heutzutage am Markt sind, Byteadressierung. Zwei Anwendungsbeispiele:

1. die DMA-Adressierung der PCs (DMA-Kanäle 5...7; Abschnitt 3.2.6.),
2. Verzweigungsadressen der Sparc-Architektur. Die bedingten Verzweigungsbefehle haben ein 22-Bit-Displacementfeld, das keine Byte-, sondern eine Befehlsadresse enthält. (Jeder Befehl ist 4 Bytes lang. Deshalb wird die Displacementangabe vor der Verrechnung mit 4 multipliziert. Damit ergibt sich ein Displacement (als Byteadresse) von 24 Bits Länge.)

3.2.4. Addition

Die ursprüngliche 16-Bit-Adresse wird zu einer - längeren - Basisadresse addiert. Wie bei der Konkatenation kann man getrennte Basisadressen für verschiedene Zugriffe bzw. Maschinenzustände vorsehen.

Vorteil: Der zugängliche Adreßraum-Ausschnitt kann an beliebiger Stelle des Speicheradreßraumes angeordnet werden (Aufteilung ist nicht so starr wie bei der Konkatenation).

Nachteil: Die Addition ist aufwendiger und langsamer als die Konkatenation.

Anwendungsbeispiel: Segmentierung in der x86-Architektur
In der x86-Architektur wird aus 16-Bit-Angaben eine 20-Bit-Adresse erzeugt (Abbildung 3.3). Die Basisadressen werden in vier Segmentregistern (CS, SS, DS, ES) bereitgestellt. Ein Segmentregister enthält die höchstwertigen 16 Bits der 20-Bit-Adresse; die verbleibenden 4 Bits sind fest mit Null belegt. Dazu wird die programmseitige 16-Bit-Adresse addiert. Segmente sind somit stets 64 kBytes lang und beginnen stets an integralen 16-Byte- (durch 16 teilbaren) Adressen.

Der Paragraph
Ein solcher 16-Byte-Block an einer integralen Adresse wird als Paragraph bezeichnet; die Länge von Speicherbereichen, Programmen usw. wird gelegentlich in Paragraphen angegeben.

Segmentregisterauswahl
Je nach Art des Zugriffs wird eines der Segmentregister ausgewählt. Für Datenzugriffe sind alle Segmentregister nutzbar. Die Segmentregister (außer CS) sind mit besonderen Befehlen ladbar (CS wird bei Verzweigungen, Unterbrechungen usw. automatisch geladen).

Abbildung 3.3 Adreßverlängerung durch Segmentierung (x86)

3.2.5. Umsetzung

Man kann eine beliebige binär codierte Angabe a in eine beliebige andere binäre Angabe b umsetzen, wenn man a als Adresse für einen Direktzugriffsspeicher verwendet, der für jeden Wert von a den zugeordneten Wert von b enthält (Abbildung 3.4).

Abbildung 3.4 Adreßumsetzung über Umsetzungsspeicher

Erklärung:

a) vollständige Umsetzung. Beispielsweise wird eine 16-Bit-Adresse einem Umsetzungsspeicher zugeführt, der 64k Einträge zu 32 Bits hat und somit eine 32-Bit-Adresse abliefern kann,

b) teilweise Umsetzung. Im Beispiel werden die 4 höchstwertigen Adreßbits einem Umsetzungsspeicher zugeführt, der 16 Einträge zu 12 Bits hat und hiermit die verbleibenden 12 Bits der ursprünglichen Adresse auf eine 24-Bit-Adresse verlängert.

Vorteil: Man kann grundsätzlich jede beliebige programmseitige Adresse in eine beliebige andere (längere) wandeln.

Nachteil: Hardware- und Verwaltungsaufwand.

Anwendung:
In reiner Form (Abbildung 3.4a) wird das Prinzip kaum eingesetzt. Ein Beispiel für eine Lösung ähnlich Abbildung 3.4b: Expanded Memory (Abschnitt 3.2.8.). Ansonsten werden Adreßumsetzungen typischerweise mittels Software erledigt, wobei nur deren Aktivierung von Hardware unterstützt wird - die Hardware prüft, ob die jeweils anliegende Adresse eine Umsetzung erfordert oder nicht. Man spricht hierbei auch vom *Abfangen* der Zugriffe (Heft 6).

Unterstützung in der Architektur IA-32:

- über die E-A-Erlaubnisbitliste. Diese enthält für jede der 64k E-A-Adressen ein Bit. Ist ein solches Bit gesetzt, so werden die betreffenden E-A-Zugriffe nicht ausgeführt, sondern es wird jedesmal eine Ausnahmebedingung wirksam. Mit anderen Worten: die Zugriffe werden abgefangen (Heft 6).
- über die Seitenverwaltung (Abschnitt 3.5.). Die Seitenverwaltung wird vielfältig ausgenutzt: zum Abfangen von E-A-Zugriffen über den Speicheradreßraum (Memory Mapped I/O), zur Steuerung des Caching und zur Verlängerung der Adresse auf 36 Bits. Näheres in Heft 6 sowie (Adreßverlängerung) in Abschnitt 3.5.5.

3.2.6. Anwendungsbeispiel (1): DMA-Adressierung

Adreßverlängerung über Bankregister
Die DMA-Hardware beruht auf zwei DMA-Steuerschaltkreisen 8237, die 16-Bit-Adressen liefern können. Diese werden durch weitere Adreßbits aus Bankregistern ergänzt (Abbildung 3.5).

Abbildung 3.5 Die DMA-Hardware eines AT-kompatiblen PCs (Intel)

Erklärung:
Jeder DMA-Schaltkreis liefert - unter Zwischenschaltung eines Latch-Registers*) - eine 16-Bit-Adresse (die Adreßsignale beider Schaltkreise (A16...A0) werden gemeinsam auf den ISA-Bus geschaltet). Der mit Memory Mapper**) bezeichnete Funktionsblock enthält je DMA-Kanal ein 8-Bit-Bankregister, das die Adreßbits 23...17 liefert. Die einzelnen Bankregister können über E-A-Zugriffe umgeladen werden.

*): Abbildung 5.8 in Heft 6 veranschaulicht, wozu dieses Register dient.

**): der ursprüngliche Schaltkreistyp: 74LS612 - ein Spezialschaltkreis zur Adreßumsetzung, der an sich für Anwendungen gemäß Abbildung 3.4b vorgesehen ist. (Er enthält 16 adressierbare Bankregister zu 12 Bits. Im PC wird das Leistungsvermögen dieses Schaltkreises nicht voll ausgenutzt.)

Hinweis:
Die Bankregister heißen hier gelegentlich auch Seitenregister (Page Registers). Das hat aber mit der eigentlichen Seitenverwaltung (Abschnitt 3.5.) nichts zu tun!

Größere adressierbare Informationseinheiten
Die DMA-Kanäle 5, 6 und 7 sind für 16-Bit-Übertragungen vorgesehen, so daß man mit einer 16-Bit-Adresse jeweils 128 kBytes überstreichen kann. Aus Abbildung 3.5 ist die technische Lösung ersichtlich: die 16 Adreßbits, die der Schaltkreis DMA2 liefert, sind auf die Adreßleitungen A16...1 geschaltet (und nicht auf A15...0), liefern also praktisch eine Adresse für 16-Bit-Worte an integralen Grenzen (Adreßbit 0 wird in der Hardware als Null angenommen).

Teilweise Adreßumsetzung
Dies ist im Rahmen der verteilten DMA-Steuerung vorgesehen (Distributed DMA; siehe Heft 6): die "industriestandardmäßigen" E-A-Adressen der herkömmlichen DMA-Hardware (gemäß Abbildung 3.5) werden vom Steuerschaltkreis in E-A-Adressen der Schaltkreise umgesetzt, in denen sich die Hardware des jeweiligen DMA-Kanals tatsächlich befindet.

