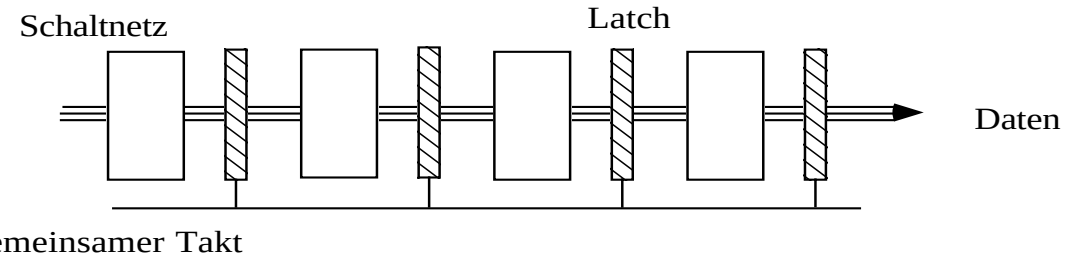
Kapitel 3

Pipelines und Vektorrechner

3.1. Arithmetische Pipelines

3.1.1. Prinzip einer Pipeline

Es wird eine Aufgabe, die mehrfach für verschiedene Daten durchzuführen ist, aufgeteilt in Teilaufgaben, die durch ein Schaltnetz erbringbar sind. Die Ergebnisse werden in Zwischenpuffern (Latches) aufgefangen und an die Folgestufe weitergereicht.



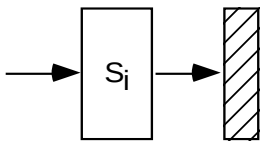
Die Latches werden gemeinsam getaktet, und während ein Datum die Stufe i der Pipeline erreicht hat, kann in Stufe i-1 schon das nächste Datum bearbeitet werden.

3.1.1.1. Zeitverhalten

Für die Ermittlung der Taktperiode gilt das **Geleitzugprinzip**: das langsamste Schiff bestimmt die Geschwindigkeit.

Sei τ_L die nötige Laufzeit in einem Flip-Flop (= $L \cdot \tau_g$, τ_g = Gatterlaufzeit, auf dem Chip ≈ 1 ns) und τ_{si} die Laufzeit durch das Schaltnetz der i-ten Stufe.

Sei $\tau_s = \max_{i=1,...,n} (\tau_{si})$



$\tau_i = \tau_{Si} + \tau_L \qquad \tau \geq \max(\tau_i)$

Dann ist $\tau = (\tau_s + \tau_L)$ die Taktperiodendauer.

Typische Werte liegen bei rund 10 ns auf dem Chip; bei Clockfrequenzen von 400 MHz, untersetzt auf 200 MHz, sind die Flip-Flops getaktet mit 5 ns, das schließt die Durchlaufzeiten durch die rechnenden Schaltnetze mit ein (bei Gatterlaufzeiten von weniger als 1 ns).

Bei n Stufen braucht ein Datum zum Durchlauf durch die Pipeline

$$\sum_{i=1}^{n} \cdot \tau = n \left(\max_{i=1,\ldots,n} (\tau_{si}) + \tau_L\right)$$

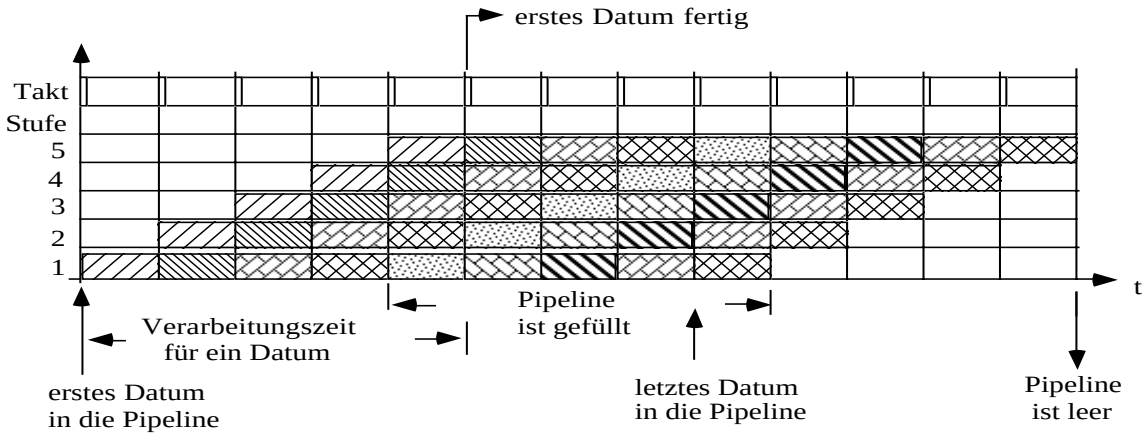
Das kontrastiert mit $\tau_{normal} = \sum_{i=1}^{n} \tau_{si}$ für ein entsprechendes Schaltnetz ohne Zwischenspeicher.

Der Overhead sind $n \cdot \tau_L$.

Eine Pipeline verarbeite $n + m$ Daten hintereinander.

Dann ist die Gesamtdauer

- $n \cdot \tau$ Schritte zum Füllen der Pipeline
  dann erscheint das erste Datum am Ausgang
- $m \cdot \tau$ Schritte zum Abarbeiten von m Daten
- $(n-1) \cdot \tau$ Schritte zum Entleeren der Pipeline



Es ist $\frac{\text{Gesamtdauer}}{\text{Anzahl Daten}} = \frac{(n + m + (n - 1)) \cdot \tau}{n + m} = \left(1 + \frac{n - 1}{n + m}\right) \cdot \tau = \frac{\text{Dauer}}{\text{Datum}}$

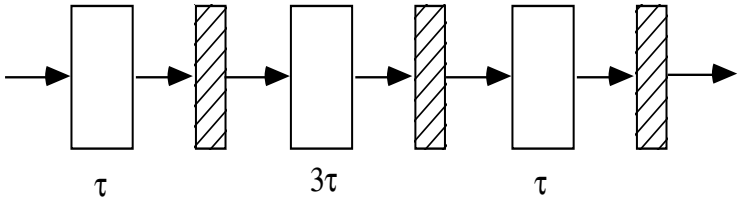
Rechnet man in Takteinheiten, dann ist

$1 + \frac{n-1}{n+m}$ = Anzahl der Takte für ein Datum .

Für große Werte von m wird der Quotient klein gegen Eins. Die Abarbeitungszeit für m Daten mit Pipeline ist dann $(1 + \varepsilon) \cdot m \cdot \tau$. Ohne Pipeline dauerte der Vorgang $n \cdot m \cdot \tau_{si}$.

3.1.1.2. Angleichen der Geschwindigkeit

Sei eine Pipeline mit einem "langsamen Schiff" gegeben. Sei $(\tau_{si} - \tau_L) = 3\,\tau$ dann kann die Taktung schneller gemacht werden durch



a) Aufteilung des langsamen Schaltnetzes in k schnelle Schaltnetze mit Taktperiode τ, es gelte im o. g. Beispiel
$3\,\tau = (\tau_{si} + \tau_L)$; es soll gelten

$$\tau = \frac{\tau_{si}}{k} + \tau_L$$

Einsetzen folgt

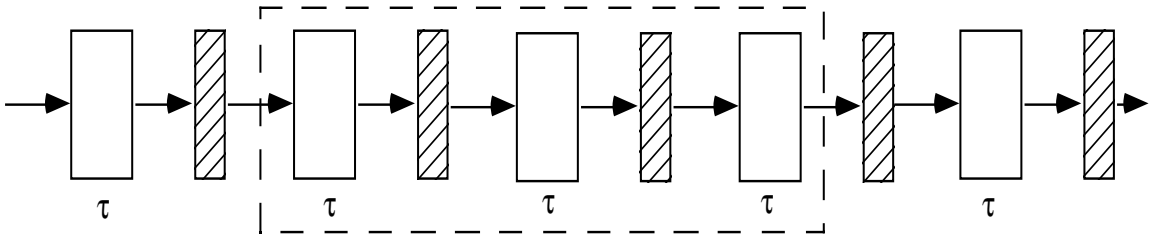
$$3 \cdot \frac{\tau_{si}}{k} + 3\,\tau_L = \tau_{si} + \tau_L$$

$$k = \frac{3\,\tau_{si}}{\tau_{si} - 2\,\tau_L} \text{ ; sei } \tau_{si} = 3\,\tau_L$$

