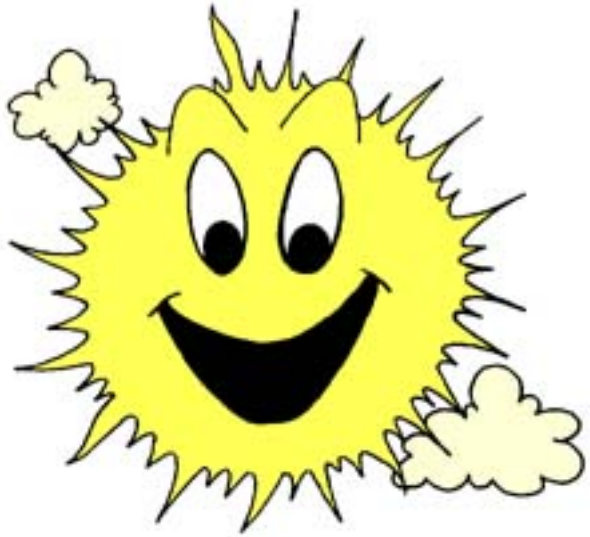
Wiederholung der 11. Vorlesung

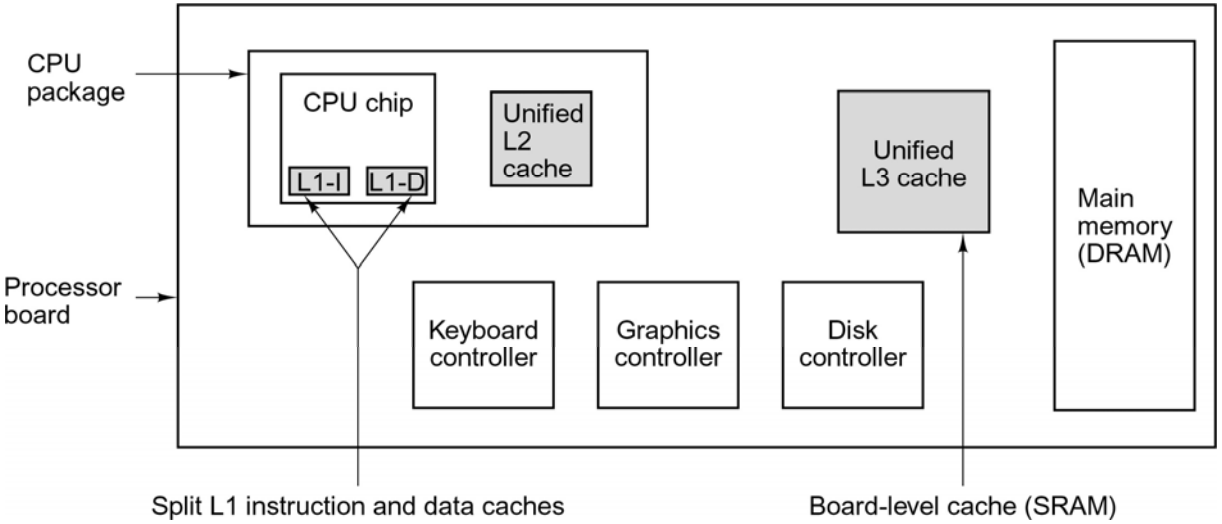


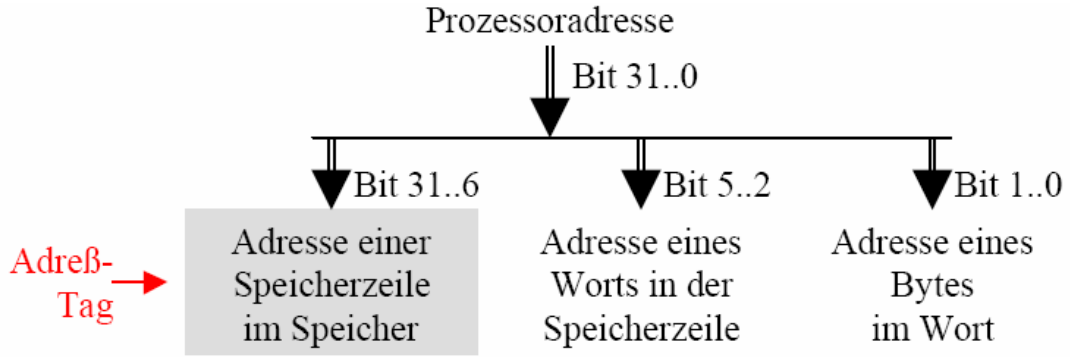
Bitte beachten:

Das Skript wurde korrigiert und aktualisiert. Einige Themen aus der DT sind nach RG gewandert, stehen aber noch im ursprünglichen Skript. Beachten Sie die Hinweise aus den Vorlesungen!