3.2.7. Anwendungsbeispiel (2): Bildspeicheradressierung

Herkömmlicherweise steht dem Bildspeicher nur ein Adreßraum von 64 kBytes (vorzugsweise) oder 128 kBytes (maximal) zur Verfügung, es müssen aber wenigstens 256 kBytes adressiert werden (bei Super-VGA-Karten 1 MBytes und mehr). Die Graphikkarten enthalten hierfür Bankregister, die über E-A-Zugriffe geladen werden können. Abbildung 3.6 veranschaulicht einige Lösungen der Adreßverlängerung.

Abbildung 3.6 Adreßverlängerung zur Bildspeicheradressierung (Beispiele)

Erklärung:

a) Konkatenation. Die 16 niederwertigen Adreßbits kommen vom Prozessor. Sie werden aus einem Bankregister mit entsprechend vielen höherwertigen Adreßbits ergänzt. 3 Bits bilden eine 19-Bit-Adresse (für 512 kBytes), 4 Bits eine 20-Bit-Adresse (für 1 MBytes) usw. (Beispiele: ATI, Genoa). Manche Graphikschaltkreise haben 2 derartige Bankregister: eines für Lese- und eines für Schreibzugriffe.

b) Konkatenation. Es sind 2 Bankregister vorgesehen. Adreßbit 15 vom Prozessor wählt aus, welches der beiden Bankregister verwendet wird (BS = Bank Select). Das ausgewählte Bankregister liefert die höherwertigen 6 Adreßbits, der Prozessor die niederwertigen 15. Hiermit wird eine 21-Bit-Adresse gebildet, mit der ein Bildspeicher von maximal 2 MBytes adressiert werden kann (Beispiel: ATI).

c) Addition. Es werden die 19 niederwertigen Adreßbits vom Prozessor ausgewertet (die Bits 18...16 sind dabei praktisch fest belegt). Zu den Bitpositionen 18...12 werden 7 Bits eines Bankregisters addiert, um eine 20-Bit-Bildspeicheradresse (für 1 MBytes) zu bilden (Beispiel: Western Digital/Paradise).

Hinweis:
Jeder Graphikschaltkreis hat seine Besonderheiten. Die Einarbeitung ist sehr mühevoll (zumal nur zu wenigen Schaltkreistypen die genaue Dokumentation ohne weiteres zugänglich ist). Falls Sie mit Graphikprogrammierung experimentieren wollen - in üblichen PC-Umgebungen gibt es 2 bequemere Zugänge: (1) über das VGA-BIOS (unter DOS), (2) über die API-Funktionen des Graphic Display Interface GDI (Windows) bzw. über die jeweilige Entwicklungsumgebung.

3.2.8. Anwendungsbeispiel (3): Expanded Memory

Wir erinnern uns: beim herkömmlichen PC mit x86-Prozessor ist der Speicheradreßraum auf 1 MBytes beschränkt. Wenn wir mehr Speicher adressieren wollen, müssen wir die 20-Bit-Adresse irgendwie verlängern. Dafür hat sich ein Industriestandard durchgesetzt, der als EMS-Standard (Expanded Memory Specification) bezeichnet wird. Dieser Standard gibt keine Schaltung vor, sondern spezifiziert die Funktionen aus der Sicht des Programmierers, er beschreibt eine Software-Schnittstelle (zum Expanded Memory Manager EMM). Wir wollen im folgenden aber ausschließlich das Prinzip der Adreßabbildung betrachten. Im Interesse der Einfachheit beschränken wir uns auf Version 3.2 des EMS-Standards.

Hinweis:
Expanded Memory ist heutzutage eigentlich nur noch von Bedeutung, wenn ältere Software weitergenutzt werden soll, die nicht ohne weiteres zu ersetzen ist.

Zur Ausdrucksweise: die EMS-Spezifikation benutzt die Begriffe "Seite" (Page) und "Seitenrahmen" (Page Frame). Zwecks Abgrenzung gegenüber der "eigentlichen" Seitenverwaltung (Abschnitt 3.5.) werden hier die Begriffe in Schrägschrift (*kursiv*) wiedergegeben.

Prinzip
Es werden nur Ausschnitte des Adreßraums umgesetzt (teilweise Umsetzung gemäß Abbildung 3.4b). Der erweiterte Speicher ist je nach Größe in eine Anzahl aufeinanderfolgender *Seiten* von 16 kBytes Länge eingeteilt (die *Seiten* sind an integralen 16k--Grenzen angeordnet). Im adressierbaren Teil des Speichers (in den untersten 1 MBytes) sind 64 kBytes als *Seitenrahmen* reserviert, und zwar an einer integralen 64k-Grenze. Der *Seitenrahmen* kann 4 *physische Seiten* aufnehmen. Der Erweiterungsspeicher kann bis zu 8 MBytes groß sein. Er enthält *logische Seiten*. Es ist nicht so, daß die logische *Seite* durch Software in die jeweilige physische kopiert wird. Dazu müßte man den Prozessor im

Protected-Modus betreiben (Abschnitt 9.1.2.). Beim 8088 bzw. 8086 geht das ohnehin nicht, und vom 286 an wäre das Hin- und Hertransportieren viel zu zeitaufwendig. Deshalb ist eine hardwaremäßige Adreßumsetzung unbedingt notwendig. Um zu einem Byte im Erweiterungsspeicher zuzugreifen, braucht man eine 23-Bit-Adresse. Die 20-Bit-Adresse des PC wird hierfür folgendermaßen interpretiert:

- die niedrigstwertigen 14 Bits wählen ein Byte in einer *Seite* von 16 kBytes aus,
- zwei weitere Adreßbits bestimmen einen von 4 Einträgen des Umsetzungsspeichers, den man als eine adressierbare Anordnung aus 4 9-Bit-Bankregistern ansehen kann*),
- die verbleibenden 4 Adreßbits der PC-Adresse werden bei jedem Zugriff mit der fest eingestellten 64k-Adresse des Seitenrahmens verglichen. Bei Gleichheit wird ein Zugriff zum Erweiterungsspeicher ausgeführt, wobei die 23-Bit-Adresse aus 14 Bits vom Prozessor und aus den 9 Bits des ausgewählten Bankregisters zusammengesetzt ist. Abbildung 3.7 veranschaulicht das Umsetzungsprinzip.

*): der Umsetzungsspeicher gehört zur Hardware, die die EMS-Funktionen unterstützt (die Bankregister sind z. B. in einem der Motherboard-Steuerschaltkreise oder auf Speichererweiterungs-Steckkarten angeordnet)

Die jeweilige Adreßumsetzung ist programmseitig einzustellen (durch Ruf des Expanded Memory Manager EMM). Dann gewährleistet die Hardware, daß die Software bei Adressierung einer physischen *Seite* im *Seitenrahmen* auf die zugeordnete logische *Seite* zugreift (zum *Seitenrahmen* im untersten Megabyte wird technisch gar nicht zugegriffen, vielmehr wird jeder Zugriff mit einer *Seitenrahmen*adresse zur jeweiligen logischen *Seite* umgelenkt (es wird also nicht das unterste Megabyte, sondern der Erweiterungsspeicher angesprochen).

Abbildung 3.7 Adreßumsetzung beim PC: Expanded Memory

3.3. Speicherbereiche

Das Speichersubsystem umfaßt typischerweise verschiedenartige Speicherbereiche (ROM, Arbeitsspeicher, Bildspeicher, über Speicherzugriffe zugängliche E-A-Einrichtungen usw.). Jeder dieser Speicherbereiche ist durch besondere Zugriffsbedingungen gekennzeichnet, also durch Organisationsform, Art der Speicherschaltkreise, Zugriffsbreite, Zugriffsmöglichkeiten und Zugriffszeiten (einzufügende Wartezustände) sowie durch den jeweils zugeordneten Ausschnitt des Adreßraums (Adreßbereich) und durch die Besonderheiten, die beim Einsatz von Caches zu beachten sind.

