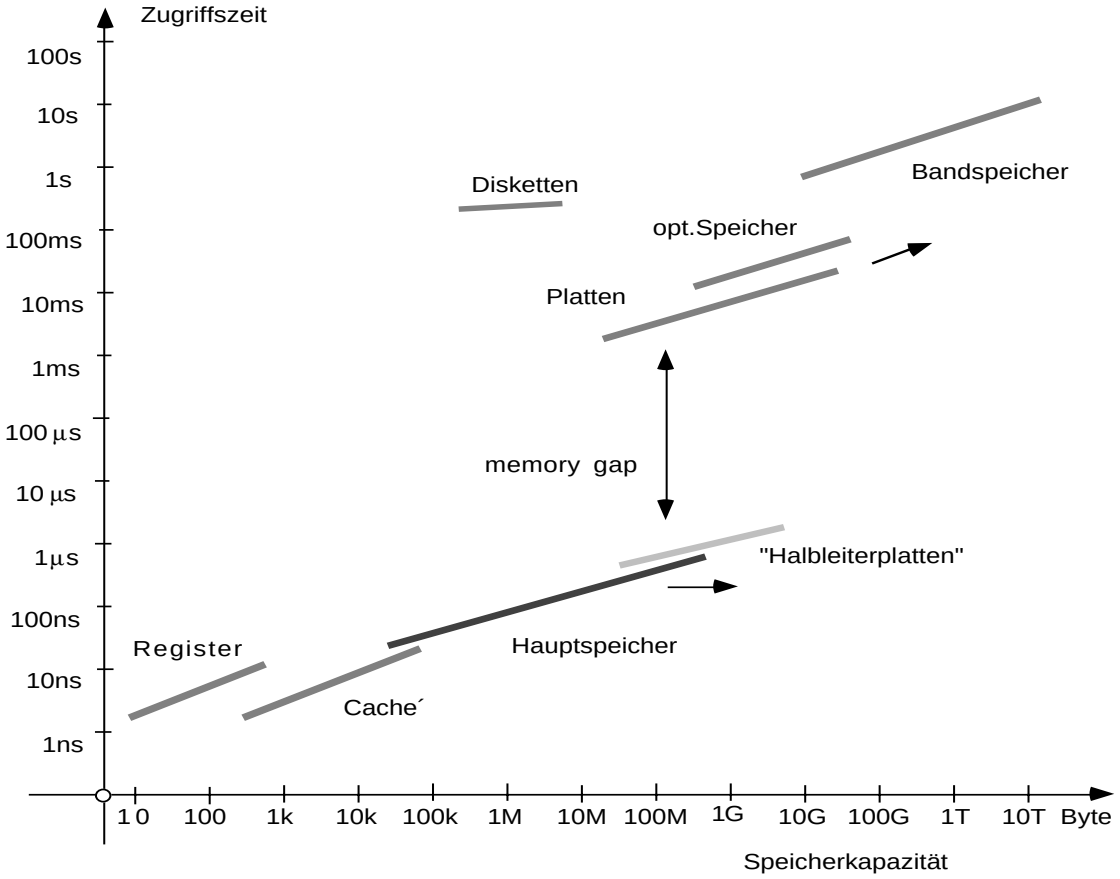
Kapitel 8 Speicherorganisation

8.1. Klassifikation von Speichern

Es gibt zwei große Gruppen von Speichern: **Halbleiterspeicher** und **mechanisch bewegte Speicher**, die sich in ihrer Zugriffszeiten um rund drei Größenordnungen unterscheiden, das sog. memory gap.



In der gleichen Weise unterscheiden sich die Kapazitäten. Massenspeicher sind nach wie vor mechanisch bewegte Speicher, seien es Platten oder optische Speicher. Es gibt allerdings inzwischen "Halbleiterplatten"; viele MByte aufgebaut aus Halbleiterspeichern und mit einer Batterie gepuffert, so daß die Information erhalten bleibt, auch wenn der Speicher aus dem Rechner für eine Weile entfernt ist oder im Rechner der Strom zeitweilig abgestellt ist.

Die technische Entwicklung geht bei den mechanisch bewegten Speichermedien sehr stark hin in Richtung optische Speicher, von den Nurlesespeichern (CD-ROM) über ein-

mal beschreibbare Speicher (write once, read multiple, WORM) hin zu optischen Schreib-Lesespeichern. Letztendlich wird der Platz, den man für die Speicherung eines Bit braucht, entscheidend sein. Halbleiterspeicher wie magnetische Speicherplatten wie auch optische Speicher bringen Datenbits auf einer planaren Struktur unter:

• Halbleiterspeicher auf der Oberfläche eines Chip, repräsentiert durch Flip-Flops oder als Ladungen auf einem Kondensator, mit Abmessungen von 1 x 1 μm^2, d.h. 10^8 Bit / cm^2 und Chipflächen von einigen cm^2 . Speicherchips haben zur Zeit bis zu 256 MBit.

• Magnetische Speicher als magnetisierte Domänen in der Größenordnung von 1 x 1 μm^2, d. h., 10^8 Bit / cm^2 auf Flächen von einigen 100 cm^2. Plattenspeicher speichern einige GByte pro Platte.

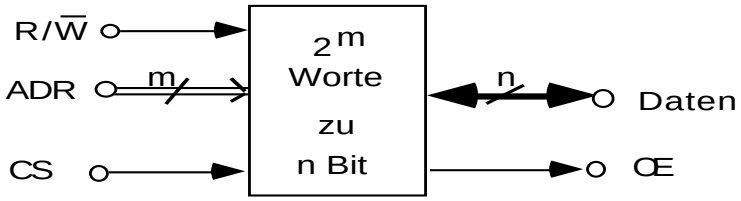
• Optische Speicher wie eine CD-ROM speichern ein Bit als Vertiefung (pit) von ca 1 x 1 μm^2. Die Wellenlänge des Lasers, mit dem die Bits ausgelesen werden, bestimmt die Dimension des Pits, in dem ein Bit gespeichert ist: 600 M Byte auf der Oberfläche einer CD-ROM sind heute (noch) Standard.

Noch größere Speicherkapazitäten erreichen optische Speicher, die die Information als Beugungsbilder (Hologramm) in einem Volumen speichern. Dort erscheinen 10^8 Bit / mm^3 möglich, d. h. 10^{14} Bit / l. Das ist auch die Größenrordnung der Synapsendichte im Gehirn höherer Lebewesen..

8.2. Physikalische Adressierung

8.2.1. Halbleiterspeicher

Der einzelne Speicherchip wird von außen angesprochen durch ein Chip-Select-Signal CS, die Adressbits, die ein Wort in der Speichermatrix, d. h. n Speicherebenen bei Wortbreite n Bit, parallel ansprechen, und ein Signal R/W, das zwischen Einschreiben und Auslesen des Inhalts entscheidet. Das angesprochene Wort steht auf den Datenleitungen zur Verfügung (read) oder wird vom Rechner dort angeboten (write). Mit einem Signal "output enable" (OE) signalisiert der Chip, daß beim Lesen nun das Wort auf den Datenleitungen gültig ist.

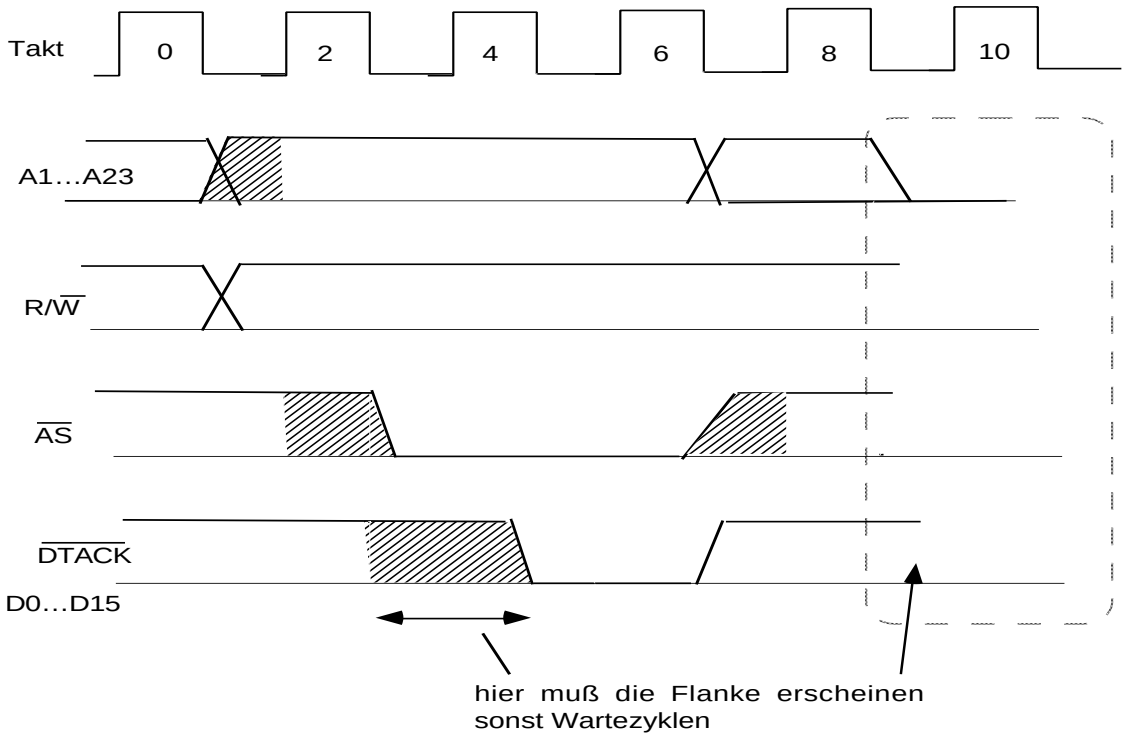


Beim Schreiben wird das R/W-Signal von der CPU als Strobe-Signal benutzt. Dazu muß dieses Signal während einer Haltezeit konstant auf Null gehalten werden.