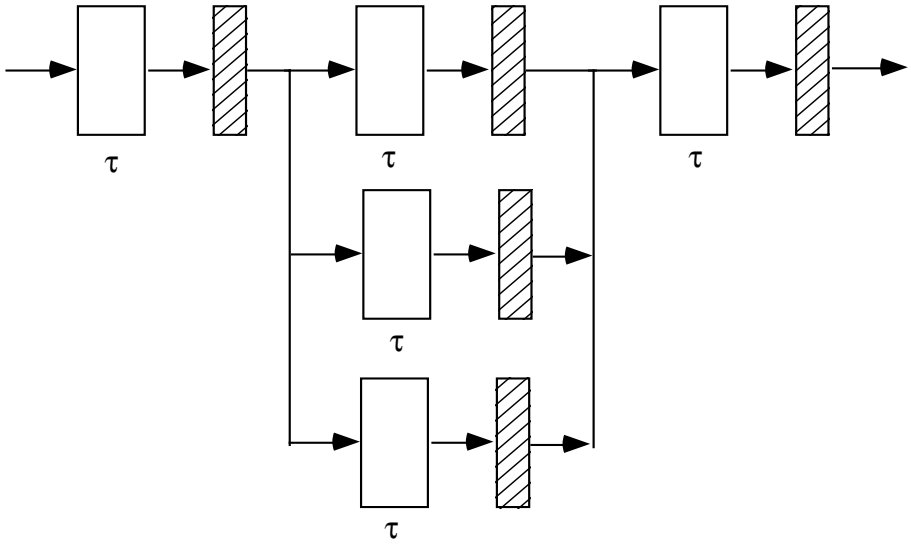
$$\Rightarrow \quad k = 3$$

$$\text{sei } \tau_{si} = 10\,\tau_L$$

$$= k = \frac{30}{8} \cdot \rightarrow 4$$



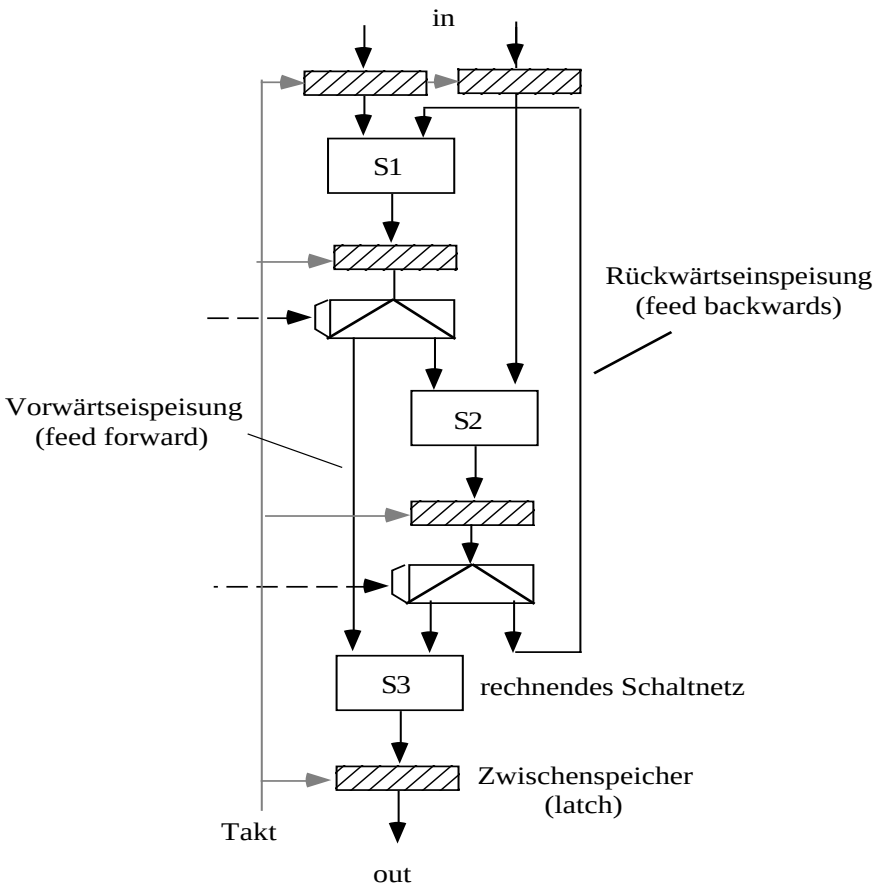
b) Parallelschaltung
Die Pipeline wird bei der langsamen Stelle aufgeteilt in drei parallele Ströme, jeder braucht nur τ_L um sein Ergebnis zu rechnen.
Die Teilergebnisse werden zusammengeführt.



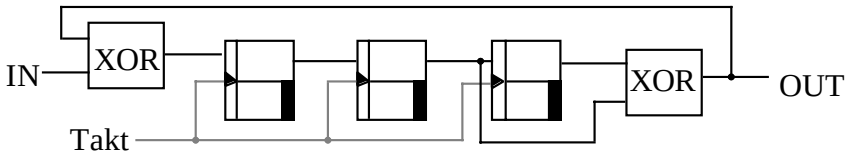
Voraussetzung ist die Datenunabhängigkeit der Teilströme.

3.1.1.3. Pipeline mit Belegungsplan

In einer Pipeline kann ein Datum an Stufen vorbei- (feed forwards) oder zurückgereicht (feed backwards) werden. Man spricht auch von Vorwärts- und Rückwärtseinspeisung.

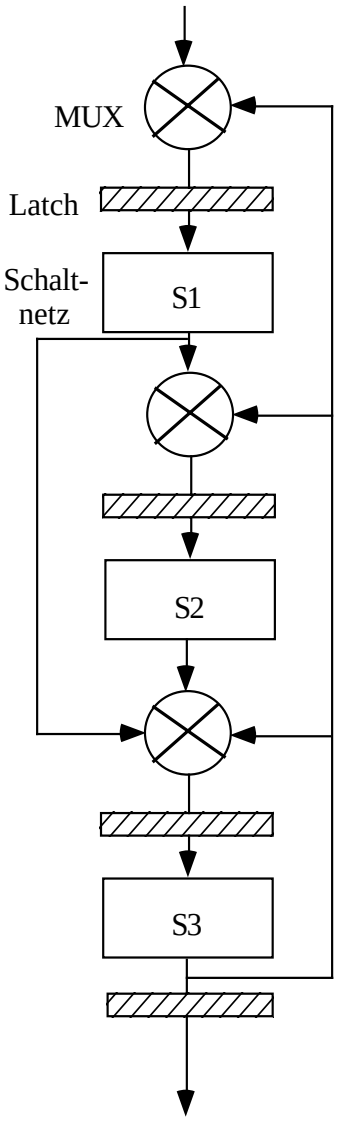


In dieses Bild paßt hinein z. B. die Schaltung für die Generierung eines CRC-Checks mit dem Generierungspolynom $G(n) = n^3 + n^2 + 1$.



I. allg. kann nicht davon ausgegangen werden, daß die Pipeline immer gefüllt ist.

Das nächste Bild zeigt eine dreistufige Pipeline mit der Möglichkeit von Vorwärts- und Rückwärtseinspeisungen.



Reservierungstafeln

	t0	t1	t2	t3	t4	t5	t6	t7
S1	A			A			A	
S2		A						A
S3			A		A	A		

S1	B				B			
S2			B			B		
S3		B		B			B	

Dazu sind zwei Reservierungstafeln gezeigt, die deutlich machen, daß in beiden Fällen A und B verschiedene Daten nicht hintereinander bearbeitet werden können: Die Pipeline wird blockiert für weitere Daten bis nach 7 bzw. 6 Schritten die Abarbeitung fertig ist.

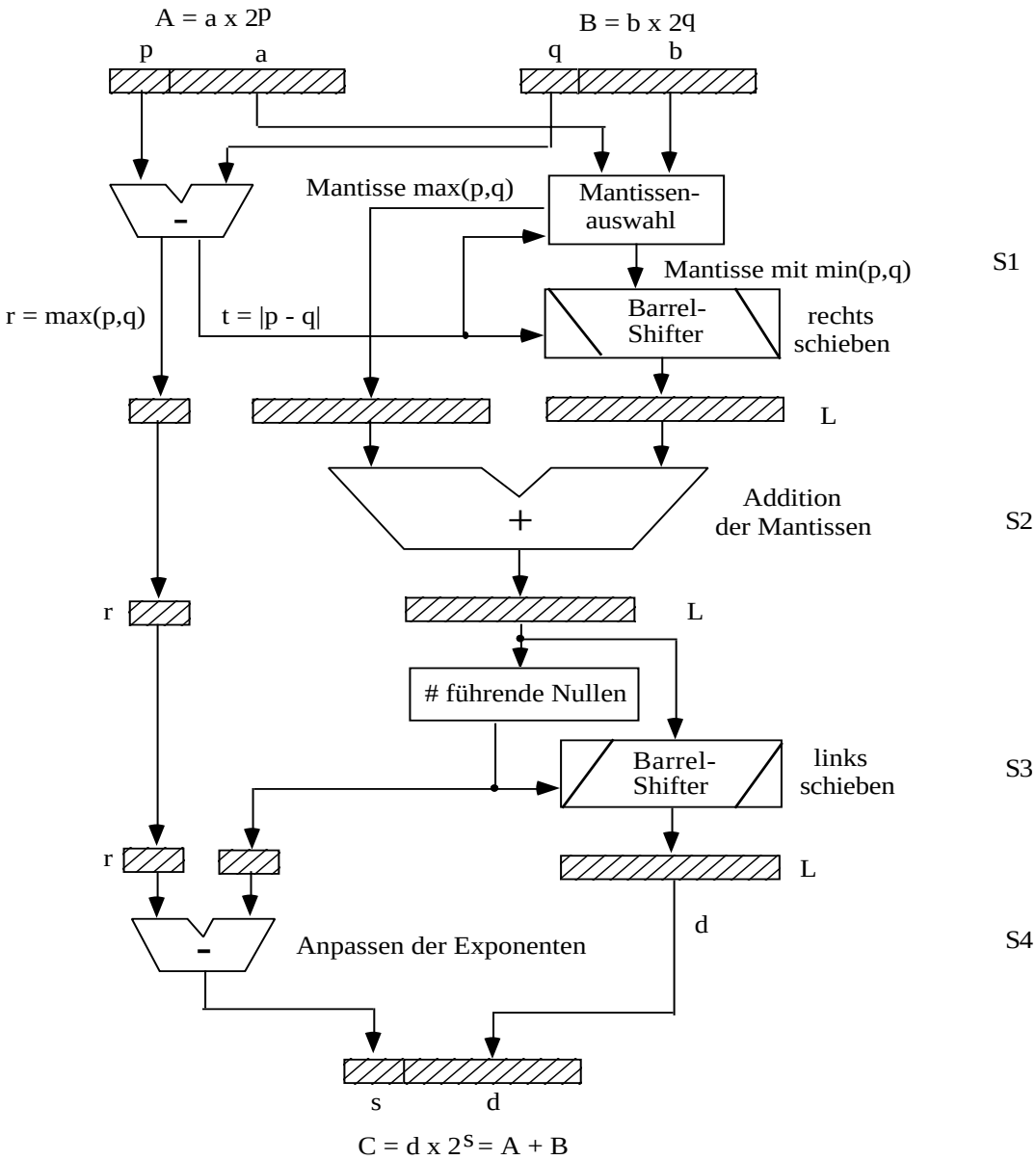
Dieser ungünstige Fall sollte vermieden werden.