3.3.1. Herkömmliche Lösungen

Die Maschinenbefehle der heutzutage üblichen Prozessoren “sehen” nur einheitliche Adreßräume ohne jede technische Nebenbedingung (es gibt keine besonderen Befehle zum Zugreifen auf den ROM, auf den Bildspeicher usw.). Die Zuordnung der Zugriffsadresse

zum jeweils technisch ausgeführten Speicherbereich und die Steuerung der Zugriffsabläufe sind vielmehr Angelegenheit von Schaltmitteln außerhalb des Prozessors.

Adreßdecodierung
Die herkömmliche Lösung. An die Adreßleitungen des jeweiligen Bussystems sind Decodierschaltungen angeschlossen, die erkennen*), zu welchem Speicherbereich jeweils zugegriffen werden soll. Demgemäß werden die jeweiligen Steuerschaltungen aktiviert, es werden erforderlichenfalls Wartezustände eingefügt usw.

*): z. B. durch Vergleich der angebotenen Adresse mit einer Bereichsangabe.

Zentrale Decodierung
Es gibt eine einzige Einrichtung zur Adreßdecodierung, an die alle Speicheranordnungen, E-A-Schaltkreise usw. angeschlossen sind. Anwendungsbeispiele:

- einfache Systeme auf Grundlage von Mikrocontrollern und Mikroprozessoren (gelegentlich ist die Adreßdecodierung in den Prozessorschaltkreis eingebaut - vgl. Abbildung 2.1 in Heft 6),
- Motherboards. Die Steuerschaltkreise enthalten entsprechende Decodierschaltungen und programmseitig ladbare Bereichsregister.

Dezentrale Decodierung
Jede Einrichtung erkennt sich gleichsam selbst. Hierzu wird allen Einrichtungen die gesamte Adresse angeboten. Anwendung: dies ist das typische Prinzip der Adreßdecodierung in universellen Bussystemen (ISA, PCI usw.).

Varianten der Bereichszuordnung und Konfigurationseinstellung:

1. feste bzw. manuell vorzunehmende Zuordnung/Einstellung. Beispiele: (1) einfache Systeme mit Mikrocontrollern usw., (2) der "klassische" PC (XT, AT) mit ISA-Bus,
2. variable (programmseitige) Zuordnung/Einstellung auf Grundlage gespeicherter Konfigurationsdaten. Beispiele: (1) die Bussysteme MCA und EISA, (2) typische Arbeitsspeichersubsysteme auf Motherboards,
3. variable (programmseitige) Zuordnung/Einstellung auf Grundlage gespeicherter Anforderungen (diese werden typischerweise in den einzelnen Einrichtungen selbst gespeichert). Beispielsweise gibt eine Steckkarte an, wie groß der Ausschnitt aus dem Speicheradreßraum sein muß, den sie benötigt, nicht aber, an welcher Adresse dieser Bereich beginnen soll. Die Bereichszuordnung selbst wird von der Software gleichsam ausgerechnet. Beispiele: (1) ISA gemäß Plug&Play-Spezifikation, (2) PCI, (3) AGP.

3.3.2. Memory Type Range Registers (MTRRs)

Die Speicherbereichszuordnung aus der Prozessorarchitektur herauszuhalten, führt bei modernen Hochleistungsprozessoren zu besonderen Problemen - und zwar vor allem deshalb, weil Steuerangaben, die mit den Speicherbereichen zusammenhängen, im Innern des Prozessorschaltkreises benötigt werden. Diese Angaben betreffen die Nutzung der

Caches sowie Maßnahmen der Zugriffsbeschleunigung (vorbeugendes Lesen, Zugriffe außerhalb der Reihe usw.).

So hat beispielsweise Intel von den P6-Prozessoren an entsprechende Vorkehrungen in die Architektur (IA-32) aufgenommen. Die Lösung: es ist eine Anzahl programmseitig zugänglicher Register vorgesehen, über die der Speicheradreßraum in Bereiche verschiedenen Typs gegliedert werden kann (Memory Type Range Registers (MTRRs)).

Speicherbereiche
Der physische Adreßraum von maximal 2^{36} = 64 GBytes (Adreßverlängerung; vgl. Abschnitt 3.5.5.) kann in maximal 96 Bereiche eingeteilt werden (Abbildung 3.8). Die Einteilung ist teils fest (88 Bereiche), teils veränderlich (8 Bereiche). Den verbleibenden (nicht von den MTRRs erfaßten) Bereichen kann eine weitere gemeinsame Typangabe zugewiesen werden (vgl. Abbildung 3.9c).

Abbildung 3.8 Einteilung des Speicheradreßraums über MTTRs

Hinweis:
Diese Einteilung nimmt offensichtlich auf die Belange der herkömmlichen PCs besondere Rücksicht - das berühmte erste MByte kann besonders feinstufig eingeteilt werden.

Feste Bereiche
Es sind 11 Bereichsregister (Fixed Range MTRRs) zu je 64 Bits vorgesehen (Abbildung 3.9a). Jedes Register enthält acht 8-Bit-Felder mit Speichertypangaben, wobei jedes Feld einem festen Adreßbereich zugeordnet ist (die Zuordnung steht im Architekturhandbuch).

Veränderliche Bereiche
Je Bereich sind 2 Register (Variable Range MTRRs) zu je 64 Bits vorgesehen, die den jeweiligen Bereich nach Anfangsadresse, Länge und Typ beschreiben (Abbildung 3.9b).

Allgemeine Steuerung
Hierzu ist ein weiteres 64-Bit-Register vorgesehen, das MTRRdefType-Register (Abbildung 3.9c).

Abbildung 3.9 MTRR-Registerstrukturen (Überblick)

Erklärung:

a) MTRR für feste Bereiche mit 8 Typangaben,
b) MTRR-Paar für einen variablen Bereich. Die Bereichsgrenzen müssen an integralen 4k-Grenzen liegen. Die Register enthalten 24 Bits lange Angaben, aus denen, wie dargestellt, Adreßstrukturen von 36 Bits Länge gebildet werden (die 12 niederwertigen Bits sind fest Null). V - Gültigkeitsbit (Validity).
c) Steuerregister (MTRRdefType). Enthält eine Typangabe für jene Speicherbereiche, die nicht von den MTRRs erfaßt werden. E - MTRRs einschalten (Enable); FE - MTRRs der festen Bereiche einschalten.

Typangaben
Diese werden in den 8-Bit-Feldern der MTRRs codiert (Tabelle 3.3). Es sind lediglich die Belegungen 00H, 01H, 04H, 05H, 06H zulässig.

Bezeichnung	Einlagerung in Caches	Cache-Schreibverfahren Write Back	voreilendes (spekulatives) Lesen	Reihenfolge der Speicherzugriffe
Uncacheable (UC)	nein	nein	nein	strikt gemäß Absicht des Programmierers
Write Combining (WC)	nein	nein	ja	Zusammenfassung von Zugriffen zulässig, Adreßreihenfolge und Cache-Kohärenz nicht garantiert
Write Through (WT)	ja	nein	ja	vom Prozessor bestimmt (spekulativ)
Write Protected (WP)	Lesen: ja, Schreiben: nein	nein	ja	
Write Back (WB)	ja	ja	ja	

*): Cacheability; betrifft alle Cache-Ebenen (L1, L2)

Tabelle 3.3 Typangaben in MTRRs (P6-Prozessoren)

Hinweis:
Ersichtlicherweise enthalten die MTRRs keine Angaben, die technische Fragen des Speichersubsystems betreffen - die Adreßdecodierung zum Ansprechen der einzelnen Speicherbereiche ist nach wie vor außerhalb des Prozessorschaltkreises zu erledigen.

Die Seitenattributtabelle (Page Attribute Table PAT)
Ein Architekturmerkmal, das mit den P6-Prozessoren eingeführt wurde und ähnlich wirkt wie die MTRRs, allerdings in Verbindung mit der Seitenverwaltung (siehe Abschnitt 3.5.5.).

