



**Universität Karlsruhe (TH)**  
Fakultät für Informatik  
Institut für Rechnerentwurf und Fehlertoleranz

---

## **Materialen zur Vorlesung**

# **Technische Informatik II**

**Prof. Dr. R. Dillmann**  
**Tamim Asfour**

---

# Inhaltsverzeichnis

<b>Inhaltsverzeichnis</b>	<b>i</b>
<b>1 Einleitung</b>	<b>1</b>
1.1 Historische Entwicklung der Rechenmaschinen . . . . .	1
1.2 Erste Gliederungsform: „Von-Neumann-Rechner“ . . . . .	3
1.3 Wichtige Begriffe: . . . . .	5
1.4 Hierarchische Gliederung der Steuerungsebenen im Rechnersystem . . . . .	6
1.5 Prinzipielle Struktur eines von-Neumann-Rechners . . . . .	7
<b>2 Mikroprozessoren</b>	<b>9</b>
2.1 Grundlegender Aufbau eines Mikroprozessors: . . . . .	9
2.1.1 Steuerwerk . . . . .	9
2.1.2 Rechenwerk (Operationswerk) . . . . .	12
2.1.3 Registersatz . . . . .	18
2.1.4 Adreßwerk . . . . .	18
2.1.5 Systembus-Schnittstelle ( <i>Bus Interface Unit</i> ) . . . . .	18
2.1.6 Interne Busse . . . . .	19
2.1.7 Weitere Komponenten . . . . .	19
2.2 Zeitverhalten des Systembusses . . . . .	20
2.3 Ablauf der Befehlsabarbeitung . . . . .	20
<b>3 Pipeline-Verarbeitung</b>	<b>21</b>
3.1 Pipeline-Prinzip . . . . .	21
3.2 Pipeline-Stufen und Pipeline-Register . . . . .	21
3.3 DLX-Pipeline . . . . .	21
3.3.1 Phasen der Befehlsausführung . . . . .	21
3.4 Pipelinekonflikte . . . . .	21
3.4.1 Daten- Steuerfluss- und Ressourcenkonflikte . . . . .	21
3.4.2 Software- und Hardware-Lösungen . . . . .	21
3.5 RISC & CISC . . . . .	21

<b>4</b>	<b>Organisation des Arbeitsspeichers</b>	<b>23</b>
4.1	Zusammenstellung der Speichertechnologien . . . . .	23
4.2	Wichtige Begriffe . . . . .	24
4.2.1	Größen zur Charakterisierung der Arbeitsgeschwindigkeit eines Speicherbausteins . . . . .	25
4.3	Klassifizierung von Halbleiterspeichern . . . . .	26
4.3.1	Festwertspeicher (ROM, <i>Read Only Memory</i> ) . . . . .	26
4.3.2	Schreib/Lese-Speicher (RAM, <i>Random Access Memory</i> ) . . . . .	27
4.3.3	Nicht-flüchtige RAM's (NVRAM, <i>non volatile RAM</i> ) . . . . .	27
4.4	Aufbau von Schreib-/Lese-Speicherzellen . . . . .	27
4.4.1	Statische MOS-Speicherzellen . . . . .	27
4.4.2	Dynamische MOS-Speicherzellen . . . . .	27
4.5	Organisation von Speicherbausteinen . . . . .	29
4.5.1	Gewinnung der Zeilen- und Spalten-Adressen aus der anliegenden Adresse . . . . .	29
4.5.2	Steuerlogik und Bausteinauswahl . . . . .	29
4.5.3	Möglichkeiten zur Auswahl eines Speicherelements . . . . .	30
4.5.4	Dynamische RAM-Bausteine . . . . .	31
4.5.5	Seitenzugriff bei DRAMs ( <i>Fast-Page-Mode-DRAM</i> ) . . . . .	33
4.5.6	EDO-RAM-Bausteine ( <i>extended data output</i> ) . . . . .	34
4.5.7	Synchrones DRAM ( <i>Synchronous DRAM (SDRAM)</i> ) . . . . .	34
4.5.8	Synchrones DRAM ( <i>Synchronous DRAM (SDRAM)</i> ) . . . . .	34
4.5.9	Double Data Rate RAMs (DDRAM) . . . . .	35
4.6	Auffrischen dynamischer RAM-Bausteine . . . . .	35
4.7	Speicherhierarchie . . . . .	37
4.8	Cache-Speicher . . . . .	38
4.8.1	Lesezugriffe . . . . .	39
4.8.2	Schreibzugriffe . . . . .	39
4.8.3	Aufbau des Cachespeichers . . . . .	40
4.8.4	Ersetzungsstrategien . . . . .	43
4.9	Virtuelle Speicherverwaltung . . . . .	44
4.9.1	Segmentierungs- und Seitenwechselverfahren . . . . .	44
4.9.2	Adress-Umsetzung bei Segmentierungsverfahren . . . . .	45
4.9.3	Adress-Umsetzung bei Seitenwechselverfahren . . . . .	47
4.9.4	Probleme der virtuellen Speicherverwaltung . . . . .	51
4.10	Direkter Speicherzugriff ( <i>direct memory access DMA</i> ) . . . . .	52
4.10.1	Prinzip des direkten Speicherzugriff . . . . .	53
4.10.2	DMA-Übertragungsraten . . . . .	53

<b>5</b>	<b>Digitale Signalprozessoren und Mikrocontroller</b>	<b>57</b>
	<b>Literatur</b>	<b>59</b>



# Vorwort

Der vorliegende Umdruck stellt ein Begleitmaterial zur Vorlesung „Technische Informatik II“ an der Fakultät für Informatik an der Universität Karlsruhe dar. Der Umfang der Themen ist im Allgemeinen *nicht* korreliert mit ihrer Wichtigkeit für die Vorlesung oder die Klausur. Es ist deshalb davon abzuraten, sich nur anhand dieser Unterlagen für die Klausur oder evtl. eine mündliche Prüfung vorzubereiten. Ausdrücklich betont sei, dass der vorliegende Text nicht als ein „Skript“ anzusehen ist.

Die Unterlagen sind unvollständig und beinhalten bestimmt einige Fehler. An die Studentinnen und Studenten sei deshalb die herzliche Bitte gerichtet, durch Hinweise auf Fehler, Mängel zur weiteren Verbesserung und Erweiterung dieser Unterlagen beizutragen.

Karlsruhe, den 30 Juli 2003

T. Asfour



# Kapitel 1

## Einleitung

### 1.1 Historische Entwicklung der Rechenmaschinen

**1623 Wilhelm Schickard:** Mechanische Rechenmaschine für die vier Grundrechenarten.

**1642 Blaise Pascal:** Mechanische Rechenmaschine; nur Addition, Subtraktion über Addition des Komplements. (Mechanik wahrscheinlich nicht funktionsfähig bei Dauerbetrieb).

**1672 Gottfried Wilhelm Leibniz:** Mechanische Rechenmaschine für die 4 Grundrechenarten (Mechanik nicht voll funktionsfähig).

**1722-74 M. Hahn:** Mech. Rechenmaschine für die 4 Grundrechenarten (Mech. Probleme weitgehend gelöst).

**1822 Charles Babbage:** *Difference Engine*: Nicht funktionsfähig, da nicht fertig gebaut. *Analytical Engine*: Erster *Rechenautomat*; nur als Konzept entwickelt, nicht gebaut. Erstmals Programmsteuerung (über Lochkarten). Möglichkeit, im Programm zu springen. Enthält bereits die meisten Funktionsbaugruppen moderner Rechenautomaten. *Ada Augusta Countess of Lovelace* griff das Konzept auf und schrieb ein Programm zur Berechnung der Bernoulli-Zahlen.

**1936 Konrad Zuse** Entwurf einer programmgesteuerten Rechenmaschine mit folgenden Funktionsbaugruppen: Eingabe-, Speicher-, Rechen-, Plansteuer- und Ausgabewerk. Anwendung des Dualsystems und der halbblogarithm. Zahlendarstellung (Gleitpunktdarstellung) sowie des Aussagenkalküls. 1941 stellte Zuse die erste voll arbeitsfähige programmgesteuerte, elektromechanische Rechenmaschine der Welt vor. Gebaute Maschinen: Z1 (1937), Z2, Z3 (1941), Z4 (1942-50). Ein Nachbau der Z3 steht im Deutschen Museum in München.

Die erste Maschine Z1 wurde 1938 fertig gestellt und war ein völlig mechanisches Gerät, das die vier Grundrechenoperationen ausführen sollte. Da die mechanischen Bauteile nicht zuverlässig arbeiteten, stieg Konrad Zuse auf Reliastechnik um. Im Jahre 1941 stellte Zuse eine funktionsfähige Maschine der Öffentlichkeit vor. Die Z3 war, nach einigen Experimenten mit einer „Zwischen-Maschine“ namens Z2, geboren. Die Originale beider Maschinen sind im zweiten Weltkrieg verlorengegangen. Konrad Zuse hat später sowohl die Z3 (1966) als auch die Z1 (1987 bis 1989) mit fast achtzig Jahren nachgebaut.

- 1938 Howard Aiken:** Bau des Rechenautomaten ASCC (Automatic Sequence Controlled Computer, auch „Harvard Mark I“ genannt) an der *Harvard University* (1944). Dezimales Zählrad-Prinzip, sehr große Maschine, relativ schnelle Funktion (Addition von 23-stelligen Dezimalzahlen in 0.3 sec, Multiplikation: 65 sec, Division: 115 sec. Erster programmgesteuerter Rechenautomat der USA.
- 1943-49 I. P. Eckert, J.W. Mauchly:** Bau des Rechenautomaten ENIAC (*Electronic Numerical Integrator and Calculator*) an der *University of Pennsylvania*. Anwendung elektronischer Schaltelemente: 17.468 Elektronenröhren, 1500 Relais, 30 t Gewicht, 174 kW Leistungsverbrauch. Addition zweier 10-stelliger Zahlen in 0.2 ms, Multiplikation in 2.8 ms, Programmierung durch Verschalten von Schalttafeln (sehr umständlich und fehleranfällig).
- 1944-46 J. v. Neumann, A. W. Burcks, H.H. Goldstine:** Rechenautomat EDVAC (*Electronic Discrete Variable Automatic Computer*) an der *Princeton University*. Das Konzept lehnte sich an das von Charles Babbage, welches eine klare Trennung zwischen Speicher und Rechenwerk vorsah (s. Abbildung 1.1). Erster speicherprogrammierter Rechenautomat. Das Programm mit Befehlen und Adressen wurde erstmals *intern* gespeichert und dabei in der gleichen Art codiert und dargestellt, wie die zu verarbeitende Information. Adressen und Befehle konnten damit von der Maschine selbst verändert werden. Aufgrund bedingter Befehle war die Maschine selbst in der Lage, den Programmablauf in Abhängigkeit von Zwischenergebnissen zu verändern.

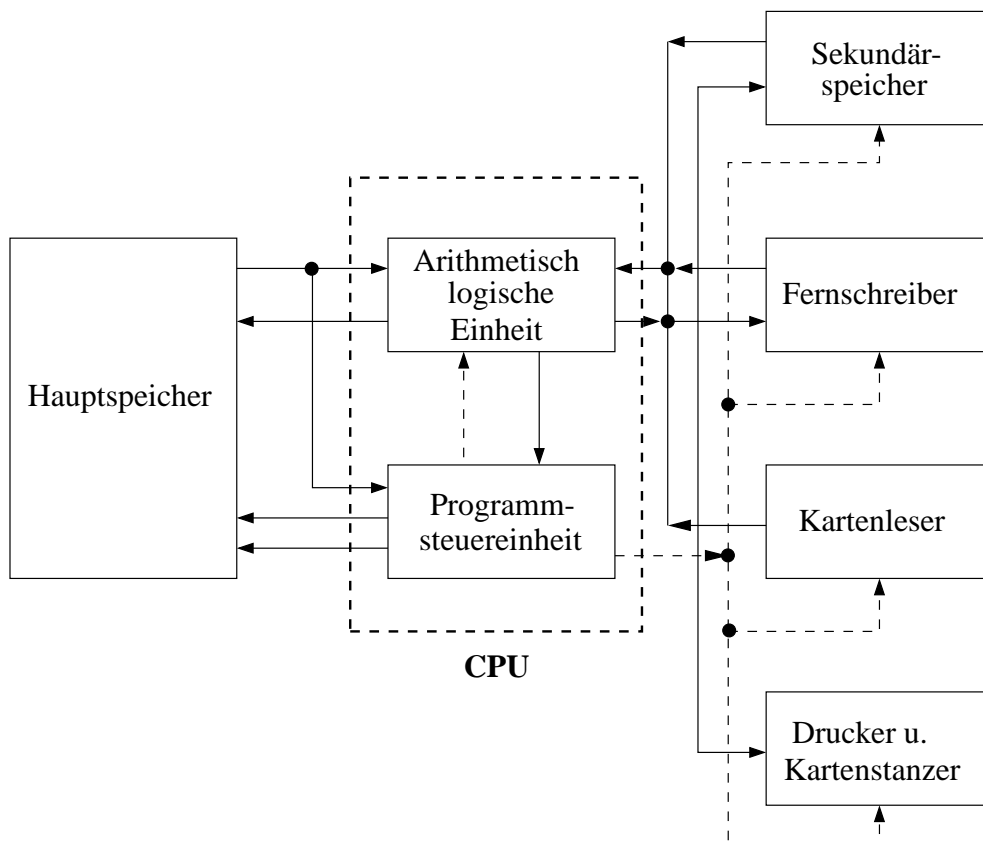


Bild 1.1: Aufbau des EDVAC

## 1.2 Erste Gliederungsform: „Von-Neumann-Rechner“

Nach dem 1944-1946 von Burks, Goldstine und von-Neumann in Princeton entwickelten Rechnerkonzept EDVAC besteht ein Rechner aus fünf **Komponenten**: Eingabe, Rechenwerk, Steuerwerk, Speicher, Ausgabe

- Speicher als lineare Folge von adressierten Speicherzellen organisiert,
- Programme stehen als lineare Folge von Befehlen und Daten im Speicher:
  - Abarbeitung mit Hilfe eines Befehlszählers,
  - Sprünge möglich,
- Ausführung der arithmetischen Operationen durch das Rechenwerk,
- Ausführung bzw. Abarbeitung der Programme durch das Steuerwerk („Leitwerk“),
- Programme und Daten stehen in gleicher Form im Speicher (als Folge von Binärzeichen),
- Befehle bestehen aus Operationsteil, Adressteil und weiteren speziellen Angaben (z.B. zur Adressmodifikation, Adresssubstitution, Indexregister usw.)

- Abarbeitung eines Befehls in definierten festgelegten Phasen:
  - Befehl holen und decodieren
  - Adresse aufbereiten
  - Operand(en) holen
  - Befehl ausführen und Ergebnis zurückschreiben

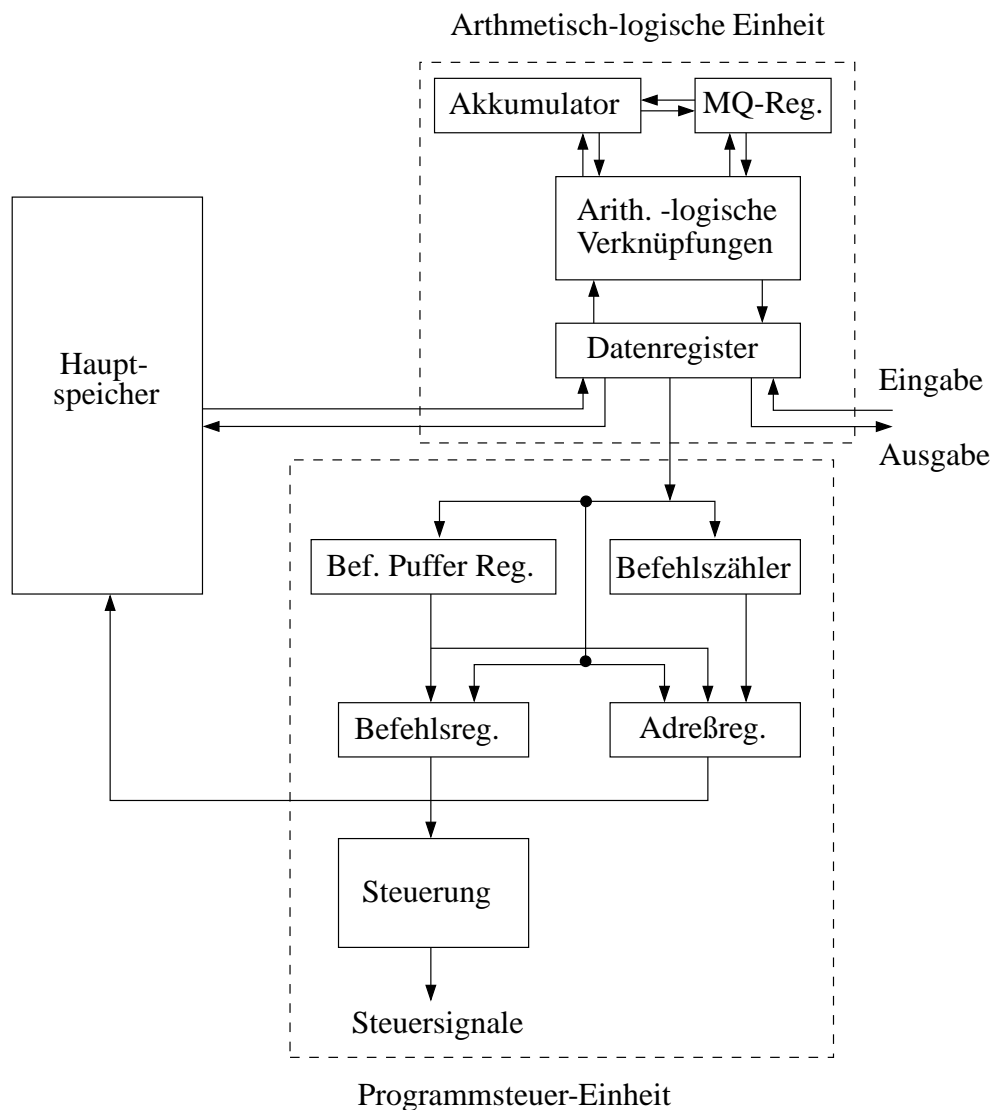


Bild 1.2: Aufbau des IAS-Rechners

Das Ziel war es, Rechnerimplementierungen mit so wenig Hardware wie möglich zu erreichen, da diese damals teuer war. Software spielte damals noch keine Rolle.

## 1.3 Wichtige Begriffe:

- **Hardware:** Alle zur physikalischen Implementierung einer Funktionalität notwendigen Teile. Das beinhaltet allen mechanische und elektronischen Teile eines Rechners.
- **Software:** Alle zur logischen Implementierung einer Funktionalität notwendigen Teile. Das sind alle Programme, die auf dem Rechner ablaufen, wie z. B. das Betriebssystem und die Benutzerprogramme, ... usw.
- **Firmware:** Bildet eine Mittelstellung zwischen Hardware und Software. Interne Programme (Mikroprogramme), welche die Ausführung der Befehle steuern. Dazu werden häufig auch das sog. *BIOS* (*Basic Input/Output System* des Betriebssystems, dessen Aufgabe in der Steuerung der Ein- und Ausgabegeräte liegt. Firmware ist dem Benutzer normalerweise nicht zugänglich.

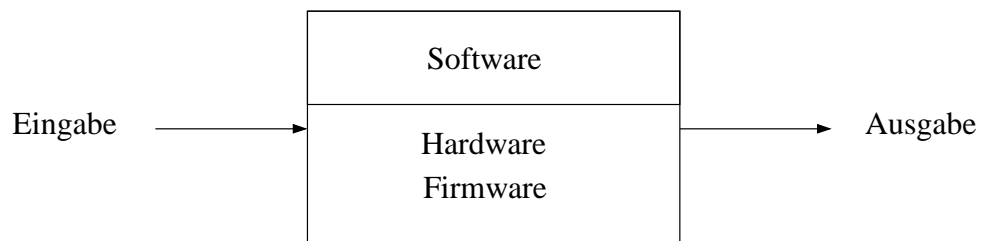


Bild 1.3: Gliederung eines Rechners



## 1.5 Prinzipielle Struktur eines von-Neumann-Rechners

Das von-Neumann-Konzept eines Rechners sieht folgende Komponenten vor:

- **Zentraleinheit (engl.: *central processing unit, CPU*):** Sie enthält alle Komponenten, die zur Abarbeitung eines Programms notwendig sind. Sie besteht aus einem Steuerwerk, das die Befehle interpretiert und einem Rechenwerk, das Daten gemäß eines Programms bearbeitet.
- **Speicher:** Gemeinsamer Speicher für Programme und Daten<sup>1</sup>. Der Speicher lässt sich unterteilen in *Peripheriespeicher*<sup>2</sup> zur permanenten Speicherung großer Datenmengen und *Hauptspeicher*<sup>3</sup> zur Aufnahme von Befehlen und Operanden im laufenden Betrieb unterteilen.
- **Ein-/Ausgabe-Einheiten:** Alle Geräte zur Eingabe von Daten und zur Ausgabe der verarbeiteten Daten (Bildschirm, Maus, Tastatur, ... usw.).
- **Verbindungseinrichtung:** Die Gesamtheit aller Leitungen, über die die oben erwähnten Komponenten Daten austauschen.