3.1.2. Ein Schaltwerk als Pipeline

Die in einem Schritt eines Schaltwerkes parallel ausrechenbaren Daten werden in Latches festgehalten, und an die rechnenden Einheiten des folgenden Schrittes übergeben.

Es mögen keine Schleifen in dem Schaltwerk programmiert sein.

Ein Beispiel ist ein Gleitpunktaddierwerk.



Gleitkommaaddierwerk mit vierstufiger Pipeline

Es soll Zahlen $A = a \cdot 2^p$ und $B = b \cdot 2^q$ addieren.

Die Addition läßt sich in vier Schritten beschreiben:

1. Subtraktion der Exponenten und Herausfinden des größeren Exponenten.
   Schieben der Mantisse beim kleineren Exponenten um die Differenz der Exponenten nach rechts.
   Übernahme von $r = \max(p, q)$ und der größeren Mantisse und geschobenen Mantisse in einen Zwischenspeicher.
2. Addition der Mantissen.
   Übernahme von $r = \max(p, q)$ und der Summe der Mantissen in einen Zwischenspeicher.
3. Ermittlung der Zahl der führenden Nullen (insbesondere wichtig bei Subtraktion mit Auslöschung).
   Ansteuerung eines Barrelschifters mit dieser Zahl und Links-Schieben der Mantisse.
   Übernahme der Zahl, der Mantisse und von r in einen Zwischenspeicher.
4. Subtraktion der Zahl der führenden Nullen von r und Übernahme des Exponenten und der normalisierten Mantisse in den Ergebnispuffer.

3.1.3. Aufrollen von Schleifen in einer Pipeline

Sei ein Divisionsalgorithmus gegeben (Division mit Rückstellen).

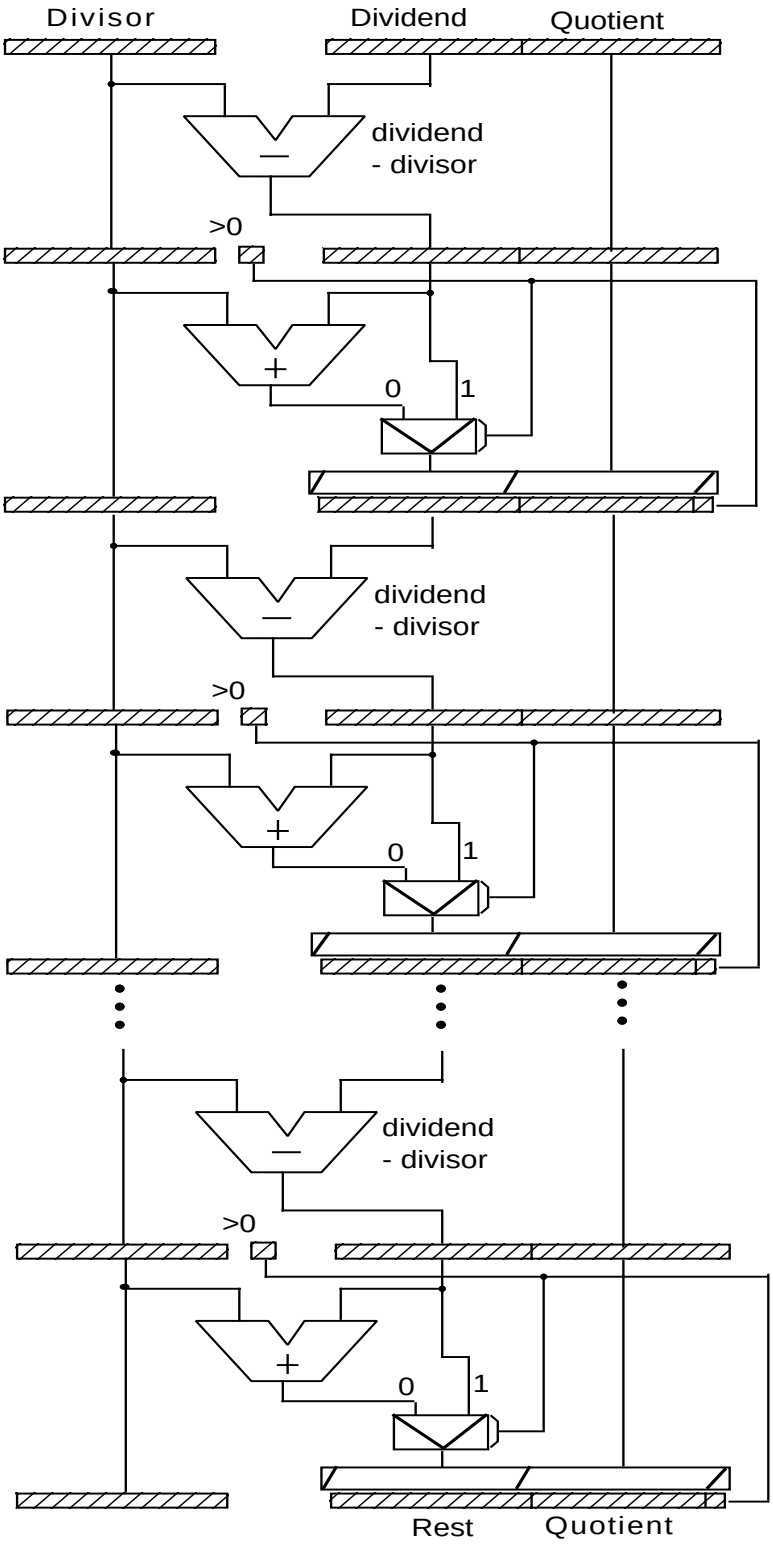
```
        dividend := dividend / 2 ^n;  {* kopiere in den rechten Teil des Registers*}

        i := n - 1;
        repeat
(1)         dividend := dividend - divisor;
            if (dividend > 0)
(2a)            then   q(0) := 1;          {*LSB des Quotienten*}
(2b)            else   dividend := dividend + divisor ; q(0) = 0
(3)     ⎡       Schiebe nach links dividend und Quotient;
        ⎬               i := i - 1;
        ⎣
        until   i < 0
```

Die Schleife wird n-mal durchlaufen (n = Anzahl Bits des Divisors).

Aufrollen heißt

Das Addier-Subtrahierwerk mit Abfrage ist n-mal zu installieren.
Nach jedem Schritt ist ein Zwischenpuffer einzubauen.



Die Pipeline wird damit entsprechend tief.

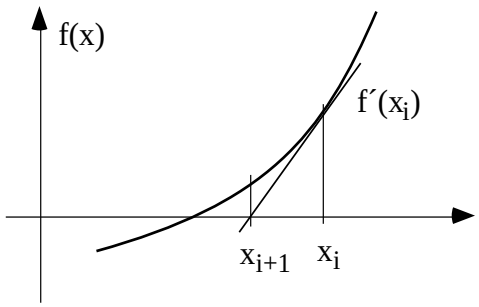
Besser ist es ggf. die Division zurückzuführen von a/b auf a ∗ 1/b und 1/b mit einem schnellen Iterationsalgorithmus (Newton-Verfahren) zu rechnen.

$b \text{ Æ } x \stackrel{!}{=} 1 \quad \Rightarrow \quad b = \frac{1}{x} \quad \Rightarrow \quad b - \frac{1}{x} = 0$

$$f(x) = b - \frac{1}{x} \; ; \quad f'(x) = \frac{1}{x^2}$$

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)} = x_i - \frac{b - \frac{1}{x_i}}{1/x_i^2}$$

$$x_{i+1} = x_i - b \cdot x_i^2 + x_i$$



$$\boxed{x_{i+1} = 2\,x_i - b \cdot x_i^2}$$

quadratisch konvergierendes Verfahren

3.1.4. Aufspaltung eines Schaltnetzes in eine n-stufige Pipeline

Sei die Laufzeit durch ein Schaltnetz $S \cdot \tau_g$.

Es werde aufgeteilt in n Stufen der Laufzeit

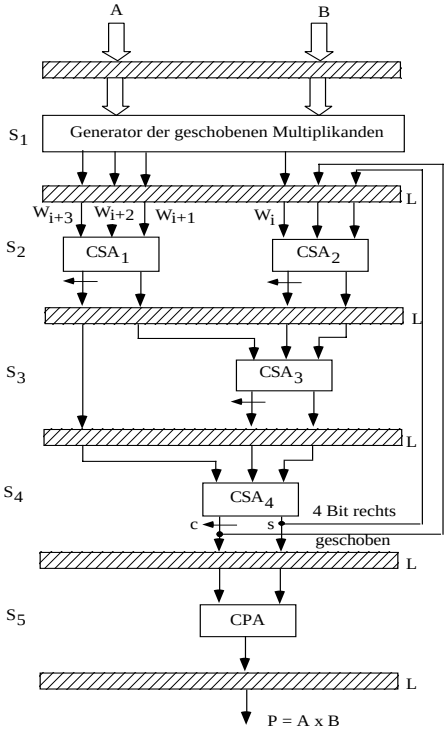
$\tau_n \leq \frac{S}{n} \cdot \tau_g$.