http://www.fbi.fh-darmstadt.de/~jwietzke/RG.pdf

8.2 Wiederholung Cache

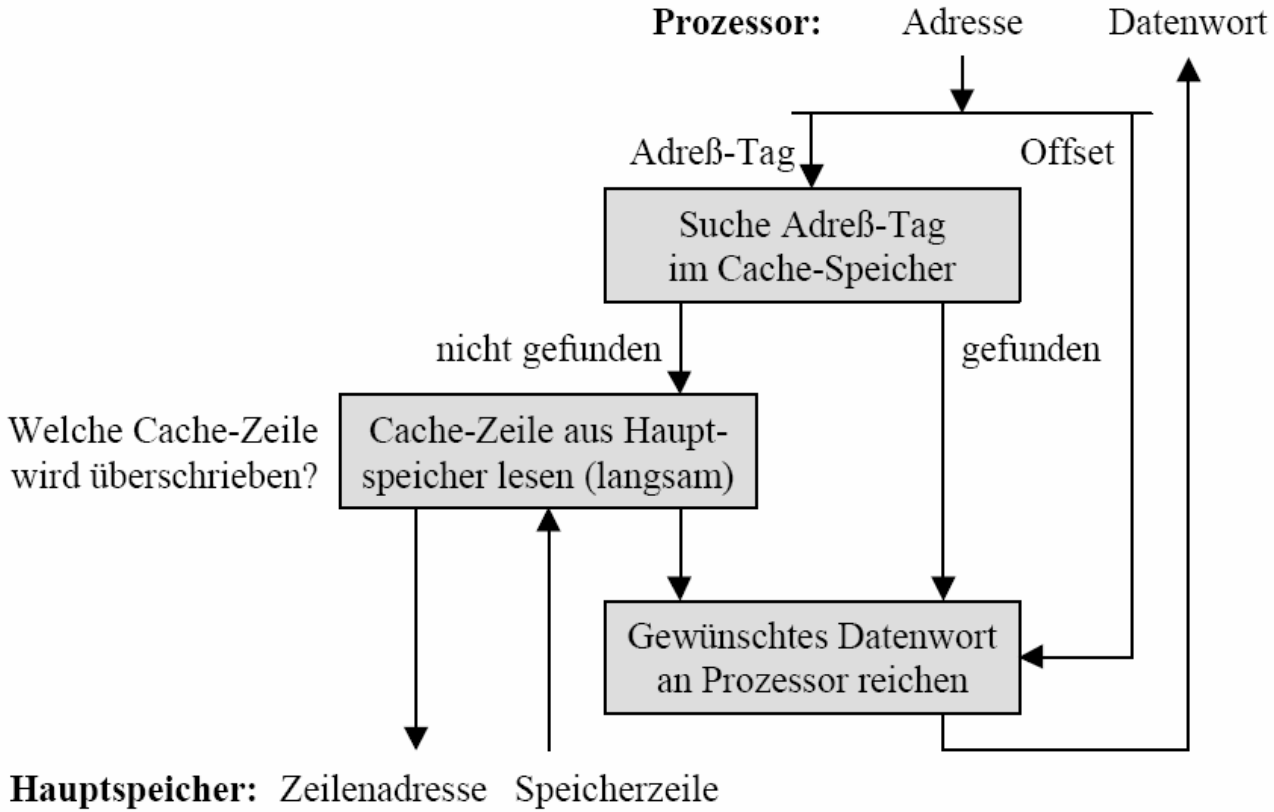




Stichworte: Cache-Level, Speicherzeile, Adress-Tag

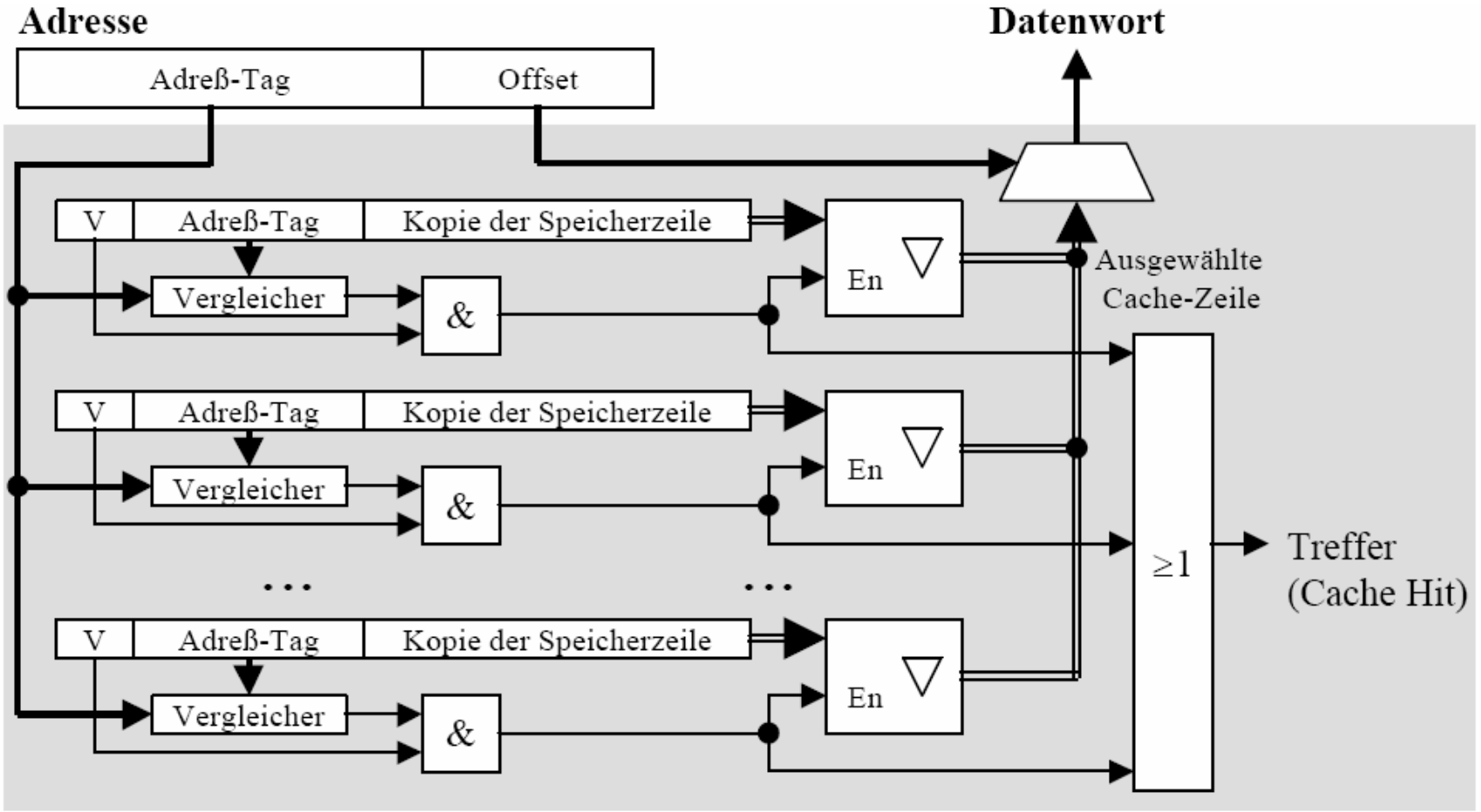
Warum Caching?

8.2 Cache-Organisation: Lesen



- **Voll assoziativ** (Fully associative): Ein Datenwort kann in einem beliebigen Cache-Eintrag abgelegt sein.
- **Direkt abgebildet** (Direct mapped): Ein Datenwort kann genau in einem Eintrag abgelegt sein.
- **Mengen-assoziativ** (Set associative): Ein Datenwort kann in wenigen Cache-Einträgen (typischerweise 2 bis 8) abgelegt sein.

8.2 Voll assoziativer Cache



8.3 Kenngrößen

- Liegt ein gesuchtes Wort im Cache, erhält man Treffer (hit).
  Trefferrate a in Prozent (hit rate).
- Liegt gesuchtes Wort nicht im Cache (miss), muss es aus dem Speicher gelesen werden:
  Fehlrate (1-a) in Prozent (miss rate).
- Mit Zugriffszeit t_c eines Wortes im Cache und der Zugriffszeit t_m eines Wortes im Hauptspeicher ist die
  mittlere Zugriffszeit t_a auf ein Wort:
  $t_a = a \cdot t_c + (1-a) \cdot t_m$
- Ist ein Wort nicht im Cache vorhanden, muss aus Speicher in den Cache geladen werden.
- Bei **Direkt abgebildetendem Cache** ist Zuordnung der Adresse des Wortes zu einer Cache-Zeile eindeutig.
- Beim **Assoziativen Cache** sind mehrere Strategien möglich:
  1. Wähle älteste Zeile aus
  2. Wähle Zeile, auf die am längsten nicht mehr zugegriffen wurde.
  3. Wähle zufällig eine Zeile aus.

  Alle drei Strategien haben weitgehend gleichen Trefferraten.

8.4 Schreiben von Daten

Informatik

Beim Schreiben von Daten muss sichergestellt sein, dass Daten in Cache und Hauptspeicher konsistent bleiben.