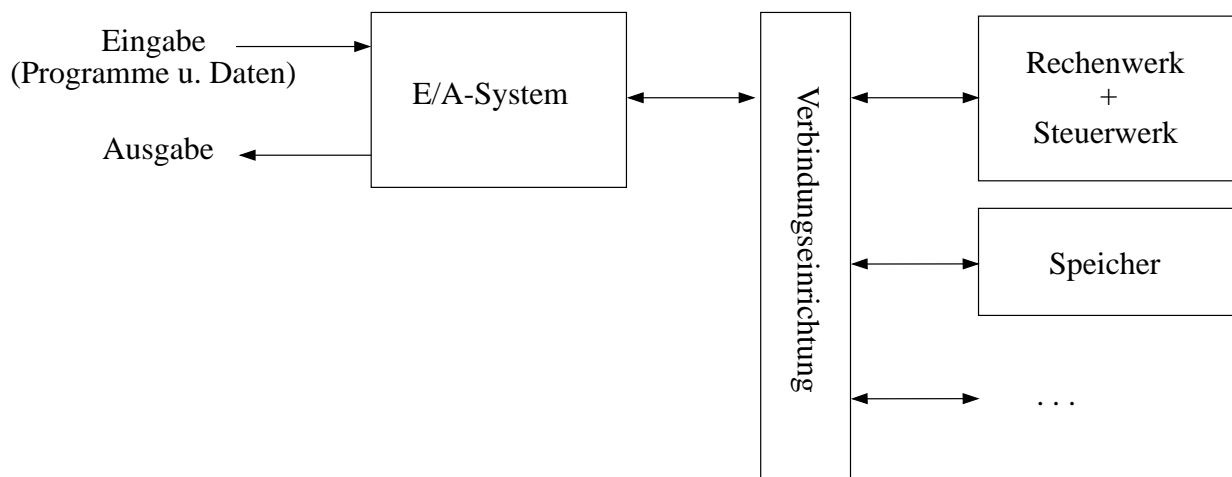


Bild 1.5: Prinzipielle Struktur eines von-Neumann-Rechners

<sup>1</sup>Alternativ: **Harvard-Architektur** mit getrennten Programm- und Datenspeicher.

<sup>2</sup>Auch *Massenspeicher* genannt

<sup>3</sup>Auch *Arbeitsspeicher* genannt



# Kapitel 2

## Mikroprozessoren

In diesem Kapitel werden die internen Komponenten eines Mikroprozessors beschrieben. Bei modernen Mikroprozessoren sind Erweiterungen der hier beschriebenen Komponenten, wie Cache-Speicher, virtuelle Speicherverwaltungseinheit auch dem Prozessor-Chip integriert. Diese Komponenten werden in Kapitel ?? behandelt.

### 2.1 Grundlegender Aufbau eines Mikroprozessors:

In Bild 2.1 ist der interne Aufbau eines Mikroprozessors dargestellt. Im Folgenden gehen wir auf die einzelnen Komponenten näher ein. Eine ausführliche Beschreibung findet man in [Bähring 02b]

#### 2.1.1 Steuerwerk

Das Steuerwerk ist ein synchrones Schaltwerk, welches die Systemkomponenten steuert. Die Realisierung des Steuerwerks kann mit Hilfe von festverdrahteter Logik<sup>1</sup> oder durch ein Mikroprogramm<sup>1</sup>, die in einem Festwertspeicher abgelegt sind. Im Steuerwerk lassen sich die folgenden Bestandteile unterscheiden:

- **Befehlsregister (*Instruction Register*)** enthält den den Teil des gerade ausgeführten Maschinenbefehl, der die gewünschte Operation und die zu ihrer Ausführung benötigten Prozessorressourcen spezifiziert. Das Befehlsregister ist häufig als FIFO-Registerspeicher realisiert. Die Gründe hierfür liegen auf der einen Seite in der unterschiedlich langen Befehlsformate, bei denen der OpCode ein oder mehrere Register belegt. Auf der andern Seite werden diese Register zum Vorabladen von Befehlen (*OpCode Prefetching, Pipelining*) verwendet. Diese Maßnahme zur Steigerung der Verarbeitungsgeschwindigkeit, die nächsten auszuführenden Befehle in
- **Dekodierer** wird von den Statussignalen beeinflusst und erzeugt die Steuersignale zur Ausführung der Befehle. Somit stellt diese Einheit die eigentliche Steuerlogik dar. So würde eine gesetztes IE-Bit (*Interrupt Enable*) in diesem Register bedeuten, dass der Prozessor Unterbrechungsanforderung zu

---

<sup>1</sup>In diesem Fall spricht man von einem **Mikroprogramm-Steuerwerk**

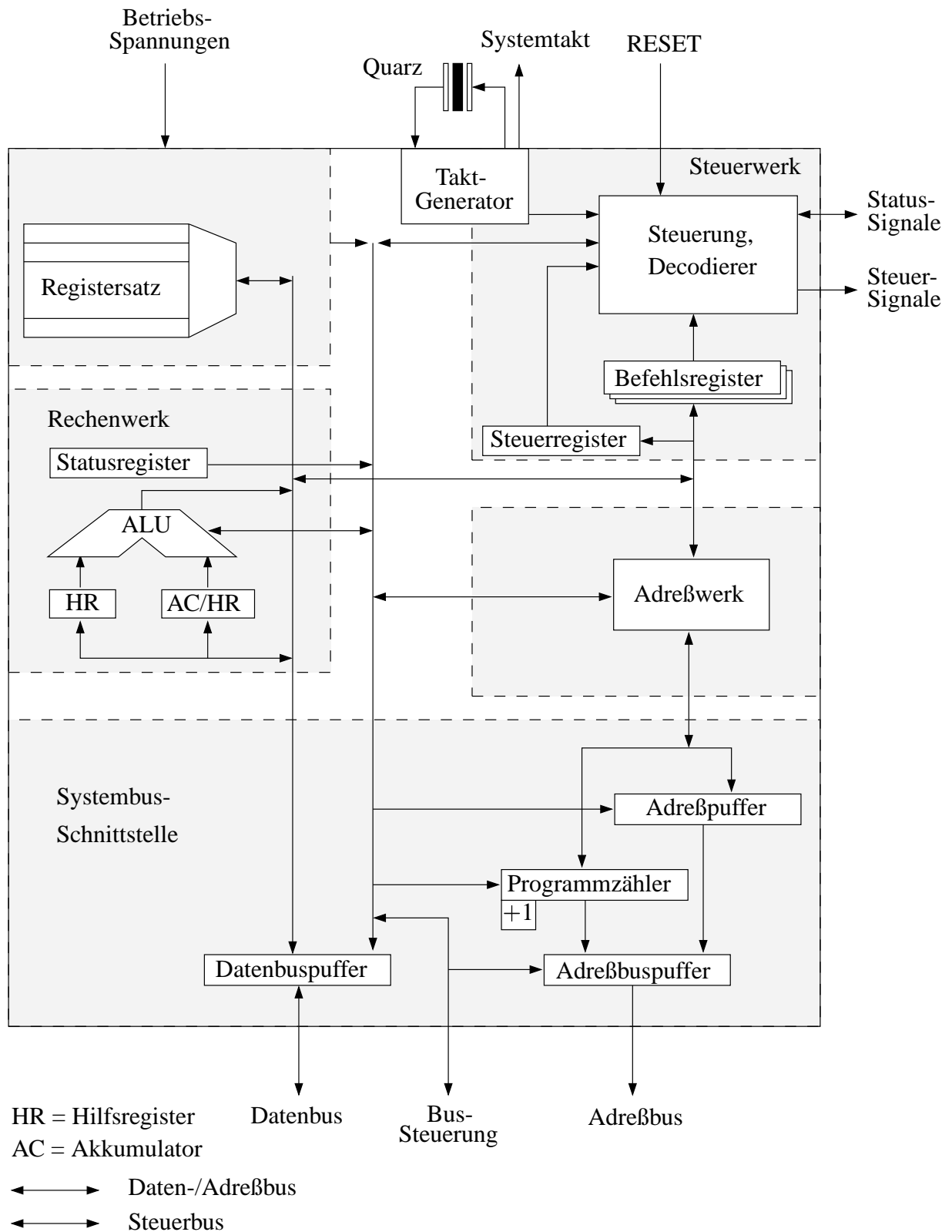


Bild 2.1: Aufbau eines Mikroprozessors

- **Steuerregister (Control Register)** Die in diesem Register gespeicherten Informationen können die augenblickliche Arbeitsweise des Steuerwerks beeinflussen. Die einzelnen Bits dieses Registers besitzen unterschiedliche Funktionen. So würde ein gesetztes IE-Bit (*Interrupt Enable Bit*) bedeuten, dass Interruptsanforderungen am Interrupteingang des Prozessors stattgegeben werden soll.
- **Taktgenerator** erzeugt den vom externen Quarz festgelegten Systemtakt. Häufig werden dem Taktgenerator weitere Aufgaben übertragen, wie z. B. die Erzeugung eines mit dem Prozessortakt synchronisierten Rücksetzesignals (RESET)

### 2.1.1.1 Ein-/Ausgabesignale des Steuerwerks

- Signale zur Zuteilung des Systembusses: Können andere Komponenten aktiv auf den Bus zugreifen (Bsp. DMA-Controller, Coprozessoren), so muss der Zugriff auf den Bus geregelt werden. Hierzu dient ein 3-Leitungs-*Handshake*-Betrieb.
- Signale der Unterbrechungsanforderung: Interrupteingänge
- Weitere Signale zur Fehlermeldung anderer Komponenten und zur Mitteilung anderer Komponenten über die augenblicklich ausgeführten Operation.

### 2.1.1.2 Fallstudie: Das Steuerwerk des Intel 80486

Bild 2.2 stellt schematisch das Steuerwerk des Intel-Prozessors 80486 dar. Es lassen sich folgende Bestandteile unterscheiden:

- **Befehlsregister-Block:** in Form eines FIFO-Speichers mit 32 Bytes (*Prefetch-Queue*). Die *Prefetch*-Steuerung sorgt für weitestmögliche Füllung der Queue. Die Füllung erfolgt meist aus dem *on-chip-cache* über einen 128 Bit breiten Datenpfad. Muß jedoch ein Befehl aus dem Hauptspeicher geholt werden, so haben diese Zugriffe höchste Priorität. Die in diesem Fall evtl. auszugebenden Daten werden in Schreibpuffern (*write buffers*) zwischengespeichert. Die Befehle in der Warteschlange bestehen aus OpCode und Operanden.
- **Befehls-Vordecoder:** Hier werden die Befehle aus der Warteschlange vorverarbeitet. Die unmittelbar in den Befehlen angegebenen Operanden und Adreßdistanzen (*offset*) zu bestimmten Basisadressen werden dabei in einem separaten Speicher abgelegt.
- **Befehls-FIFO:** Die vordecodierten Befehle gelangen in den Befehls-FIFO (eine weitere Warteschlange für maximal 3 Befehle).
- **Befehls-Decoder:** Entnimmt den obersten vordecodierten Befehl aus dem Befehls-FIFO und ermittelt die Startadresse des zugehörigen Mikroprogramms.
- **Mikrobefehls-Steuerung:** Synchrones mikroprogrammiertes Schaltwerk, das die Steuersignale erzeugt und die Meldesignale interpretiert.
- **Zeit-Steuerung:** Synchronisiert die erzeugten Steuersignale mit dem Systemtakt.



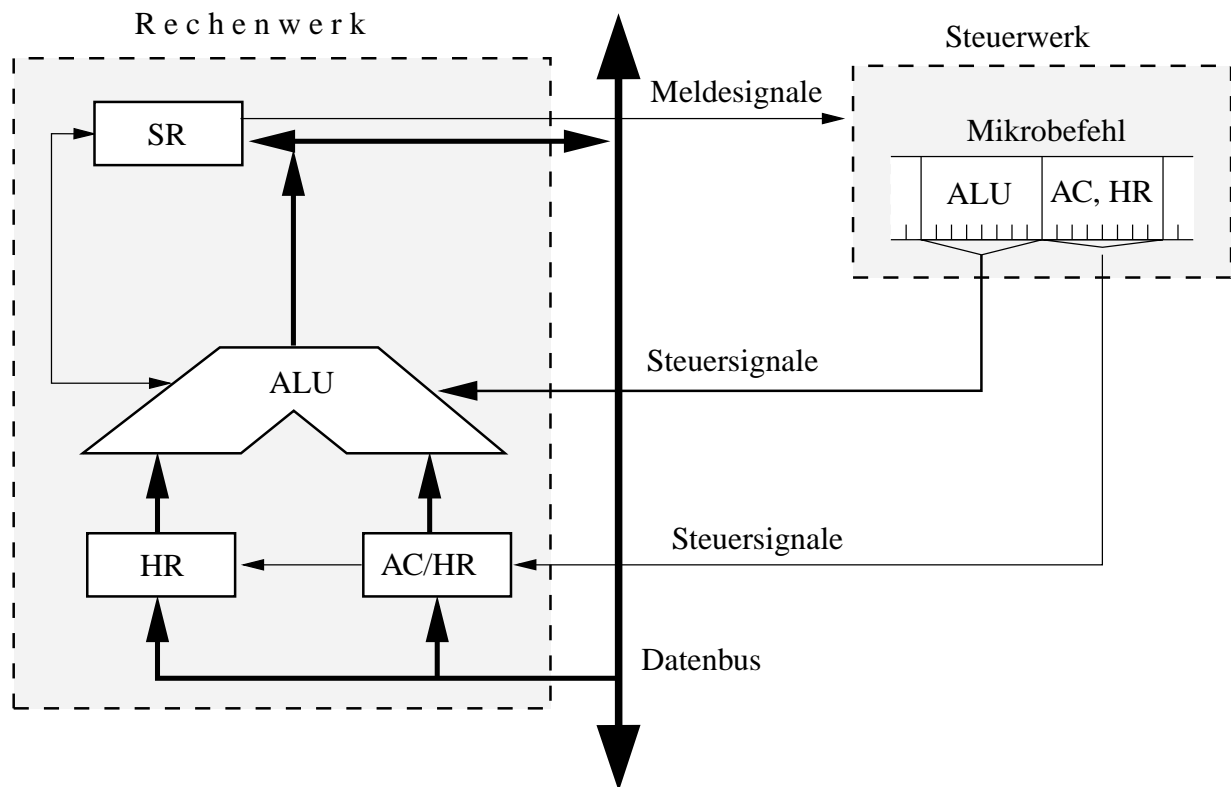


Bild 2.3: Prinzipieller Aufbau und Komponenten eines Rechenwerkes

Neben den Dateneingängen besitzt die ALU weitere Eingänge, welche mit *Steuersignalen* vom Steuerwerk zur Ablaufsteuerung der ALU-Operationen belegt sind und *Meldesignale* für den Status der abgelaufenen Operation. Dieser Status wird im Statusregister **SR** zwischengespeichert. Die einzelnen Bits dieses Registers spiegeln das Ergebnis der ALU-Operation wieder.

Der Akkumulator **AC** ist ein spezielles Register, welches die Ergebnisse der ALU „aufsammelt“. Der Akkumulator besteht meistens aus dem eigentliche Akkumulator-Register und einen nachgeschalteten Hilfsregister. In Bild 2.6 ist der Akkumulator aus einem Akkumulator-Register (AC) als Vorspeicherregister und einem Hilfsregister (HR) aufgebaut. Prinzipiell kann der Akkumulator aus *Master-Slave-Flipflops* oder aus zwei getrennt getakteten Registern aufgebaut werden.

### 2.1.2.1 Beispiele von Rechenwerken

Moderne Prozessoren besitzen eine ganze Reihe von Rechenwerken mit speziellen Aufgaben. Häufig benutzte Rechenwerke sind mehrfach vorhanden. Beispiele sind:

- Rechenwerke für Festkomma- und Fließkomma-Arithmetik
- Schnelle parallele Addierer und Multiplizierer
- Schiebe- und Rotationseinheiten

- Rechenwerk zur Ausführung von Bitoperationen
- Rechenwerke für graphische und Multimediaoperationen

### 2.1.2.2 Rechenwerksvarianten

- Hilfsregister des Akkumulators wird hinter die ALU verlegt (siehe Bild 2.4). Dadurch sind ALU-Operationen ohne Veränderung des Akkumulators möglich. Diese Variante findet hauptsächlich bei älteren Prozessoren Verwendung, bei denen die Adreßrechnung in der ALU des Rechenwerks durchgeführt wird.

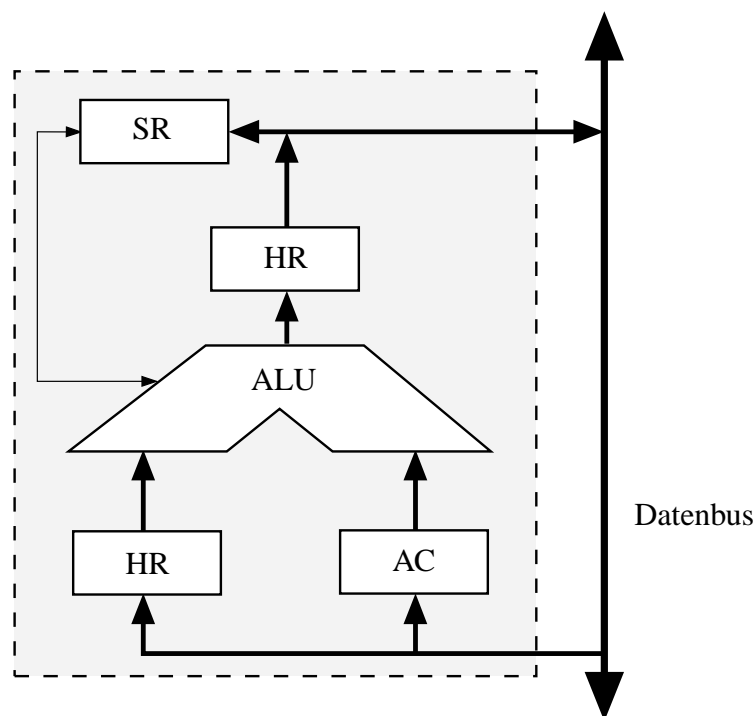


Bild 2.4: Eine andere Rechenwersvariante

- Rechenwerk ohne Akkumulator. Der Akkumulator wird in den Registersatz des Prozessors verlegt (siehe Bild 2.5). Damit können mehrere Register des Registersatzes eine Akkumulatorfunktion übernehmen. Alle modernen 16/32-Bit Prozessoren nutzen dieses Konzept.

### 2.1.2.3 Zeitverhalten des Rechenwerkes

In Bild 2.7 ist das Zeitverhalten des in Bild 2.6 angegebenen Rechenwerks dargestellt.

#### Funktionweise

Während der positiven Takthälfte von T liegen die Operanden auf den Bussen B1 und B3. Dann werden die Operanden mit der negativen Taktflanke in die Hilfsregister übernommen und stehen somit

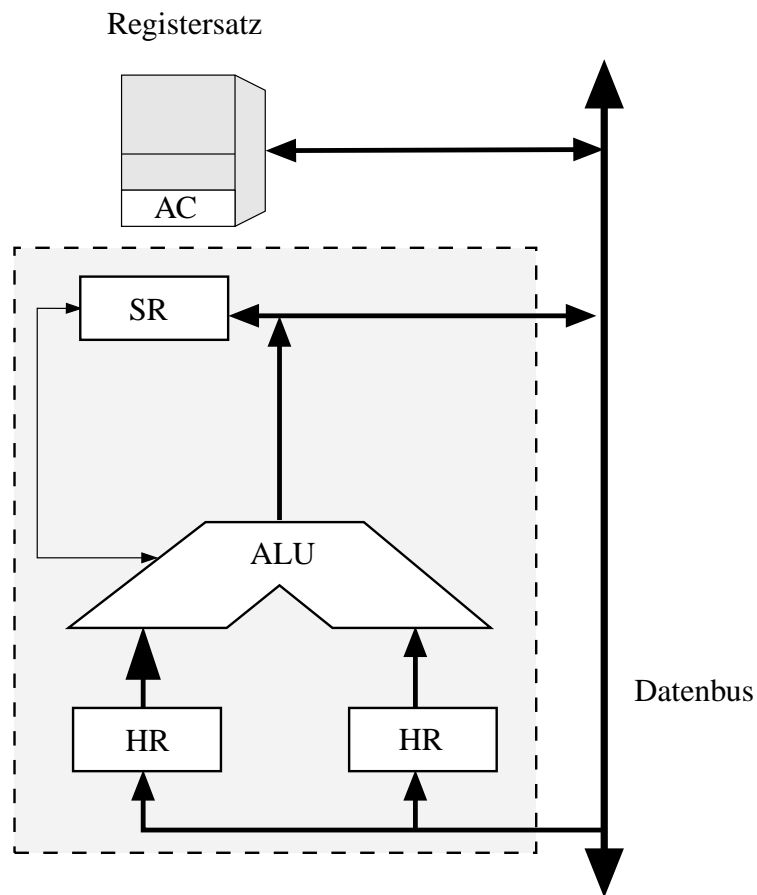


Bild 2.5: Ein Rechenwerk ohne Akkumulator

auf B2 und B4 stabil zur Verfügung. Die ALU berechnet in einer Ausführungszeit  $t_{ALU}$  das Ergebnis. Dieses Ergebnis wird mit der positiven Taktflanke in den Akkumulator übernommen. Ohne Hilfsregister würden sich die während der ALU-Rechenzeit ergebenden Schwankungen am Ausgang (Hasards, Wettläufe) sofort wieder auf den ALU-Eingang auswirken (Ein asynchrones Schaltwerk mit allen bekannten Problemen und Fehl-Ergebnissen würde entstehen).