Die folgenden Bilder zeigen im Prinzip den Schreib-Lesevorgang, hier für den Rechner Motorola 68000.

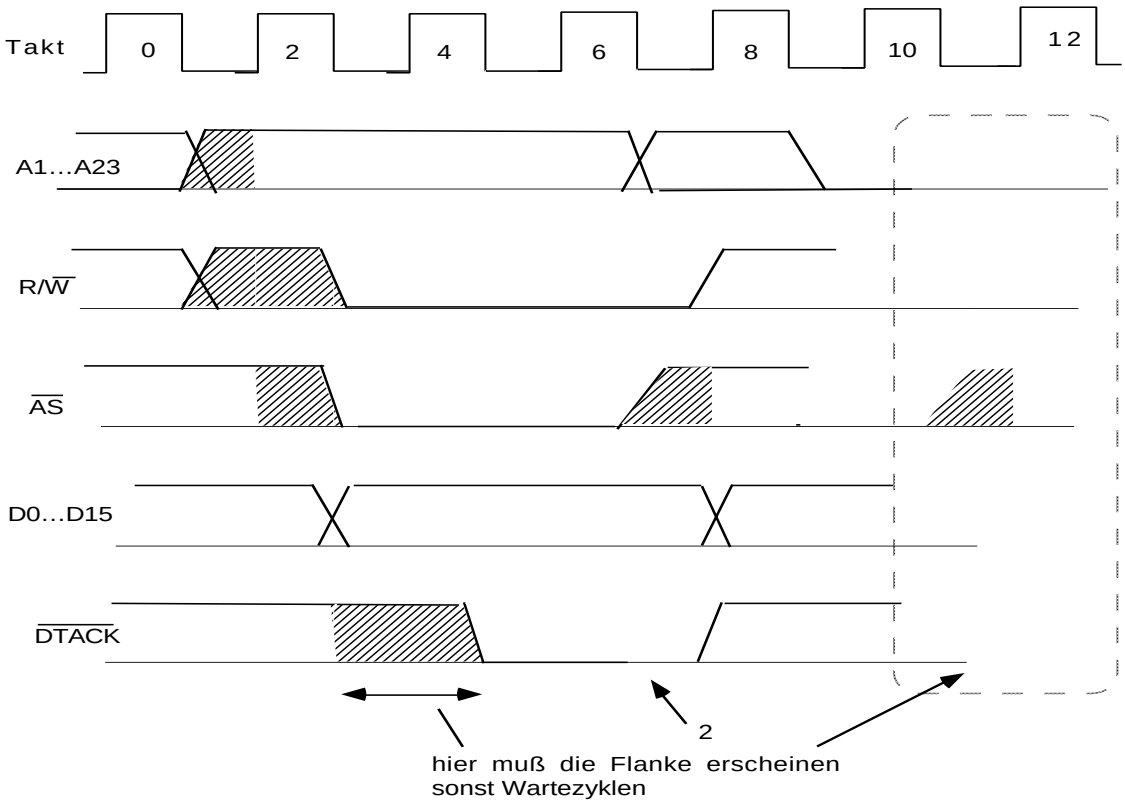
Lesen:

Lesezyklus auf dem MC68000



Schreiben:

Schreibzyklus auf dem MC68000



8.2.2. Physikalische Adressierung eines Plattenspeichers

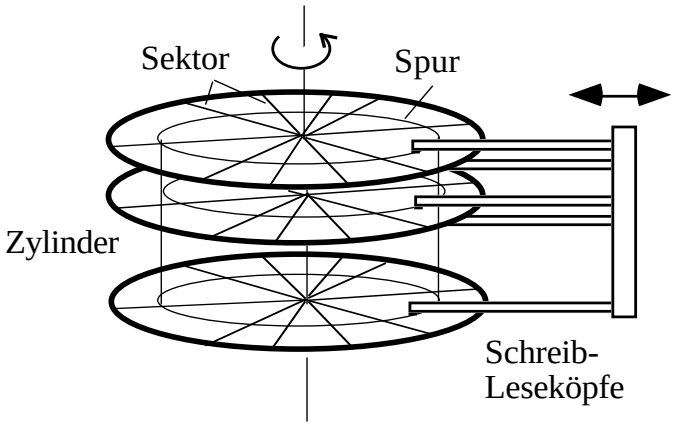
Auf das Medium "Plattenspeicher" wird vom Rechner her über den Bus und einen Plattencontroller zugegriffen. Ein Plattenspeicher besteht aus einem Stapel von rotierenden Platten, bei denen auf jede Oberfläche mit einem beweglichen Schreib-Lesekopf zugegriffen wird.

Die Schreib-Leseköpfe sind starr gekoppelt und werden gemeinsam bewegt. Eine Plattenoberfläche ist aufgeteilt in **Sektoren** (16 - 80) und **Spuren** (~ 400 - 1200).

Alle übereinanderliegenden Spuren in einem Stapel von 5 - 10 Platten bilden einen **Zylinder**.

Jede Oberfläche hat ihren eigenen Schreib-Lesekopf, so daß n Bit parallel verarbeitet werden können.

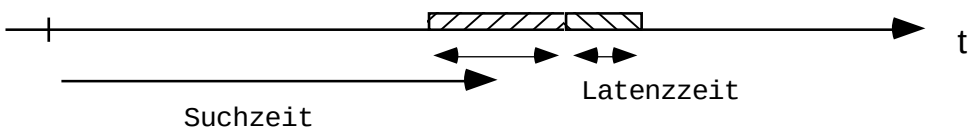
Mit 9 Oberflächen ist es möglich, 8 Bit + 1 Paritätsbit (odd parity) parallel zu verarbeiten oder 8 Bit + 1 Taktbit zu schreiben oder zu lesen, womit in beiden Fällen eine Synchronisation möglich wird.



Die hardwaremäßige Adresse ist festgelegt durch Angabe einer Spur und eines Sektors in dieser Spur, und es wird ein Sektor auf einmal geschrieben oder gelesen.

Durch den parallelen Zugriff über alle Schreib-Leseköpfe wird ein Sektor eines Zylinders jeweils geschrieben oder gelesen.

Für den Zugriff auf eine Platte spielen zwei Zeiten eine Rolle: die **Suchzeit** zur Positionierung der Schreib-Leseköpfe auf der gesuchten Spur und die **Latenzzeit**, die mittlere Zugriffszeit auf den gewünschten Sektor, i. allg. eine halbe Umdrehung der Platte.



Durch die Entwicklung immer kleinerer Schreib-Leseköpfe und die damit einhergehende höhere Bitdichte auf der Platte und durch die RLL-Codierung (run lenth limited) ist die Kapazität von Platten stetig angestiegen.

Ein Beispiel zeigt das folgende Bild:

Seagate ST 11 200 N

Kapazität : unformatiert : 1248 MB

formatiert : 1054 MB

1872 Zylinder

15 Köpfe

74 Sektoren / Spur

Aufzeichnungsverfahren : RLL 1.7

Drehzahl : 5411 U / min

Suchzeit : 12 - 12,7 ms

Latenzzeit : 5,5 ms

Transferrate : max. 5 MB / s

Interface : SCSI-2 fast

Durchmesser : 5,25 Zoll

Einbauhöhe : 82 mm

Das Aufzeichnungsverfahren RLL1.7 bedeutet, daß zwischen zwei Wechseln in der Magnetisierungsrichtung, die eine "1" repräsentieren, mindestens eine und höchstens 7 Nullen liegen. Eine beliebige Folge von Null und Eins wird dann in eine Folge mit dieser Eigenschaft umcodiert. Damit wird das Aufzeichnungsverfahren selbsttaktend: bei ungefährer Vorgabe der Umdrehungsgeschwindigkeit kann sich die Lese-Schreibeinheit immer wieder synchronisieren.