3.4. Die Segmentierung im Protected-Modus (IA-32)

Segmente bilden die Grundlage der Speicheradressierung. Das einzelne Segment ist eine in sich bis auf's Byte adressierbare Ganzheit. Die Länge eines Segments ist zwischen einem Byte und maximal 4 GBytes wählbar. Das jeweils laufende Programm hat über die Segmentregister gleichzeitigen Zugriff auf bis zu sechs Segmente (1 Programmsegment, 1 Stacksegment, 4 Datensegmente). Das einzelne Segment erscheint dem Programmierer praktisch als ein zusammenhängender, von Null an adressierbarer separater Speicher. Die Lage des Segments im linearen Adreßraum wird durch den zugehörigen Segmentdeskriptor beschrieben. Die Deskriptoren der sechs aktuell zugänglichen Segmente stehen in Deskriptor-Cache-Registern der Segmentierungseinheit zur Verfügung, so daß praktisch kein Zeitverlust auftritt, um die tatsächliche Speicheradresse auszurechnen und um die Berechtigung des Zugangs zu prüfen. Die Informationsstrukturen der Segmentierung kennen wir aus Heft 1 (Abschnitt 1.5.). Hier folgt ein Einblick in deren Nutzungsweise.

Segmentselektoren
Um ein Segment zur Nutzung auszuwählen, muß ein entsprechender Segmentselektor in das jeweilige Segmentregister geladen werden.

Deskriptortabellen
Die Segmentdeskriptoren sind in Deskriptortabellen zusammengefaßt. Eine Deskriptortabelle kann bis zu 8k Deskriptoren von 8 Bytes Länge enthalten. Die 13-Bit-Indexangabe im Segmentselektor ist praktisch die Ordinalzahl des gewünschten Deskriptors.

Dieses Zugriffsschema wird auch verwendet, um kontrollierte Eintrittspunkte in Programme bzw. Tasks vorzusehen. Dazu können die Deskriptortabellen neben Segmentdeskriptoren auch Gate-Deskriptoren aufnehmen. Des weiteren verwendet das Interruptsystem eine ähnlich aufgebaute und adressierte Interruptdeskriptortabelle IDT (vgl. Abschnitt 4.3.). Die IDT enthält jedoch ausschließlich Interrupt Gate-, Trap Gate- bzw. Task Gate-Deskriptoren.

Globale Deskriptortabelle
Die globale Deskriptortabelle GDT wird direkt durch das Systemadressenregister GDTR adressiert, das die lineare Anfangsadresse und die Länge der GDT enthält. Es ist seinerseits mit einem besonderen Befehl ladbar.

Lokale Deskriptortabelle
Die lokale Deskriptortabelle LDT ist ihrerseits ein Segment, das in der GDT beschrieben ist. Der aktuelle Segmentselektor wird im Systemsegmentregister LDTR gehalten. Er kann direkt geladen werden. Der aktuelle LDT-Selektor einer Task steht im Taskzustandssegment TSS und wird bei der Taskumschaltung automatisch in das LDTR gebracht (jede Task kann somit eine andere LDT haben). Das LDTR ist, wie die anderen Segmentregister, um ein Deskriptor-Cache-Register erweitert.

Abschaltbarkeit
Die Segmentierung ist *nicht abschaltbar*. Man kann allerdings die Segmentparameter so wählen, daß sich der Effekt einer abgeschalteten Segmentierung ergibt. Die Segmentdeskriptoren erlauben es, dem einzelnen Segment den Umfang des gesamten linearen Adreßraums (4 GBytes) zu geben (das ist das sog. "flache" Speichermodell; Abschnitt 3.6.). Damit verhält sich IA-32 wie viele andere Architekturen mit linearem 32-Bit-Adreßraum.

Schutzvorkehrungen der Segmentierung
Wenn ein Segment angesprochen wird, laufen verschiedene Prüfungen ab:

a) Prüfungen beim Auswählen eines Segments (Laden eines Segmentregisters):

- Längenprüfung. Alle Bytes des ausgewählten Deskriptors müssen in den Grenzen der Deskriptortabelle liegen. Dazu wird der Index aus dem Segmentselektor mit der Längenangabe der Deskriptortabelle verglichen.
- Berechtigungsprüfung. Typ- und Privilegangaben im Segmentdeskriptor werden gegen die Art des Zugriffs und die aktuelle Privilegebene geprüft.

- Anwesenheitsprüfung. Es wird geprüft, ob das betreffende Segment im Speicher anwesend ist oder nicht (dazu wird das P-Bit im Segmentdeskriptor ausgewertet).

b) Prüfungen bei Zugriffen auf Segment-Inhalte:

- Berechtigungsprüfung. Die Art des aktuellen Zugriffs (Holen von Befehlen, Lesen von Daten, Schreiben) wird gegen die Typangabe im Deskriptor geprüft.
- Längenprüfung. Die effektive (Offset-) Adresse wird mit dem Längenwert verglichen, der sich aus dem Segmentdeskriptor ergibt.

Virtualspeicherorganisation auf Grundlage der Segmentierung
Segmente können im Deskriptor als "nicht anwesend" gekennzeichnet sein (das sog. Presence-Bit P - vgl. Abbildung 1.39 - ist nicht gesetzt). Wird versucht, auf ein solches Segment zuzugreifen, so wird eine Ausnahmebedingung wirksam. Der Befehl, der diese Ausnahmebedingung bewirkt hat, wird nicht ausgeführt, und alle Änderungen (z. B. von Registerinhalten), die der Befehl bereits vorgenommen hat, werden automatisch wieder rückgängig gemacht. Somit kann nach der Rückkehr aus der Ausnahmebehandlung derselbe Befehl nochmals abgearbeitet werden, diesmal aber mit anwesendem (vom Ausnahmebehandler geladenem) Segment.

So wird es möglich, Segmente auf Externspeichern (z. B. Magnetplatten) zu halten und nur bei Bedarf in den Arbeitsspeicher zu laden. In den Deskriptoren nicht anwesender Segmente sind 7 Bytes nutzbar, um beispielsweise die Position des Segments auf einer Festplatte zu bezeichnen. Das Behandlungsprogramm muß das gewünschte Segment in den Arbeitsspeicher bringen und dessen Deskriptor entsprechend umladen. Die Rückkehr aus der Ausnahmebehandlung bewirkt den Wiederanlauf des betreffenden Befehls, der nunmehr das Segment als anwesend vorfindet und somit normal beendet wird.

Adressierungsvermögen der Segmentierung
Die GDT kann bis zu 8 191 Segmentdeskriptoren aufnehmen und die LDT bis zu 8 192, insgesamt sind also bis zu 16 383 Segmente gleichzeitig zugänglich. Jedes Segment kann bis zu 4 GBytes lang sein. Damit liegen (8k-1) · 8k · 4 GBytes ≈ 64 Terabytes (genau: 65 532 GBytes) theoretisch im unmittelbaren Zugriffsbereich der Segmentierung.

Diese 64 TBytes werden gelegentlich in der Werbung herausgestellt. Über den herkömmlichen Prozessorbus kann man aber nicht mehr als 4 GBytes wirklich adressieren[*)]. Die praktische Nutzung der 64 TBytes erfordert Kunstgriffe in Hard- und Software, die beim heutigen Stand der Technik niemand implementiert hat. Auch die modernsten Betriebssysteme beschränken sich darauf, die Segmentierungsvorkehrungen zu nutzen, um "flache" Speicheradreßräume von 2...4 GBytes zu organisieren.

*): die Adreßverlängerung auf 36 Bits - bei manchen P6-Typen - hat man über die Seitenverwaltung implementiert (Abschnitt 3.5.5.).

3.5. Seitenverwaltung (Paging)

3.5.1. Grundlagen

Im folgenden beschreiben wir die Prinzipien der Seitenverwaltung am Beispiel IA-32 (wobei wir Einzelheiten außer acht lassen)*). Andere Architekturen unterscheiden sich nur in einigen Parametern (z. B. in der Seitengröße); das Prinzip ist aber letztlich das gleiche.

*): wir betrachten - der Überschaubarkeit wegen - zunächst die herkömmliche Auslegung, wie sie mit dem 386 eingeführt wurde. Weiterentwicklungen beschreiben wir in Abschnitt 3.5.5.