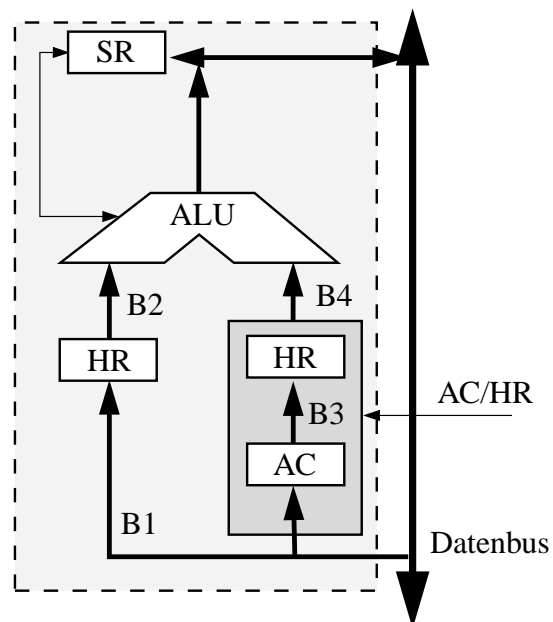


Bild 2.6: Aufbau des Rechenwerkes und Akkumulators

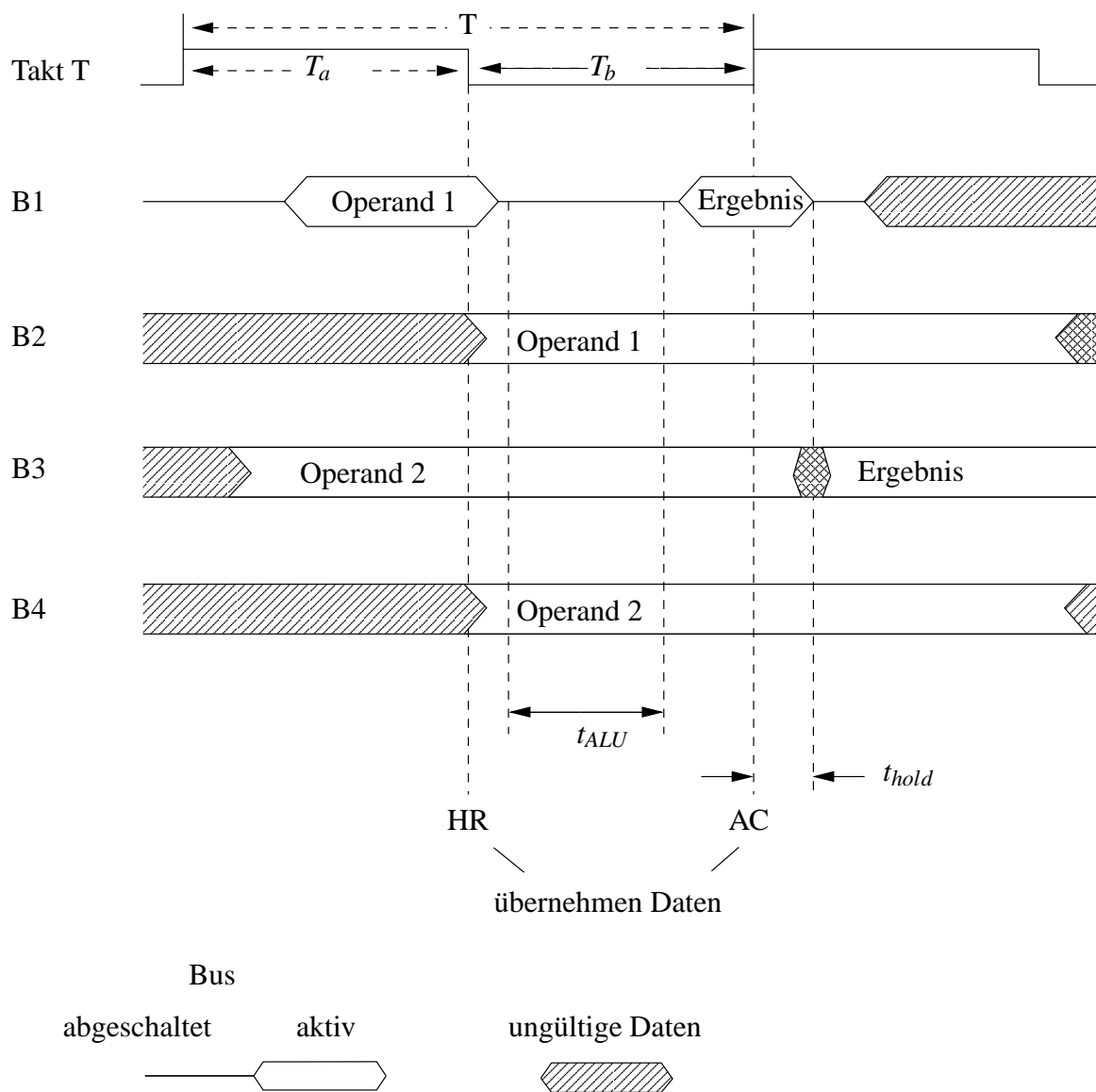


Bild 2.7: Zeitverhalten des Rechenwerkes

### 2.1.3 Registersatz

Erweiterung des Operationswerkes zur Zwischenspeicherung von Variablen und Ergebnissen. Häufig benutzte Operanden können dort zwischengespeichert werden, so dass auf sie schneller zugegriffen werden können als auf den Hauptspeicher. Dies wird dadurch auf der einen Seite erreicht, dass die Register auf dem Chip untergebracht werden. Auf der anderen Seite erfolgt die Auswahl der Register, aufgrund ihrer relativ kleinen Anzahl, über individuelle Steuerleitungen, wodurch die Zeit zur Adressdekodierung entfällt. „Größere“ Registersätze werden mit zusätzlichem Adressdekodierer realisiert.

Der Registersatz wird bei modernen Prozessoren als Multiport-Speicher realisiert, auf den gleichzeitig mehrfach lesend und schreibend zugegriffen werden kann.

### 2.1.4 Adreßwerk

Berechnet nach den Vorschriften des Steuerwerkes die Adresse eines gewünschten Befehls oder Operanden.

- Früher häufig Bestandteil des Rechenwerkes. Später sehr komplex (viele verschiedene komplexe Adressierungsarten) und deshalb eigenständig. Sie führen neben einfacher Adressrechnung auch Aufgabe der Umsetzung virtueller Adresse in physikalische Adressen.
- Digitale Signalprozessoren besitzen drei Adreßwerk, da sie in jedem Taktzyklus einen Befehl in einem Programmspeicher und zwei Operanden in zwei getrennten Daten-Speichern selektieren müssen (vergleiche Folien zur Vorlesung 18: Digitale Signalprozessoren).

### 2.1.5 Systembus-Schnittstelle (*Bus Interface Unit*)

Stellt die Verbindung des Mikroprozessors zu seiner Außenwelt dar. Sie enthält diverse Zwischenspeicher-Register (Puffer) zur kurzfristigen Aufbewahrung von Adressen und Daten, z. B.

- **Befehlszähler (*Program Counter: PC*):** enthält die Adresse des nächsten Befehls. Wird nach jedem Befehl, außer bei Sprüngen und Unterprogrammaufrufen, inkrementiert
- **Adressbuspuffer:** Adresse des gewünschten Operanden im Hauptspeicher.
- **Datenbuspuffer:** Wert des gerade bearbeiteten Operanden.

Die Systembus-Schnittstelle enthält daneben Aus- und Eingangstreiber (*Tristate-Treiber*), die zur „Abkoppelung“ des Prozessors vom Systembus<sup>2</sup> und zur elektrischen Anpassung der Signale an die Systemspezifikationen dienen.

---

<sup>2</sup>Wenn der Systembus einer anderen Komponente, wie z. B. einem DMA-fähigen Baustein, zugeteilt wird

### 2.1.6 Interne Busse

Interne Busse dienen zur Verbindung der Systemkomponenten auf dem Prozessor-Chip untereinander. Über die gleichen Leitungen werden sowohl Operanden als auch Adressen übertagen, deshalb ist hier keine Unterscheidung zwischen einem Daten- und Adreßbus möglich ist, wie das beim externen Bussystem der Fall ist.

Durch die Architektur des internen Bussystems kann der Grad der prozessorinternen Parallelität erhöht werden (siehe Vorlesungsfolien 4). Dies wird durch mehrere Teilbusse (auf dem Chip) erreicht. Dieser Maßnahme steht jedoch entgegen, dass das interne Bussystem einen großen Teil der Chipfläche belegt und deshalb nicht beliebig ausgebaut werden kann.

### 2.1.7 Weitere Komponenten

Bei modernen Mikroprozessoren ist eine Reihe weiterer Funktionseinheiten integriert, die in Bild 2.1 der Übersichtlichkeit wegen nicht eingezeichnet wurden, z. B.

- Speicherverwaltungseinheit (*Memory Management Unit, MMU*).
- Arithmetik-Coprozessor.
- Cache-Speicher (schnelle Zwischenspeicher) für Befehle und Daten.

Diese werden ausführlich in Kapitel ?? behandelt.

## 2.2 Zeitverhalten des Systembusses

Siehe Folien zur Vorlesung Nr. 5

## 2.3 Ablauf der Befehlsabarbeitung

Siehe Folien zur Vorlesung Nr. 5. Weiterhin bieten die auf der TI-Homepage<sup>3</sup> bereitgestellten Animationen zum Thema Adressierungsarten weitere Beispiele an und sind sehr empfehlenswert, das Verständnis.

---

<sup>3</sup><http://i61www.ira.uka.de/users/asfour/TI>

# **Kapitel 3**

## **Pipeline-Verarbeitung**

Siehe die Folien zu den Vorlesungen 6, 7 und 8 :)

### **3.1 Pipeline-Prinzip**

### **3.2 Pipeline-Stufen und Pipeline-Register**

### **3.3 DLX-Pipeline**

#### **3.3.1 Phasen der Befehlsausführung**

### **3.4 Pipelinekonflikte**

#### **3.4.1 Daten- Steuerfluss- und Ressourcenkonflikte**

#### **3.4.2 Software- und Hardware-Lösungen**

### **3.5 RISC & CISC**

Siehe die Folien zur Vorlesung 9



# Kapitel 4

## Organisation des Arbeitsspeichers

### 4.1 Zusammenstellung der Speichertechnologien

Prinzip	Technische Ausführung	Bemerkungen
<i>magnetisch</i>	Plattenspeicher	Speicherung <b>ohne</b> Energie- zufuhr
	Trommelspeicher	
	Flex. Magnetplatte (Diskette)	
	Winchester-Platte	
	Magnetbandspeicher	
	Kernspeicher	
	Magnetdrahtspeicher	
<i>Halbleiter</i>	Bubblespeicher	Speicherung <b>nur bei</b> Energie- zufuhr
	Komplementäre MOSFET (CMOS)	
	MOSFET	
	Elektronenstrahl-MOS	
	Nitrid-MOS	
	Silizium auf Saphir	
	Ladungskopplung (CCD)	
	Integrierte Injektions-Logik (I <sup>2</sup> L)	
<i>Optik</i>	Bipolar Halbleiter	Derzeit ohne Bedeutung
	Amorphe Halbleiter (OVONICs)	
<i>Supraleitung</i>	Optische Platte	Derzeit ohne Bedeutung
	Holographische Speicher	
	Josephson-Effekt	
<i>Elektrostatik</i>	Speicherröhren	Derzeit ohne Bedeutung
<i>Ferroakustik</i>	Magnetostriktive Drähte	

## 4.2 Wichtige Begriffe

Der Hauptspeicher heutiger Mikrorechnersystem bestehen aus Halbleiterbausteinen. In Bild 4.1 ist der prinzipielle Aufbau des Hauptspeichers skizziert.

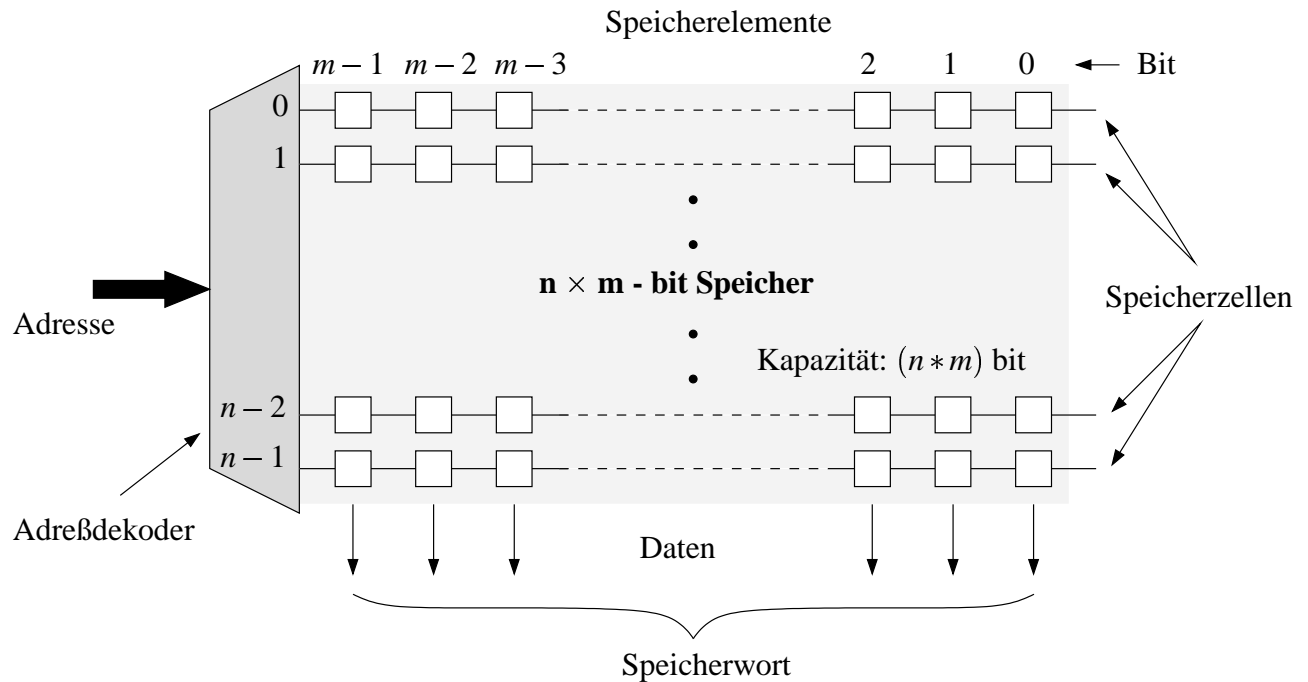


Bild 4.1: Prinzipieller Aufbau des Arbeitsspeichers

**Speicherelement:** 1 Bit Speicher.

**Speicherzelle:** feste Anzahl von Speicherelementen, die durch eine einzige Adresse ausgewählt werden, z. B. 8, 16, 32, 64 bit.

**Speicherwort:** maximale Anzahl von Speicherelementen, die in einem Buszyklus zwischen den Mikroprozessor und dem Speicher übertragen werden können. (Speicherwortbreite).

**wahlfreier Zugriff:** jede Speicherzelle kann direkt angesprochen werden (ohne vorher andere Zellen ansprechen zu müssen). Die Selektion der Speicherzelle erfolgt über einen Adressdekoder. Die Adresse wird in einem 1-aus- $n$  Code umgeformt.

**Organisation:** ein Speicherbaustein wird definiert durch die Anzahl  $n$  seiner Speicherzellen und die Anzahl der Speicherelemente pro Zelle  $m$ . Die Organisation wird in der Form:  $n \times m$ -bit angegeben.

**Beispiel:**

Ein  $4K \times 8$  -bit Speicher enthält 4096 Speicherzellen je 8 bit. Der Speicher kann aus acht  $4K \times 1$  -bit Bausteinen oder zwei  $4K \times 4$  -bit oder ... aufgebaut sein.

Heute sind  $1M \times 1$  -bit gängige Organisationen von großen dynamischen Speicherbausteinen.

**Kapazität:** Die Informationsmenge (in Bits), die im Speicher untergebracht werden können  
 $n \bullet m$  -bit

### 4.2.1 Größen zur Charakterisierung der Arbeitsgeschwindigkeit eines Speicherbausteins

In Bild 4.2 sind zwei charakteristische Größen eines Speicherbausteins skizziert.

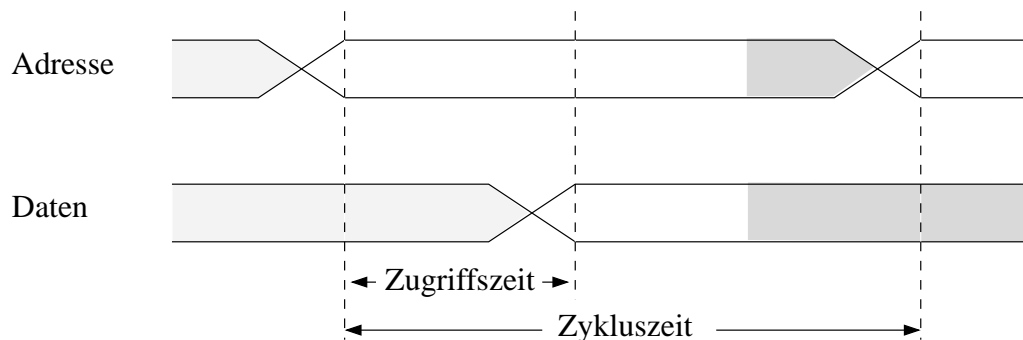


Bild 4.2: Zugriffszeit und Zykluszeit

**Zugriffszeit** (*access time*):

Die maximale Zeitdauer, die vom Anlegen einer Adresse an den Speicher bis zur Ausgabe der gewünschten Daten vergeht.

**Zykluszeit** (*cycle time*):

Die minimale Zeitdauer, die zwischen zwei aufeinander folgenden Aufschaltungen von Adressen auf den Speicher vergehen muss.

Die Zykluszeit kann erheblich länger als die Zugriffszeit sein! Dies kann folgende Gründe haben:

- Die Speicherzelle muss sich nach einem Zugriff „erholen“.
- Bei einigen Speicherarten wird die Information durch das Auslesen zerstört und muss erst wieder eingeschrieben werden (*refresh*)

Im Idealfall gilt:  $\text{Zykluszeit} = \text{Zugriffszeit}$ . In der Realität gilt meist:  $\text{Zykluszeit} > \text{Zugriffszeit}$  (bis zu 80%).

## 4.3 Klassifizierung von Halbleiterspeichern

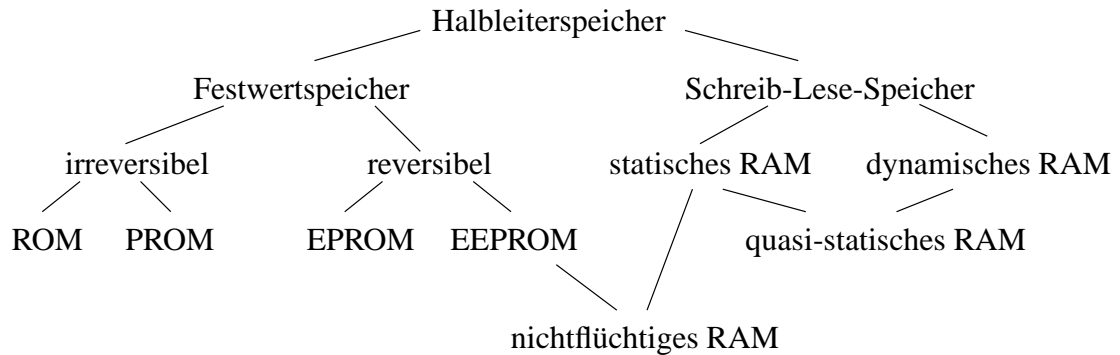


Bild 4.3: Wichtigste Typen von Halbleiterspeichern

### 4.3.1 Festwertspeicher (ROM, *Read Only Memory*)

Der Inhalt des Speichers ist während des Normalbetriebs nur lesbar. Er geht bei Abschaltung der Versorgungsspannung nicht verloren. Man spricht dann von einem „nicht flüchtigen Speicher“ (*non volatile*).

#### Irreversible Festwertspeicher:

Das Schreiben einer Information kann nicht rückgängig gemacht werden. Hier unterscheidet man zwischen

- **Maskenprogrammierten Festwertspeichern (ROM)**, deren Programmierung bei der Herstellung erfolgt. Wegen der hohen Programmierkosten sind Speicher als maskenprogrammierte Festwertspeicher erst in sehr großen Stückzahlen wirtschaftlich einsetzbar.
- **Programmierbaren Festwertspeichern (PROM)**, *programmable read only memory*, deren Programmierung durch den Anwender mit einem Programmiergerät erfolgen kann. Bei dem Programmiervorgang werden z.B. physikalische Verbindungen (*fusable links*) in den Speicherelementen durchgebrannt.

#### Reversible Festwertspeicher:

Das Schreiben einer Information kann wieder rückgängig gemacht werden. Die eingeschriebene Information kann zwar verändert werden, aber nicht während des normalen Betriebs

- **UV-löschbare Festwertspeicher (EPROM)**, *erasable and programmable read only memory*. Das Löschen des Speicherinhaltes erfolgt mit UV-Licht und die neue Programmierung über spezielle Geräte.
- **Elektrisch löschbare Festwertspeicher (EEPROM)**, *electrically erasable and programmable read only memory*. Das Löschen und das erneute Programmieren des Speichers wird elektrisch durch den Mikroprozessor selbst veranlaßt. Dieser Vorgang ist jedoch sehr langsam und kann nur begrenzt oft wiederholt werden.