8.3. Speicherhierarchie

Da ein Programm i. allg. nicht wild im verfügbaren Adressraum herumspringt und der Zugriff auf Programme und Daten ein lokales Verhalten zeigt, und da schnelle Speicher nach wie vor teuer sind, hat man eine Hierarchie von Speichern im Rechner:

- Register mit einigen 100 Byte und Zugriffen mit der Geschwindigkeit des internen Taktes (5 - 10 ns).
- Cache, Schreib-Lesespeicher an Bord der CPU, einige k Byte bis 100 k Byte, mit Zugriffszeiten nur wenig langsamer (50 ns) als auf Register.
- Hauptspeicher in Form von Halbleiterspeichern von 8 - 512 M Byte mit Zugriffszeiten von weniger als 500 ns außerhalb der CPU, der über den Zentralbus der Anlage angesprochen wird.
- Massenspeicher als Platten von 100 M Byte bis viele G Byte angesprochen über den Zentralbus und eigene Plattencontroller.

8.4. Logische Adressierung

Der durch Maschinenbefehle ansprechbare Adressraum ist der **logische Adressraum** des Rechners. Die Adressen in diesem Raum sind **logische** oder **virtuelle Adressen.**

Der Speicher ist meist organisiert in Byte von 8 Bit als kleinste adressierbare Einheiten; man findet auch wortadressierte Speicher mit Wortlängen von 4 Byte = 32 Bit. Bei einem Zugriff wird häufig auch bei Byteadessierung ein Wort von 32 Bit herausgelesen. Der Adressumfang ist festgelegt durch die Anzahl der Bits in den Adressregistern. Das sind bei kleinen Rechnern (Microcontroller) 16 Bit, die einem Adressraum von 64 k Byte entsprechen, sonst meist 32 Bit, entsprechend einem Adressraum von 4 G Byte und neuerdings bis zu 64 Bit, entsprechend einem logischen Adressraum von 2^{64} adressierbaren Worten.

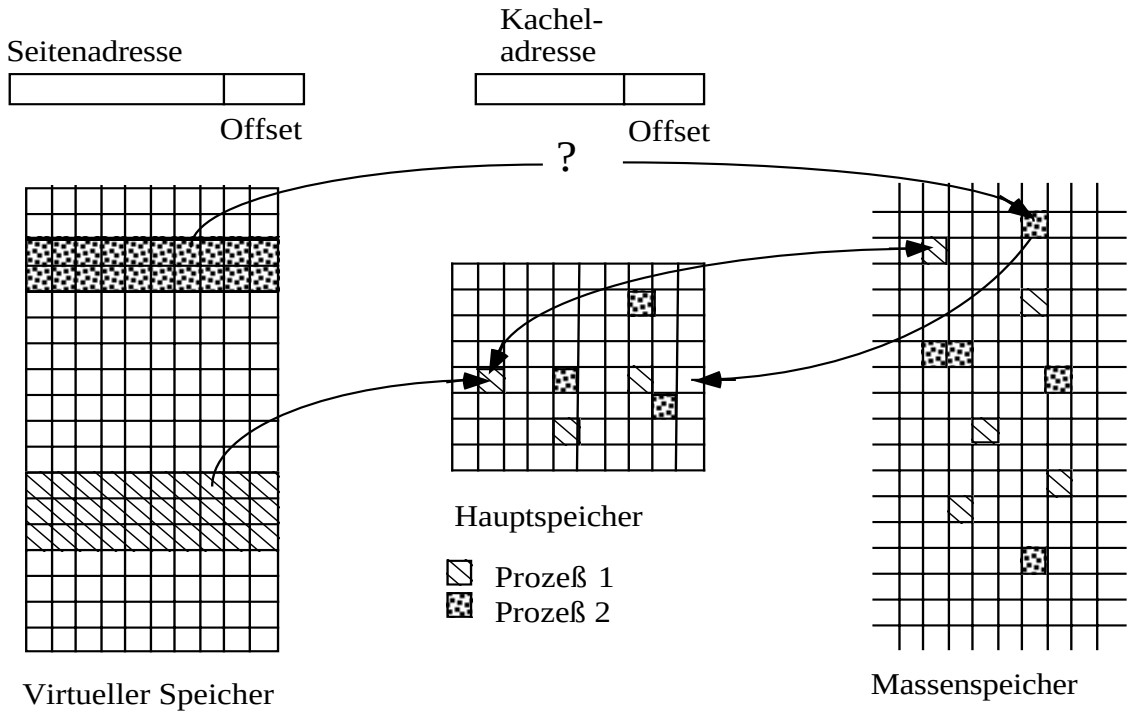
In diesen großen Adressräumen führt man zusätzliche Unterteilungen ein:

8.4.1. Seiten

Der logische Adressraum wird eingeteilt in **Seiten (pages)** gleicher Größe, typisch 0.5 - 4 k Byte.

Die logische Adresse zerfällt dann in zwei Anteile: die **Seitenadresse** und den **Offset** innerhalb der Seite. Der physikalische Speicher (Cache, Hauptspeicher, Massenspeicher) wird ebenfalls in **Kacheln (frames)** der gleichen Größe wie die Seiten eingeteilt.

Die physikalische Adresse ist dann (insbesondere im Hauptspeicher) aufgeteilt in **Kacheladresse** und Offset. Im Cache werden häufig kleinere Blöcke verwaltet.



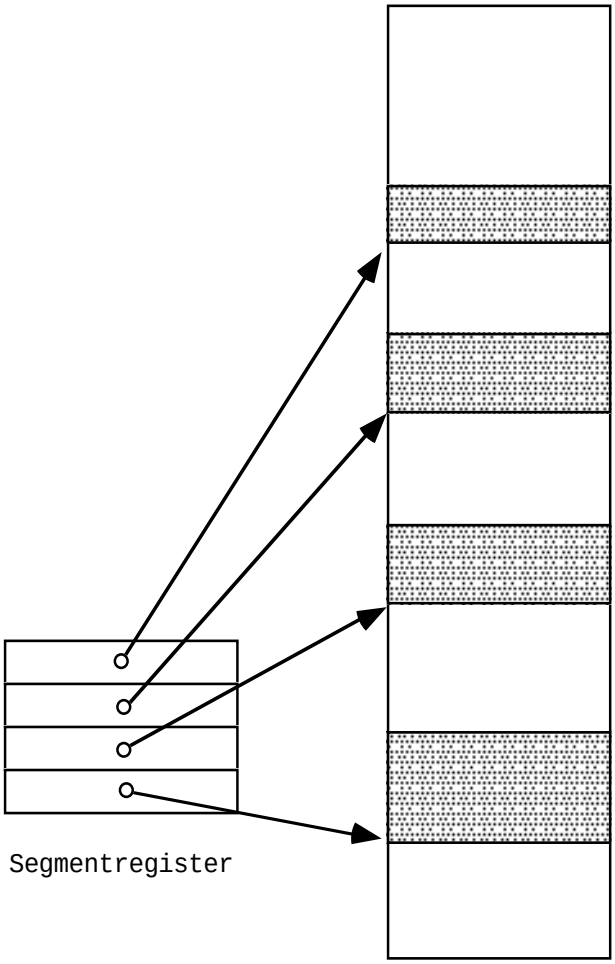
Da Ein- Auslagerung in Form von Seiten geschieht, wird sich im Laufe der Zeit eine **interne Fragmentierung** des Speichers einstellen: einzelne Kacheln sind nur zum Teil gefüllt.

Da von der Platte, dem primären Massenspeicher, sektorweise gelesen wird, macht man häufig die Seitengröße gleich einem Sektor: Dann beansprucht ein Ein-oder Auslagervorgang einer Seite genau einen Zugriff auf die Platte.

8.4.2. Segmente

Man teilt den einem Benutzer zur Verfügung gestellten Adressraum auf in Teilbereiche (Segmente) für Daten, Stackbereiche, Programm und ein Extra-Segment und bildet die logischen Adressen als Summe aus einer Adresse im Segment und der jeweiligen Segment-Anfangsadresse, die in eigenen Segmentregistern gehalten wird.

Bei der Adressbildung kann zugleich überprüft werden, ob die angegebene Adresse noch in die Grenzen fällt, die für die Größe des Segments angegeben wurde



Wird die Verwaltung der Segment-Anfangsadressen dem Betriebssystem übertragen und die Länge der Adressen innerhalb eines Segments begrenzt, kann damit eine Überwachung der Zugriffe eines Benutzers erreicht werden: Er kann nichts außerhalb der ihm zugewiesenen Segmente adressieren.