Mehrer Strategien:

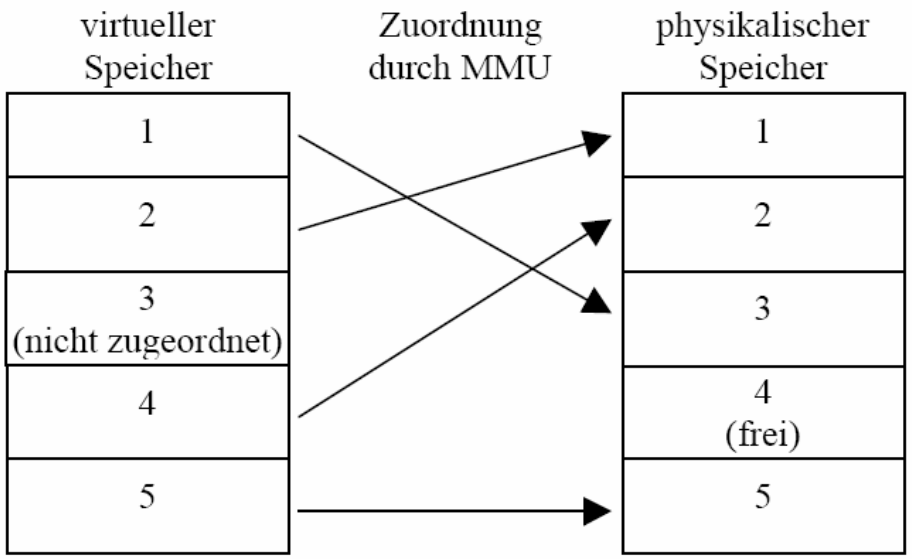
- **No-Write**: Wort nur in Hauptspeicher schreiben, nicht in Cache. Zugehörige Zeile im Cache als ungültig markieren.
  → langsam, Schreiben in HS, ggf. Lesen aus HS
- **Write-Through**: Cache- und Hauptspeicher beide schreiben. Cache-Zeile bleibt gültig. Übliches Verfahren.
  → langsame Schreibzugriffe.
- **Write-Back**: Nur Cache-Speicher schreiben, aber Cache-Zeile als modifiziert („Dirty“) markieren. Vor Überschreiben der Zeile muss der Inhalt in Hauptspeicher gesichert werden. Dirty-Bit ist der Cache-Zeile angefügt, wird beim Laden einer Zeile aus dem HS gelöscht, beim Schreiben eines Wortes in der Cache-Zeile gesetzt.

8.5 Lesen von Daten

Zwei prinzipielle Lese- und Füllstrategien

- **on demand, demand fetching**
  Beim ersten Lesen einer Information aus dem Hauptspeicher wird die gesamte Zeile auch in den Cache übertragen. Benötigte Blöcke, die noch nicht im Cache liegen, werden nachgeladen. Diese Strategie wird von jedem Cache verwendet
- **Prädiktiv, prefetch**
  Es wird versucht, vorherzusagen, welche Speicherzeilen zukünftig gebraucht werden und diese während Leerlaufphasen schon in den Cache vorgeladen.
  Ein zweiter Programmzähler (remote PC) im Cache wird dazu verwendet, den künftigen Programmverlauf vorherzubestimmen und entsprechende Blöcke vorzuladen.

9.2 virtuelle und physikalische Adresse



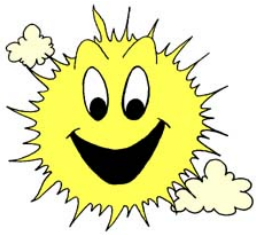
Der virtuelle und der physikalische Speicherbereich wird in Speicherseiten fester Größe aufgeteilt
→ pages (Kacheln)

Übliche Seitengrößen sind 4 kBytes oder 8 kBytes (2er Potenz), aber auch deutlich größere Seiten werden verwendet.

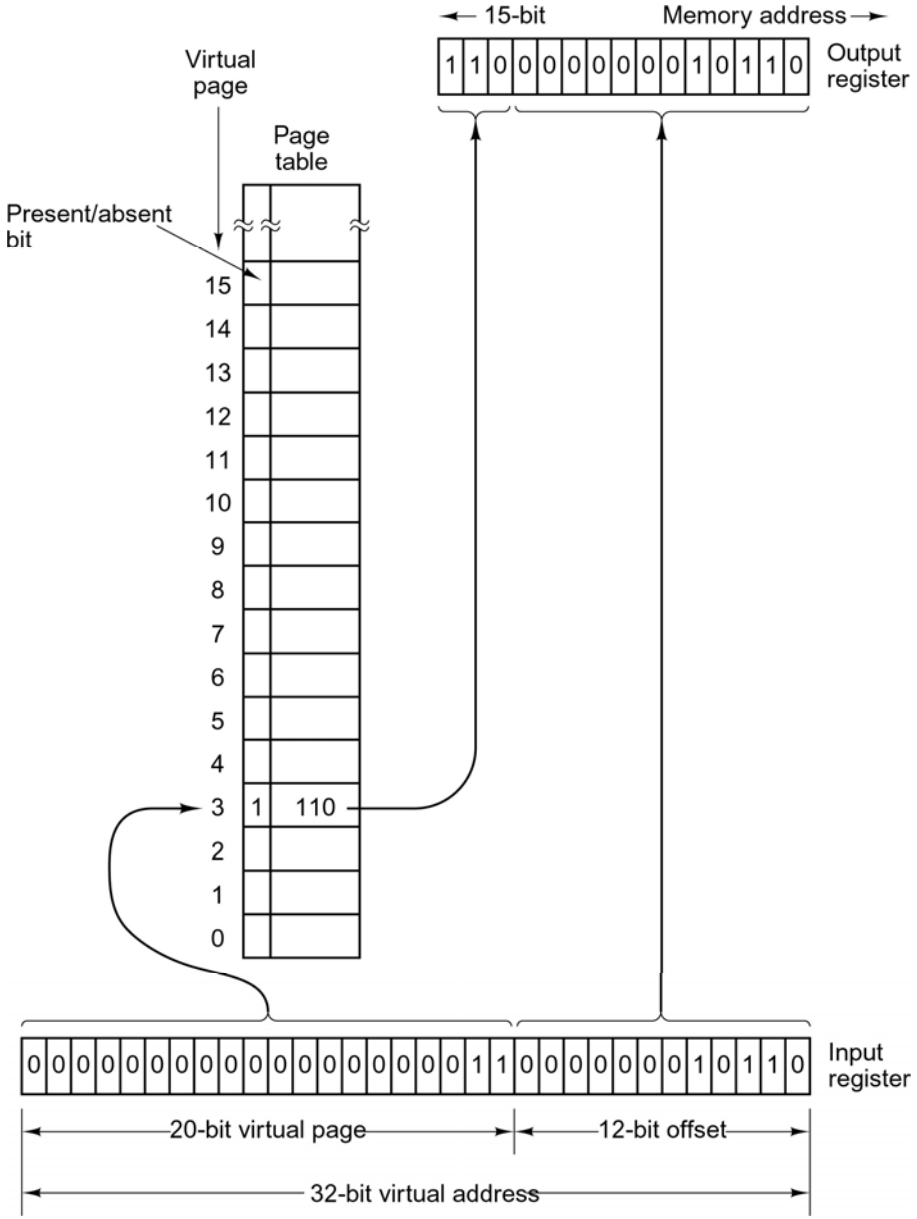
Virtuelle Speicherseiten werden durch die MMU physikalischen Seiten zugeordnet.

Ist bei Zugriff auf eine virtuelle Seite keine physikalische zugeordnet, liegt ein **Seitenfehler** (page fault) vor. Die MMU erzeugt dann in der CPU einen Interrupt, die Ausnahmebehandlung lädt die fehlende Seite.