### 4.3.2 Schreib/Lese-Speicher (RAM, *Random Access Memory*)

Der Inhalt des Speichers ist jederzeit lesbar und schreibbar. Er geht bei Abschaltung der Versorgungsspannung verloren. Man spricht dann von einem flüchtigen Speicher *volatile memory*).

**Statische Schreib/Lese-Speicher (SRAM)** speichern die Information in Zellen, die aus Flipflops bestehen. Der Inhalt des Speichers ist stabil, solange die Versorgungsspannung anliegt.

**dynamische Schreib/Lese-Speicher (DRAM)** speichern die Information als elektrische Ladung in einem Kondensator. Das Lesen einer Speicherzelle bedingt das Entladen des Kondensators (*destructive read*), so dass der gelesene Wert nach dem Lesen erneut eingeschrieben werden muss. Außerdem geht diese Ladung nach einiger Zeit auch durch Leckströme verloren, so dass sie in regelmäßigen Abständen aufgefrischt werden muss (*refresh*). Hierzu ist eine Steuerlogik erforderlich, die das Auffrischen der Zellen steuert.

**Quasi-statische Schreib/Lese-Speicher (iRAM, *integrated RAM*)** sind dynamische Speicher, bei denen die Steuerlogik für das Wiederauffrischen mit auf dem Chip integriert ist. Diese Logik sorgt für das regelmäßige Auffrischen der Speicherzellen. Dadurch wirkt der Speicher nach außen wie ein statischer RAM-Baustein.

### 4.3.3 Nicht-flüchtige RAM's (NVRAM, *non volatile RAM*)

Nicht-flüchtige RAM's stellen eine Kombination aus Festwertspeicher und Schreib/Lese-Speicher dar. Jede Speicherzelle ist doppelt ausgelegt, einmal als statische RAM-Zelle und einmal als EEPROM-Zelle. Im normalen Betrieb wird der Baustein wie ein RAM benutzt. Eine spezielle Steuerschaltung erlaubt es jedoch, den gesamten RAM-Inhalt in das EEPROM zu kopieren oder von dort zu laden.

## 4.4 Aufbau von Schreib-/Lese-Speicherzellen

### 4.4.1 Statische MOS-Speicherzellen

Eine statische CMOS-Speicherzelle (Bild 4.4) besteht aus zwei kreuzweise rückgekoppelten CMOS-Invertern ( $T_0, T_2$ ) und ( $T_1, T_3$ ), die ein Flipflop bilden. Die Transistoren  $T_4$  und  $T_5$  dienen zur Ankopplung der Zelle an die beiden Bitleitungen  $B_0$  und  $B_1$ . Diese Zelle wird auch 6-Transistorzelle genannt. CMOS-Zellen haben eine geringe Verlustleistung, weil nur zum Umschaltzeitpunkt ein nennenswerter Strom fließt; der Schaltungsaufwand ist jedoch relativ hoch.

### 4.4.2 Dynamische MOS-Speicherzellen

Dynamische MOS-Speicherzellen (Bild 4.5) besitzen von allen Zellen für Schreib-/Lese-Speicher den geringsten Aufwand und benötigen daher die geringste Halbleiterfläche. Deshalb werden sie heute

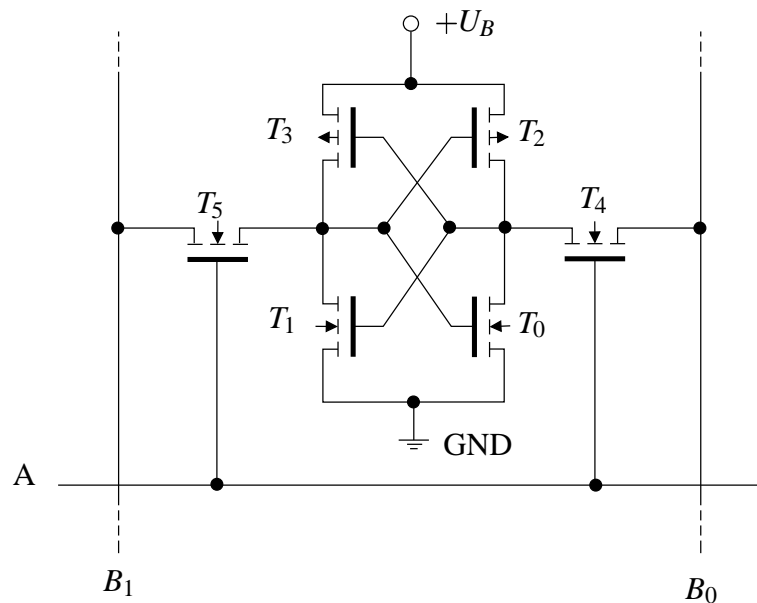


Bild 4.4: Statische CMOS-Zelle

in allen hochintegrierten Speicherbausteinen eingesetzt. Die Information wird als elektrische Ladung in einem Kondensator gespeichert. Dieser Kondensator wird durch eine vergrößerte Drain-Zone gebildet.

### Lesen:

Die trotz allem noch recht kleine Speicherkapazität führt zu dem Problem, dass die Speicherkapazität ungefähr die gleiche Größe hat wie die parasitäre Kapazität der Bitleitung (Leitungskapazität). Dadurch wird es schwierig, den Zustand der Zelle direkt durch Leseverstärker auszuwerten. Deshalb wird die Leitungskapazität zu Beginn des Lesevorgangs vorgeladen (*precharge*), indem die Bitleitung *B* über den Lasttransistor  $T_L$  kurzzeitig mit  $+U_B$  verbunden wird. Dieser Vorgang wird über die Spannung  $U_p$  gesteuert. Zum Lesen der Zelle wird dann über *A* eine positive Spannung an das Gate des Speichertransistors angelegt. Ist die Speicherkapazität geladen, so findet ein Ausgleich mit den Ladungsträgern der Bitleitung statt. Ein Leseverstärker am Ende der Bitleitung stellt den Zustand der Zelle an dem Ausgleichsstrom fest.

### Schreiben:

Durch Anlegen einer positiven Spannung  $U_{GS}$  wird der Speichertransistor leitend. Liegt die Bitleitung *B* auf Masse, so werden Elektronen in die Drain-Zone gebracht. Dadurch wird der Speicherkondensator aufgeladen. Liegt jedoch die Bitleitung *B* auf  $U_B$ , so wird der Speicherkondensator nicht geladen.

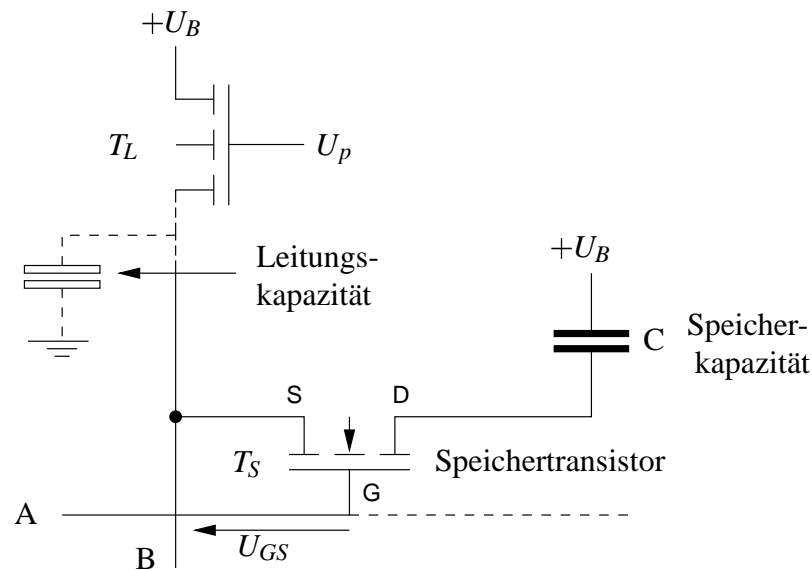


Bild 4.5: Dynamische MOS-Speicherzelle

## 4.5 Organisation von Speicherbausteinen

Speicherzellen werden in einer matrixförmigen Anordnung zu einer **Speichermatrix** zusammengefaßt, so dass jede Zelle im Schnittpunkt einer Matrix-Zeile und einer Matrix-Spalte liegt. Bild 4.6 zeigt den prinzipiellen Aufbau eines Speicherbausteins.

Ein Speicherelement kann über die Zeilen-Auswahlleitungen  $A_0, \dots, A_i$  und die Spalten-Auswahlleitungen  $A_{i+1}, \dots, A_n$  adressiert werden. Die Auswahl einer Zeile bzw. einer Spalte erfolgt über Decoder. Die Spalten-Auswahlleitungen werden am Rand der Matrix mit Verstärkern abgeschlossen.

### 4.5.1 Gewinnung der Zeilen- und Spalten-Adressen aus der anliegenden Adresse

Die niedrigstwertigen Adressbits ( $A_0, \dots, A_i$ ) adressieren über Interface-Treiber und einen Decoder die Zeilen-Auswahlleitungen. Hierdurch wird eine Zeile angesprochen. Die Auswahl einer bestimmten Spalte erledigt ein Spaltenauswahl-Schalter, der über Treiber und Decoder von den höchstwertigen Adressbits ( $A_{i+1}, \dots, A_n$ ) gesteuert wird.

### 4.5.2 Steuerlogik und Bausteinauswahl

Diese Einheit übernimmt die Steuerung aller Komponenten im Baustein. Sie besitzt eine Schnittstelle zum Steuerbus des Mikroprozessors. Einige Schnittstellen-Signale sind in Bild 4.6 eingezeichnet.

- $\overline{\text{CS}}$  (*chip select*): Durch dieses Signal wird der Baustein aus der Menge aller System-Bausteine ausgewählt.

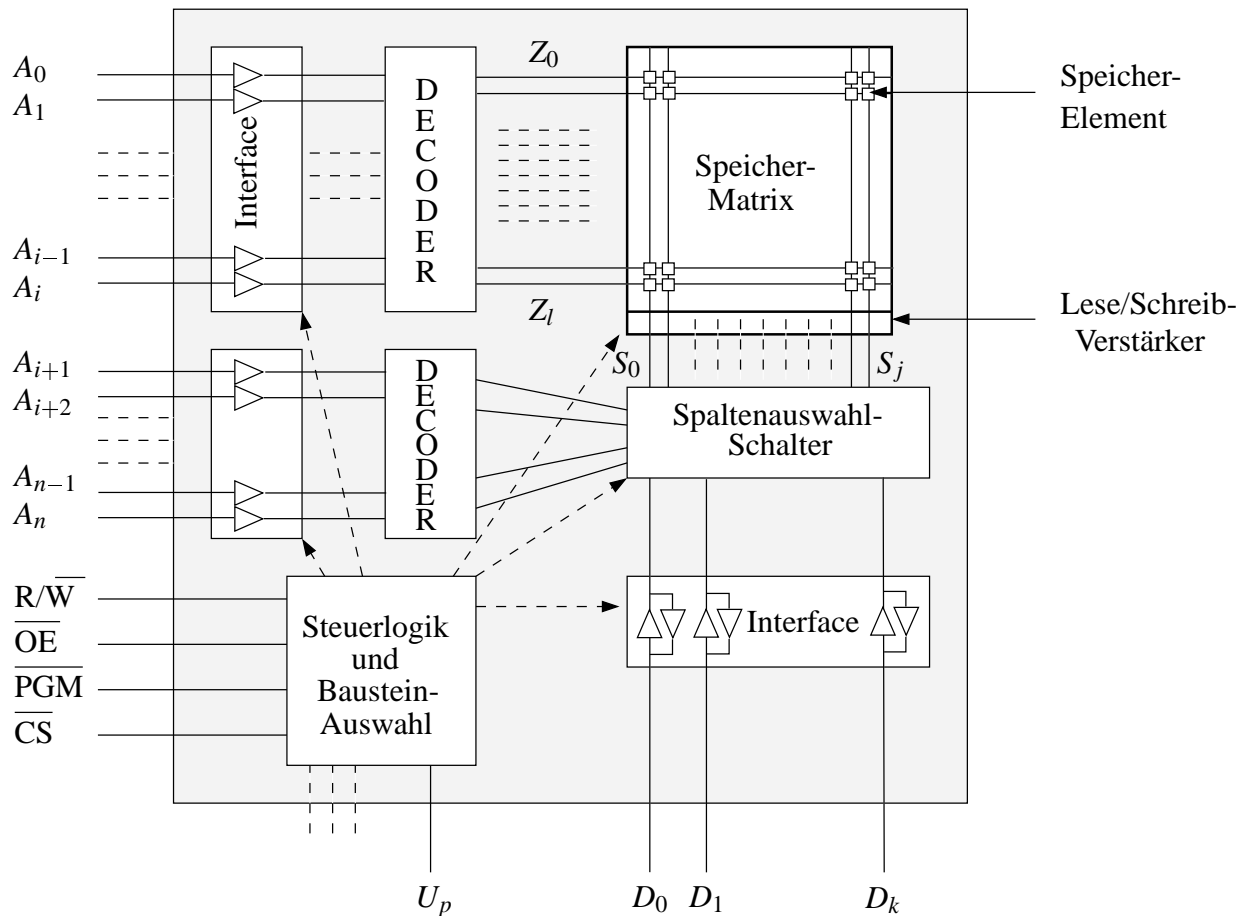


Bild 4.6: Prinzipieller Aufbau eines Speicherbausteins

- $\overline{R/W}$  (read/write): Steuert die Richtung der Treiber in der Datenbus-Schnittstelle und die Lese-/Schreib-Verstärker. Dieses Signal wird nur bei Speichertypen gebraucht, deren Inhalt verändert werden kann (RAM, EEPROM).
- $\overline{OE}$  (output enable): aktiviert die Ausgangstreiber in der Datenbus-Schnittstelle.
- $\overline{PGM}$  (program): muss aktiviert werden, wenn der Baustein neu programmiert werden soll (im Falle von EPROM, EEPROM, NVRAM).

### 4.5.3 Möglichkeiten zur Auswahl eines Speicherelements

Bild 4.7 zeigt die verschiedenen Möglichkeiten zur Auswahl eines Speicherelements.

- a) Zeilen-Auswahlleitung/Bitleitung ( $Z_i/B_k$ ).
- b) wie a), es sind jedoch zwei Bitleitungen vorhanden, was den Betrieb vereinfacht.

- c) Koinzidente Ansteuerung: Alle Zellen einer Zeile  $i$  sind mit einer Leitung  $Z_i$ , alle Zellen einer Spalte  $j$  durch eine Leitung  $S_j$  verbunden. Die Bitleitungen  $B_0$  und  $B_1$  müssen dazu parallel an alle Speicherzellen geführt werden.

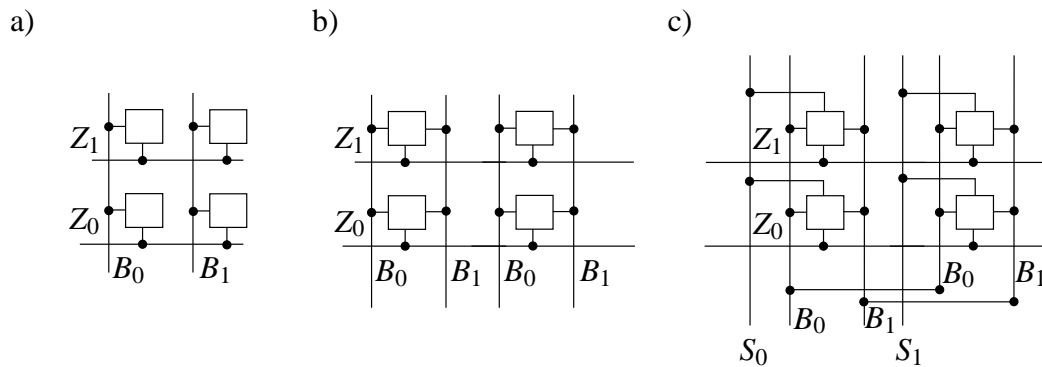


Bild 4.7: Auswahl eines Speicherelements

#### 4.5.4 Dynamische RAM-Bausteine

Dynamische RAM-Bausteine (DRAM) weisen die größte Integrationsdichte aller Halbleiterspeicher auf. Dies ist auf den einfachen Aufbau der Zellen zurückzuführen. Um die Anzahl der Baustein-Anschlüsse trotz der großen Zahl von Speicherzellen möglichst klein zu halten, werden DRAM-Bausteine häufig bitweise organisiert, d. h. sie haben eine Organisation der Form  $n \times 1$ -bit. Bitweise organisierte DRAMs verfügen meist über einen Dateneingang  $D_{in}$  und einen davon getrennten Datenausgang  $D_{aus}$  (siehe Bild 4.8). Ein Mikrorechnersystem mit einer Datenbus-Breite von  $N$  bit benötigt eine „Bank“ von mindestens  $N$  DRAM-Bausteinen.

Um die Anzahl der Chip-Anschlüsse zu reduzieren, können einerseits die Zeilen- und Spalten-Adressen dem Baustein im Multiplexverfahren zugeführt werden. Diese Vorgehensweise spart zwar Anschlüsse auf dem Chip; sie benötigt jedoch eine zusätzliche Steuerlogik zur Adressbildung. Die Steuersignale  $\overline{RAS}$  (row address strobe) und  $\overline{CAS}$  (column address strobe) steuern die Auswahl von Spalten- bzw. Zeilenadressen.

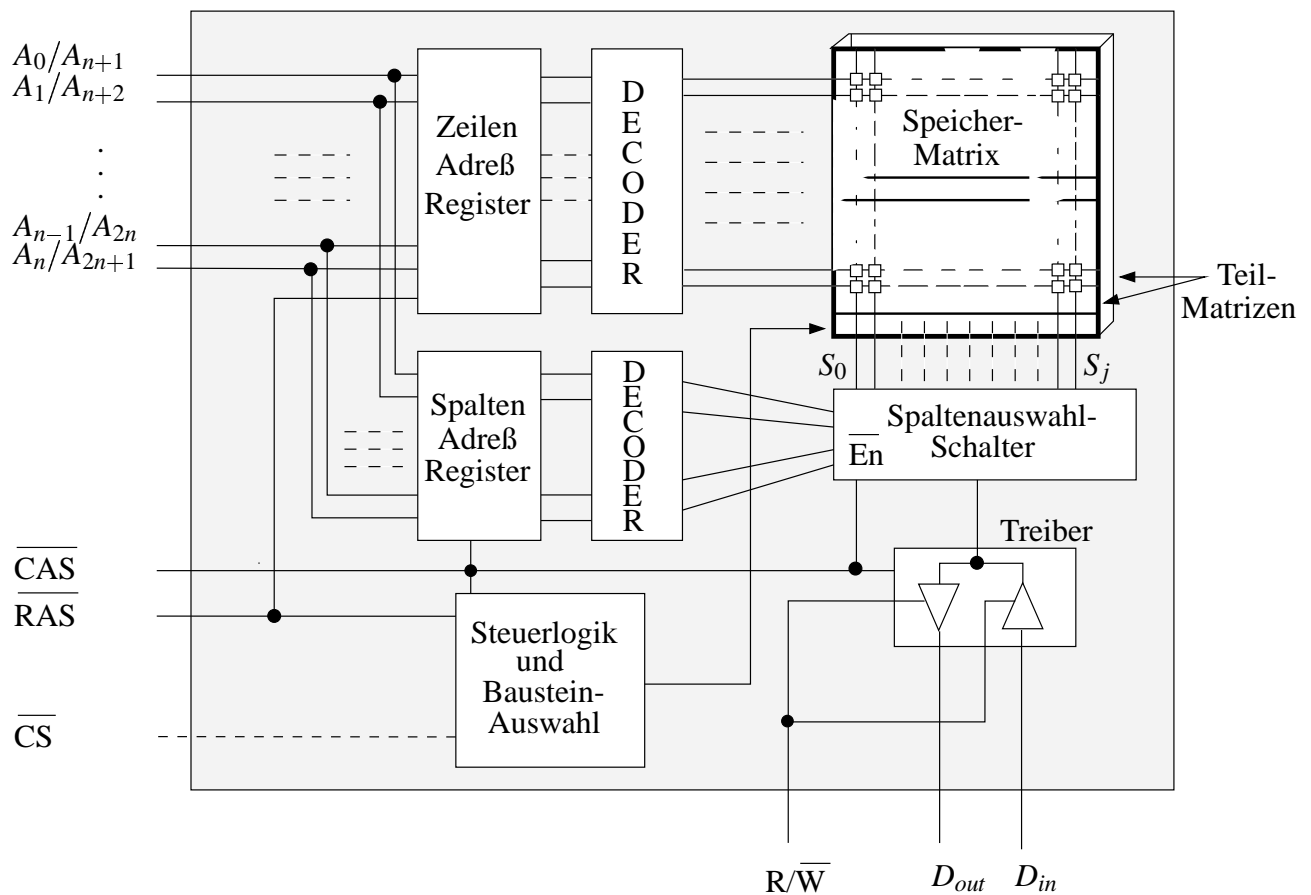


Bild 4.8: Dynamischer RAM-Baustein

#### 4.5.4.1 Adressierung eines dynamischen RAM-Bausteins

Die Steuersignale  $\overline{\text{RAS}}$  und  $\overline{\text{CAS}}$  sind zunächst inaktiv (*high*). Mit der negativen  $\overline{\text{RAS}}$ -Flanke wird die Zeilenadresse in das Zeilen-Adressregister übernommen. Die Spaltenadresse wird dann mit der negativen  $\overline{\text{CAS}}$ -Flanke in das Spalten-Adressregister übernommen (siehe Bild 4.9).