Legt man die Segmentgröße fest, und verwendet als Basisadressen nicht Vielfache der Segmentgröße, dann entsteht im Laufe der Zeit eine **externe Fragmentierung** des Speichers: zwischen Segmenten sind Speicherbereiche, die zu klein sind, um sie noch als Segmente nutzen zu können.

8.5. Adressumrechnung

Die Umrechnung der logischen Adresse in die physikalische Adresse im jeweils interessierenden Speicher soll nun untersucht werden.

Sie kennt zwei Grenzfälle.

- Der logische Adressraum ist gleich dem physikalischen Adressraum.

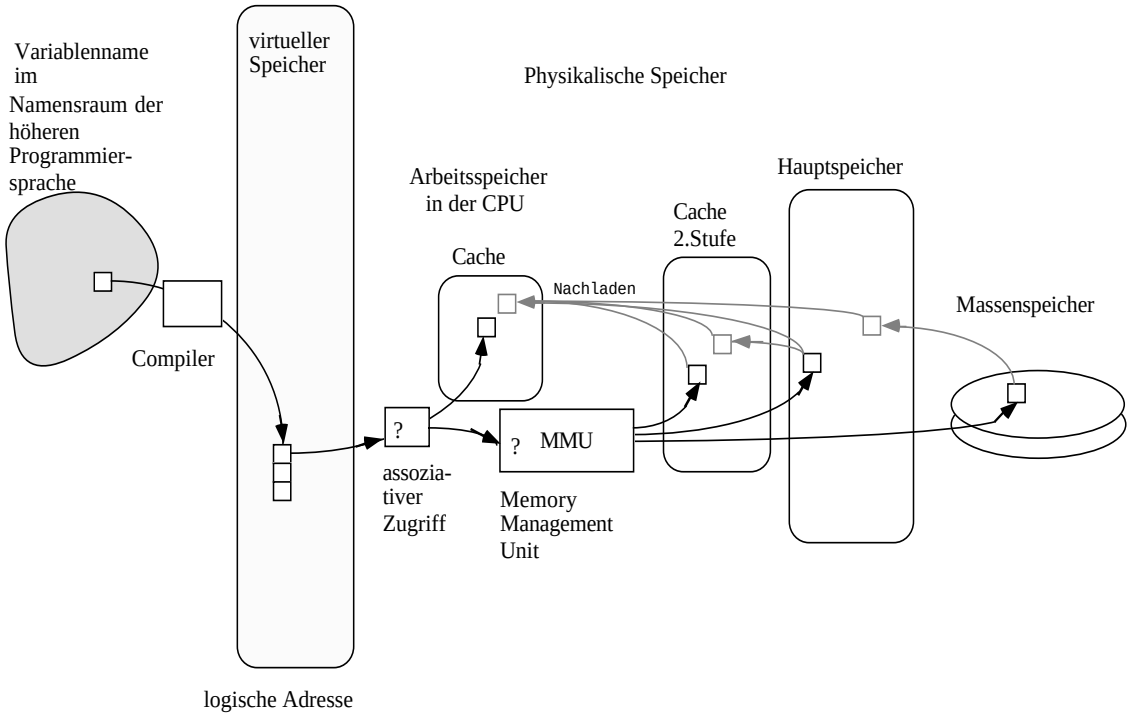
  Das ist der Fall bei kleinen Rechnern, z. B. 8-Bit-Mikroprozessoren mit 16-Bit-Adressen. Einer logischen Adresse entspricht eine physikalische Adresse. Der Fall, daß der physikalische Speicher kleiner ist als der logische Adressraum liegt hier in der Verantwortung des Programmierers.

  Der Hauptspeicher ist der Gesamtspeicher von 64 k Byte, ggf. aufgeteilt in interne Speicher, Spezialregister, die wie Speicherplätze angesprochen werden und Programmspeicher.

- Der logische Adressraum ist sehr viel größer als der dem Hauptspeicher vorgeschaltete Cache und der Hauptspeicher.

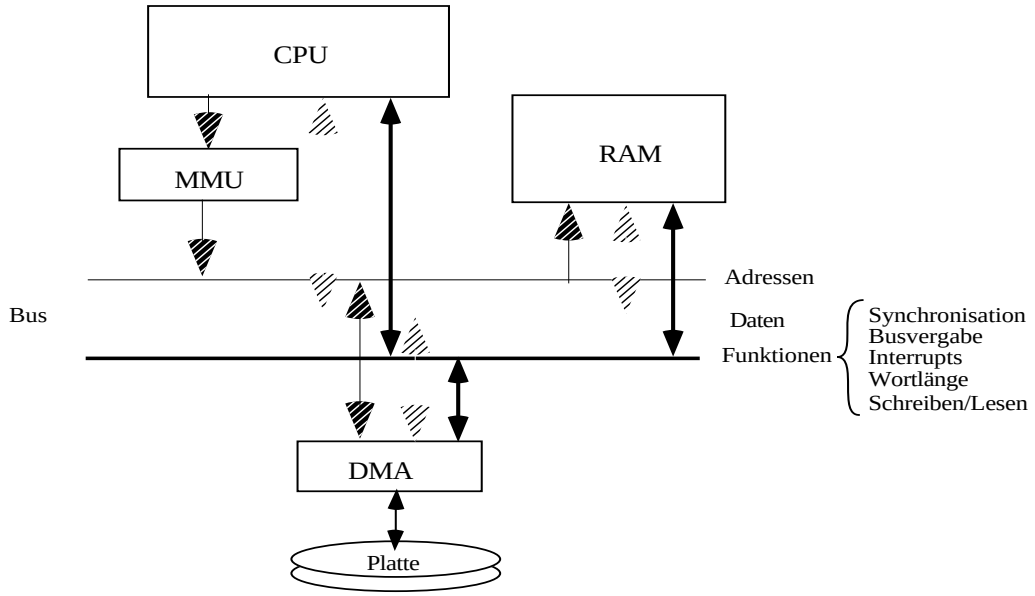
  Dann ist zunächst zu prüfen, ob die Adresse im Cache liegt , abgeprüft durch assoziativen Zugriff auf den Cache, sonst, ob aus dem Hauptspeicher nachgeladen werden kann, abgeprüft durch die Speicherverwaltungseinheit (memory management unit, MMU), die zunächst ggf. in einem weiteren Cache nachschaut und erst dann die Adresse im Hauptspeicher sucht und bei einem Fehlzugriff den Massenspeicher adressiert.

  Entsprechend müssen dann Daten in Cache oder Hauptspeicher nachgeladen werden.

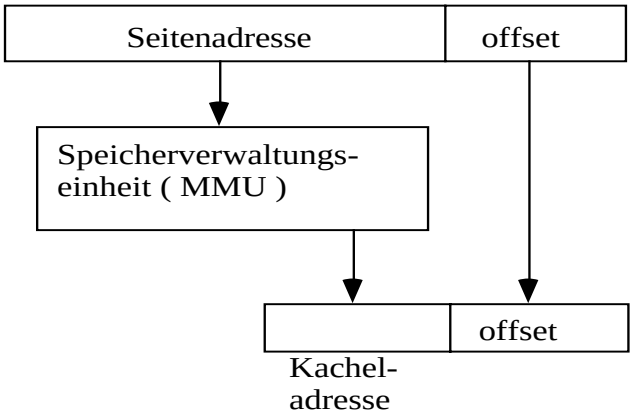


8.5.1. Umrechnung der logischen Adresse in die physikalische Adresse ohne Cache

Zwischen CPU und Speicher liegt eine Speicherverwaltungseinheit (memory management unit, MMU).