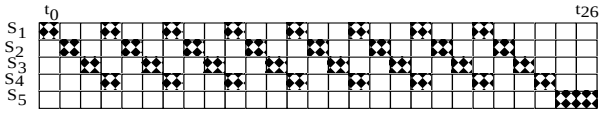
Jede Stufe wird mit einem Latch abgeschlossen. Das Latch fordert $L \cdot \tau_g$ Gatterlaufzeiten. Dann ist die Taktperiode

$\tau = \left(\frac{S}{n} \cdot \tau_g + L \cdot \tau_g \right)$.

Dieses Verfahren ist sinnvoll, wenn Vektoren zu bearbeiten sind.

Beispiel:
Ein Festpunktmultiplizierwerk mit Berechnung der Teilprodukte und schrittweiser Addition in carry-save-Addieren (CSA)

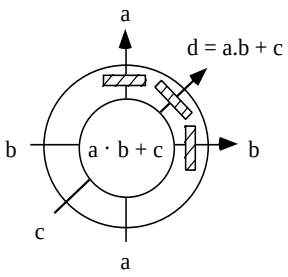




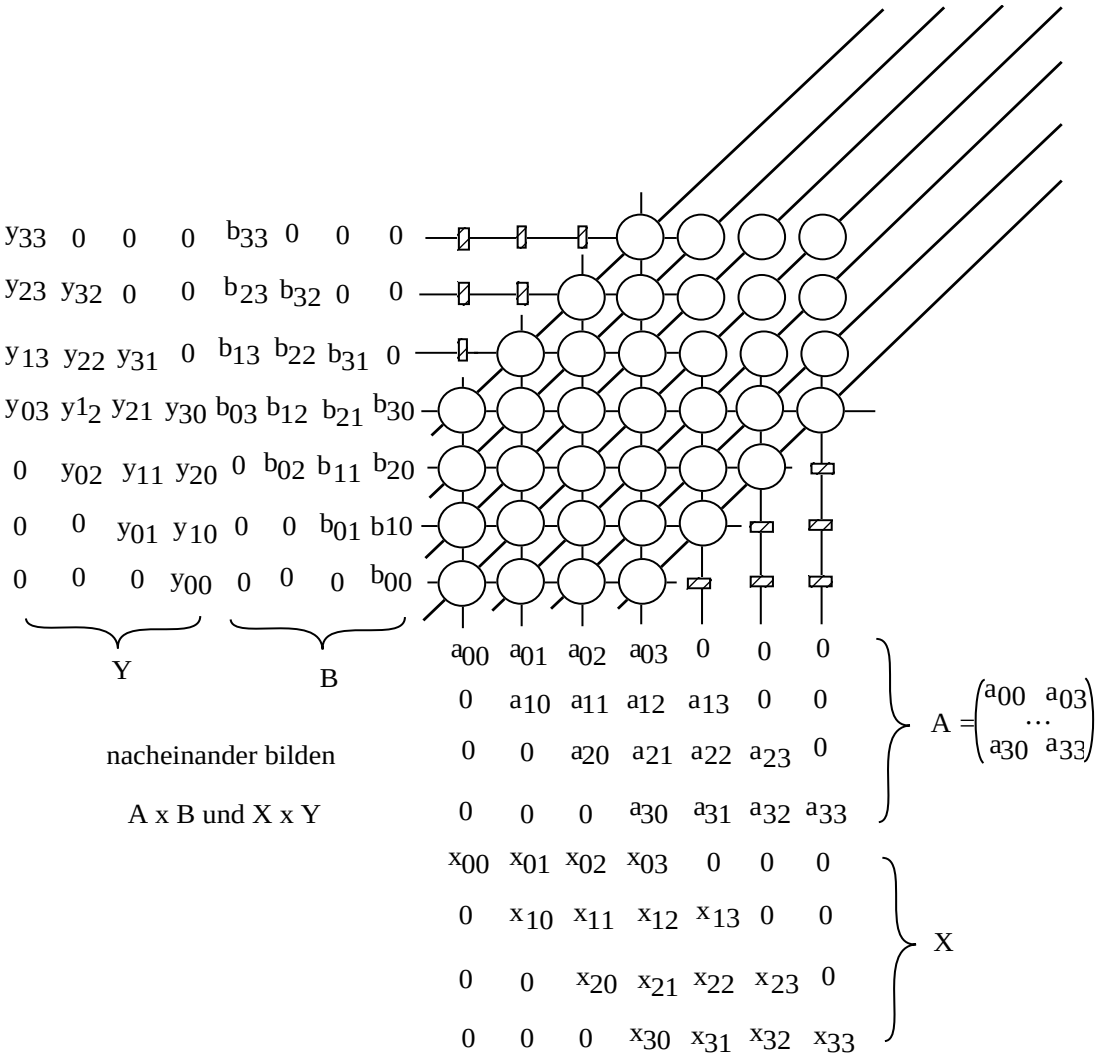
Reservierungstafel für einen 32-Bit- Pipeline-Multiplizierer
(4-Bit Parallelverarbeitung)

3.1.5. Matrix-Array-Multiplizierer

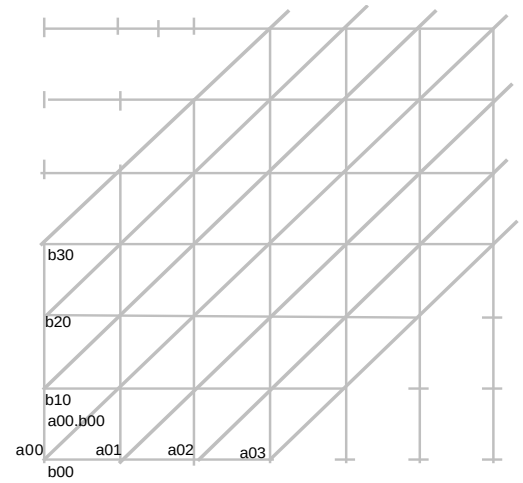
Schließt man an das Grundelement eines Matrixmultiplizierers den Zwischenspeicher einer Pipeline an, dann kann man eine zweidimensionale Pipeline realisieren zur Matrixmultiplikation. Das folgende Bild zeigt den elementaren Baustein.



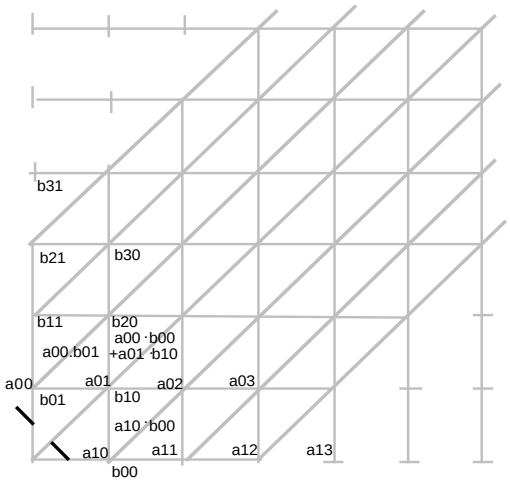
Eine Zusammenschaltung in einer Rautenform erlaubt die Multiplikation von Matrizen.



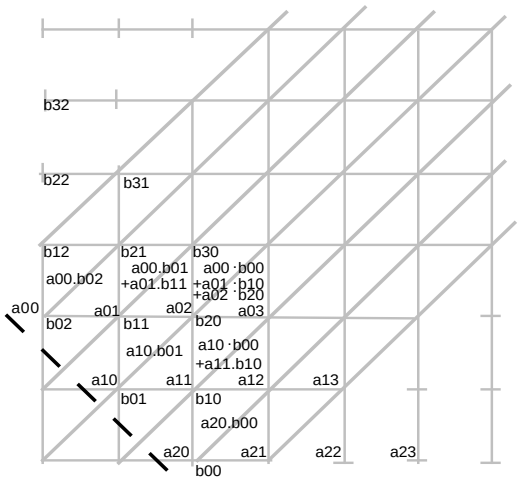
Die Hintereinanderausführung von zwei Matrixmultiplikationen A x B und X x Y zeigen die nächsten Abbildungen am Beispiel von 4 x 4 - Matrizen.