**Lesen:** Beim Lesen erscheinen eine gewisse Zeit nach der negativen  $\overline{\text{CAS}}$ -Flanke die Daten am Ausgang  $D_{out}$ . Der Zugriffszyklus wird durch das Setzen der  $\overline{\text{CAS}}$ -Signal beendet ( $\overline{\text{CAS}} = 1$ ).

**Schreiben:** Beim Schreiben muss das zu schreibende Datum am Eingang  $D_{in}$  gleichzeitig mit der Spaltenadresse angelegt werden. Die Datenübernahme geschieht mit der negativen  $\overline{\text{CAS}}$ -Flanke.

Die Zugriffszeit  $t_{Zugriff}$  ergibt sich als die Zeit von der negativen Flanke von  $\overline{\text{RAS}}$  bis zu dem Zeitpunkt, zu dem das Datum verfügbar ist. Die Zykluszeit  $t_{Zyklus}$  umfaßt zusätzlich die Zeit, die für das notwendige interne Zurückschreiben des Datums in die Speicherzelle gebraucht wird. Sie gibt die Zeit an, die zwischen zwei aufeinanderfolgenden Zugriffen vergehen muss.

Eine detaillierte Beschreibung der Parameter  $t_{\text{RAC}}$ ,  $t_{\text{RC}}$ ,  $t_{\text{CAC}}$ ,  $t_{\text{RCD}}$  und  $t_{\text{RP}}$  eines DRAM-Bausteins ist den Folien zur Vorlesung 11 zu entnehmen.

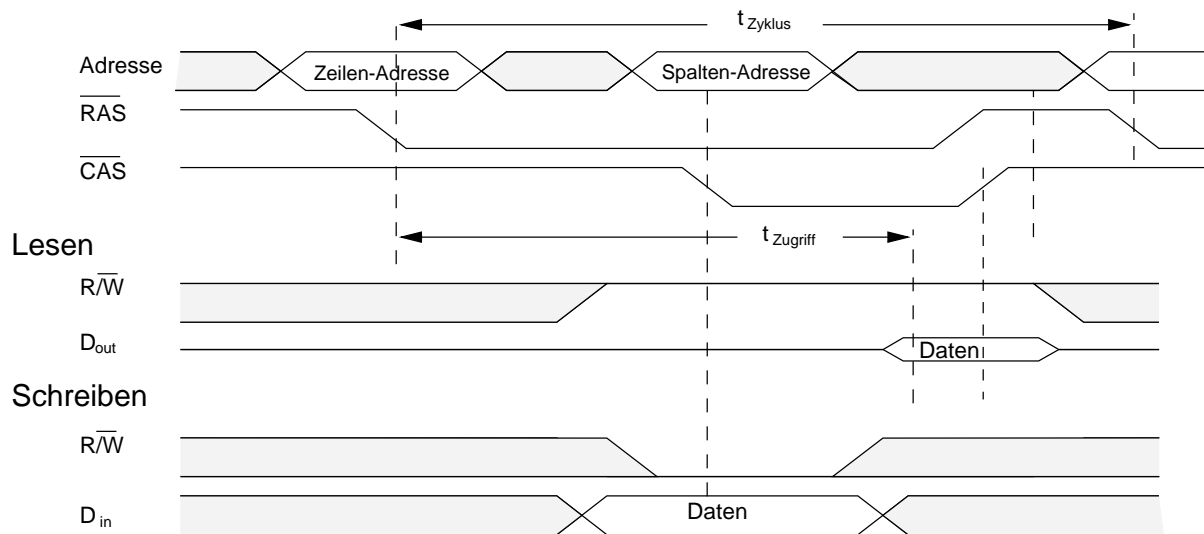


Bild 4.9: Adressierung eines dynamischen RAM-Bausteins

#### 4.5.5 Seitenzugriff bei DRAMs (*Fast-Page-Mode-DRAM*)

Diese Adressierungsart der DRAM-Bausteine beschleunigt wesentlich die Lese- und Schreibzugriffe auf den Baustein. Nach dem einmaligen Anlegen einer Zeilenadresse können alle Speicherzellen dieser Zeile (*Seite*) durch eine Folge von Spaltenadressen adressiert werden (siehe Bild ??). Dazu muss das  $\overline{\text{RAS}}$ -Signal aktiv bleiben und mit jeder abfallenden Flanke von  $\overline{\text{CAS}}$  wird eine neue Spaltenadresse übernommen, so dass das korrespondierende Bit adressiert wird. Die Zugriffszeit ist häufig um die Hälfte kürzer als bei einem vollen RAS-CAS-Zyklus, weil nur die kurze Spalten-Zugriffszeit benötigt wird. Genaugenommen stellen derartige DRAM-Bausteine keinen echten Speicher mit wahlfreiem Zugriff (*random access memory*) dar, weil die Zugriffe auf eine adressierte Zeile schneller ist als auf andere Speicherzellen erfolgen.

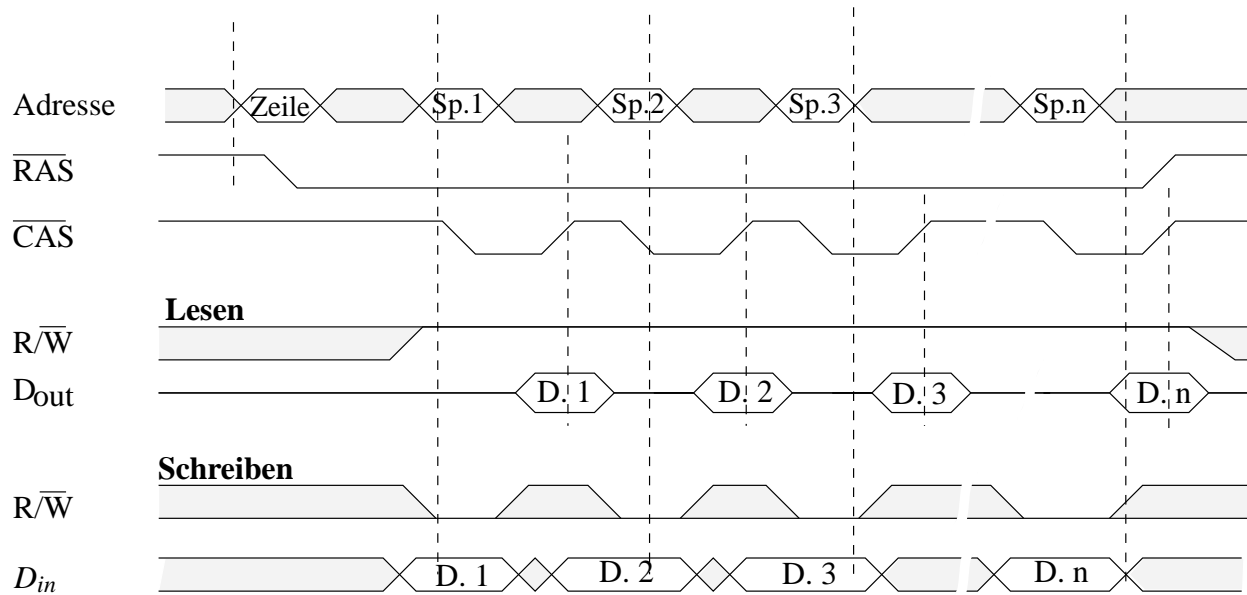


Bild 4.10: Seitenzugriff in einem DRAM (Page-Mode-DRAM)

#### 4.5.6 EDO-RAM-Bausteine (*extended data output*)

EDO-RAMs stellen eine neue Entwicklung dar, die Page-Mode-DRAMs in vielen neuen Mikroprozessoren ersetzen. Bei EDO-RAM-Bausteinen stehen die Daten am Ausgang länger zur Verfügung. Mit Hilfe eines zusätzlichen Steuersignals werden die Daten am Ausgang gehalten, auch nachdem das  $\overline{\text{CAS}}$ -Signal inaktiv wird. Diese Möglichkeit kann in Pipeline-Systemen vorteilhaft sein, wenn z. B. eine Bearbeitungseinheit mit der Durchführung ihres Auftrages beginnt, bevor die Daten der letzten Bearbeitungseinheit vom Bus gelesen werden können.

#### 4.5.7 Synchrones DRAM (*Synchronous DRAM (SDRAM)*)

SDRAMs speichern die Adressangaben und Ein-/Ausgabedaten in Registern zwischen und synchronisieren den Zugriff durch den Bustakt. Sie bestehen aus mehreren Speicherbänken. Sie arbeiten mit Takten von 66-133 MHz. Alle Ein-/Ausgangssignale sind synchron zum Takt.

#### 4.5.8 Synchrones DRAM (*Synchronous DRAM (SDRAM)*)

SDRAMs speichern die Adressangaben und Ein-/Ausgabedaten in Registern zwischen und synchronisieren den Zugriff durch den Bustakt. Sie bestehen aus mehreren Speicherbänken. Sie arbeiten mit Takten von 66-133 MHz.

### 4.5.9 Double Data Rate RAMs (DDRAM)

DDRAMs stellen die nächste Stufe der SDRAM-Entwicklung dar. Eine Erhöhung der Bandbreite wird durch die Nutzung beider Taktflanken des Bustaktes erreicht. Daten werden bei steigender und fallender Taktflanke übertragen. Die Zugriffsrates wird somit gegenüber SDRAMs verdoppelt.

## 4.6 Auffrischen dynamischer RAM-Bausteine

Das Auffrischen dynamischer RAMs geschieht zeilenweise, indem jede Zeile gelesen und erneut geschrieben wird, um die ständigen Ladungsverluste durch Leckströme der Speicherkapazität zu kompensieren. Je nach Baustein-Typ muss jede Zeile alle 2-4 ms aufgefrischt werden. Das Auffrischen einer Zeile erfolgt dadurch, dass die Zeilenadresse an den Baustein angelegt und  $\overline{\text{RAS}}$  aktiviert wird. Während der Auffrischzeit einer Zeile muss  $\overline{\text{CAS}}$  inaktiv bleiben (siehe Bild 4.11). Der Auffrischvorgang kann von der CPU oder durch einen DRAM-Controller gesteuert werden.

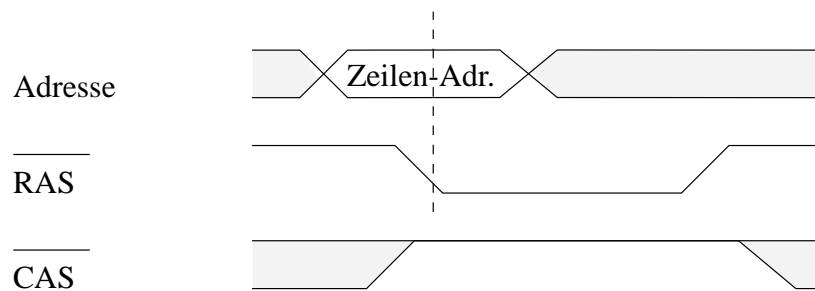


Bild 4.11: Auffrischzyklus bei dynamischen RAMs

Die Spaltenadresse steuert die Spalten-Auswahl-Schalter. Die Zeilenadresse wird über den Multiplexer dem Zeilendecoder zugeführt. Die Takt- und Zeitsteuerung erzeugt einen freilaufenden internen Takt und kontrolliert den Zähler. Zur Synchronisation der Auffrisch-Vorgänge und der externen Prozessor-Zugriffe benutzt die Steuerlogik den RDY-Ausgang.

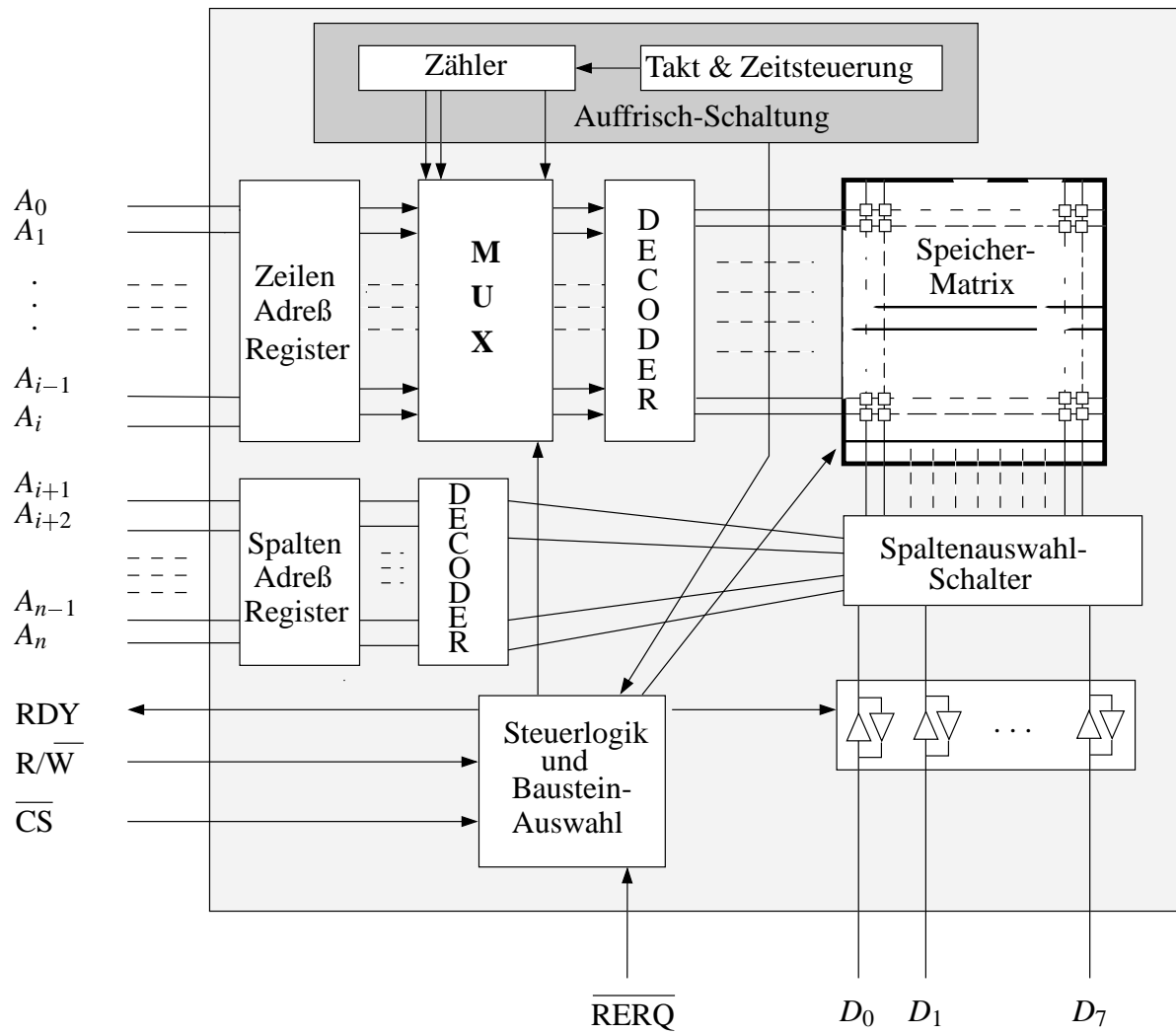


Bild 4.12: Pseudo-statischer RAM-Baustein

## 4.7 Speicherhierarchie

Zwischen der Verarbeitungsgeschwindigkeit von Prozessoren und der Zugriffsgeschwindigkeit von DRAM-Speicherchips des Hauptspeichers klafft eine Lücke, die ständig wächst. Ein technologisch einheitlicher Speicher mit **kurzer Zugriffszeit** und **großer Kapazität** ist aus Kostengründen i.a. nicht realisierbar.

### Lösung:

Schichtenweise Anordnung verschiedener Speicher und Verschiebung der Information zwischen den Schichten (Speicherhierarchie).

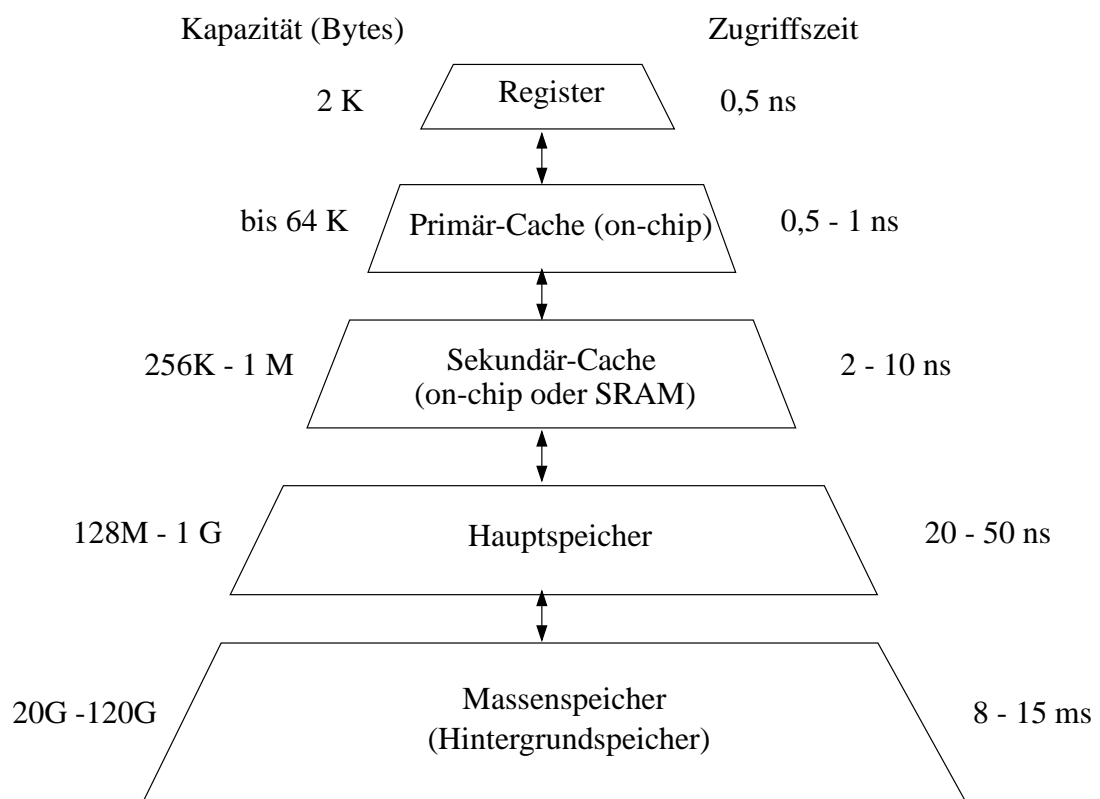


Bild 4.13: Die Speicherhierarchie eines Rechners

**Wirkung** wie *ein* großer und schneller Speicher, **wenn**

- Lokalitätsverhalten der Programmverarbeitung,
- Umlagerung der Information rechtzeitig (*Umlagerungsstrategien*),
- Inhomogenität des Speichersystems für Benutzer nicht sichtbar ist (*Virtueller Speicher*)

**Leistungsfähigkeit** der Hierarchie bestimmt durch:

- Eigenschaften der Speichertechnologien (Zugriffsart, Zugriffszeiten, ...)

- Adressierung der Speicherplätze
- Organisation des Betriebs

Die Einführung einer Speicherhierarchie in einem Mikroprozessorsystem soll die unterschiedlichen Zugriffszeiten der CPU und des Hauptspeichers ausgleichen. Dazu wird auf der einen Seite ein Pufferspeicher (*Cache*) mit kurzen Zugriffszeiten zwischen den schnellen Registerspeicher des Prozessors und den langsameren Hauptspeicher geschaltet. Auf der anderen Seite wird die Speicherkapazität des Hauptspeichers durch Einbeziehung von Hintergrundspeichern erweitert (*virtueller Hauptspeicher*). Dabei befinden sich jeweils nur die aktuell benötigten Daten im physikalisch vorhandenen Hauptspeicher, während die übrigen Informationen auf einer Festplatte ausgelagert sind.

## 4.8 Cache-Speicher

Ein Cachespeicher (*Pufferspeicher*) ist ein kleiner schneller Speicher, in dem Kopien der Hauptspeichereinhalte bereitgehalten werden, auf die aller Wahrscheinlichkeit nach vom Prozessor als nächstes zugegriffen wird. Man spricht dann von einem **CPU-Cachespeicher**. Der Prozessor soll auf den Cachespeicher fast so schnell wie auf seine Register zugreifen können. Deshalb wird der Cachespeicher direkt auf dem Chip angelegt oder in schnellen SRAM-Bausteinen realisiert.