Ende der Wiederholung



9.2 virtuelle Adressierung durch MMU



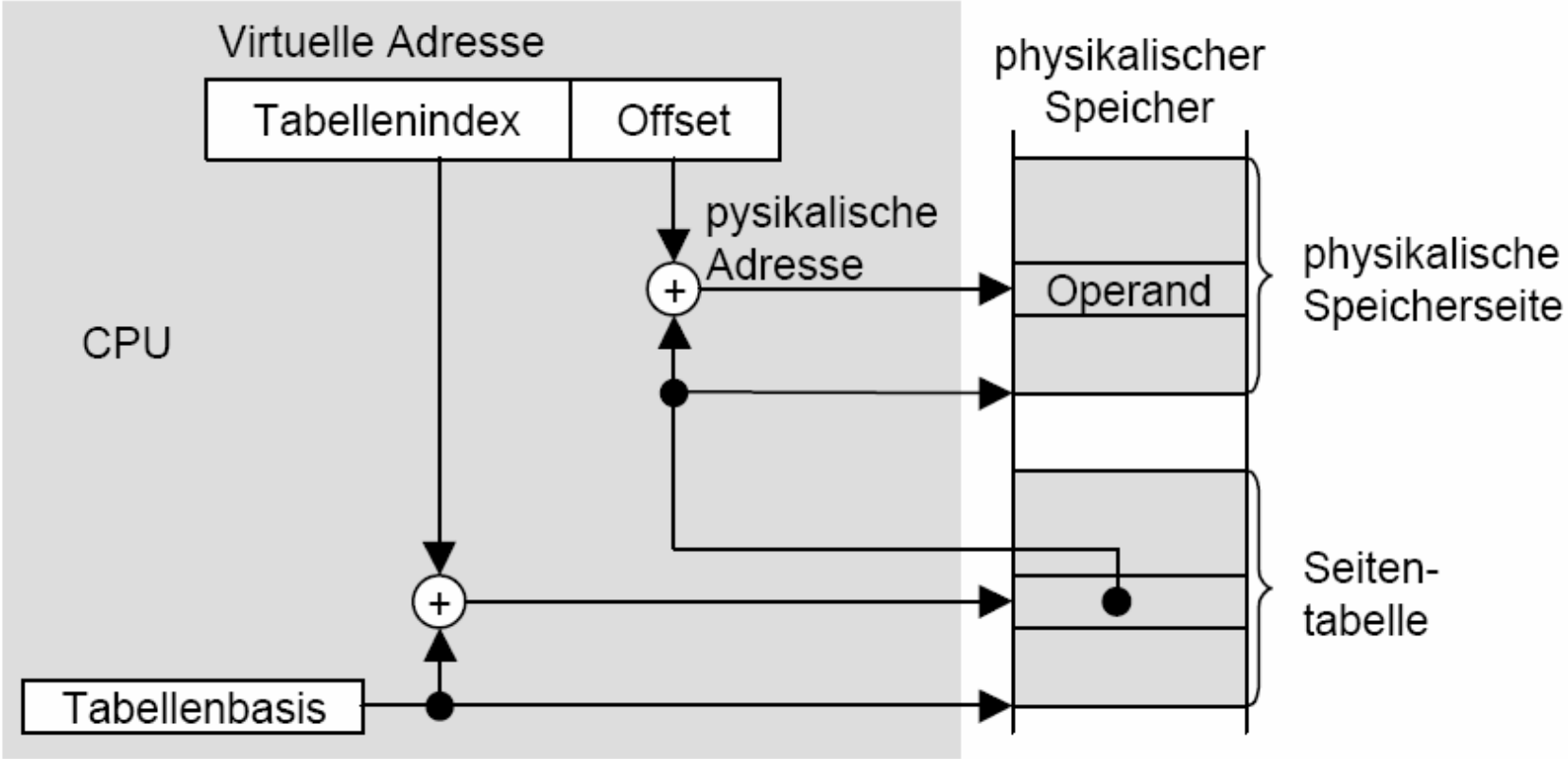
Beispiel: eine 32 Bit-Adresse wird zerlegt in 20 Bit Seiten-Nummer und 12 Bit Offset.

Der Offset ist die Adresse des gesuchten Byte in der 4096 Byte (12 Bit) großen Seite.

Mit der Seitennummer wird in der Tabelle die physikalische Adresse gesucht.

Physik. Adresse + Offset = Adresse des Operanden im Speicher.

9.2 virtuelle Adressierung durch MMU (2)



Die Seitentabelle liegt im physikalischen Speicher, ihre Anfangsadresse ist im Register Tabellenindex abgelegt.

Index auf Seitentabelle = Tabellenindex + Tabellenbasis

Adresse Operand = physik. Adresse der Seite + Offset

9.2 Beschleunigung der Adressumsetzung

Informatik

Zugriff auf Seitentabelle im physikalischen Speicher und anschließender Zugriff auf die physikalische Speicherzelle:

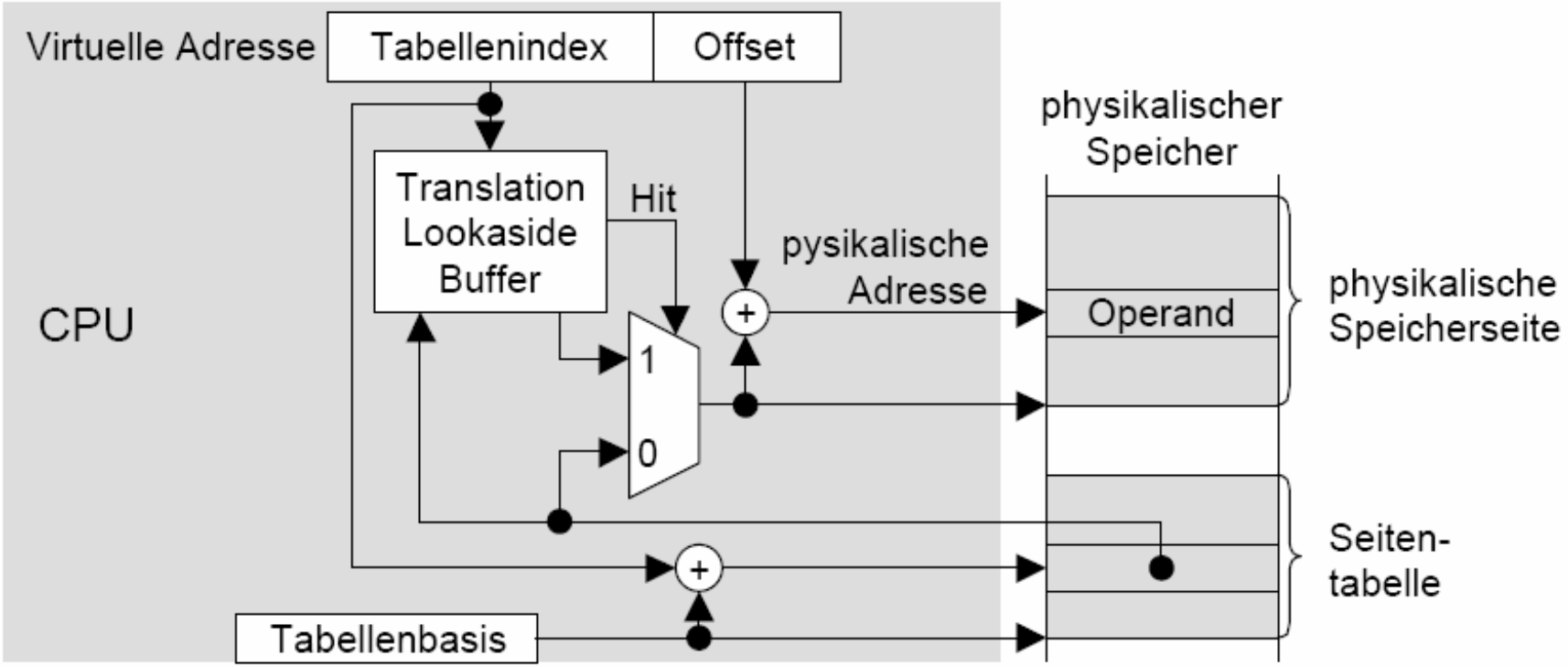
→langsam!