Die in ihr vorgenommene Umrechnung ist



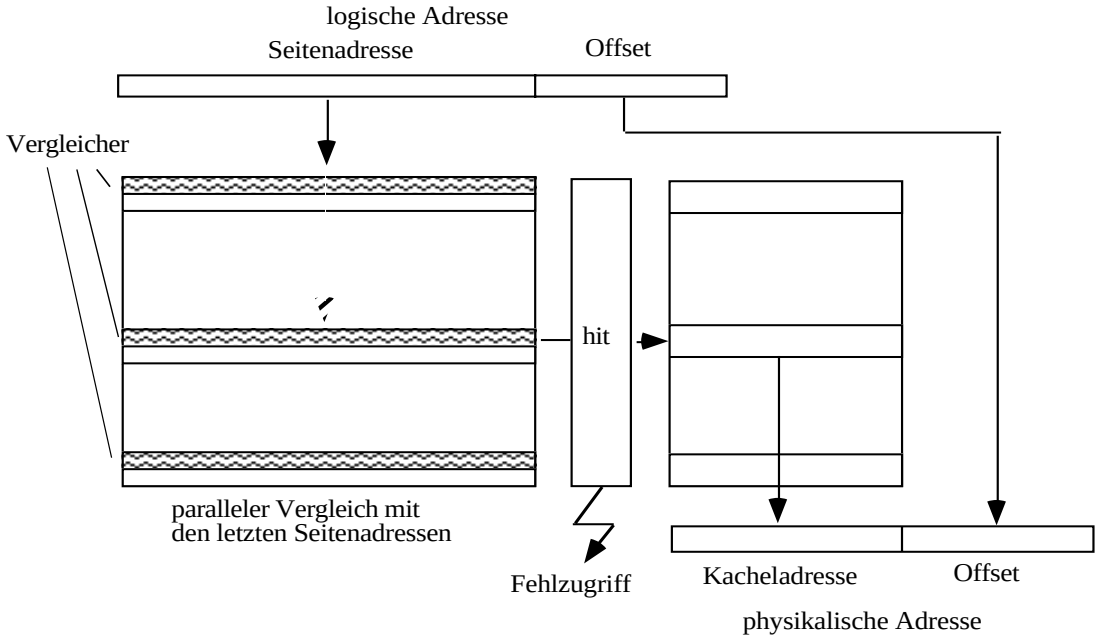
Dazu ein Beispiel: die logische Adresse habe 32 Bit, die physikalische Adresse 24 Bit (16 MB). Die Seitengröße sei 4 k B.

Dann ist die Seitenadresse 20 Bit groß und die Kacheladresse hat 12 Bit (4096 Kacheln).

Die Abbildung innerhalb der MMU geschieht dann in zwei Stufen:

8.5.1.1. Zugriff auf Assoziativspeicher in der MMU

In der MMU ist ein Hardware-Assoziativspeicher, der z. B. die letzten 32 Seitenadressen kennt. Diesen Speicher nennt man auch ein **translation lookaside buffer, TLB**.



Der Vergleich der aktuellen Seitennummer mit den letzten 32 Einträgen geschieht parallel und sehr rasch (~20 - 50 ns). Bei Erfolg wird die Kachelnummer ausgegeben und mit ihr der Hauptspeicher adressiert. Der Offset wird direkt übernommen. Bei einem Fehlzugriff (miss) wird die Seitennummer und Kacheladresse im obersten Platz des TLB in eine zufällig ausgewählte Seitennummer und Kacheladresse geschrieben und die Seitennummer, die den Fehlzugriff veranlasste, in den obersten Platz des TLB eingeschrieben und ein Interrupt in das Betriebssystem erzeugt mit Übergabe der Seitennummer.

8.5.1.2. Maßnahmen im Betriebssystem bei einem Seitenfehler (page fault)

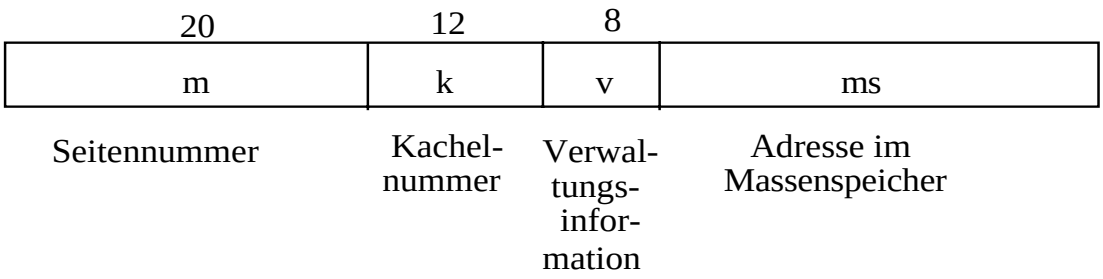
Zugriff auf eine Hashtabelle (scatter table oder randomized table) im Hauptspeicher mit der Seitennummer als Suchschlüssel.

Im Beispiel: 16 MB Hauptspeicher, 4096 Seiten

4 GB virtueller Speicher, 10^6 Seiten

=> eine Tabelle mit 8192 Einträgen (13 Bit) als Hashtabelle
Die zugehörige Hashfunktion bildet ab
20 Bit -> 13 Bit
m -> h (m) , möglichst zufällig.
h (m) ist der Index in der Tabelle.

Ein Tabelleneintrag hat die Form



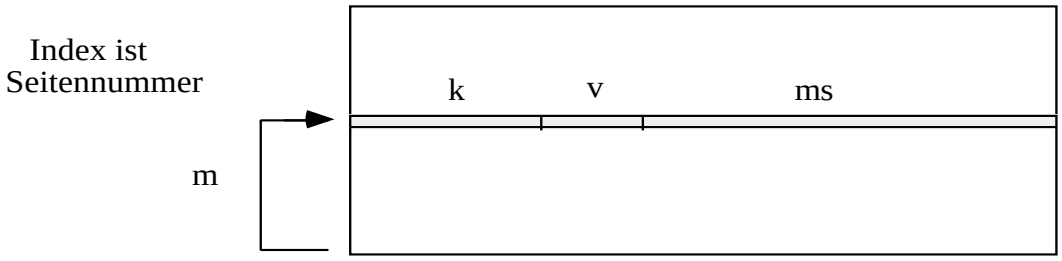
Bei halb gefüllter Tabelle sind weniger als zwei Zugriffe nötig um festzustellen, ob die Seitennummer im Hauptspeicher ist ; (Bei Mehrfachtreffern ist dann entschieden, ob die Seitennummer da ist).

a.) Bei erfolgreichem Zugriff wird im Verwaltungsinformationsteil des Tabelleneintrages nachgeprüft, ob der Zugriff im Kontext des Prozeßzustandes erlaubt ist (z. B. Schreiben in eine als "Programmcode" angewiesene Seite). Wenn der Zugriff erlaubt ist, wird die Kachelnummer in die erste Zeile des TLB im Speicherteil eingetragen und auf die Kacheladresse im Hauptspeicher zugegriffen.

Andernfalls verzweigt das Betriebssystem in eine Fehlerroutine, die den nicht erlaubten Zugriff auf eine Seite behandelt.

b.) Bei erfolglosem Zugriff wird in der Haupt-Seitentabelle die Adresse im Massenspeicher gesucht, bei der die Seite zu finden ist.

Diese Tabelle ist lang: im Beispiel 8 x 10^6 Byte mit einer logischen Organisation der Art

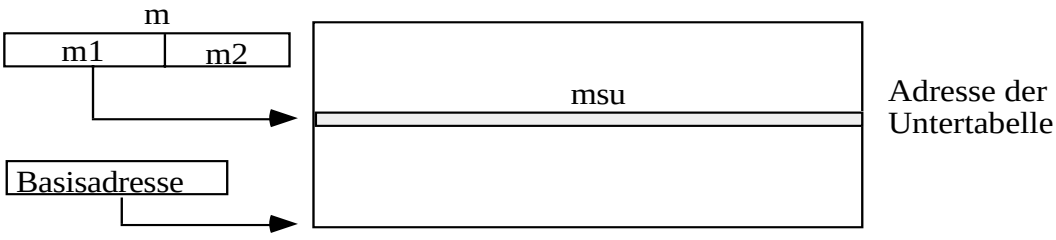


und wird nicht als Ganzes im Hauptspeicher stehen sondern nur in Untertabellen geladen werden.

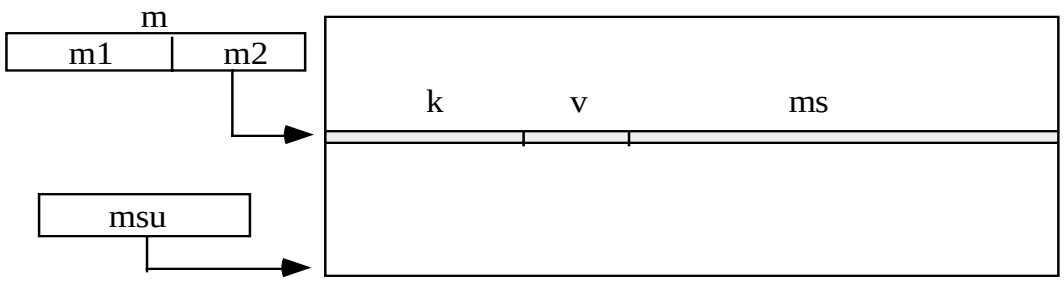
Dazu wird so vorgegangen:

Eine Seite wird aus dem Hauptspeicher verdrängt (welche ?) und an ihre Stelle ein Seitendirectory geladen. Die Basisadresse ist eine globale Adresse des Betriebssystems. Die Seitennummer m wird in zwei Selektoren m_1 und m_2 aufgeteilt:

m_1 indiziert das Seitentabellendirectory



Die dort gefundene Untertabelle verdrängt dann das Seitendirectory auf dieser Kachel.