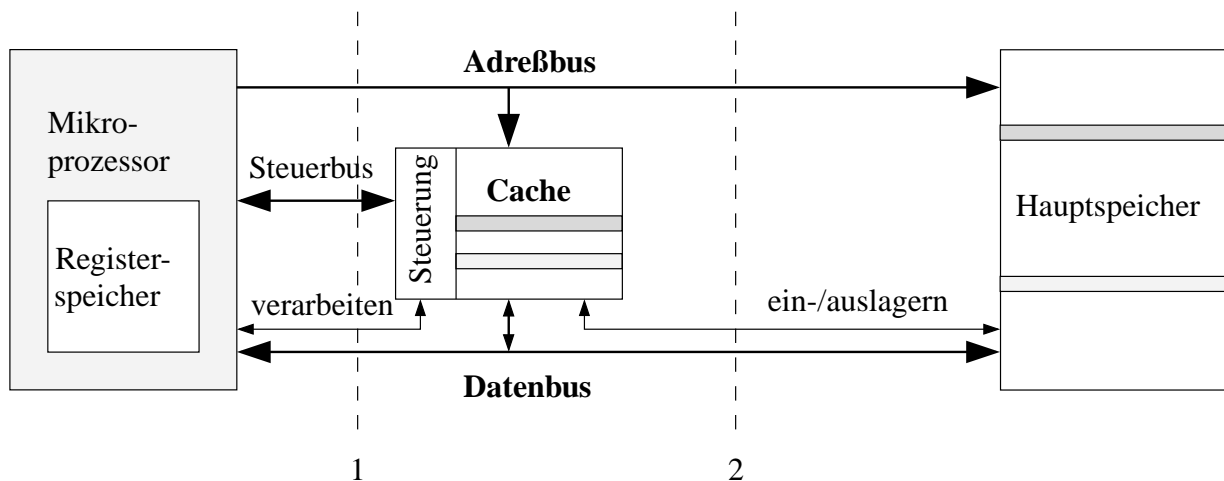


Bild 4.14: Cachespeicher zwischen dem Registerspeicher des Prozessors und dem Hauptspeicher  
Schnittstelle 1: Off-chip-cache, Schnittstelle 2: On-chip-cache

Die **Cacheverwaltung** (Steuerung) sorgt dafür, dass sich die am häufigsten benötigten Daten im Cachespeicher befinden. Die Cachespeicherverwaltung muss sehr schnell sein und ist deshalb meist vollständig in Hardware realisiert (**Cache-Controller**). Sie kopiert alle Daten in den Cache, auf die der Prozessor zugreift. Im Laufe der Zeit füllt sich der Cache mit den aktuellen Daten, so dass erneute Zugriffe wesentlich schneller ausgeführt werden, sofern die benötigten Daten bereits im Cachespeicher stehen.

### 4.8.1 Lesezugriffe

Vor jedem Lesezugriff prüft der Prozessor, ob das Datum im Cache steht. Man spricht von einem **Cache-Hit** (Treffer), falls das angeforderte Datum im Cachespeicher vorhanden ist. Das Datum kann dann ohne Verzögerung (ohne Wartezyklen) aus dem Cache entnommen werden. Von einem **Cache-Miss** (kein Treffer) spricht man, falls das angeforderte Datum nur im Hauptspeicher steht. Das Datum wird dann aus dem Hauptspeicher in den Cache geladen.

Die **Hit-Rate** bezeichnet die Trefferquote im Cache und ist definiert als

$$\text{Hit-Rate} = \frac{\text{Anzahl der Treffer}}{\text{Anzahl der Zugriffe}}$$

Bei einer hohen Trefferrate (meist über 95%) erniedrigt sich die mittlere Zugriffszeit deutlich. Dadurch beschleunigt sich sowohl der Zugriff auf den Programmcode (Programmschleifen werden möglichst vollständig im Cache geladen) als auch der wiederholte Zugriff auf die aktuellen Daten. Die **mittlere Zugriffszeit** berechnet sich näherungsweise wie folgt:

$$t_{\text{access}} = (\text{Hit-Rate}) * t_{\text{Hit}} + (1 - \text{Hit-Rate}) * t_{\text{Miss}}$$

$t_{\text{Hit}}$ : Die Zugriffszeit des Caches

$t_{\text{Miss}}$ : Die Zugriffszeit ohne den Cache

### 4.8.2 Schreibzugriffe

Liegt beim Schreiben ein *Cache-Miss* vor, so wird das Datum sowohl in den Hauptspeicher als auch in den Cache geschrieben. Liegt jedoch ein *Cache-Hit* vor, d.h. ein im Cache stehendes Datum wird durch den Prozessor verändert, so existieren verschiedene Organisationsformen:

#### **Rückschreib-Verfahren** (*write back policy*):

Ein Datum wird von der CPU nur in den Cachespeicher geschrieben und durch ein spezielles Bit (*dirty bit*) gekennzeichnet. Der Hauptspeicher wird nur dann verändert, wenn ein so gekennzeichnetes Datum aus dem Cache verdrängt wird. Diese Methode hat den Vorteil, dass auch Schreibzugriffe mit der schnellen Cache-Zykluszeit abgewickelt werden können. Als Nachteil ergeben sich Konsistenzprobleme zwischen dem Cache- und Hauptspeicher.

#### **Durchschreib-Verfahren** (*write through policy*):

Ein Datum wird von der CPU immer gleichzeitig in den Cache- und den Hauptspeicher geschrieben. Die Schreibzugriffe benötigen also immer die lange Zykluszeit des Hauptspeichers. Durch dieses Verfahren wird aber die Konsistenz zwischen Cache- und Hauptspeicher garantiert.

### 4.8.3 Aufbau des Cachespeichers

Cachespeicher bestehen aus zwei Speicher-Einheiten. Ein *Datenspeicher*, der die im Cache abgelegten Daten enthält und ein *Adressspeicher*, der die Adressen dieser Daten im Hauptspeicher enthält. Bild 4.15 zeigt die Organisation und Steuerung eines Cachespeichers: jede Cache-Zeile enthält ein (Adress, Daten)- Paar und Statusbits. Ein **Block** ist dabei eine zusammengehörende Reihe von Speicherplätzen im Cachespeicher. Dazugehörig wird ein Adresstikett (*address tag*), Index oder auch **Cache-Tag** genannt, im Adressspeicher abgelegt. Das Cache-Tag enthält die Adresse des aktuellen Blocks im Hauptspeicher. Die Statusbits sagen aus, ob die Daten im Cache gültig sind.

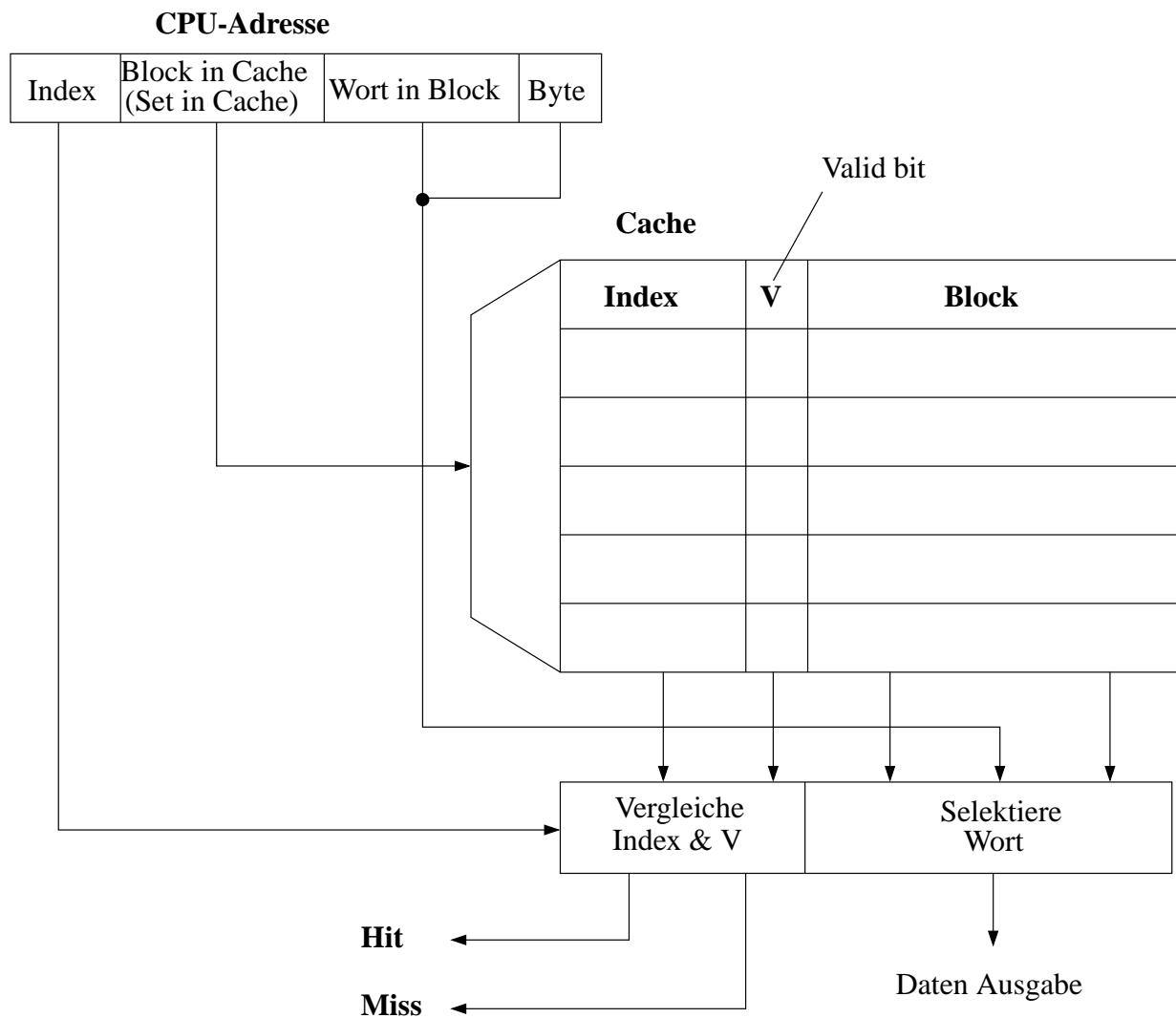


Bild 4.15: Blockschaltbild einer Cache-Steuerung

Die Adressierung eines Cachespeichers bedeutet das Abbilden der Hauptspeicheradressen auf einen Adressraum geringeren Umfangs. Dazu gibt es drei Organisationsformen:

#### 4.8.3.1 Direct-mapped-Cache:

Jeder Block des Hauptspeichers wird auf eine bestimmte Cache-Zeile abgebildet. Bei einem Cache mit  $n$  Blöcken (Zeilen) wird der Block  $m$  des Hauptspeichers auf den  $(m \bmod n)$ -ten Block im Cache abgebildet.

- **Nachteil:** Ständige Konkurrenz der Blöcke (z. B. 1, 65, 129, ...), obwohl andere Blöcke im Cache frei sein können.
- **Vorteil:** Geringer Hardwareaufwand für die Adressierung, da nur ein Vergleich für alle Tags benötigt wird.

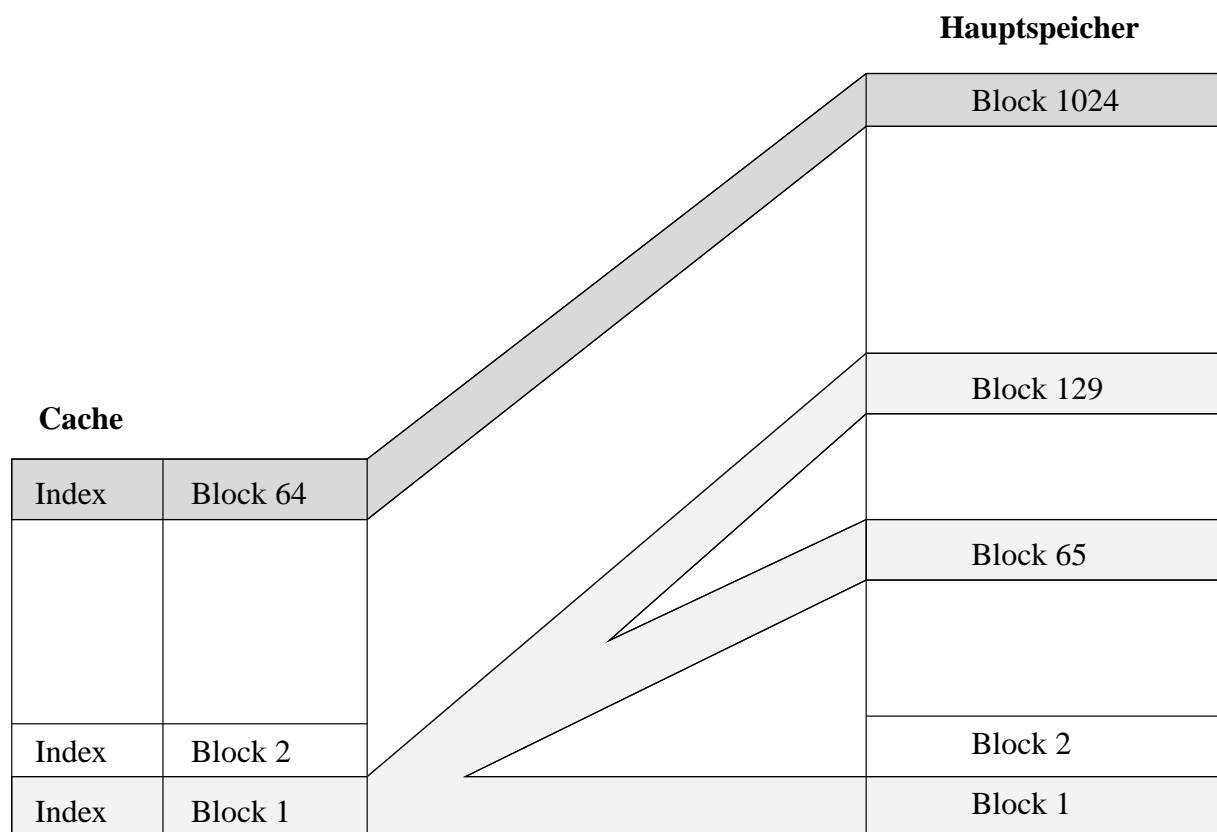


Bild 4.16: Direct-mapped-Cache-Organisation

#### 4.8.3.2 Vollasoziativer Cache

Jeder Block des Hauptspeichers kann in jede Zeile im Cache abgebildet werden. Dadurch wird der Cache optimal ausgenutzt.

- **Nachteile:** Hoher Hardwareaufwand, da eine Hauptspeicheradresse mit allen im Cache gespeicherten Tags (Indexfelder) parallel in einem einzigen Taktzyklus verglichen werden muss. Hierzu ist ein Vergleich für jede Cache-Zeile erforderlich. Bei großer Cache-Kapazität entsteht ein hoher Hardwareaufwand. Deshalb sind vollassoziative Cachespeicher nur mit sehr kleiner Kapazität realisiert (bis ca. 128 Byte). Die große Flexibilität der Abbildungsvorschrift bei dieser Cachearchitektur erfordert eine weitere Hardware, welche die Ersetzungsstrategie (welcher Block soll überschrieben werden, wenn der Cache voll ist) realisiert.

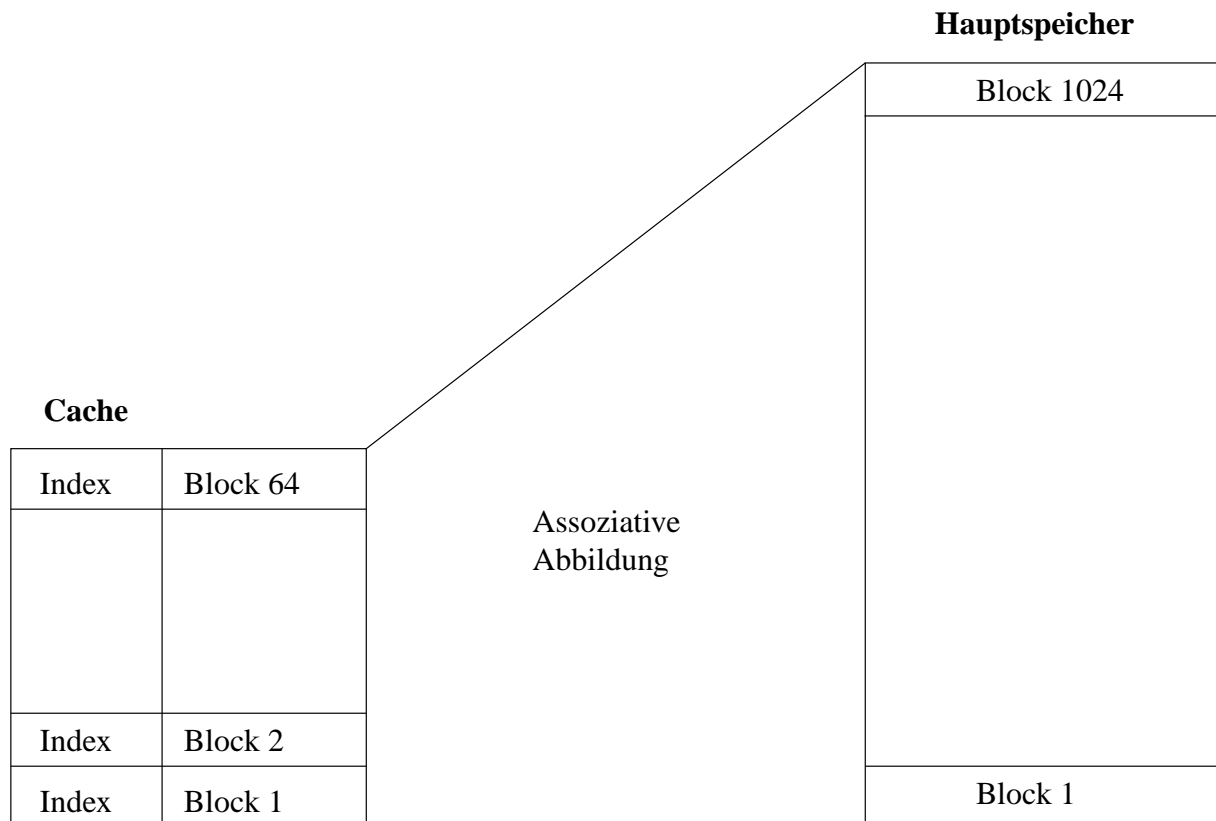


Bild 4.17: Vollassoziative Cache-Organisation

#### 4.8.3.3 n-way-set-assoziativer Cache

Mehrere Cache-Zeilen werden zu Sätzen („Sets“) zusammengefaßt. Ein Block des Hauptspeichers wird auf einen Satz abgebildet. Innerhalb der Sets kann gewählt werden (assoziatives Prinzip, aber in kleinerem Rahmen, so dass der Hardwareaufwand geringer ist). Bild 4.18 zeigt einen 2-way-set assoziativen Cache. Er stellt einen Kompromiß zwischen dem direct-mapped-Cache und dem vollassoziativen Cache dar.

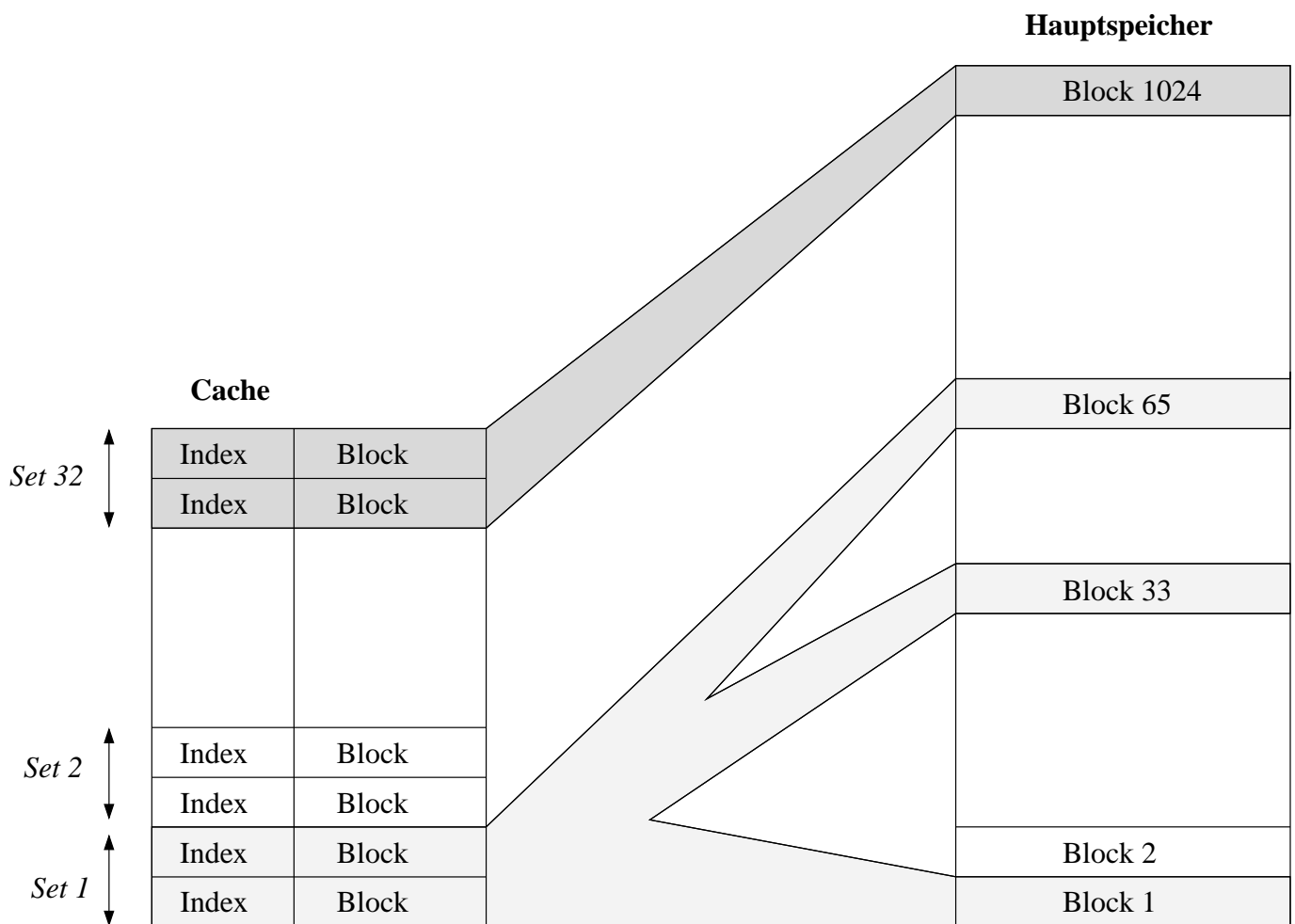


Bild 4.18: Satz-assoziative Cache-Organisation

#### 4.8.4 Ersetzungsstrategien

Bei allen satz-assoziativen Caches muss bei keinem Treffer „Miss“ ermittelt werden, welcher Block innerhalb eines Satzes ersetzt werden darf. Die Ersetzungsstrategien lassen sich in zwei Gruppen einteilen:

1. An der Benutzung der Blöcke orientiert (*usage based*).
  - **LRU** (*least recently used*), d.h. der am längsten nicht mehr angesprochene Block wird ersetzt.
2. Nicht an der Benutzung der Blöcke orientiert (*non usage based*).
  - **FIFO** (*first-in-first-out*), d.h. der Block, der als erster in den Cache gebracht wurde, wird auch als erster ausgelagert.
  - **RAND** (*Random*), ein Pseudozufallsgenerator ermittelt irgendeinen Block. Dieser Block wird dann ersetzt.