Ist die Seitenverwaltung aktiv, so werden die linearen Speicheradressen nicht unmittelbar zu den Adressenleitungen des Bus-Interfaces durchgeschaltet, sondern es findet eine Adreßumsetzung statt. Dabei wird der gesamte lineare Adreßraum von 4 GBytes in 1 M (2^{20}) Seiten zu je 4 kBytes zerlegt. Der eigentliche verfügbare Arbeitsspeicher wird in Seitenrahmen zu 4 kBytes unterteilt, so daß jede beliebige Seite in jedem beliebigen Seitenrahmen untergebracht werden kann. Die Seitenverwaltungseinheit (Paging Unit) gewährleistet, daß:

- zu jeder Seite, die sich im Arbeitsspeicher befindet, mit entsprechenden linearen Adressen korrekt zugegriffen werden kann, gleichgültig in welchem Seitenrahmen sie gerade untergebracht ist,
- bei Zugriffen zu einer Seite, die sich nicht im Arbeitsspeicher befindet, eine Ausnahme signalisiert wird,
- der Betriebssoftware Gelegenheit gegeben wird, die gewünschte Seite in den Arbeitsspeicher zu bringen,
- die Betriebssoftware entscheiden kann, welche Seite aus dem Arbeitsspeicher zu entfernen ist, um Platz für eine neue zu schaffen,
- nach dem Bereitstellen der Seite im Arbeitsspeicher der betreffende Befehl korrekt ausgeführt werden kann.

3.5.2. Adreßumsetzung

Die Abbildung von der linearen zur physischen Adresse bzw. das Feststellen, daß die betreffende Seite nicht im Arbeitsspeicher verfügbar ist, wird über eine zweistufige Tabellenstruktur verwirklicht (Abbildung 3.10).

Abbildung 3.10 Prinzip der Adreßumsetzung

Die umzusetzende lineare 32-Bit-Adresse wird in zwei 10-Bit-Seitenauswahlangaben und in eine 12-Bit-Offsetangabe zerlegt. Die Offsetangabe adressiert das Byte innerhalb der Seite bzw. des Seitenrahmens, und die beiden Seitenauswahlangaben wählen die betreffende aus den insgesamt 1M (2^{20}) Seiten aus.

Eine einzige Tabelle mit maximal 1M Seitenumsetzungseinträgen könnte die gewünschte Abbildung zwar leisten, ist aber unwirtschaftlich, da sie ihrerseits 4 MBytes im Arbeitsspeicher belegen würde.

Deshalb ist das Tabellensystem zweistufig aufgebaut. Die Umsetzungsangaben werden selbst als Seiten betrachtet und können so der Verlagerung auf Externspeicher unterworfen werden. Das ist naturgemäß nicht vollständig möglich; auf irgendeine beschreibende Angabe muß die Hardware immer direkt zugreifen können. In der IA-32-Architektur ist dies folgendermaßen gelöst:

Seitentabellen
Die Umsetzungsangaben für alle Seiten sind in Seitentabellen zusammengefaßt. Jede Seitentabelle (Page Table) ist ihrerseits eine Seite, die 4 Bytes lange Umsetzungseinträge für jeweils 1 k (=1024) Seiten aufnimmt. Für 1 M (2^{20}) Seiten werden somit maximal 1 k Seitentabellen gebraucht. Jede Seitentabelle ist, wie die anderen Seiten auch, Gegenstand der Adreßumsetzung bzw. Auslagerung.

Das Seitentabellenverzeichnis
Die Umsetzungsangaben für alle Seitentabellen sind in einem Seitentabellenverzeichnis (Page Directory) zusammengefaßt. Dieses hat ebenfalls den Umfang einer Seite (1k Umsetzungseinträge zu 4 Bytes für die Seitentabellen), ist aber *ständig im Arbeitsspeicher anwesend*. Es belegt dort einen Seitenrahmen, liegt also an einer integralen 4k- (Seiten-) Adresse. Die Adresse dieses Seitenrahmens steht in einem Steuerregister (CR3).

Prinzip der Adreßumsetzung
Die höchstwertigen 10 Bits der umzusetzenden linearen Adresse wählen die Seitentabelle aus dem Seitenverzeichnis aus. Die nachfolgenden 10 Bits der umzusetzenden linearen Adresse wählen im Seitenverzeichnis die Umsetzungsangabe der betreffenden Seite aus, und die niederwertigen 12 Bits adressieren ein Byte im Seitenrahmen, in dem sich die betreffende Seite befindet.

3.5.3. Tabelleneinträge der Seitenverwaltung

Die allgemeine Struktur kennen wir bereits aus Heft 1 (Abbildung 1.44). Hier folgen einige Einzelheiten.

Seitenrahmenadresse
Da ein Seitenrahmen 4 kBytes lang ist (entspricht 12 Adreßbits), genügt eine 20-Bit-Angabe, um die physische Anfangsadresse des Seitenrahmens zu bilden.

Zugriffssteuerung
Die beiden Bits A, D sind anfänglich gelöscht. Sie dienen dazu, zu erkennen, ob auf die Seite überhaupt zugegriffen wurde bzw. ob der Inhalt der Seite verändert wurde.

A - Accessed. Wird gesetzt, sobald ein Zugriff auf die Seite ausgeführt wird (kennzeichnet Nutzung der Seite).

- D - Dirty. Wird gesetzt, sobald ein Schreibzugriff auf die Seite ausgeführt wird (kennzeichnet Änderung des Inhalts der Seite).

Cache-Steuerung
Über die Bits PCD, PWT kann die Nutzung des Cache-Subsystems in Bezug auf den Adreßbereich der jeweiligen Seite gesteuert werden:

- PCD - Page Level Cache Disable. Bei Zugriffen auf diese Seite werden alle Caches grundsätzlich umgangen (Adreßbereich ist “uncacheable”).
- PWT - Page Level Write Through. Bei Zugriffen auf diese Seite wird in allen Caches das Schreibverfahren Write Through (Heft 6) erzwungen (Schreibzugriffe führen stets zum Arbeitsspeicher).

Privilegierung
Die Seitenverwaltung unterscheidet zwischen einer Supervisorebene und einer Anwenderebene. Nur höher privilegierte Programme können auf Seiten der Supervisorebene zugreifen. Auf Seiten der Anwenderebene kann immer zugegriffen werden. Steuerung: über das Bit U/S (User/Supervisor):

- U/S = 0. Supervisorebene. Zugriff ist nur erlaubt, wenn die aktuelle Privilegebene kleiner als 3 ist.
- U/S = 1. Anwenderebene. Zugriff ist in allen Privilegebenen erlaubt.

Schreibschutz
Seiten können gegen Schreibzugriffe geschützt werden. Steuerung: über das Bit R/W:

- R/W = 0. Es sind nur Lesezugriffe erlaubt.
- R/W = 1. Es sind Lese- und Schreibzugriffe erlaubt.

Seitenfehler
Das Anwesenheitsbit P (Present) kennzeichnet die Belegung bzw. Gültigkeit des Eintrags:

- P = 0. Die verbleibenden Bits des Eintrags enthalten *keine* physische Seitenrahmenadresse. Demzufolge wird eine Seitenfehlerausnahme wirksam. Die umzusetzende lineare Adresse wird dabei in einem Steuerregister (CR2) hinterlassen und kann so vom Seitenfehlerbehandler ausgewertet werden. Des weiteren stehen die meisten Bits des Tabelleneintrags zur Verfügung, um dort eine entsprechende Angabe (z. B. zur Position der Seite auf einem Massenspeicher) unterzubringen.
- P = 1. Die verbleibenden Bits des Eintrags sind mit einer physischen Seitenrahmenadresse sowie mit den weiteren Steuerbits belegt. Also kann auf die Seite im Arbeitsspeicher zugegriffen werden (Adressierung gemäß Abbildung 3.10).