Mit dem Selektor m_2 der Seitennummer wird die Untertabelle indiziert. In ihr stehen die Informationen zum Eintrag in der Hashtabelle. Nach Überprüfen, ob der Zugriff auf die Seite im Kontext des Prozesses gestattet ist, wird die Information in die Hashtabelle eingetragen, und mit der Massenspeicheradresse wird diese Kachel nun überschrieben mit der gesuchten Seite, die Kachelnummer in der Seitentabelle des TLB eingetragen, und auf die Adresse aus Kachelnummer und Offset zugegriffen.

Von außen her gesehen hat der Zugriff auf die Seite lange gedauert, sonst ist aber nichts geschehen.

Es müssen 3 - 4 Seiten aus dem Massenspeicher geladen werden: Eine Seite ist zu verdrängen, dann sind nacheinander das Directory, die Untertabelle und die gesuchte Seite auf die Kachel der verdrängten Seite zu laden. Wenn in die zu verdrängende Seite nicht geschrieben wurde, braucht sie nicht extra in den Massenspeicher zurückgeschrieben werden. Der Vorgang dauert mindestens 30 - 40 ms.

In vielen Systemen ist ein Seitenfehler in der Hashtabelle Anlaß, das laufende Programm zu unterbrechen, ein anderes rechenwilliges Programm zu starten, und erst wieder am unterbrochenen Programm weiterzuarbeiten, wenn der Plattencontroller via Interrupt zurückgemeldet hat, daß er die Seite in den Hauptspeicher geschafft hat.

8.5.2. Verdrängen einer Seite aus dem Hauptspeicher

Das Verdrängen von Seiten aus dem Hauptspeicher in den Massenspeicher ist unvermeidlich.

Wie kann es organisiert werden ?

LRU (least recently used) - Verfahren

Die am längsten nicht referenzierte Seite wird verdrängt.
Wie wird dann das Mitzählen organisiert ?

LFU (least frequently used) - Verfahren

Bei jeder Kachel wird ein Zähler mitgeführt, bei einem Zugriff inkrementiert und der kleinste Wert gesucht.

FIFO (first in first out) - Verfahren

Es wird eine Ringliste der Kachelnummern mitgeführt, und die erste geladene Seite auch als erste entfernt. Es muß allerdings in dem Fifo mit vermerkt werden, ob die Kachel Teil der Hashtabelle ist oder Systeminformation enthält: Die darf auf keinen Fall aus dem Hauptspeicher verdrängt werden, sondern sollte wieder an die Spitze der Liste geschrieben werden und dann erneut am Schwanz der Ringliste nachgeschaut werden.

Statistisches Verfahren

Es wird statistisch eine Kachel herausgepickt und verdrängt; es darf, s. o., allerdings keine Systeminformation in der Kachel stehen.

Bewertung der Verfahren

FIFO und statistisches Verfahren lassen sich am einfachsten implementieren und haben gegenüber LRU und LFU keine gravierenden Nachteile.

8.5.3. Aufgaben der Speicherverwaltungseinheit (MMU)

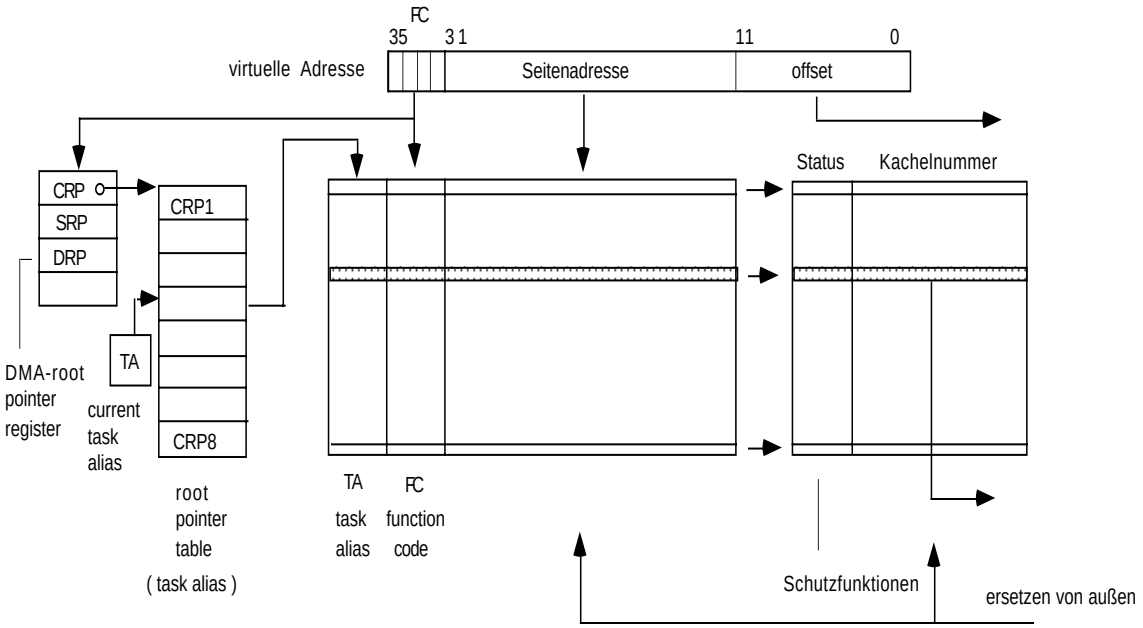
Die schon mehrfach angesprochene MMU hat folgende Aufgaben wahrzunehmen:

- Umrechnung der logischen in die physikalische Adresse im Hauptspeicher
- Speicherschutzfunktionen
  - Überprüfen der Korrektheit von Daten beim Nachladen vom Massenspeicher (Paritätsprüfung und Hammingcode mit 1-Bit-Fehlererkennung auf dem Speicher selbst)
  - Verwaltung der Schreib/Lese/Ausführungsberechtigung
    vergeben jeweils für eine Seite mit Überprüfung beim erstmaligen Zugriff auf die Seite
  - Sperren von Seiten außerhalb des dem Benutzer zugeteilten Bereichs. Das ist Voraussetzung für Multitasking: Die Benutzer sind voreinander zu schützen.

    Ausnahmen
    * Leseberechtigung für globale Daten
    * Ausführungsberechtigung für gemeinsam genutzten Code

    Schutzverletzungen (protection violation) lösen einen Interrupt in das Betriebssystem aus.
- Ver- und Entschlüsseln von Daten
  Insbesondere beim Übertragen über Netze und auf der Platte; ggf. dort durch Plattencontroller mit Schlüsselübermittlung durch die MMU.

Beispiel: MMU der MC 68030 CPU



Dargestellt ist der Adress-Übersetzungs-Cache (address translation cache, ATC), der voll assoziativ auf 22 Einträge zugreift.

Ein Eintrag im ATC besteht aus der Seitenadresse, einem Funktionscode FC und einem Task-Alias, einer Kennung der laufenden Task.

Es wird verglichen mit der Seitenadresse und dem Funktionscode der virtuellen Adresse; der Funktionscode spezifiziert eines von mehreren Root-Pointer-Registern (Ccurrent, Supervisor, DMA).

Sie enthalten jeweils einen Verweis auf eine Root-Pointer-Tabelle, die mit dem momentanen Task-Alias indiziert wird. Der Inhalt der Tabelle wird mit verglichen mit dem Task-Alias im ATC.

Damit kann die logische Organisation der Software in Form verschiedener Tasks überwacht werden. Sie laufen einmal für den Benutzer und sind in der Root-Pointer-Tabelle verzeichnet, auf die das CRP (current root pointer) Register verweist, oder sind Tasks der Supervisors, verzeichnet in einer Tabelle, auf die das SRP (supervisor root pointer) Register verweist oder sind Tasks der DMA, auf deren Root-Pointer-Tabelle das Register DRP (DMA root pointer) weist.

Im Ergebnisteil des ATC stehen Status(Schutz)informationen und die Kachelnummer.

Damit kann die MMU auch bei jedem erfolgreichen Zugriff die mögliche Verletzung von Schutzfunktionen erkennen.

8.6. Cache

Man macht die Beobachtung, daß in den meisten Fällen Daten und Programme **Lokalität** zeigen, d. h. daß die Daten, auf die in einer Zeitspanne zugegriffen werden, geclustert sind in Gruppen mit nebeneinanderliegenden Adressen und in Programmen Schleifen durchlaufen werden oder kurze Hilfsprozeduren aufgerufen werden, wobei wieder Adressen konsekutiv nebeneinanderliegen.

Es gibt Gegenbeispiele, insbesondere Programme in der Sprache **Lisp**, wo der zur Verfügung stehende Speicher schnell zu einem Spaghettihaufen von Pointern wird, da dort Daten und Programme als Listen formuliert sind und Programme als Listen erzeugt werden können.