## 4.9 Virtuelle Speicherverwaltung

Das Betriebssystem führt Buch über die freien Speicherbereiche und lagert bei nicht ausreichendem Freiplatz im Hauptspeicher diejenigen Speicherinhalte auf den Hintergrundspeicher aus, die gegenüber ihrem Original auf dem Hintergrundspeicher verändert worden sind. Der Mechanismus zur eindeutigen Abbildung des großen virtuellen Speichers auf die effektive Hauptspeicherkapazität wird von der Hardware durch Speicherverwaltungseinheiten (*memory management units, MMU*) unterstützt. Die erforderliche Abbildungsinformation stellt das Betriebssystem in Form von einer oder mehreren *Übersetzungstabellen* zur Verfügung (siehe Bild 4.19).

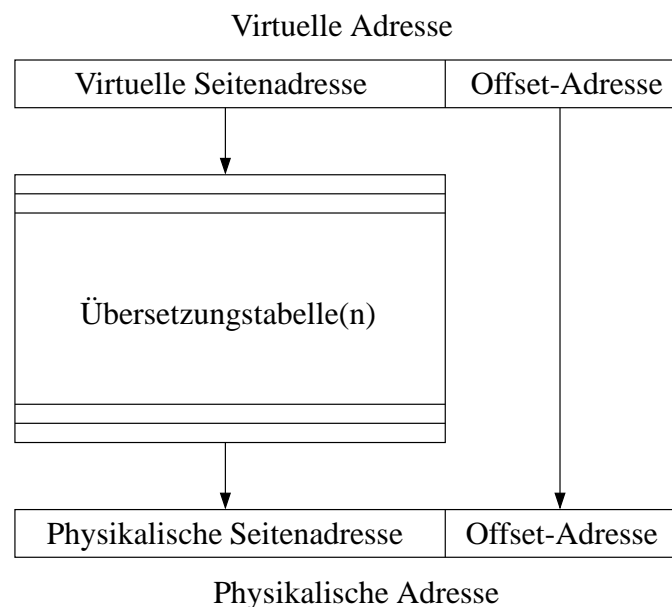


Bild 4.19: Abbildung virtueller in physikalische Adressen

Die virtuelle Speicherverwaltung stellt während der Programmausführung fest, welche Daten gerade gebraucht werden, transferiert Daten zwischen Haupt- und Massenspeicher und aktualisiert die Referenzen zwischen virtueller und physikalischer Adresse in einer Übersetzungstabelle.

Die Lokalitätseigenschaften von Programmen und Daten gewährleisten eine hohe Wahrscheinlichkeit, dass die Daten, welche die CPU anfordert, im physikalischen Hauptspeicher zu finden sind.

### 4.9.1 Segmentierungs- und Seitenwechselverfahren

Um den Umfang der Übersetzungstabellen gering zu halten, bezieht man die Abbildungsinformation nicht auf einzelne Adressen, sondern auf zusammenhängende Adressbereiche. Dazu gibt es zwei Möglichkeiten, die *Segmentierung* und die *Seitenverwaltung*.

#### 4.9.1.1 Segmentierung

Hierbei werden die Adressbereiche so groß gewählt, daß sie logisch zusammenhängende Einheiten, wie Programmcode, Daten oder Stack vollständig umfassen. Der Adressraum wird somit in Segmente

variabler Größe zerlegt. Die einzelnen Segmente können relativ groß sein.

- **Vorteile:**

- Die Segmentierung spiegelt die logische Programmstruktur wider.
- Durch große Segmente ist der erforderliche Datentransfer seltener.

- **Nachteile:**

- umfangreicher Datentransfer, da ganze Segmente übertragen werden.
- Wenn ein Programm z. B. nur aus einem Code- und Datensegment besteht, dann kann es nur vollständig in den Hauptspeicher eingelagert werden.

#### 4.9.1.2 Seitenverwaltung

Bei diesem Verfahren wird der logische und physikalische Adressraum in Einheiten fester Größe, sog. Seiten unterteilt. Die Seiten sind relativ klein (256 Byte - 4 KByte).

- **Vorteile:**

- Durch die relativ kleine Seitengröße ist eine bessere Anpassung an den aktuell benötigten Teil eines Programms im Hauptspeicher möglich.
- geringerer Verwaltungsaufwand bei der Zuweisung von Hauptspeicherplatz.

**Nachteile:**

- häufiger Datentransfer

#### 4.9.2 Adress-Umsetzung bei Segmentierungsverfahren

Eine segmentorientierte Speicherverwaltung hat eine Strukturierung des virtuellen als auch des realen Adressraumes zur Folge. Bild 4.20 zeigt die Wirkungsweise einer solchen MMU. Bei dieser Variante liegen die Segmentgrenzen im Hauptspeicher an Bytegrenzen. Die Adressumsetzung erfolgt über eine Segmenttabelle. Sie enthält für jedes Segment eine 32-bit-Segmentbasisadresse und eine 24-bit-Segmentlängenangabe. Die reale (physikalische) Adresse ergibt sich aus der Segmentbasisadresse, zu der die virtuelle Bytenummer addiert wird. Durch die angegeben Unterteilung der virtuellen Adresse kann ein Segment in diesem Fall bis zu  $2^{24} = 16$  MByte umfassen.

Bild 4.21 zeigt die Strukturierung des virtuellen und realen Adressraumes am Beispiel dreier Segmente mit den in der Segmenttabelle in Bild 4.20 angegebenen Basisadressen und Segmentlängen.

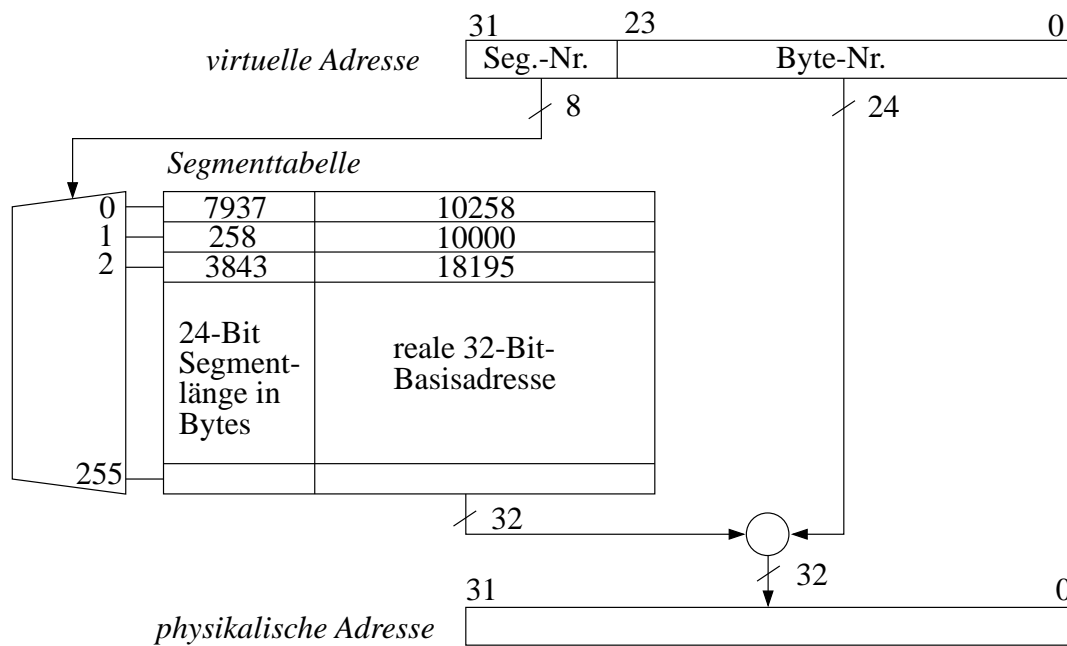


Bild 4.20: Adressumsetzung mit Segmentadressen im Hauptspeicher an Bytegrenzen

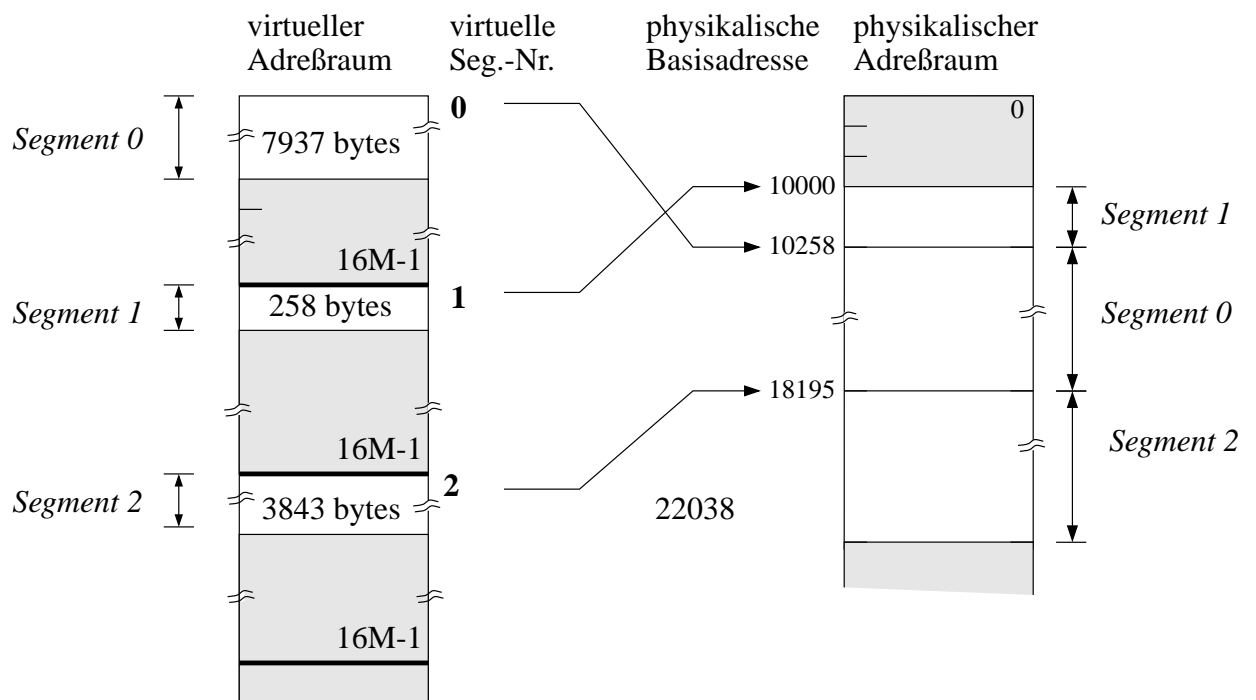


Bild 4.21: Strukturierung des virtuellen und physikalischen Adressraumes

Eine andere Art der Strukturierung erhält man mit der in Bild 4.22 dargestellten MMU. Bei dieser Variante der Segmentierung liegen die Segmentgrenzen im Hauptspeicher nicht an Bytegrenzen, sondern an Vielfachen von Blöcken (hier von 256 Bytes). Die Segmente werden im virtuellen und realen Adressraum in Blöcke von 256 Bytes unterteilt.

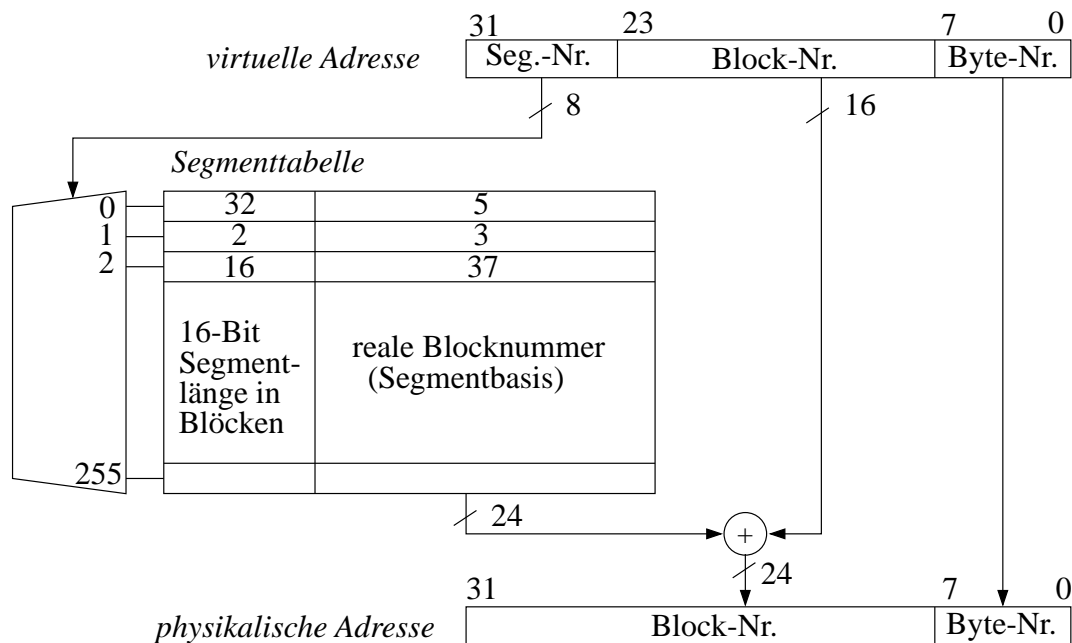


Bild 4.22: Adressumsetzung mit Segmentadressen im Hauptspeicher an Blockgrenzen

Die sich dadurch ergebende Strukturierung des virtuellen und realen Adressraums ist in Bild 4.23 anhand der drei in der Segmenttabelle eingetragenen Segmente gezeigt. Die Segmentgrenzen ergeben sich im virtuellen Adressraum aus den Vielfachen von 16 Mbyte mit jeweils 64K Blöcken. Im Hauptspeicher liegen die Segmentgrenzen an Vielfachen von 256 Bytes.

### 4.9.3 Adress-Umsetzung bei Seitenwechselverfahren

Bei der Seitenverwaltung wird der Adressraum in Seiten unterteilt, welche dann auf entsprechende Rahmen (*frames*) im Hauptspeicher abgebildet werden. Dabei wird eine virtuelle Seitennummer auf eine Rahmennummer abgebildet. Im Bild 4.24 ergeben sich Seiten der Größe 4 KBytes. Die Strukturierung des virtuellen und physikalischen Adressraums ist im folgenden Bild 4.25 dargestellt.

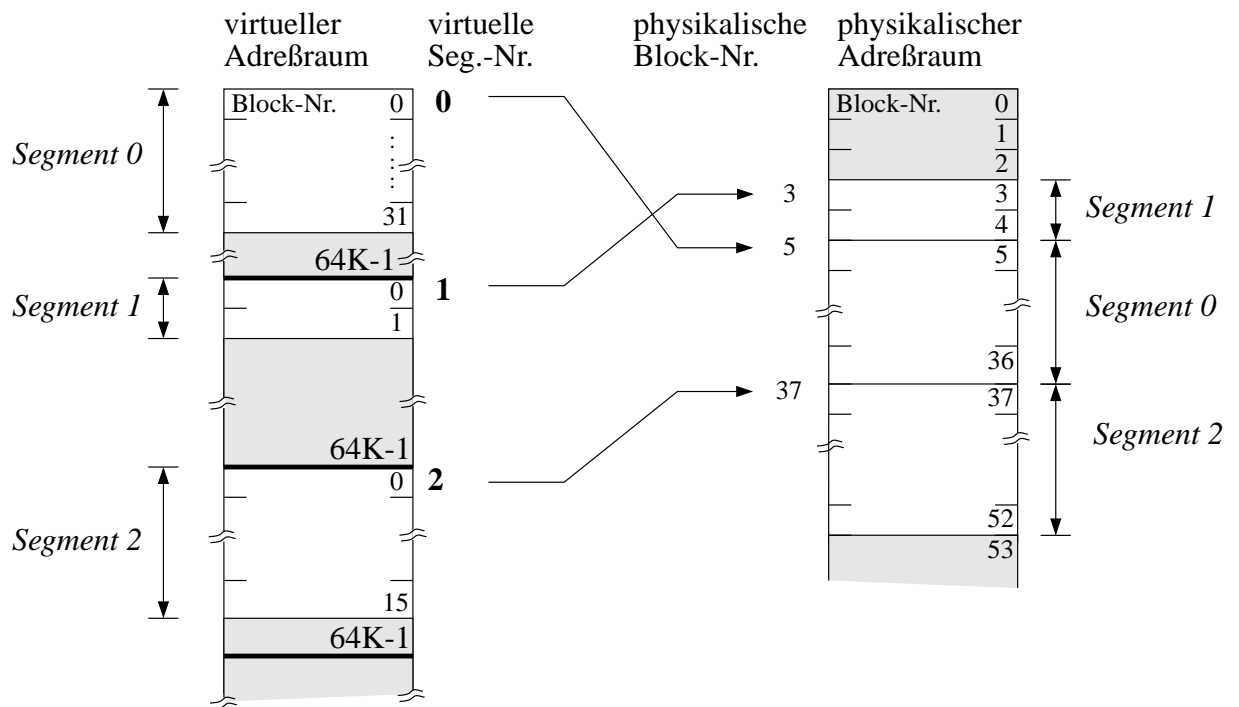


Bild 4.23: Strukturierung des virtuellen und physikalischen Adressraumes

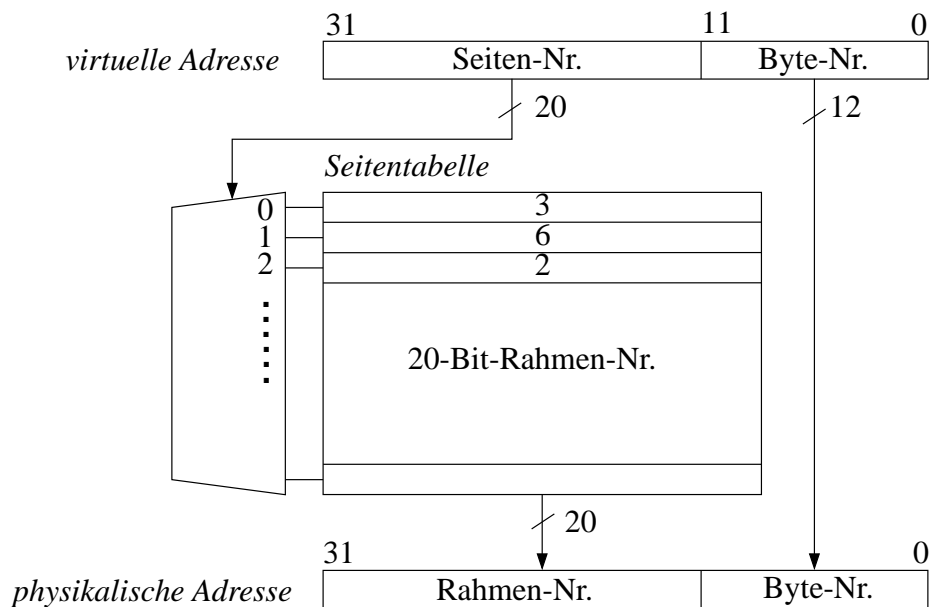


Bild 4.24: Adressumsetzung bei Seitenverwaltung

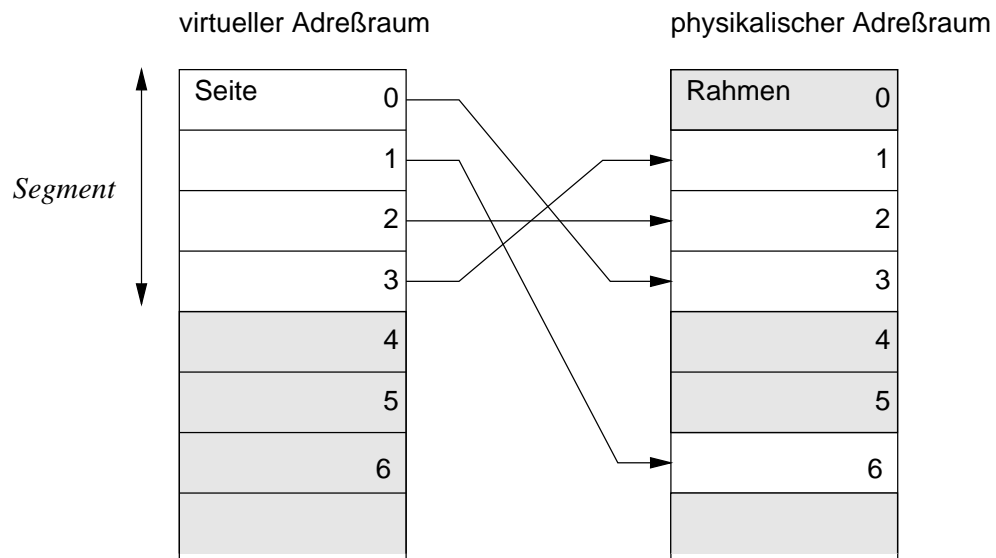


Bild 4.25: Strukturierung des virtuellen und physikalischen Adressraumes

Die Übersetzung einer virtuellen Adresse in eine physikalische Adresse geschieht meist nicht über *eine*, sondern *mehrstufig* über mehrere Übersetzungstabellen. Oft werden zwei Übersetzungstabellen, eine Segmenttabelle und eine Seitentabelle, benutzt, so daß ein Teil der virtuellen Adresse einen Eintrag in der Segmenttabelle adressiert, der wiederum mit dem zweiten Teil der virtuellen Adresse einen Eintrag in der Seitentabelle bezeichnet, der dann die physikalische Adresse der Speicherseite enthält.