3.5.4. Virtualspeicherorganisation

Der wichtigste Nutzungsfall der Seitenverwaltung ist der seitenorientierte virtuelle Speicher (On-Demand Paging Virtual Memory). Prinzip: anhand des P-Bits im Seitentabelleneintrag kann die Hardware erkennen, ob sich die Seite, auf die zugegriffen werden soll, im Arbeitsspeicher befindet oder nicht.

Das Prinzip wird in Abbildung 3.11 veranschaulicht. Es hat sich seit den 60er Jahren bewährt (eine der ersten Maschinen war die IBM 360/67; im Mainframe-Bereich setzte sich das Prinzip mit dem System /370 auf dem Markt durch).

Abbildung 3.11 Prinzip des seitenorientierten virtuellen Speichers

Die Seite befindet sich im Arbeitsspeicher
Der Zugriff wird ausgeführt, wobei die physische Adresse aus den niederen 12 Bits der linearen Adresse und aus den 20 Adreßbits im Seitentabelleneintrag zusammengesetzt wird. Des weiteren werden die Steuerbits PCD, PWT, R/W und U/S ausgewertet (ergibt es sich daraus, daß der Zugriff unzulässig ist, so wird er nicht ausgeführt, und es wird eine entsprechende Programmausnahme wirksam). Im Seitentabelleneintrag wird das A-Bit gesetzt; bei einem Schreibzugriff auch das D-Bit.

Die Seite befindet sich nicht im Arbeitsspeicher
Sie befindet sich demzufolge auf dem Massenspeicher und muß in einen Seitenrahmen des Arbeitsspeichers geschafft werden. Es ist zunächst ein freier Seitenrahmen zu finden (Sache der Software)*). Ist kein Seitenrahmen mehr frei, so muß einer freigemacht werden. Hierzu ist die Seite, die diesen Seitenrahmen derzeit belegt, auf den Massenspeicher auszulagern.

*): das Problem entspricht dem Finden freier Sektoren auf einem Massenspeicher.

Einfachste Fälle der Auslagerung:

- die Seite wurde nicht verändert (erkennbar am gelöschten D-Bit): ein Auslagern ist gar nicht notwendig, da die Kopie auf dem Massenspeicher noch gültig ist. Es genügt, den Seitentabelleneintrag auf “nicht anwesend” umzustellen (Löschen des P-Bits usw.).
- die Seite wurde verändert (D-Bit gesetzt). Dann ist sie tatsächlich auszulagern.

Welche Seite soll ausgelagert werden?
Es gibt verschiedene Verfahren. Beispielsweise liegt es nahe, nach Seitenrahmen zu suchen, die mit Seiten belegt sind, zu denen in der letzten Zeit nicht zugegriffen wurde (erkennbar am gelöschten A-Bit). Fachbegriff: Last Recently Used (LRU). Um die Nutzungszeiten der Seiten näherungsweise zu berücksichtigen, löscht die Systemsoftware in bestimmten Zeitabständen (z.B. alle 250 ms) alle A-Bits. Wird nun zwischen zwei solchen Löschaktionen ein A-Bit als gesetzt vorgefunden, so ist klar, daß innerhalb dieses Intervalls auf die jeweilige Seite tatsächlich zugegriffen wurde. Demgegenüber sind dann alle Seiten mit gelöschtem A-Bit in den - beispielsweise - letzten 250 ms nicht genutzt worden, sind also womöglich geeignete Kandidaten zur Auslagerung.

Hinweise:

1. Das Problem gibt es auch bei den Caches und TLBs. Nur arbeiten diese Einrichtungen transparent - die Software kann also gar nicht eingreifen. Deshalb begnügt man sich mit recht groben Zeitkontrollen (Heft 6).
2. Das LRU-Verfahren wurde hier sehr vereinfacht dargestellt. In der Praxis sind diese Algorithmen eine Wissenschaft für sich. Nur zwei Beispiele: (1) das System wählt bestimmte Adreßbereiche aus, in denen die A-Bits gelöscht werden, (2) das zyklische Löschen der A-Bits beginnt erst dann, wenn die Anzahl der freien Seitenrahmen unter eine gewisse Schwelle gefallen ist.

Auslagerungsstrategien
Es ist eine Frage der Systemoptimierung, wann Seitenrahmen freigemacht werden. Im Extremfall geschieht dies nur dann, wenn (1) tatsächlich eine neue Seite benötigt wird, im Arbeitsspeicher aber (2) kein Seitenrahmen mehr frei ist. Diese Vorgehensweise kann allerdings die Systemleistung stark beeinträchtigen, und zwar dann, wenn bei voll belegtem Arbeitsspeicher häufige Seitenwechsel vorkommen (diesen - recht unangenehmen - Effekt bezeichnet man als *Page Thrashing*; sprich: Peetsch Ssräsching). Es ist deshalb üblich, am Anfang eine gewisse Anzahl p von Seitenrahmen freizuhalten. Diese werden zunächst nacheinander belegt. Wird dabei ein gewisser Schwellwert s unterschritten (d. h. durch das fortlaufende Belegen sind irgendwann einmal weniger als s Seitenrahmen frei), so wird dafür gesorgt, daß wieder p Seitenrahmen frei werden. Die erforderliche Anzahl an Seitenrahmen wird dann auf einmal freigemacht.

Hinweis:
Wenn Sie mit der Leistung eines solchen Systems nicht zufrieden sind, können Sie diese Parameter - falls zugänglich - ändern (aber bitte systematisch, mit exakter Protokollierung von Maßnahme und Leistungsverhalten). Grundsätzlich können Sie getrost experimentieren. Schlimmstenfalls wird das System nur noch langsamer; wirklich "abstürzen" dürfte dabei nichts.

Leistungsmessung
Hierzu gibt es Hilfssoftware. Beispiel: Windows Systemmonitor (Abbildung 3.12).

Abbildung 3.12 Anzeige des Systemmonitors (Windows 98). Die interessierenden Angaben sind unter der Datenquelle “Speicher-Manager” (hier: im rechten Feld) zu finden

Wo eingreifen?
Die Dokumentation der jeweiligen Systemplattform gibt Auskunft - oder auch nicht. Beispielsweise hält sich Microsoft seit einiger Zeit sehr bedeckt. In der Systemsteuerung (Abbildung 3.13) und auch in den Handbüchern geht es praktisch nur noch um die Größe der Auslagerungsdatei - weitere Eingriffsmöglichkeiten werden nicht offengelegt. (Bei Windows 3.1 befinden sich hingegen noch Steuerangaben zu diversen Feinheiten des Seitenwechsels in der Datei SYSTEM.INI (Abschnitt [386enh]). Zwei Beispiele: (1) LRUSweepLowWater bestimmt, von welcher Zahl freier Seitenrahmen an der LRU-Algorithmus wirksam werden soll, (2) LRUSweepFreq bestimmt das Intervall (in ms) zwischen den LRU-Durchgängen (Löschen der A-Bits).)

Abbildung 3.13 Systemsteuerung (Windows 98): so läßt sich die Auslagerungsdatei beeinflussen

Adreßumsetzungspuffer (TLB)
Wenn jedem "eigentlichen" Speicherzugriff die Tabellenzugriffe der Seitenverwaltung vorangehen müßten, würde die Verarbeitungsleistung untragbar absinken. Deshalb werden diese Aktivitäten in der Hardware besonders unterstützt: alle Maschinen mit Seitenverwaltung haben *Adreßumsetzungspuffer*, die zumeist als TLBs (Translation Lookaside Buffers) bezeichnet werden. Die Auslegung der TLBs ist herkömmlicherweise (so auch bei IA-32) an sich kein Bestandteil der Architektur; Programme merken von der Anwesenheit eines TLB gar nichts (er ist völlig transparent)[*)]. Näheres zu Aufbau und Wirkungsweise in Heft 6.

*): anders bei den typischen RISC-Architekturen. 1. Auslegung: der TLB *kann* softwareseitig geladen werden (Beispiel: Sparc). 2. Auslegung: die Software *muß* den TLB verwalten (Beispiel: Mips).