Lösung: Hardware-Erweiterung

Assoziativspeicher, in dem **Paare** aus Tabellenindex und zugehöriger Basisadresse der physikalischen Speicherseite aus vorangegangenen Zugriffen gespeichert sind.

→**Translation Lookaside Buffer** (TLB)

9.2 Translation-Lookaside-Buffer (TLB)

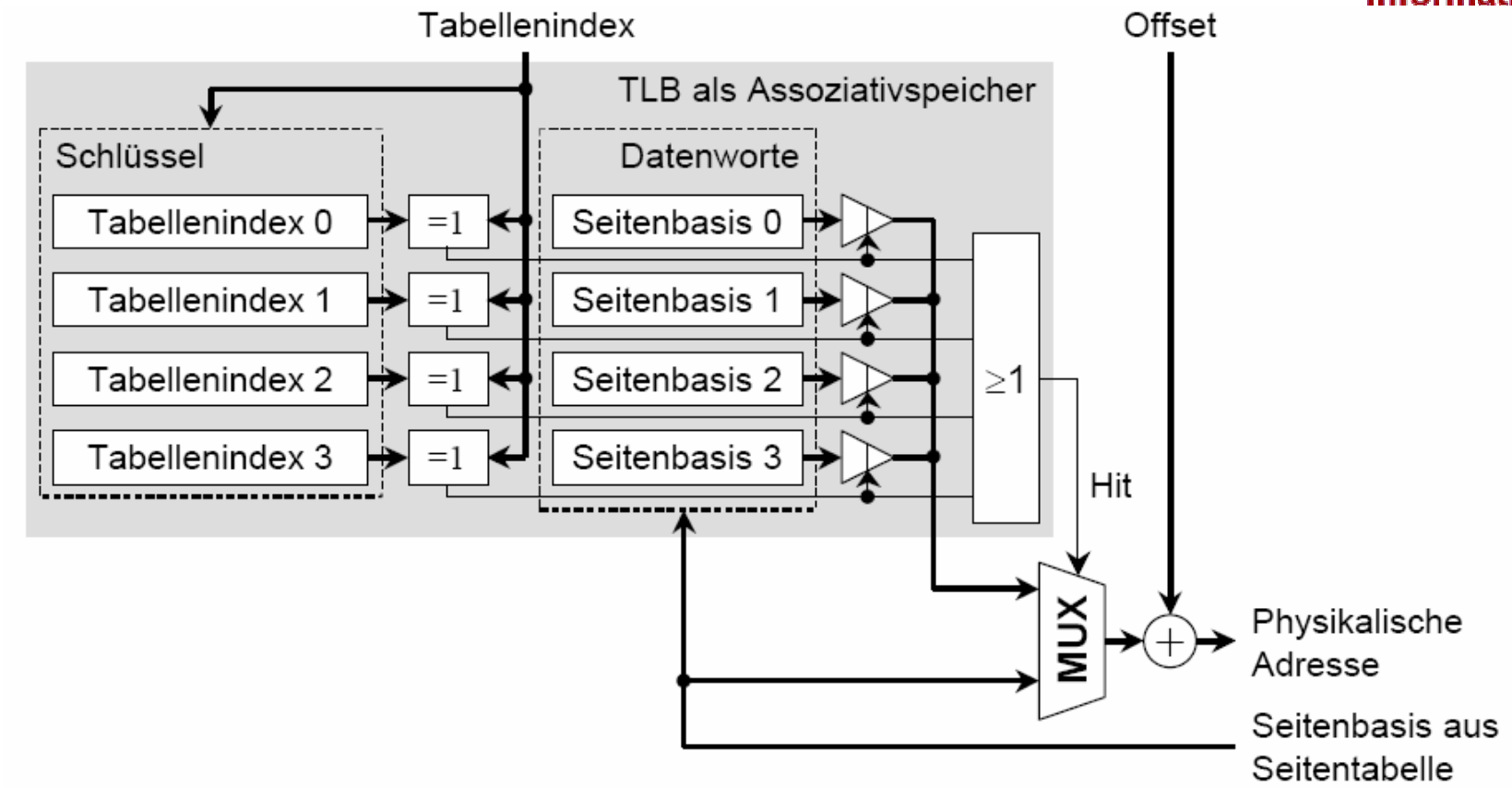


Virtuelle Adresse wird an TLB angelegt:

- Für Tabellenindex ist Eintrag vorhanden → Hit.
- Kein Eintrag: Basisadresse für physik. Speicherseite wird aus Seitentabelle beschafft und im TLB eingetragen.

In beiden Fällen: Physik. Adresse = Basisadresse + Offset

9.2 Translation-Lookaside-Buffer (TLB) (2)



TLB enthält aktuellen Tabellenindex und Seitenbasis.

Fehlt der Eintrag, erfolgt Zugriff auf Seitentabelle.

9 Virtuelle Adressierung (Zusammenf.)

Virtuelle Adressierung erlaubt, einen großen Adressraum mit kleinerem physikalischem Speicher flexibel zu realisieren.

Nur ein Teil der virtuellen Seiten wird im physikalischen Speicher gehalten.

Nicht benötigte Seiten werden auf **Festplatte** ausgelagert (Page-, Swap-Datei). Ersetzung erfolgt nach **L**east **R**ecently **U**sed, FiFo oder anderen Mechanismen.

Benötigte aber **nicht** vorhandene Seiten werden von der Festplatte nachgeladen (Paging) und ersetzen „ältere“ Seiten.