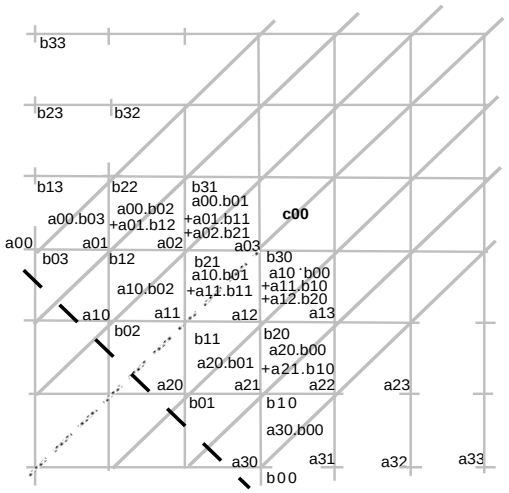
t = 1



t = 2

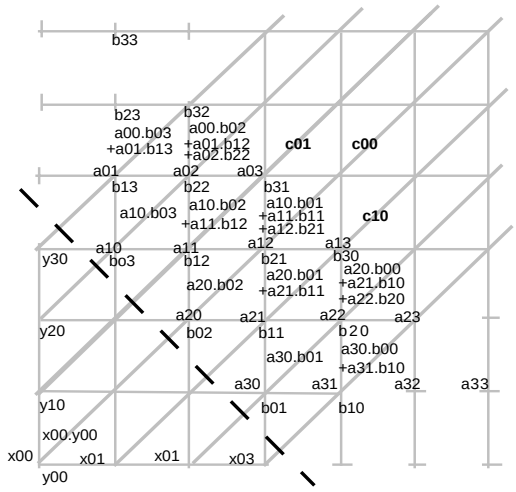


t = 3

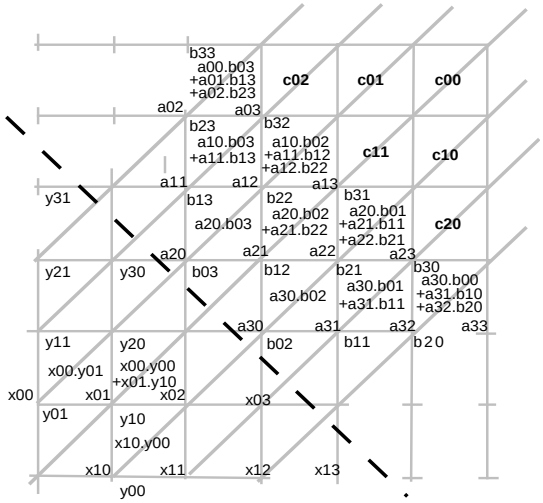


t = 4

Alle Komponenten
von A und B sind eingelesen



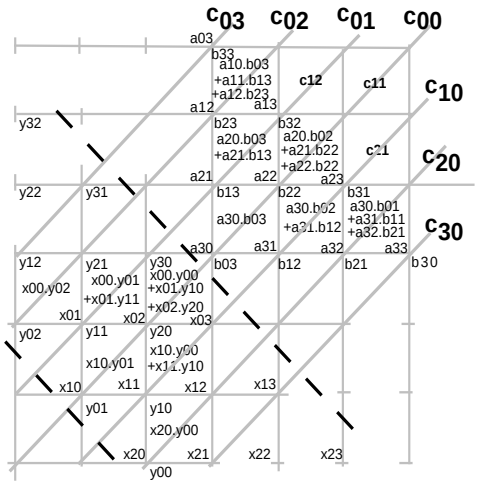
t = 5



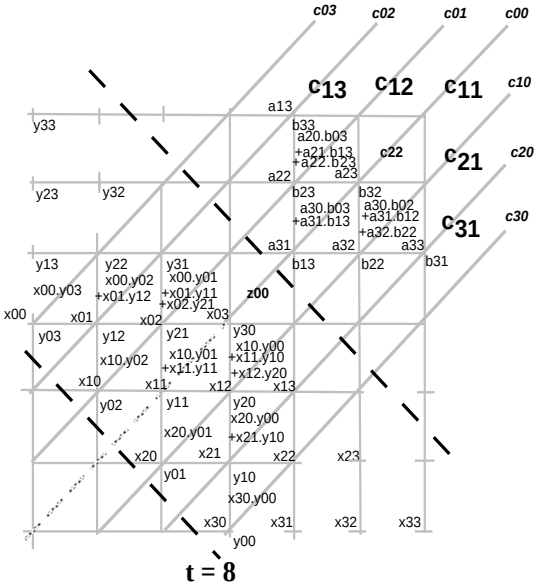
t = 6

das erste Vektorpaar
von X,Y wird verarbeitet

Die ersten Komponenten
von A x B erscheinen am Ausgang

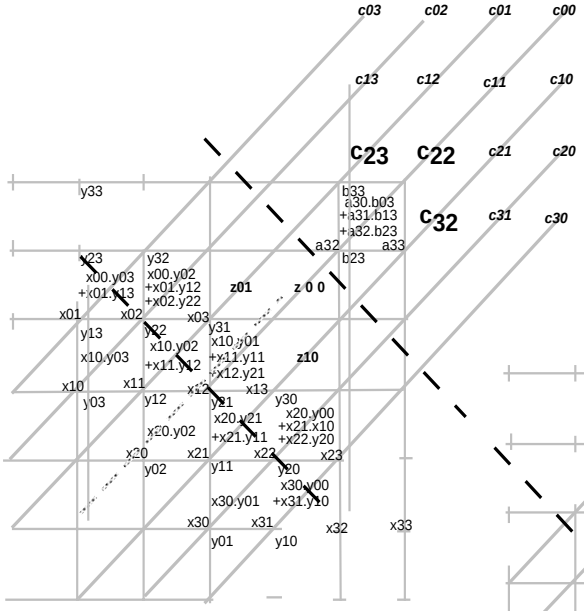


t = 7

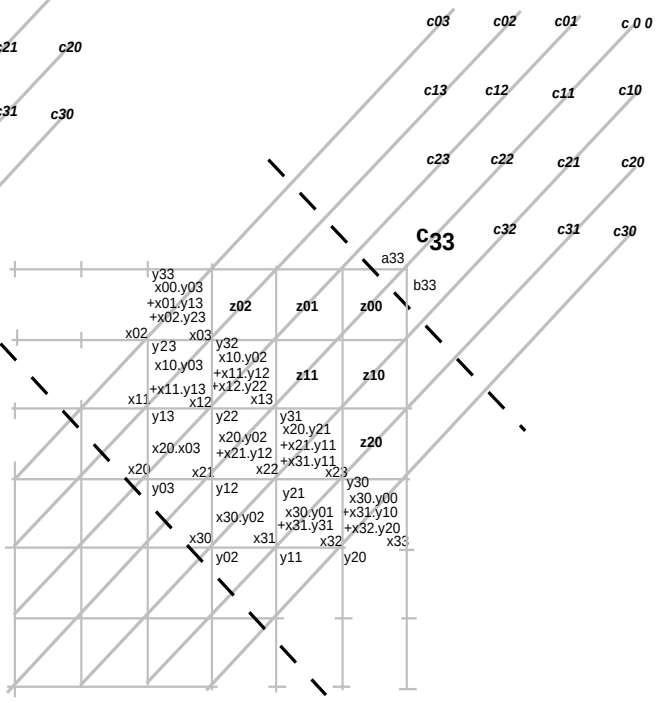


t = 8

Alle Komponenten
von X und Y sind eingelesen



t = 9



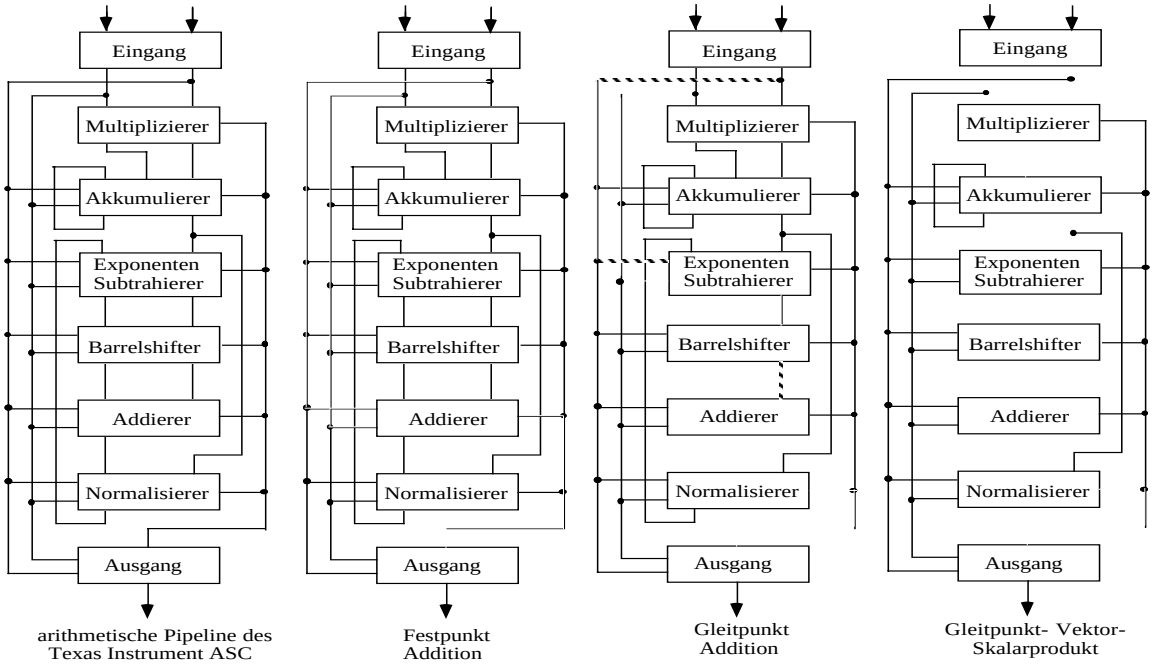
t = 10

Alle Komponenten des
Matrixproduktes A x B
sind bestimmt

Nach einer Füllphase von 2 n Takten bei einer n x n - Matrix werden mit jedem Takt Komponenten der Produktmatrix ausgerechnet; allerdings unterschiedlich viele mit einem Takt.