Solange man von Daten- und Programmlokalität ausgehen kann, macht es Sinn, diese lokal gebrauchten Daten und Programme in schnellen Speichern an Bord der CPU vorzuhalten, den **Caches**.

Der Zugriff auf den Cache erfolgt in Blöcken von 4 - 128 - (4096) Byte und die Cachegröße sind 1 - 256 k Byte.

Man kann Caches vorsehen

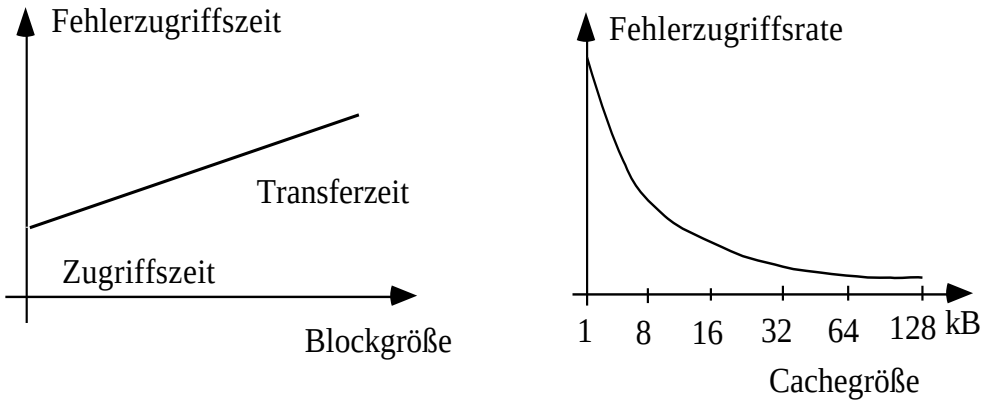
- für Befehle (Nurlesezugriff, kein selbstmodifizierender Code erlaubt)
- für Befehle und Daten gemeinsam
- je ein Cache für Befehle und Daten
  (damit erhält der Rechner intern eine Harvard-Architektur)
- Caches für Befehle im User- und BS-Mode getrennt oder durch ein extra Tag-Bit unterschieden.

8.6.1. Lesezugriff

Der Zugriff auf den Cache erfolgt über Assoziativspeicher, die zunächst den Vergleich der logischen Adresse mit den im Cache vorhandenen Adressen machen.

Im Trefferfall werden dafür **Trefferzeiten** von 1 - 4 Taktzyklen gebraucht.

Je nach Programm und Cachegröße werden ab und zu Fehlzugriffe auftreten; die **Fehlzugriffsrate** liegt zwischen 1 % und 20 %.



Im Falle eines Fehlzugriffs muß ein Block nachgeladen werden. Das dauert 8 - 32 Taktzyklen bei Blockgröße von 4 - 128 Byte.
Beim Verdrängen ist gegebenenfalls der Speicher zu aktualisieren, wenn in den Block bei Datencaches geschrieben wurde.

8.6.2. Schreibzugriff

Es gibt zwei Strategien:

write through

Bei jedem Schreibzugriff wird auch in den Speicher geschrieben.

Technisch wird das häufig so organisiert, daß erst in einen Puffer noch auf der CPU und von dort in den Speicher geschrieben wird (buffered write through). Der Schreibzugriff auf den Hauptspeicher geschieht dann parallel zu weiteren Zugriffen auf den Cache.

Es wird Datenkonsistenz garantiert. Bei Prozeßwechsel werden alle Seiten im Cache als ungültig (invalid) gekennzeichnet.

write back

Beim Schreiben wird im Cache für den Block ein "dirty bit" D = 1 gesetzt.

Bei Fehlzugriff wird der zu verdrängende Block in den Speicher zurückgeschrieben, wenn D = 1 war.

Bei Prozeßwechsel wird ein "cache flash" ausgelöst:
- alle Seiten mit D = 1 werden zurückgeschrieben
- alle Seiten werden ungültig erklärt (invalid)

Seiten (Blöcke), die als ungültig (invalid) gekennzeichnet sind, dürfen verdrängt werden.

Hier besteht keine Datenkonsistenz zwischen Speicher und Cache. Beim Zugriff auf den Bus von außen, z. B. durch eine DMA, benutzt man **snooping**: Die Busadresse wird auf den Cache umgelenkt.

Bei einem Hit - die Adresse ist im Cache - werden

- beim Lesen von Bus die Daten vom Cache geliefert
- beim Schreiben auf den Bus Daten in Speicher und Cache gemeinsam geändert.

Damit ist auch in diesem Fall die Datenkonsistenz gesichert.

Dahinter liegt ein allgemeines Problem:

Es liegen Daten in N Speichern parallel vor. Wie kann die Datenkonsistenz erhalten bleiben, wenn in einem Speicher ein Datum verändert wird ?

Die Lösung ist hier angegeben für zwei Speicher, Cache und Hauptspeicher.

Man kann auch zeitweilige Inkonsistenz zulassen:

Beim Schreiben in den Speicher wird das "dirty bit" des entsprechenden Blocks im Cache D = 1 gesetzt und nur in den Cache geschrieben.

Die **mittlere Speicherzugriffszeit** ist in allen Fällen

Trefferzeit + Fehlerzugriffsrate ∗ Fehlerzugriffszeit.

Bei jedem Fehlzugriff muß ein Block aus dem Cache verdrängt werden.

Mögliche Strategien sind hier LRU oder zufälliges Verdrängen.

LRU benötigt zusätzliche Hardware. Bei jedem Block j wird ein Zähler z(j) von ld N Bit mitgeführt, mit N = Anzahl der Blöcke im Cache . Er wird initialisiert mit 1.

Wird auf Block j zugegriffen, wird auf $z(j) \neq 0$ abgefragt.

Ist $z(j) \neq 0$ ==> $z(i) := z(i) + 1 \bmod N$ für alle $i \neq j$ und $z(j) := 0$.

Bei mehrfachem Zugriff hintereinander auf den gleichen Block ändert sich nichts.

Bei einem Fehlzugriff ist der Block mit maximalem Zählerstand zu verdrängen.

Zufälliges Verdrängen erzeugt die Blocknummer, die zu verdrängen ist, z. B. durch ein geeignetes rückgekoppeltes Schieberegister mit ld N Bit, einen Pseudozufallsgenerator, der Zahlen von 1 .. N erzeugt.

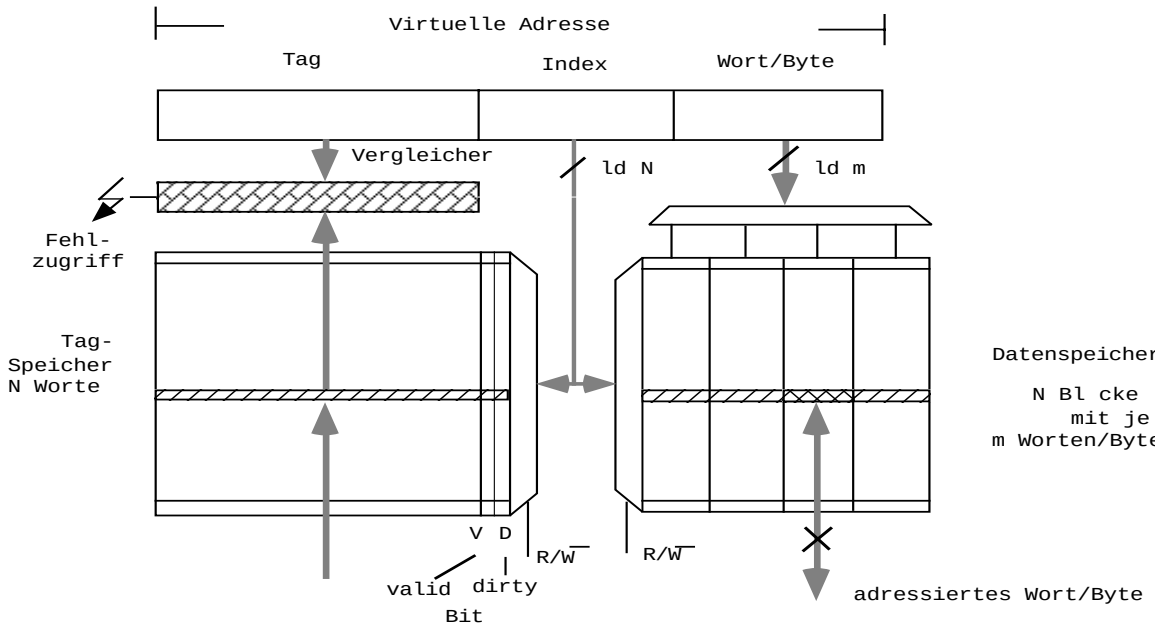
Bei größeren Caches sind beide Verfahren gleich gut; der schnellere Zugriff auf die zu verdrängende Blocknummer favorisiert zufälliges Ersetzen.