12. Rechenwerke

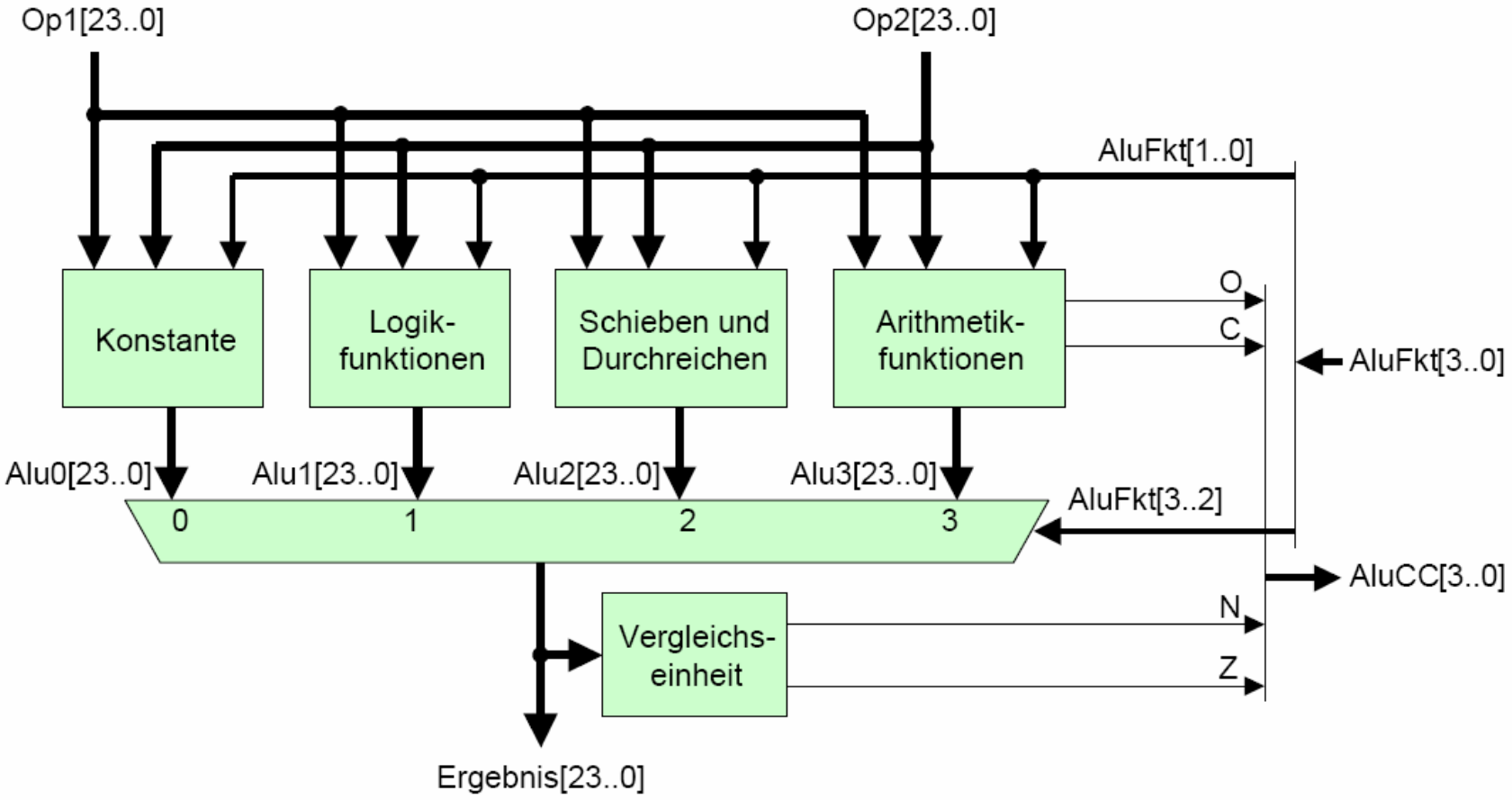
War früher teils Bestandteil der DT-Vorlesung.
(siehe auch Skript Digitaltechnik I)

- Arithmetisch-Logische Einheit (ALU)
  - Aufgaben und Aufbau der ALU (Wiederholung)
  - Funktionsblöcke der ALU (Wiederholung und Details)
- Barrel-Shifter (Schiebeoperationen) mit MUX
- Addition, Multiplikation (Beispiel)

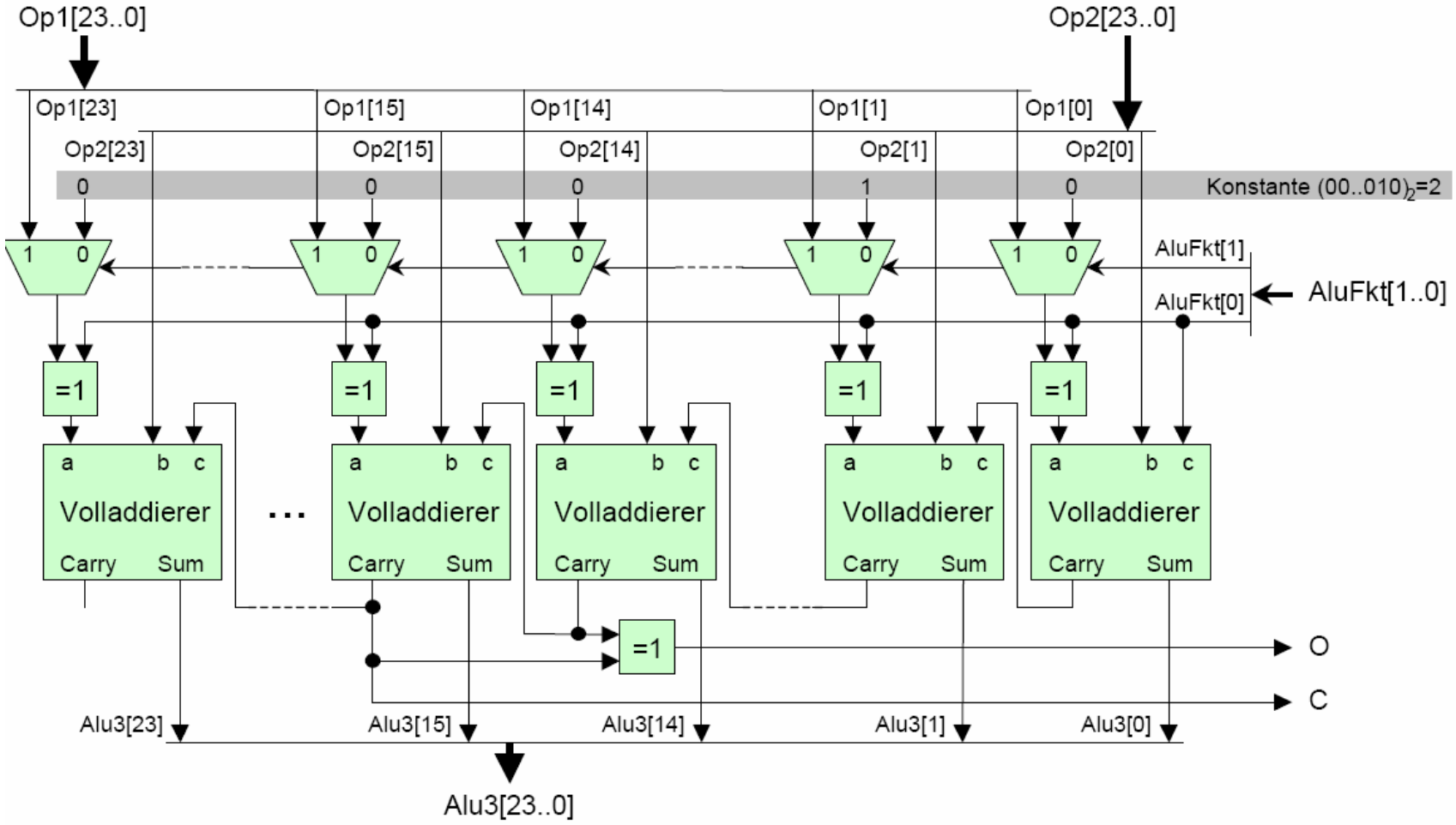
12.1 Eigenschaften einer ALU

- ALU unterstützt die Daten und Adresswortbreite (16/24 zum Beispiel)
  - Arithmetik (Addition)
  - Logik (AND, OR, NOT)
  - Konstanten generieren
  - Schieben und Durchreichen ferner
  - Komplementbildung
  - Flags erzeugen (Overflow, Carry, Negative, Null)