3.1.6. Multifunktionspipelines

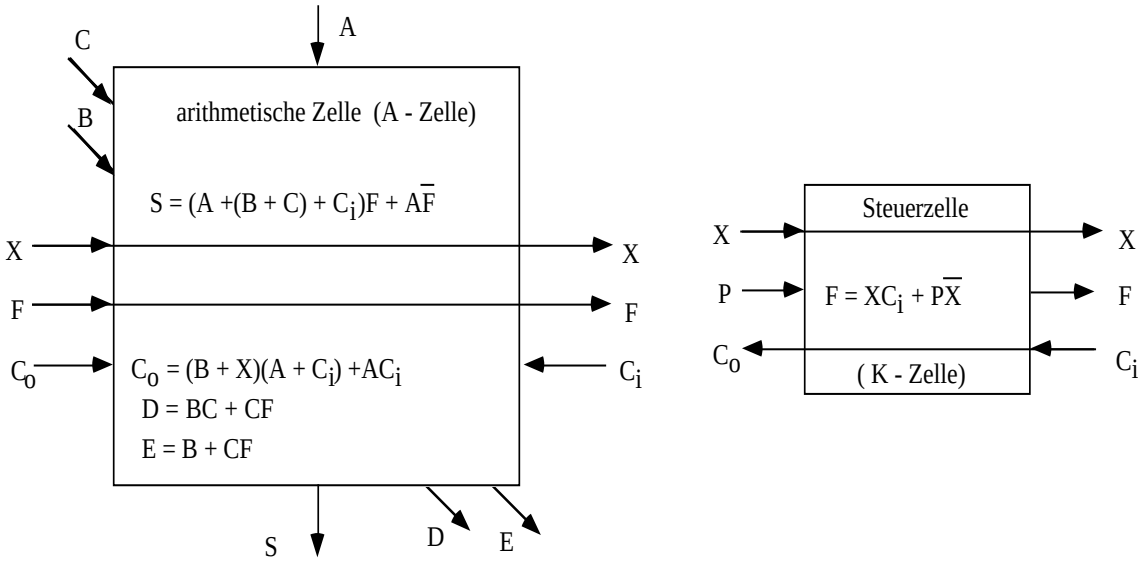
Solange Hardware teuer war lohnte es sich, Pipelines mehrfach verwendbar zu gestalten. Ein Beispiel zeigt die Multifunktionspipeline des Texas-Instruments ASC (advanced scientific computer, 1972).



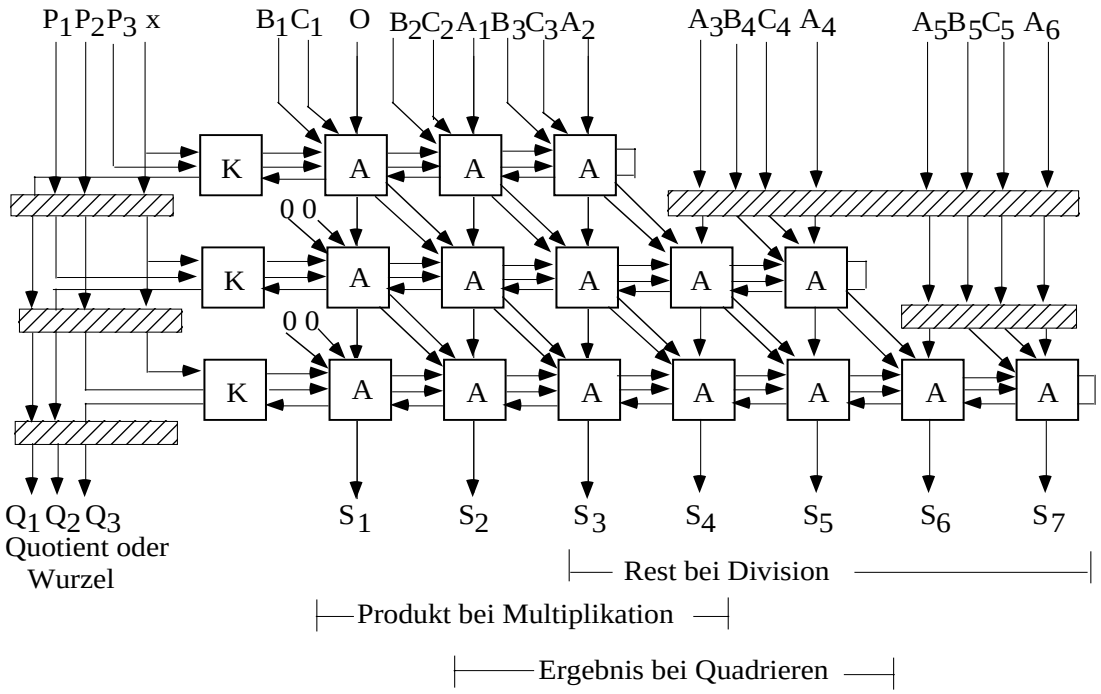
Je nach Durchschaltung kann die Pipeline Fluss- und Festkommazahlen verarbeiten.

Für besondere Zwecke lohnen sich Multifunktionsarrays, mit denen sich Quotienten und Wurzeln berechnen lassen.

Das folgende Bild zeigt ein Beispiel eines solchen Arrays, der aus 1-Bit- Rechenelementen (A-Zellen) und Steuerzellen (K-Zellen) besteht.



Eine Matrix zeigt dann das folgende Bild.



3.2. Vektorrechner

3.2.1. Vektoroperationen

Seien Vektoren $A = (a_1, ..., a_n)$, $B = (b_1, ..., b_n)$ gegeben, dann sind auf ihnen Operationen erklärt:

$V \rightarrow V$ (z. B. VSQRT : $b_i := (a_i)^{1/2}$ oder VCOMP : $b_i := \neg a_i$ oder VSIN : $b_i = \sin a_i$)

$V \times S \rightarrow V$ (c : skalare Größe $c \in S$)
(z. B. $b_i := c \cdot a_i$ oder $b_i = : a_i + c$)
(Streckung) (Translation)

$V \rightarrow S$ (z. B. $c := \sum a_i$ VSUM oder $c := \max (a_i)$ VMAX)

$V \times V \rightarrow S$ (z. B. SMPY : $c := \sum_i a_i \cdot b_i$, Skalarprodukt)

$V \times V \rightarrow V$ (z. B. $c_i := a_i + b_i$: VADD
$c_i := a_i * b_i$: VMPY
$c_i := a_i \wedge b_i$: VAND ; $a_i, b_i \in \mathbb{B}$
$c_i := \max (a_i, b_i)$: VLAR

$$c_i := \begin{cases} 0 \text{ falls } a_i < b_i \\ 1 \text{ sonst} \end{cases} \quad \text{VGTE}$$

Sei $a_i \in$ char ; $b_i \in \mathbb{B}$ (Bitvektor)

Auswahl: `A = (KAISERSLAUTERN)`

$\Rightarrow$ `(AST)`

`B = (01010000001000)`

Vektoroperationen werden durch einige höhere Sprachen wie **APL** unterstützt.

Die Formulierung als Vektoren wird unterstützt durch **vektorisierende Compiler**.

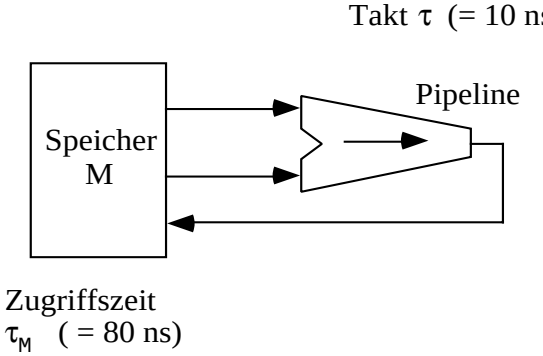
3.2.2. Organisationsformen von Vektorrechnern

Der Zeitaufwand für Operationen mit Vektoren teilt sich auf in zwei Anteile:

- set-up-time:
  Verteilzeit der Operanden auf die Pipelines und Vorbereitung der Vektoren im Speicher, um den Zugriff einfach halten zu können.
- flushing-time:
  Zeit zwischen dem Dekodieren einer Vektoroperation und dem Erscheinen des ersten Resultats am Ausgang der Pipeline.

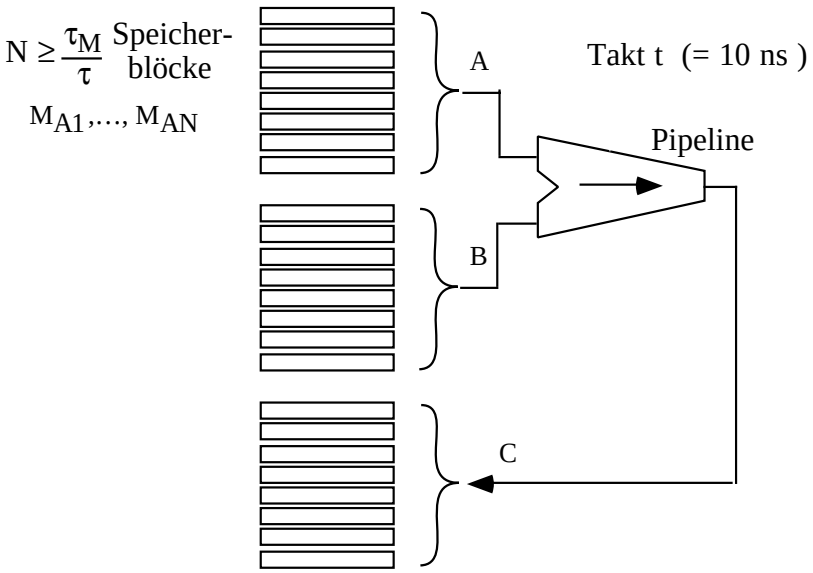
3.2.2.1. Speicher-Speicher-Organisation

Die Vektoren werden aus einem Speicher in die Pipeline gelesen und die Ergebnisse in den Speicher zurückgelesen.



Sei die Taktzeit der Pipeline τ = 10 ns , und die Speicherzugriffszeit 80 ns . Die Wortbreite sei 32 Bit. Dann sind in 10 ns 2 x 4 = 8 Byte zu lesen und 4 Byte zu schreiben , d. h. es sind 1,2 GB/s zu transportieren (es ist dabei nur eine Pipeline angenommen).