3.5.5. Weiterentwicklungen

Lange Seiten (1): 4 MBytes
Mit dem Pentium wurde die Möglichkeit eingeführt, Seiten von 4 MBytes Länge vorzusehen. Die Adreßabbildung erfolgt hierbei ausschließlich über das Seitentabellenverzeichnis (Abbildung 3.14 b, c).

Lange Seiten (2): 2 MBytes
Seiten von 2 MBytes können vorgesehen werden, falls die PAE-Adreßverlängerung auf 36 Bits genutzt wird (siehe Seite 135).

Globale Seiten
Von den P6-Prozessoren an wird die Bitposition 6 in den Tabelleneinträgen ausgenutzt, um Seiten als “global” zu kennzeichnen. Die Wirkung: hat eine solche globale Seite einen TLB-Eintrag, so bleibt dieser erhalten, auch wenn ansonsten alle TLB-Einträge ungültig gemacht werden (z. B. beim Laden des Steuerregisters CR3 oder anläßlich einer Taskumschaltung). Vgl. auch Abschnitt 3.3.6. in Heft 6. Der Zweck: es soll möglich sein, den Zugriff auf Seiten, die immer wieder gebraucht werden, stets über den TLB zu führen (Leistungssteigerung).

Tabelleneinträge
Abbildung 3.14 veranschaulicht die Belegung der Einträge im Seitentabellenverzeichnis bzw. in den Seitentabellen, wie sie mit den P5- und P6-Prozessoren eingeführt wurde.

Abbildung 3.14 Tabelleneinträge der Seitenverwaltung (P5, P6). Vgl. Abbildung 1.44 in Heft 1

Erklärung zu Abbildung 3.14:

a) Format für herkömmliche Seiten (4 kBytes). Die bisher reservierten Bits 7 und 8 wurden belegt:
   - Bit 7: PS = Page Size. 0 - Seitenlänge 4 kBytes; 1 - Seitenlänge 4 MBytes*) (nur im Seitentabellenverzeichnis). Von P5 an.
   - Bit 8: G = Global. 0 - Eintrag der Seite kann aus TLB entfernt werden; 1 - TLB-Eintrag bleibt erhalten ("globale" Seite - siehe oben). Von P6 an.

b) Eintrag im Seitentabellenverzeichnis für Seiten von 4 MBytes Länge.

c) Bildung der physischen Adresse für Seiten von 4 MBytes Länge.

*): 2 MBytes, wenn die PAE-Adreßverlängerung wirksam ist.

Adreßverlängerung auf 36 Bits
Von P6 an liefern manche Prozessoren eine physische Adresse von 36 Bits (der unterstützte Adreßraum wächst somit von 4 GBytes auf 64 GBytes). Hierzu wird die Seitenverwaltung ausgenutzt. Es gibt zwei Möglichkeiten:

1. Physical Address Extension (PAE)
Die Tabellenstrukturen der Seitenverwaltung sind jetzt 64 Bits lang. Zudem wird eine weitere Ebene der Adreßumsetzung eingeführt: die Seitentabellenverzeichnis-Zeigertabelle (Page-Directory Pointer Table). Hiermit wird eine "echte" Adreßverlängerung (für alle Seitenlängen) erreicht (Abbildungen 3.15, 3.16).

2. 36-Bit Page Size Extension (PSE)
Es werden die herkömmlichen Tabellenstrukturen (32 Bits) weiterverwendet. Die Adreßverlängerung wirkt allerdings nur für Seiten von 4 MBytes Länge (Abbildung 3.17).

Ein- und Ausschalten der verschiedenen Betriebsweisen: über Bits im Steuerregister CR4.

Abbildung 3.15 PAE: Prinzip der Adreßumsetzung

Erklärung:
Die lineare Adresse ist nach wie vor nur 32 Bits lang. Um sie auf eine physische 36-Bit-Adresse umsetzen zu können, wurden die Tabellenstrukturen auf 64 Bits lange Einträge umgestellt. Der Umsetzungsweg:

das Steuerregister CR3 adressiert eine neue Tabellenstruktur, die Seitentabellenverzeichnis-Zeigertabelle (Page Directory Pointer Table). Sie hat 4 Einträge.

die zwei Bits 31, 30 der linearen Adresse wählen einen Eintrag in der Seitentabellenverzeichnis-Zeigertabelle aus. Dieser adressiert eines (von 4) Seitentabellenverzeichnissen. Jedes Seitentabellenverzeichnis hat 512 Einträge zu 8 Bytes.

die neun Bits 29...21 der linearen Adresse wählen einen Eintrag im jeweiligen Seitentabellenverzeichnis aus. Nun gibt es zwei Möglichkeiten:
   - der Eintrag adressiert eine Seite von 2 MBytes. Dann adressieren die verbleibenden 21 Bits 20...0 der linearen Adresse das Byte in dieser Seite.

- der Eintrag adressiert eine Seitentabelle (in der Abbildung unten). Jede Seitentabelle hat 512 Einträge zu 8 Bytes. Die neun Bits 20...12 der linearen Adresse wählen einen Eintrag in der jeweiligen Seitentabelle aus, und die verbleibenden 12 Bits 11...0 der linearen Adresse adressieren das Byte in der jeweiligen Seite.

*): diese Tabellen haben also - nach wie vor - eine Gesamtlänge von 4 kBytes = einer Seite.

Abbildung 3.16 PAE: Tabelleneinträge (Überblick)

Erklärung:

a) allgemeines Format (betrifft alle Tabellenstrukturen). Die Belegung der Bits 11...0 entspricht Abbildung 3.14. Dabei sind in manchen Einträgen manche Bits mit Festwerten belegt (die Einzelheiten stehen in den Architekturhandbüchern - wir wollen sie uns hier schenken). Im Vergleich zur herkömmlichen Auslegung hat man eigentlich nichts weiter getan, als die Länge der Einträge einfach zu verdoppeln (wobei nur 4 der zusätzlichen Bits ausgenutzt werden).
b) Seitentabellenverzeichnis-Eintrag für eine Seite von 2 MBytes,
c) 36-Bit-Adresse, aus einem Eintrag gemäß a) gebildet,
d) 36-Bit-Adresse einer Seite von 2 MBytes.

Weshalb sind die langen Seiten nur 2 MBytes lang und nicht 4 MBytes? - Aus einer Zwangslage heraus. Damit die Seitenverwaltung halbwegs kompatibel bleibt, muß die (häufigste) Seitengröße von 4 kBytes erhalten bleiben. Und die einzelne Tabellenstruktur darf auch nicht größer werden (Prinzip: 1 Tabelle = 1 Seite). Bei 8 Bytes je Eintrag passen in eine Seite somit nur 512 Einträge (statt 1024). Jede der Tabellen-Stufen (Seitentabellenverzeichnis → Seitentabelle) kann somit nur 9 Bits der linearen Adresse umsetzen (bisher: 10 Bits). Bei zwei Tabellenstufen fehlen also 2 Bits, die nicht umgesetzt werden können. Deshalb die vorgeschaltete dritte Stufe der Umsetzung (über die Seitentabellenverzeichnis-Zeigertabelle). Aus dieser Struktur ergibt sich aber, daß in den ersten beiden Stufen 2 + 9 = 11 Bits umgesetzt werden. Es verbleiben also nur noch 32 - 11 = 21 Bits der linearen Adresse - und damit lassen sich nur 2 MBytes adressieren.

Denksportaufgabe: könnte man trotzdem Seiten von 4 MBytes unterstützen?

Lösung: im Prinzip ja - wenn im Seitentabellenverzeichnis jeweils 2 aufeinanderfolgende Einträge entsprechend eingerichtet und zusammen verwaltet werden (Sache der Systemsoftware).

Abbildung 3.17 PSE: Bildung der 36-Bit-Adresse

Erklärung:
Die Strukturen der herkömmlichen Seitenverwaltung (Abbildungen 3.10, 3.14) bleiben erhalten. Die Adreßverlängerung auf 36 Bits ist aber nur bei Zugriffen auf Seiten von 4 MBytes Länge wirksam.

a) Struktur eines entsprechenden Eintrags im Seitentabellenverzeichnis. Man nutzt hier einfach einige der bisher reservierten Bitpositionen, um die 4 zusätzlichen Adreßbits unterzubringen. Eine weitere dieser Bitpositionen dient als Zusatzbit zum Adressieren der nachfolgend beschriebenen Seitenattributtabelle (PAT = Page Attribute Table).

b) die aus dem Eintrag gebildete 36-Bit-Adresse.