Das Ersetzen eines Blocks geschieht hier in Form von
- Demand Fetch: nach einem Fehlzugriff.

Eine zweite Möglichkeit ist
- Prefetch: Lade den Block mit der logischen Blocknummer j + 1 wenn auf Block j zugegriffen wurde.

Die Überprüfung, ob die virtuelle Adresse im Cache vorhanden ist, geschieht durch **Assoziativspeicher**.

8.6.3. Einfach assoziativer Cache



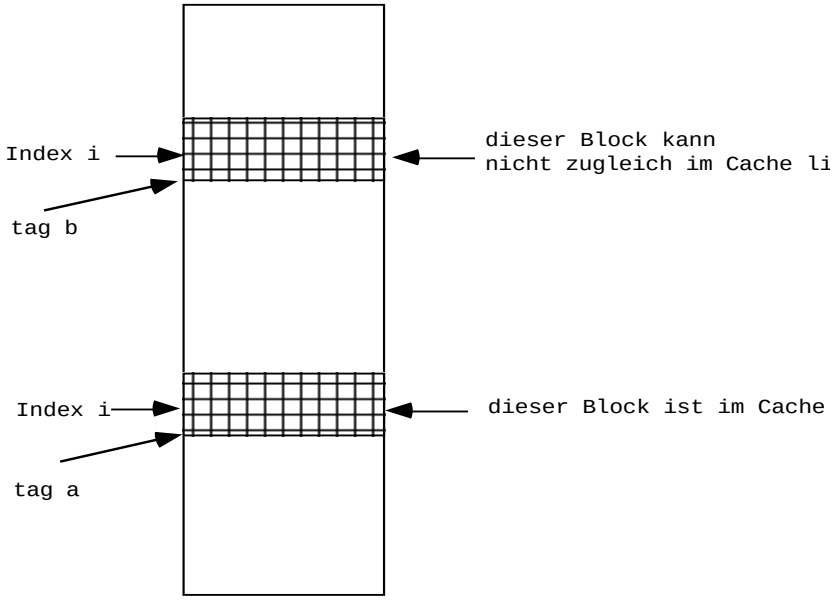
Die virtuelle Adresse wird aufgeteilt in drei Teile:

Ein Tag, das assoziativ verglichen wird, ein Index von ld N Bit, der eins aus N Worten im Tagspeicher anspricht und eine Byte / Wortadresse von ld m Bit, die ein Wort oder Byte aus dem Datenspeicher mit N Blöcken zu m Worten / Byte pro Block bei einem Treffer ausliest.

Ein Treffer liegt vor, wenn das in der virtuellen Adresse vorgegebene Tag mit dem Tag im Tagspeicher übereinstimmt. Bei Cache-Fehlzugriffen wird ein neues Tag in den Tagspeicher geschrieben und der zugehörige Datenblock ersetzt. Im Tagspeicher sind zwei zusätzliche Bit mitgeführt: "valid" und "dirty".

Vorteil des Verfahrens:
Der Zugriff ist schnell und man braucht nur einen Vergleicher; N kann relativ groß sein (2^{10} - 2^{12}).

Nachteil des Verfahrens:
Virtuelle Adressen mit dem gleichen Index können nur genau einmal im Cache gespeichert werden.



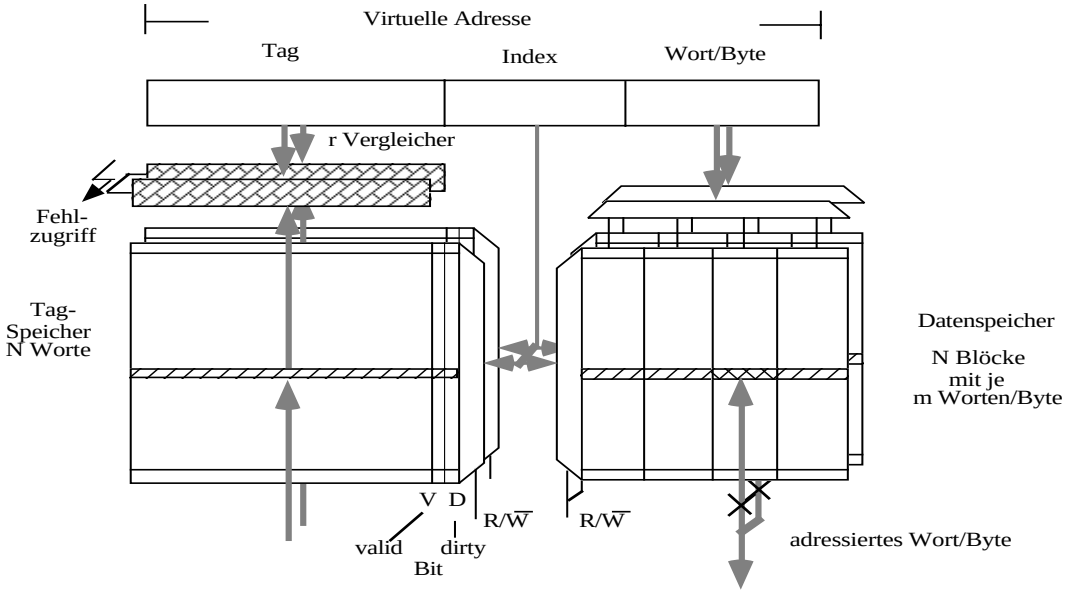
8.6.4. n-fach assoziativer Cache

Man hat n Tag- und Datenspeicher und n Vergleicher parallel.

Damit können n Blöcke mit dem gleichen Index im Cache gehalten werden, n = 2, 4, 8.

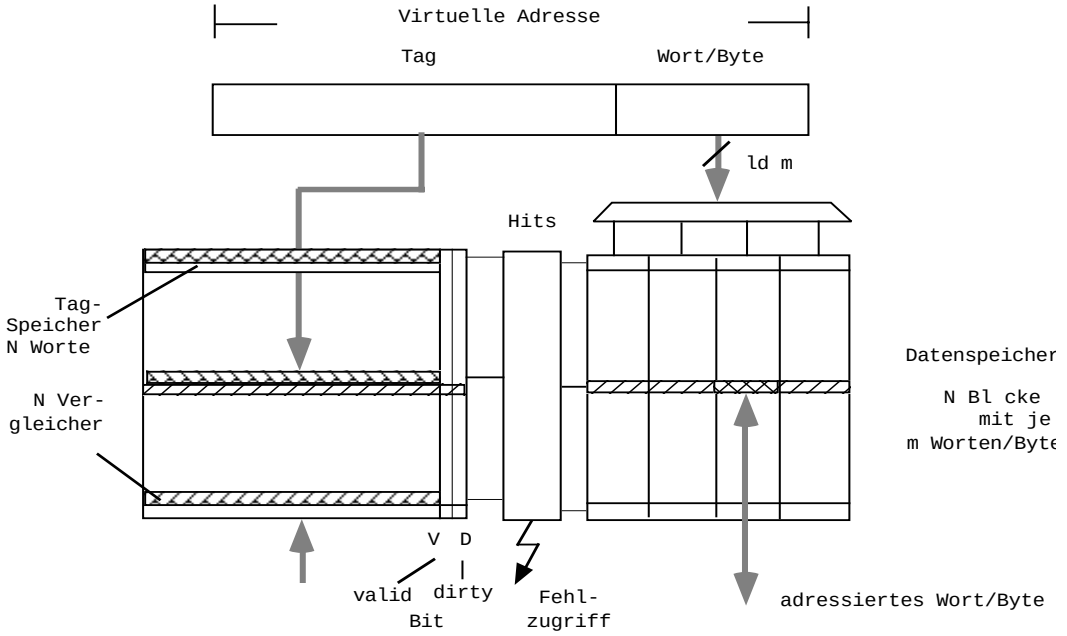
Man gewinnt durch die n-fache Hardware Flexibilität. Allerdings wird der Cache nur dann gefüllt sein, wenn die o. g. Überlappung beim gleichen Index passiert.

Sonst ist der entsprechende Eintrag im parallelen Tagspeicher leer.



8.6.5. Vollassoziativer Cache

Hier gibt es keinen Index; der Tagspeicher enthält N Tags und bei jedem Wort einen Vergleicher.



voll assoziativer Cache´

Ein Hit spricht dann den Datenspeicher an der entsprechenden Stelle an.

Da viele Vergleicher gebraucht werden ist N klein: 16 -64 - (128). Dafür wählt man große Blockgrößen, z. B. Seiten von 4086 Byte.

Dann enthält der Cache komplette Seiten.

Man hat dann allerdings große Transferzeiten beim Nachladen bei einem Cache-Miss in Kauf zu nehmen.