12.1 ALU Funktionsblöcke



12.1 Arithmetikfunktionen

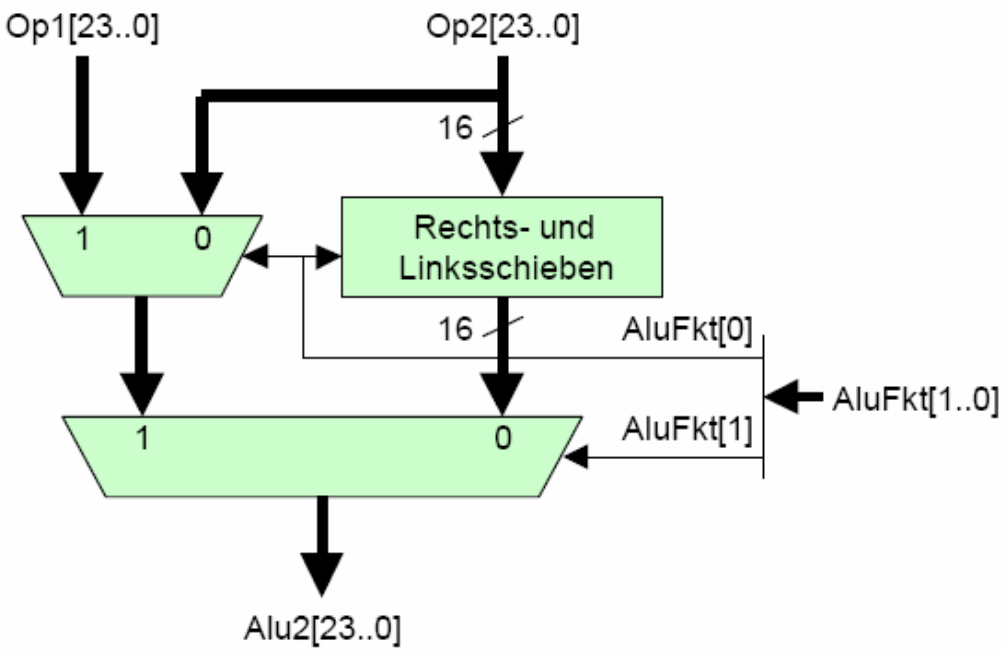


- Konstante 2 addieren für Adressberechnung
- Addition/Subtraktion

Aus 16 Bit:

- Carry
- Overflow

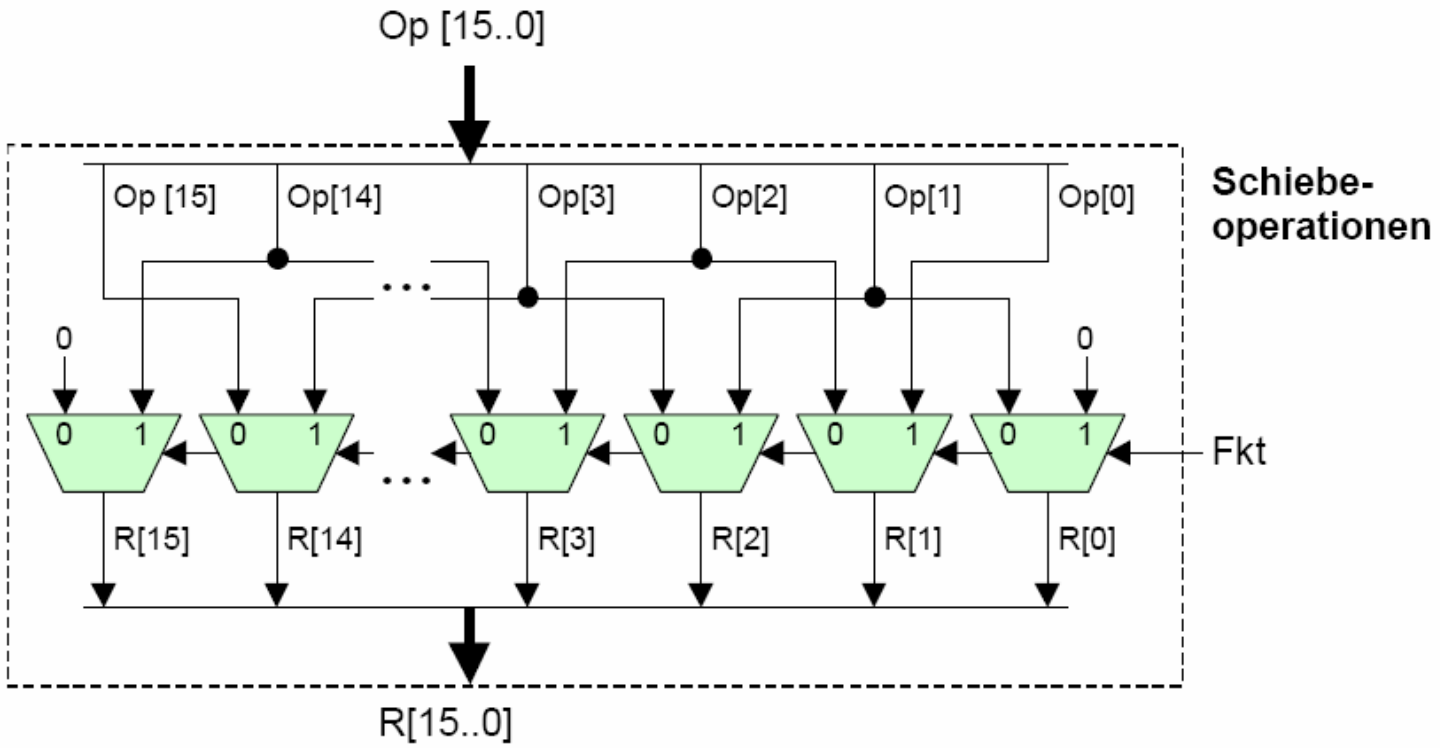
12.1 Schieben und Durchreichen



Zum einen kann in diesem Block der Operand 2 nach rechts und nach links geschoben werden. Dies belegt zwei Funktionen der vier Funktionen des Blocks.

Zusätzlich ist in diesem Block ein Durchreichen der Operanden 1 und 2 durch die ALU realisiert.