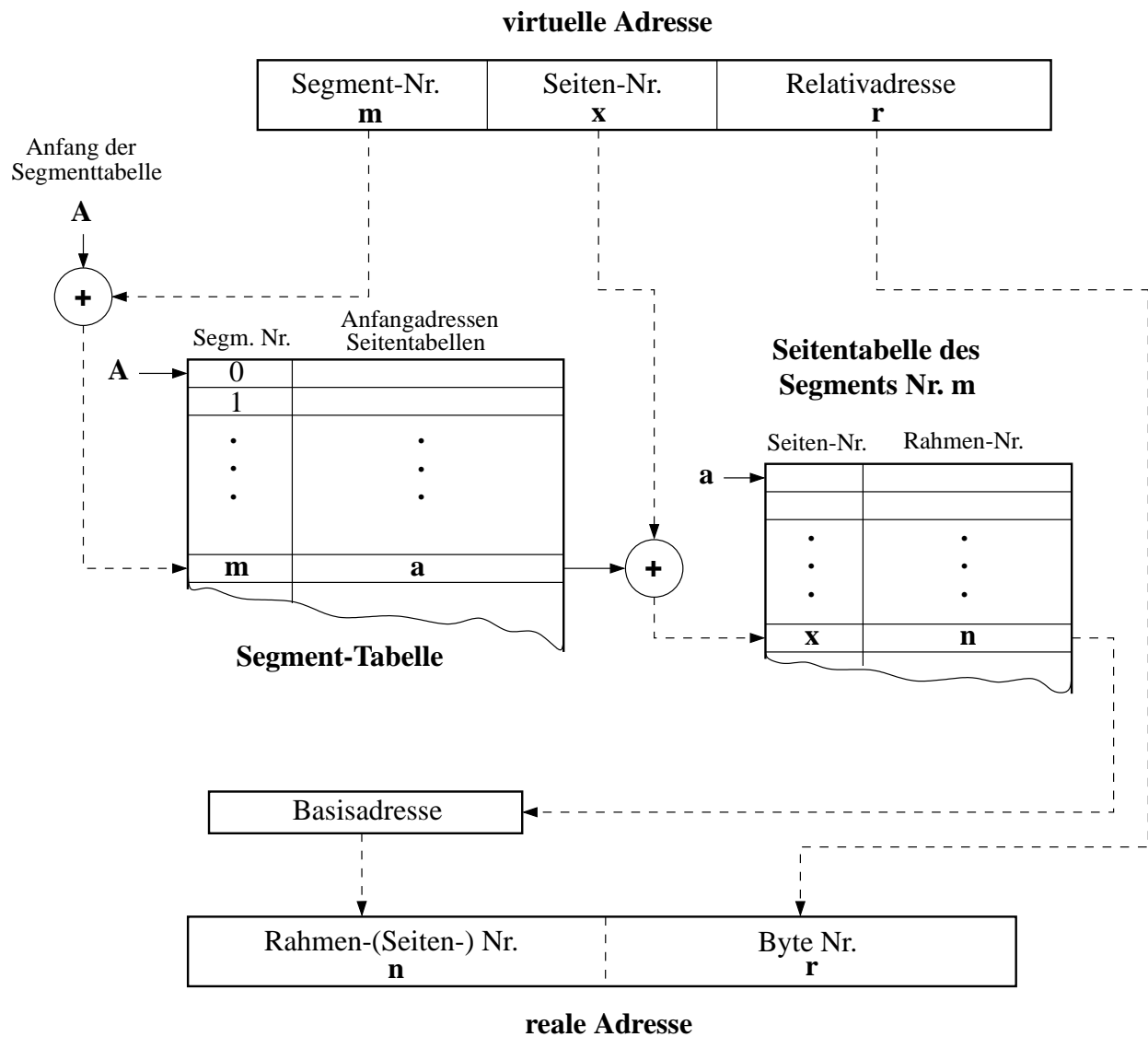


Bild 4.26: Adressumsetzung bei Segmentierung mit Seitenverwaltung

#### 4.9.4 Probleme der virtuellen Speicherverwaltung

Beim Austausch von Daten zwischen dem Hauptspeicher und dem Hintergrundspeicher können sich folgende Probleme ergeben:

**a) Der Einlagerungszeitpunkt:** Wann werden Segmente oder Seiten in den Hauptspeicher eingelagert? Das gängigste Verfahren ist die Einlagerung auf Anforderung. (*demand paging*). Hierbei werden die Daten eingelagert, sobald sie gefordert werden, sofern sie sich noch nicht im Hauptspeicher befinden. Ein Zugriff auf Daten, die sich nicht im Hauptspeicher befinden, wird als **Segmentfehler** (*segment fault*) oder **Seitenfehler** (*page fault*) bezeichnet.

**b) Das Zuweisungsproblem:** An welcher Stelle des Hauptspeichers werden die Seiten oder Segmente eingelagert?

- Bei Segmentierungsverfahren muß eine ausreichend große Lücke im Hauptspeicher gefunden werden. Hierzu gibt es drei Strategien:
  - **first-fit**: die erste passende Lücke wird genommen.
  - **best-fit**: die kleinste passende Lücke wird genommen.
  - **worst-fit**: die größte passende Lücke wird genommen.

Alle drei Strategien haben zur Folge, daß der Speicher nach einiger Zeit in belegte und unbelegte Speicherbereiche zerfällt (**externe Fragmentierung**), da die unbelegten Speicherbereiche meist zu klein sind, um noch Segmente aufnehmen zu können.

- Bei Seitenwechselverfahren taucht dieses Problem nicht auf, da alle Seiten gleich groß sind und somit immer „passende Lücken“ im Hauptspeicher entstehen. Es tritt also keine externe Fragmentierung, wohl aber eine **interne Fragmentierung** bei der Aufteilung eines Programms auf die Seiten auf.

**c) Das Ersetzungsproblem:** Welche Segmente oder Seiten müssen ausgelagert werden, um Platz für neu benötigte Daten zu schaffen?

- Bei Segmentierungsverfahren wird die Anzahl der gleichzeitig von einem Prozeß benutzten Segmente limitiert. Bei der Einlagerung eines neuen Segments wird ein zuvor für diesen Prozeß benutztes Segment ausgelagert. Es ist jedoch auch möglich, eine der im folgenden für Seitenwechselverfahren beschriebenen Methoden zu verwenden.
- Bei Seitenwechselverfahren können folgende Strategien zum Ersetzen einer Seite benutzt werden.
  - **FIFO** (*first-in-first-out*): Die sich am längsten in Hauptspeicher befindende Seite wird ersetzt.
  - **LIFO** (*last-in-first-out*): Die zuletzt eingelagerte Seite wird ersetzt.
  - **LRU** (*least recently used*): Die Seite, auf die am längsten nicht zugegriffen wurde, wird ersetzt.
  - **LFU** (*least frequently used*): Die seit ihrer Einlagerung am seltensten benutzte Seite wird ersetzt.
  - **LRD** (*least refernce density*): Mischung aus LRU und LFU; die Seite mit der geringsten Zugriffsichte (Anzahl der Zugriffe in Einlagerungszeitraum) wird ersetzt.

## 4.10 Direkter Speicherzugriff (*direct memory access DMA*)

Bisher wurde der Datentransfer zwischen dem Prozessor und dem Speicher bzw. zwischen dem Prozessor und der Peripherie betrachtet. Bild 4.27 zeigt schematisch die Datenübertragung aus dem Speicher zur Peripherie.

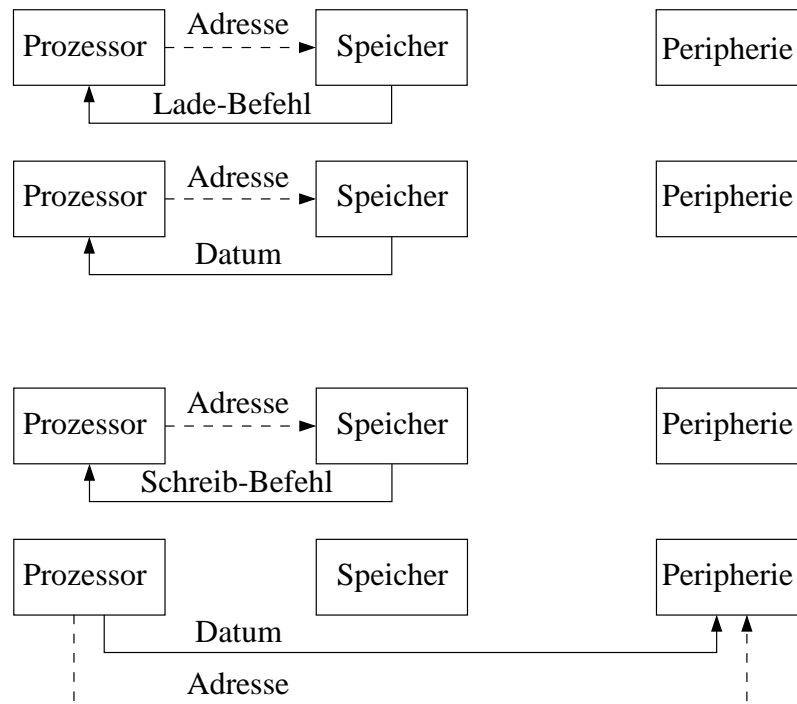


Bild 4.27: Datenübertragung aus dem Speicher zur Peripherie

Bei einem Lade-Befehl adressiert der Prozessor zunächst den Speicher, berechnet die Adresse des zu ladenden Datums (*Lese-Phase*). Im nächsten Schritt wird das Datum in ein Register der CPU geholt (*Holphase*). Handelt es sich jedoch um einen Schreib-Befehl, der Daten zur Peripherie übertragen soll, so wird die Adresse des Peripherie-Speichers, in den das Datum geschrieben werden soll, ermittelt und das Datum zur Peripherie übertragen.

Unter der Annahme, daß sowohl das Holen als auch die Ausführung jedes Befehls genau einen Speicherzugriff erfordert, sind bei der Datenübertragung durch die CPU mindestens vier Speicherzugriffe nötig. (Bei Schleifenbefehlen kommt noch der Befehl zum Inkrementieren bzw. Dekrementieren des Schleifenzählers und der Befehl zur Abfrage der Schleifen-Bedingung hinzu, so daß dann mindestens sechs Speicherzugriffe für die Übertragung eines Datums erforderlich sind). Die Übertragungsrate ist dann für viele Anwendungen bei weitem nicht ausreichend, insbesondere nicht für die Übertragung von Daten zu schnellen Festplatten. Ein weiterer Nachteil der beschriebenen Datenübertragung ergibt sich daraus, daß jede Übertragung eines Datenblocks die CPU belastet, auch wenn die zu übertragenden Daten für die CPU keine Bedeutung haben (Zwischenergebnisse).

Abhilfe schafft der **direkte Speicherzugriff** (*direct memory access DMA*). Hierbei erfolgt die Datenübertragung von und zur Peripherie ohne Beteiligung des Mikroprozessors. Eine spezielle

Steuereinheit (*DMA-Controller*) übernimmt die Koordinierung der Übertragung von Datenblöcken zwischen dem Speicher und der Peripherie.

**Vorteile:**

- Die Speicherzugriffe für das Holen der Lade-, Speicher- und Schleifenbefehle entfallen, da die Datenübertragung hardwaremäßig ausgeführt wird. Damit sind nur 2 (ggf. sogar nur 1) Speicherzugriffe für die Übertragung eines Datums erforderlich.
- Der Prozessor wird entlastet und kann zur Übertragungszeit andere Aufgaben erledigen, sofern diese den Systembus nicht benötigen.

### 4.10.1 Prinzip des direkten Speicherzugriff

Zu Beginn der Datenübertragung wird der DMA-Controller von der CPU mit den benötigten Informationen versorgt (Bild 4.28a). Diese Informationen bestehen aus der Startadresse der zu übertragenden Daten, der Zieladresse, der Anzahl der zu übertragenden Bytes sowie der Übertragungsrichtung (Lesen/Schreiben) und Steuerinformationen. Der DMA-Controller wird dabei wie jeder andere Peripherie-Baustein über den Adreßbus unter einer festgelegten Adresse angesprochen. Der Datentransfer wird dann selbständig vom DMA-Controller vorgenommen.

Zur Übertragung durch den DMA-Controller können zwei verschiedene Verfahren angewandt werden. Sie werden im folgenden für eine Übertragung vom Speicher zu einer Peripherie-Schnittstelle erläutert.

**Indirekte Übertragung:**

Der DMA-Controller führt zunächst einen Lesezugriff auf den Speicher durch und lädt das Datum in ein internes Pufferregister (Bild 4.28b). Danach spricht er den Peripherie-Baustein an und überträgt das Datum (Schreibzugriff) (Bild 4.28c). Es sind also zwei Speicherzugriffe für den Datentransfer nötig. Dieses Verfahren wird als „*two cycle transfer*“-Verfahren bezeichnet. Bei Speicher-Speicher-Übertragung wird eine Speicherzelle im Zielbereich des Speichers adressiert.

**Direkte Übertragung:**

Bei diesem Verfahren wird das Datum direkt ohne die Zwischenspeicherung in einem internen Register des DMA-Controllers übertragen (Bild 4.28d). Der DMA-Controller adressiert das Datum im Speicher über den Adreßbus und gleichzeitig auch die Peripherie-Schnittstelle über Auswahlsignale. Das Datum kann dann aus dem Speicher in die Peripherie gelangen. Damit ist nur ein Speicherzugriff erforderlich. Das Verfahren kann jedoch nicht für einen Speicher-Speicher-Transfer benutzt werden. Es wird auch „*fly-by transfer*“-Verfahren genannt.

### 4.10.2 DMA-Übertragungsraten

In der Regel kann man bei einem DMA-Baustein zwischen verschiedenen Übertragungsraten wählen, die sich durch die Belegung des Systembusses unterscheiden:

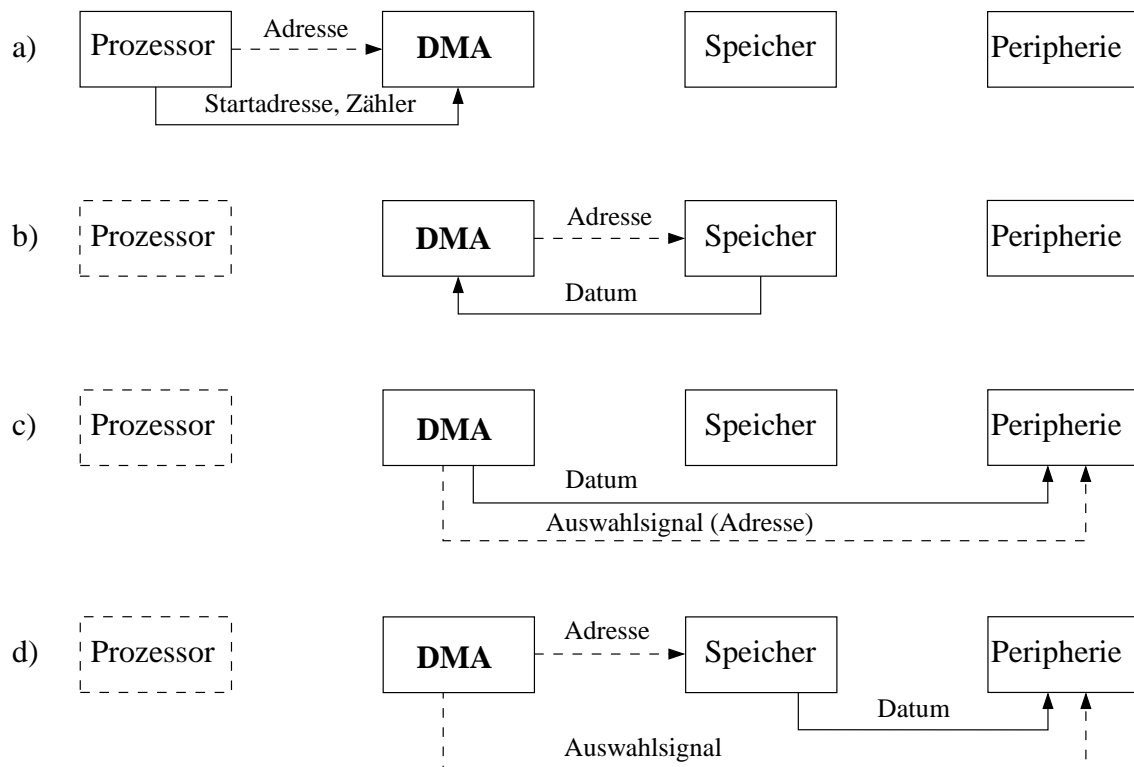


Bild 4.28: Das Prinzip des direkten Speicherzugriffs

**Einzeltransfer** (*single transfer mode*):

Hierbei wird jeweils genau ein Datum übertragen. Danach wird der Systembus für den Prozessor freigegeben. Diese Übertragungsart wird auch als *cycle stealing* bezeichnet, da dem Prozessor für jeden Datentransfer einzelne Buszyklen entzogen werden.

**Blocktransfer** (*block transfer mode, burst mode*):

Bei dieser Transferart überträgt der DMA-Controller ohne zwischenzeitliche Busfreigabe alle Daten eines Blocks, sobald er den Systembus erhalten hat.

**Transfer auf Anforderung** (*demand transfer mode*):

Diese Übertragungsart nimmt eine Mittelstellung zwischen dem Einzeltransfer und dem Blocktransfer ein. Die Datenübertragung wird solange durchgeführt, wie dies der Anforderer (*Requester*) wünscht. Diese Transferart wird bevorzugt bei Peripherie-Geräten benutzt, die ihre Daten in Schüben (*bursts*) liefern bzw. verlangen, zwischen denen lange Pausen liegen.

Besonders zu beachten bei DMA-Transfers sind:

- unterschiedliche Datenbreite von Quelle und Ziel: z.B. Übertragung von einem 8-bit breiten Peripherie-Gerät (Drucker, Terminal, ...) in den 32-bit breiten Hauptspeicher.
- Übertragung von nicht „ausgerichteten“ Daten (*non-aligned-data*): Hier sind mehrere Speicherzugriffe zum Lesen oder Schreiben eines Datums erforderlich.

DMA-Controller sind durch verschiedene Maßnahmen an diese Anforderungen angepaßt, wie z. B.

- Zerlegung bzw. Sammeln von Daten vor dem Weitertransport in interne Pufferregister bei unterschiedlicher Breite von Quelle und Ziel.
- Automatische Erkennung von Zugriffen auf nicht-ausgerichtete Daten und Zerlegung dieser Zugriffe in mehrere Speicherzyklen

Moderne DMA-Controller können mehrere DMA-Vorgänge quasi-gleichzeitig bearbeiten. Sie haben dazu bis zu acht DMA-Kanäle, d. h. bis zu acht gleichartige Registersätze und Peripherie-Schnittstellen bei einem gemeinsamen Steuerwerk für die Datenübertragung.



# **Kapitel 5**

## **Digitale Signalprozessoren und Mikrocontroller**

Siehe die Vorlesungsfolien :)



## Literaturverzeichnis

- [Bähring 02a] H. Bähring. *Mikrorechnertechnik: Busse, Speicher, Peripherie und Mikrocontroller*, Band II. Springer Verlag, 3. Auflage, 2002.
- [Bähring 02b] H. Bähring. *Mikrorechnertechnik: Mikroprozessoren und digitale Signalprozessoren*, Band I. Springer Verlag, 3. Auflage, 2002.
- [und Th. Ungerer 02] U. Brinkschulte und Th. Ungerer. *Mikrocontroller und Mikroprozessoren*. Springer Verlag, 2002.