Bei einer Zugriffsbreite von 4 Byte sind in 10 ns 3 Worte zwischen Pipeline und Speicher zu transportieren. Das bedingt bei einer Zugriffszeit von 80 ns auf den Speicher eine Aufteilung des Speichers in 3 x 8 Blöcke im Minimum.



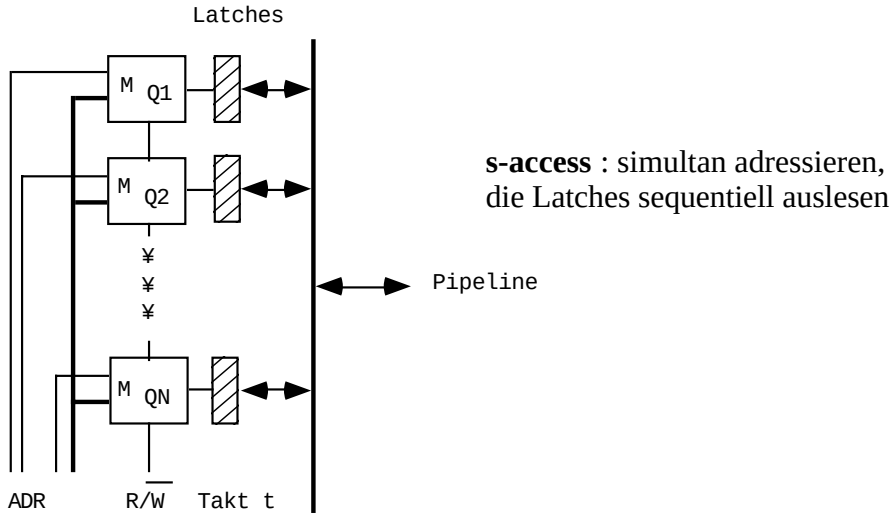
Der Zugriff auf die Speicherblöcke kann auf zwei Weisen erfolgen:

3.2.2.1.1. Simultan-Zugriff (s-access)

Alle 8 Blöcke werden parallel adressiert.

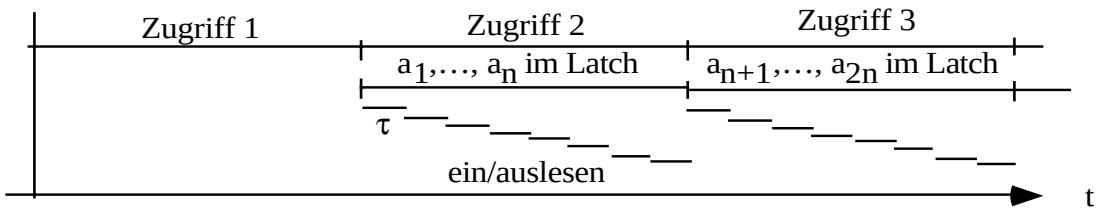
Die höchstwertigen Adressen sind separat dekodiert und sprechen jeweils einen Speicherblock an.

Nach Ablauf der Zugriffszeit werden die Daten in Latches übernommen (Lesen aus dem Speicher) oder aus Latches in die Speicherblöcke übernommen (Schreiben in die Speicherblöcke).



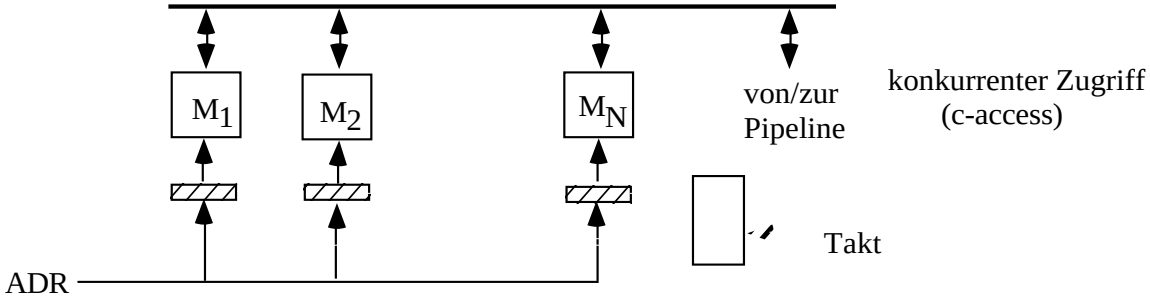
Mit dem Takt τ der Pipeline werden die Daten aus den Latches sequentiell ausgelesen und über einen Bus auf den Eingang der Pipeline gelegt bzw. vom Ausgang der Pipeline in die Latches übernommen.

Während des sequentiellen Auslesens erfolgt der nächste Speicherzugriff.
Die zeitliche Überlappung zeigt das folgende Bild.



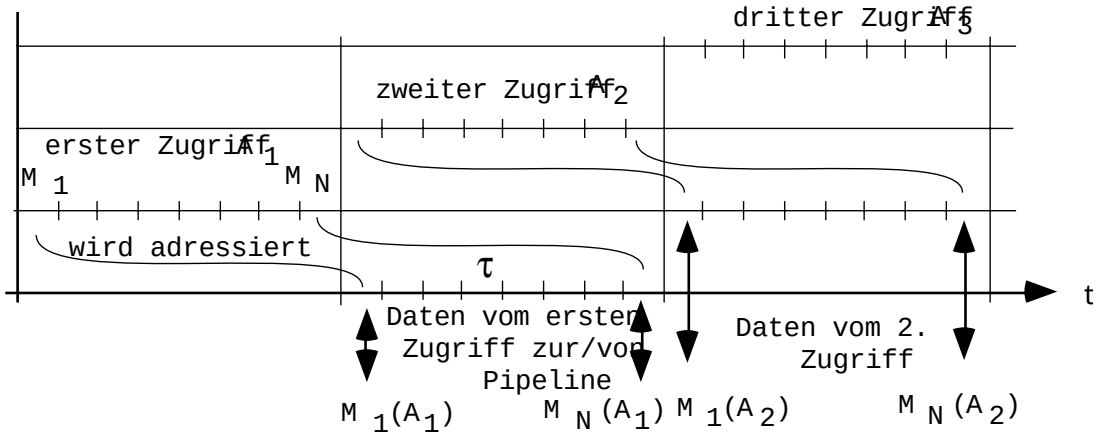
3.2.2.1.2. Konsekutiver Zugriff (c-access)

Hier werden die Adressen der Speicherblöcke in Latches zwischengespeichert, und die Speicherblöcke mit der Taktrate der Pipeline konsekutiv adressiert.



Nach einer Latenzzeit gegeben durch die Zugriffszeit auf den Speicher erscheinen nacheinander die Daten auf dem Bus, bzw. können von dort übernommen werden.

Nachteil dieses Verfahrens ist die schnelle Erzeugung konsekutiver Adressen für die Latches.



In beiden Fällen sind die Vektoren in der richtigen Weise in den Speicherblöcken abzuspeichern, so daß beim einen Speicherzugriff die Komponenten a_i, a_{i+1}, ..., a_{i+N} gelesen werden und beim folgenden Zugriff mit der nächsten Adresse die Komponenten a_{i+N+1}, a_{i+N+2}, ..., a_{i+2N} adressiert werden.

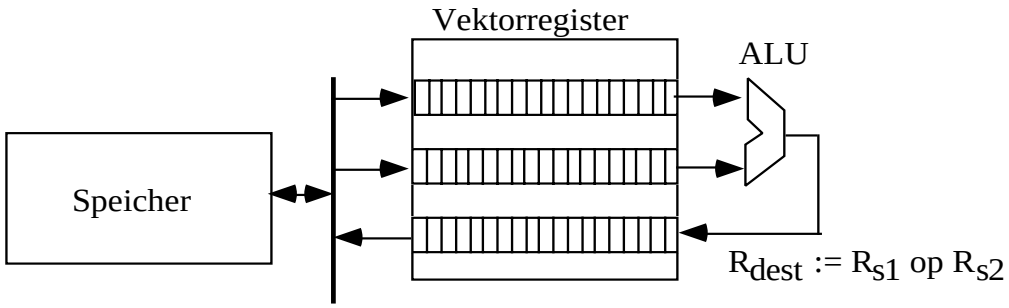
Soll am gleichen Eingang der Pipeline nach einem Vektor A ein Vektor B verarbeitet werden, der an einer anderen Stelle im Speicher steht, dann ist beim s-access das nur möglich, wenn die Pipeline Leerzyklen zwischenschaltet: Vektoren werden mit $K \cdot N$ Komponenten angesetzt, und wenn die Vektorlänge $n \bmod N$ ungleich Null ist, so ist die Anzahl der Leerzyklen der Pipeline $N - n \bmod N$.

3.2.2.2. Register-Register-Organisation

Da man den Speicher zunächst nicht auf dem Chip mit der (den) Pipeline(s) unterbringen kann, ist die Transportgeschwindigkeit der Daten eine Begrenzung. Sehr breite Busse (256 Bit und mehr) machen außerhalb des Chips Schwierigkeiten, schnelle Busse desgleichen (im Beispiel wären 1,2 GByte/s entspricht 9,6 GBit/s zu transportieren). Bei einer Busbreite von 256 Bit sind das immer noch 75 Mio. Worte à 256 Bit/s.

Die gleiche Überlegung wie für Caches greift dann auch hier: schnelle Speicher an Bord der CPU.