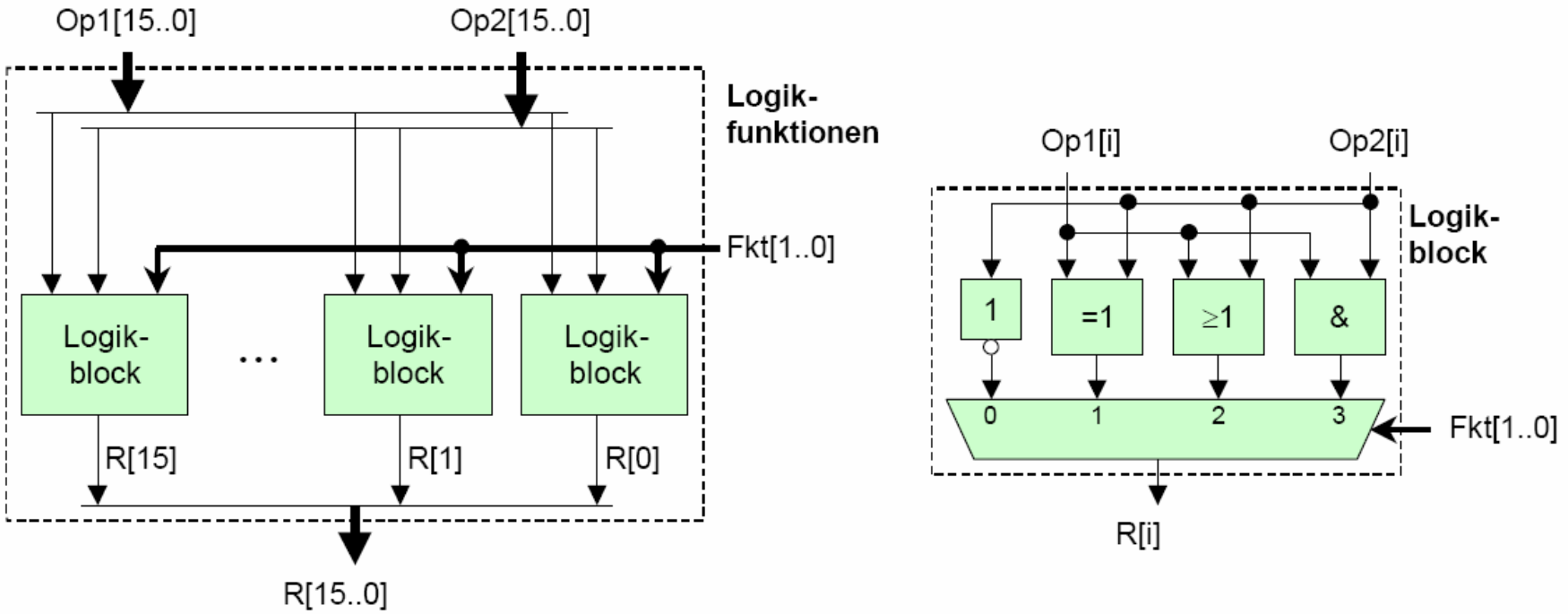
12.1 Schiebe-Operation (s.a. DT 1)



Grundsätzlich muss zuerst entschieden werden, ob der Rechner Schiebeoperationen in einem oder mehrerer Schritten durchführen soll.

Oben: „Barrel-Shifter“, der um eine Stelle nach rechts oder links verschieben kann.

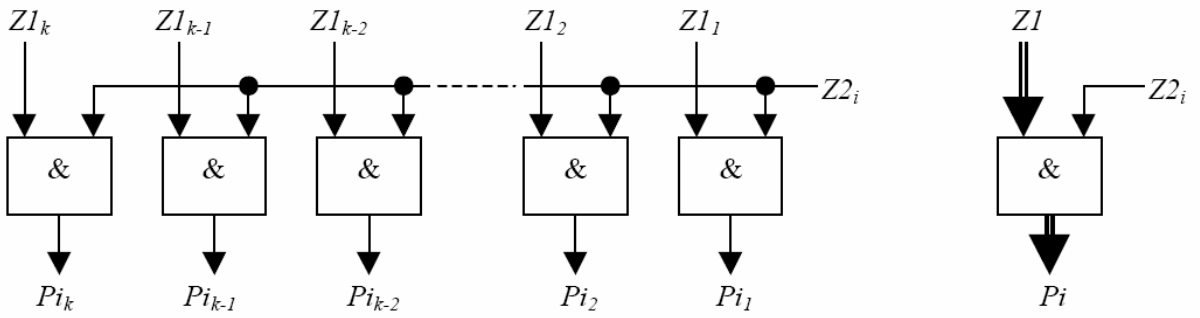
12.1 Bitweise Logische Verknüpfung



Zwei Operanden Op1 und Op2 mit beispielsweise 16 Bit sollen bitweise miteinander verknüpft werden.

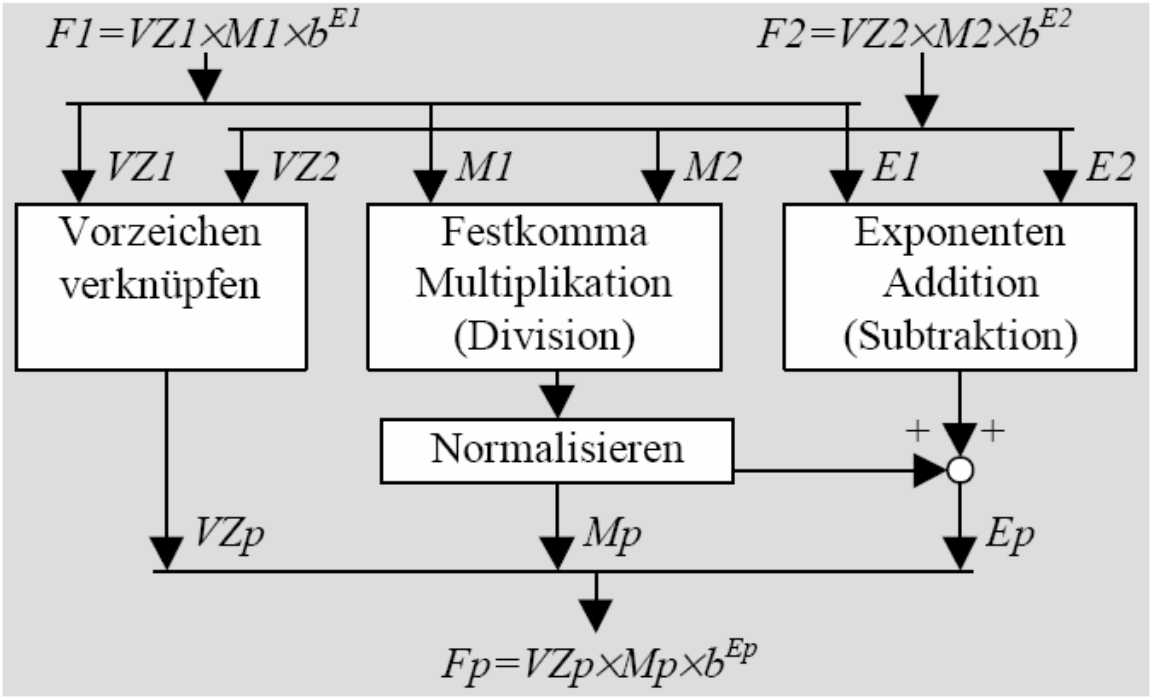
Die Innenschaltung für die Verknüpfung zweier Einzelbits in einem Logik-Block ist im rechten Ausschnitt dargestellt.

12.2 Multiplikation (Gleitkomma = „Float“)




```
0 · 0 = 0
0 · 1 = 0
1 · 0 = 0
1 · 1 = 1
```

Multiplikand · Multiplikator = Produkt



Multiplikand mit einzelnen Stellen des Multiplikators malnehmen

Wiederholung Gleit-Fließkomma-Zahlen:

... HS-Übung ...

12.2 Addition im BCD-Code (Beispiel)

numerisches Alphabet	Tetradendarstellung			
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
Pseudo-tetraden .	1	0	1	0
.	1	0	1	1
.	1	1	0	0
.	1	1	0	1
.	1	1	1	0
.	1	1	1	1

Übergelaufene Tetrade muss durch Addition von 6 korrigiert werden:

- Wenn ein Übertrag erfolgt
- Das „Zwischenergebnis“ eine Pseudoterade ist

Beispiel:

mit Übertrag		mit Pseudotetrade		
8_{10}	1000	7_{10}	0111	
+ 9_{10}	1001	+ 6_{10}	0110	
= 17_{10}	0001 0001	= 13_{10}	1101	
+	0110	+	0110	<-- Korrektur (+6)
	0001 0111		0001 0011	<-- Endergebnis