Die Seitenattributtabelle (Page Attribute Table PAT)

Es handelt sich um ein 64-Bit-Register, das in Zusammenhang mit der Seitenverwaltung betrieben wird. Der Zweck: den Speicherbereichen gemäß der logischen (linearen) Adresse der jeweiligen Seite bestimmte Typangaben bzw. Eigenschaften (Attribute) zuzuordnen. Die Seitenattributtabelle leistet praktisch das gleiche wie die Speichertyp-Bereichsregister (MTRRs; Abschnitt 3.3.2.).

Weshalb diese Lösung, obwohl es die MTRRs gibt? - Damit die Typkennzeichnung auch beim virtuellen Speicher funktioniert. Die MTRRs betreffen physische Adressen. Nun ist es aber ein Kennzeichen des virtuellen Speichers, daß die Seiten in beliebige Seitenrahmen des Arbeitsspeichers eingelagert werden können. Der Programmierer sieht aber nur lineare Adressen und möchte ggf. einzelnen Bereichen bestimmte Eigenschaften zuweisen. Diese Zuweisung muß immer gelten, gleichgültig ob die betreffende Seite in Seitenrahmen 531 oder in Seitenrahmen 28 452 eingelagert wird - sie muß also mit der Adreßumsetzung gleichsam mitwandern.

Prinzip:

Die Seitenattributtabelle hat 8 Einträge zu je 8 Bits. Diese Einträge werden aus den Einträgen der Seitenverwaltung heraus adressiert, und zwar über eine 3-Bit-Adresse, die aus den Bits PCD, PWT sowie einem PAT-Bit gebildet wird (Abbildung 3.18).

Abbildung 3.18 Die Seitenattributtabelle (Page Attribute Table PAT)

Erklärung:

a) die Seitenattributtabelle als 64-Bit-Register mit 8 adressierbaren Attribut-Einträgen,

b) Attributadresse aus Steuerregister 3 und aus diversen Verzeichniseinträgen. Da es kein PAT-Bit gibt, sind nur die ersten 4 Attribute zugänglich.

c) Attributadresse des Seitentabelleneintrags einer Seite von 4 kBytes. Als PAT-Bit wird hier Bit 7 verwendet (das in einem solchen Eintrag keine anderweitige Funktion hat).

d) Attributadresse des Seitentabellenverzeichnis-Eintrags einer Seite von 2 oder 4 MBytes. Als PAT-Bit wird hier - das ansonsten reservierte - Bit 12 verwendet (vgl. Abbildung 3.17a).

Tabelle 3.4 nennt die einzelnen Attributwerte, die in den Einträgen der PAT untergebracht werden können.

Bezeichnung	Einlagerung in Caches	Cache-Schreibverfahren Write Back	voreilendes (spekulatives) Lesen	Reihenfolge der Speicherzugriffe
Uncacheable (UC)	nein	nein	nein	strikt gemäß Absicht des Programmierers
Write Combining (WC)	nein	nein	ja	Zusammenfassung von Zugriffen zulässig, Adreßreihenfolge und Cache-Kohärenz nicht garantiert
Write Through (WT)	ja	nein	ja	vom Prozessor bestimmt (spekulativ)
Write Protected (WP)	Lesen: ja, Schreiben: nein	nein	ja	
Write Back (WB)	ja	ja	ja	
Uncached (UC-)	nein	nein	nein	strikt gemäß Absicht des Programmierers. Kann aber von WC aus den MTRRs (Tabelle 3.3) übergangen werden

*): Cacheability; betrifft alle Cache-Ebenen (L1, L2)

Tabelle 3.4 Attributangaben in der Seitenattributtabelle PAT (P6-Prozessoren)

Hinweise:

1. Eine trickreiche Angelegenheit. Die modernen Prozessoren mit ihren vielfältigen Maßnahmen der Zugriffsbeschleunigung benötigen Steuerangaben, die über das bisher Vorgesehene (z. B. die Cache-Steuerung mittels PCD und PWT) hinausgehen. Aus Gründen der Abwärtskompatibilität kann man aber die Bits in den Tabelleneinträgen nicht einfach anders belegen. Zudem fehlen - in den 32-Bit-Strukturen - Reserven für künftige Erweiterungen. Deshalb hat man in Form der PAT gleichsam einen weiteren Abbildungsmechanismus zwischengeschaltet.
2. Die 8 Einträge der PAT werden nach dem Rücksetzen hardwareseitig auf bestimmte Attribute fest eingestellt (z. B. Eintrag 1 auf WT, Eintrag 2 auf UC usw.).
3. Nicht alle P6-Prozessoren haben eine PAT (ob eine vorhanden ist oder nicht, kann programmseitig abgefragt werden).
4. PAT und MTRRs wirken zusammen. Welche Speichertypzuordnungen sich aus den Kombinationen der Angaben gemäß den Tabellen 3.3 und 3.4 ergeben, steht in den Architekturhandbüchern.
5. Die Systemprogrammierer haben es nicht einfach. Sie müssen sich (1) überlegen, wie der physische Speicher eingeteilt werden soll, und die MTRRs entsprechend laden. Weiterhin müssen sie (2) sinnvolle Attribute für die einzelnen Bereiche des virtuellen Speichers festlegen (8 verschiedene sind insgesamt möglich, weil die PAT nur 8 Einträge hat). Schließlich sind (3) alle Einträge der Seitenverwaltung so vorzubereiten, daß

die jeweiligen Bits (PCD, PWT, PAT) den richtigen Eintrag in der Seitenattributtabelle adressieren.*)

*): Beispiel: man entschließt sich, einen Speicherbereich, der einem PCI-Busmaster als Zwischenspeicher dient, vom Caching auszuschließen (Attribut UC). Nun muß man zunächst einen PAT-Eintrag auf UC stellen (das Rücksetzen bewirkt, daß Eintrag Nr. 2 bereits so eingestellt ist). Schließlich sind die Bits PCD, PWT und PAT in allen Seitentabelleneinträgen, die diesen Speicherbereich betreffen, so einzutragen, daß sich die entsprechende PAT-Adresse (im Beispiel: 2) ergibt.

3.6. Speichermodelle in der Praxis

3.6.1. Das flache Speichermodell

Obwohl es IA-32 erlaubt, eine Vielzahl von Speichermodellen zu implementieren, hat sich - auch im PC- Bereich ein Prinzip durchgesetzt: das *flache Speichermodell* (Flat Memory Model) mit einem einzigen linearen Adreßraum von 4 GBytes und Virtualspeicherorganisation auf Grundlage der Seitenverwaltung. (Alle Segmentdeskriptoren betreffen dasselbe Segment mit Anfangsadresse 0 und einer Länge von 4 GBytes; Abbildung 3.19.)

Abbildung 3.19 Beispiel eines flachen Speichermodells

3.6.2. Segmentierte Speichermodelle

Der Vorteil der Segmentierung: die Zugriffsrechte können gleichsam fein dosiert vergeben werden. Deshalb werden auch segmentierte Speichermodelle gelegentlich noch verwendet. Beispiel: die VxD-Gerätetreiber (Windows 95/98). Sie ordnen sich zwar in das flache Speichermodell von Windows ein, die Programm- und Datenbereiche der einzelnen Treiber werden aber als Segmente verwaltet. Ein VxD kann bis zu 12 Segmente haben (je ein Daten- und ein Programmsegment für die Initialisierung im Realmodus und im Protected-Modus, auslagerungsfähige (pageable) Programme und Daten, residente Programme und Daten, statische Programme und Daten sowie je ein Programm- und ein Datensegment zu Fehlersuchzwecken (Debugging)).