Es sind Vektorregister (VR); man wähle eine gängige Vektorlänge (64 - 128 Komponenten) und sehe Speicher an Bord vor mit dieser Länge. Dann erhält man folgende Architektur:

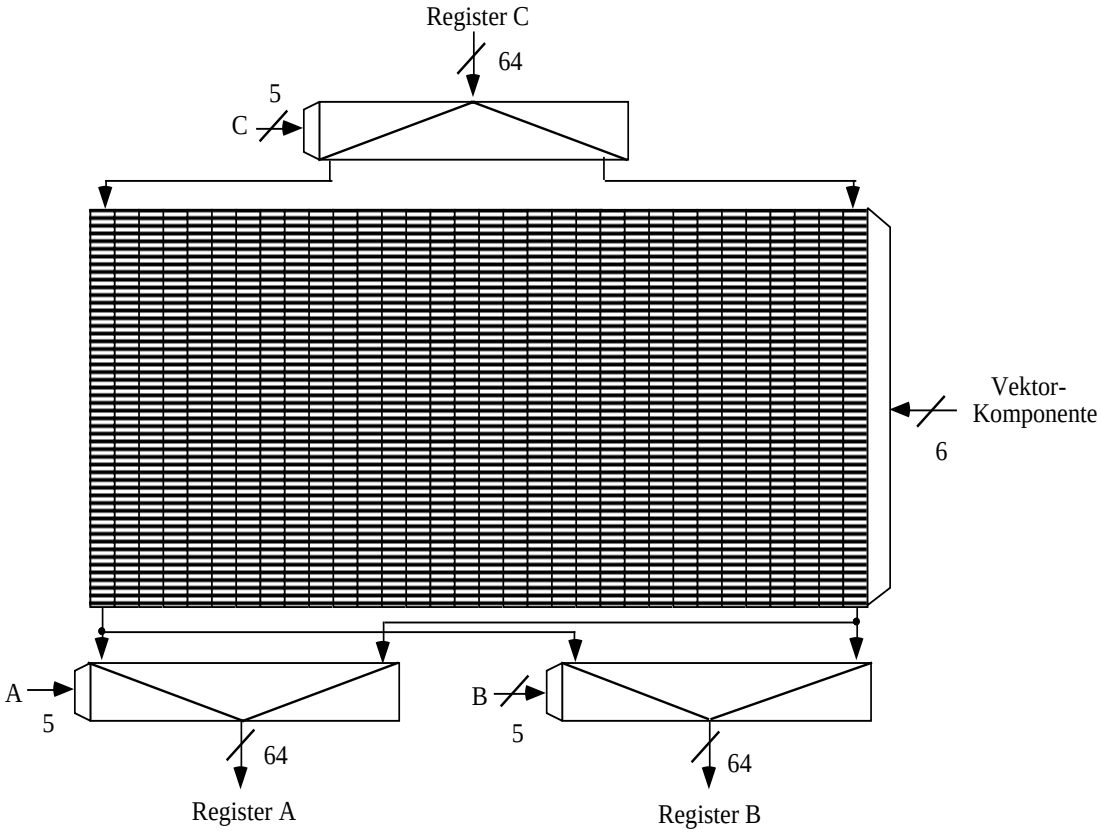


Nur ab und zu muß ein neuer Vektor aus dem Speicher nachgeladen werden; der Löwenanteil der Rechnung läßt sich mit den Vektorregistern intern bestreiten.

Intern, auf dem Chip, ist die Zugriffszeit auf die Speicher vergleichbar der Taktperiode der Pipeline.

Organisiert man die Vektorregister für 64 Komponenten so erhält man einen Speicher von 64 Worten à 64 Bit (bei 64 Bit Wortlänge) für ein VR.

Dann ist der Zugriff auf dem Chip in 10 ns problemlos möglich. (τ_g = 1ns ; Dekodieren 6 τ_g ; Auslesen/Einschreiben 2 τ_g ; Durchschalten 2 τ_g .)



Sieht man 32 Register vor wie in einer RISC-Architektur, dann braucht man 32 Speicherblöcke à 4 k Bit. Da für eine Vektoroperation nur einmal ein Register als Ganzes angesprochen wird und dann ein Adressgenerator die Adressierung der Komponenten übernimmt, geht die Adressierung des Registers in den Zeitverbrauch nur als kleiner Overhead ein.

3.2.2.3. Trennung von Skalar- und Vektoroperationen

Da sich skalare Operationen i. allg. schlecht vektorisieren lassen, sollten sie getrennt von Vektoroperationen ablaufen.

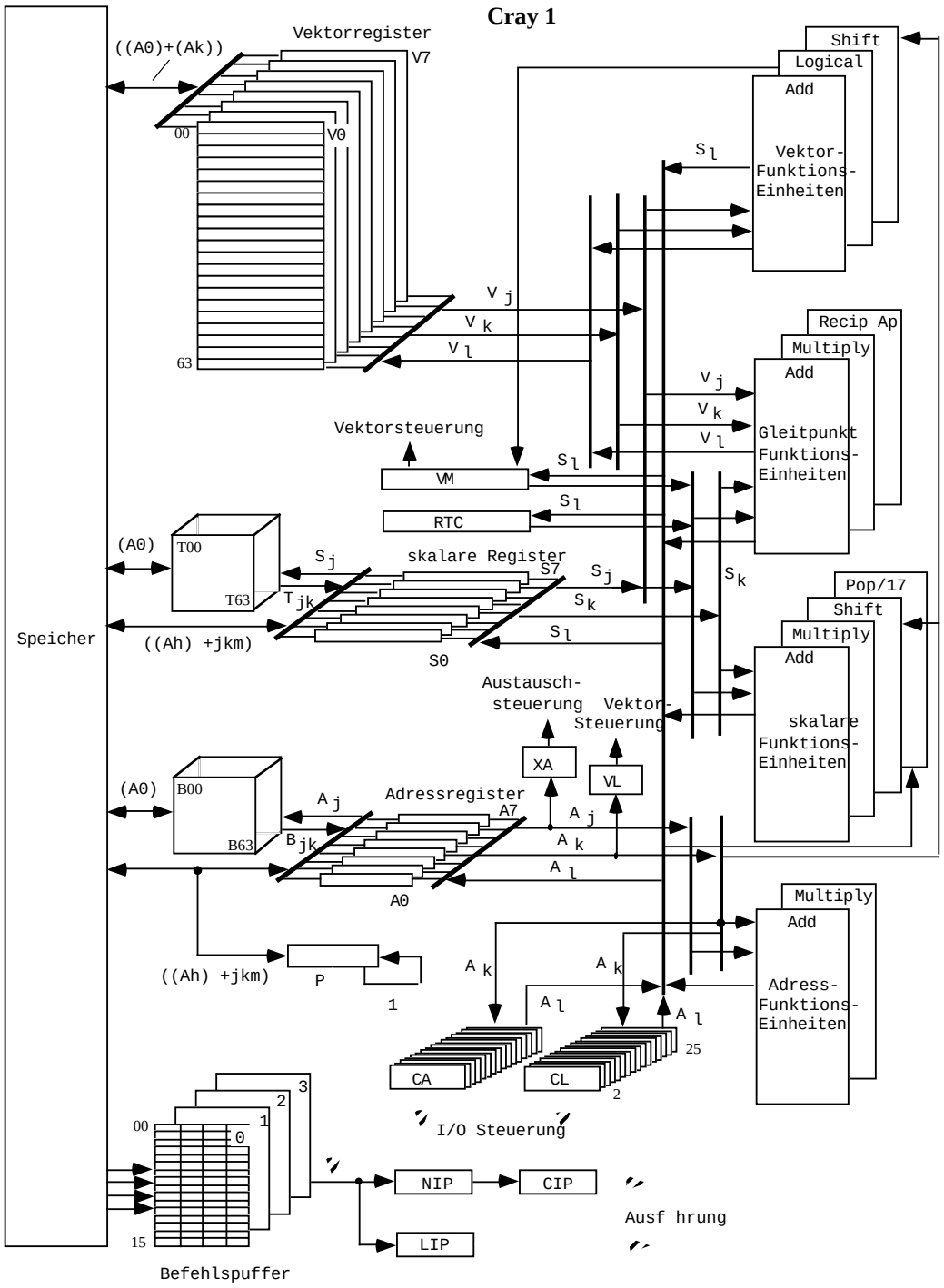
Dem trägt man Rechnung durch einen eigenen Satz von Registern für Skalare und einen eigenen Skalarprozessor, ggf. mit eigenen arithmetischen Pipelines.

3.2.3. Beispiele von Vektorrechnern

3.2.3.1. CRAY-1

Gebaut 1972 von Seymore Cray für die Nuklearforschung in Los Alamos wurde der Rechner 1976 auf den Markt gebracht.

Der Rechner arbeitet mit getrennten Skalar- und Vektorregistern und -Recheneinheiten. Er realisiert eine Register-Register-Architektur. Seine Pipeline arbeitet mit einer Taktrate von $\tau = 12{,}5$ ns . Das entspricht einer Leistung von bis zu 160 MFlop.



Die Cray-1 hat 8 Adressregister (A) mit 24 Bit Länge, 8 skalare Register (S) mit 64 Bit, 8 Vektorregister (V) mit 64 Komponenten zu 64 Bit, 64 Adress-Rettregister (B) zu 24 Bit und 64 Skalar-Rettregister zu 64 Bit für den Blocktransfer zum Speicher mit 1 Wort/12,5 ns. An Spezialregistern ist vorhanden ein Vektorlängenregister VL mit 7 Bit, eine Vektormaske VM mit 64 Bit, ein Adress-Austauschregister XA mit 24 Bit, ein Befehlszähler (P) mit 24 Bit und Instruktionsregister LIP(lower instruction parcel), NIP (next instruction parcel) und CIP (current instruction parcel) mit jeweils 16 Bit. Die Vektorpipeline kann mit den Registern zusammen die folgenden Operationen machen:

$V \times V \rightarrow V$, $V \times S \rightarrow V$, $V \times V \rightarrow S$, Laden und Speichern in den Hauptspeicher.

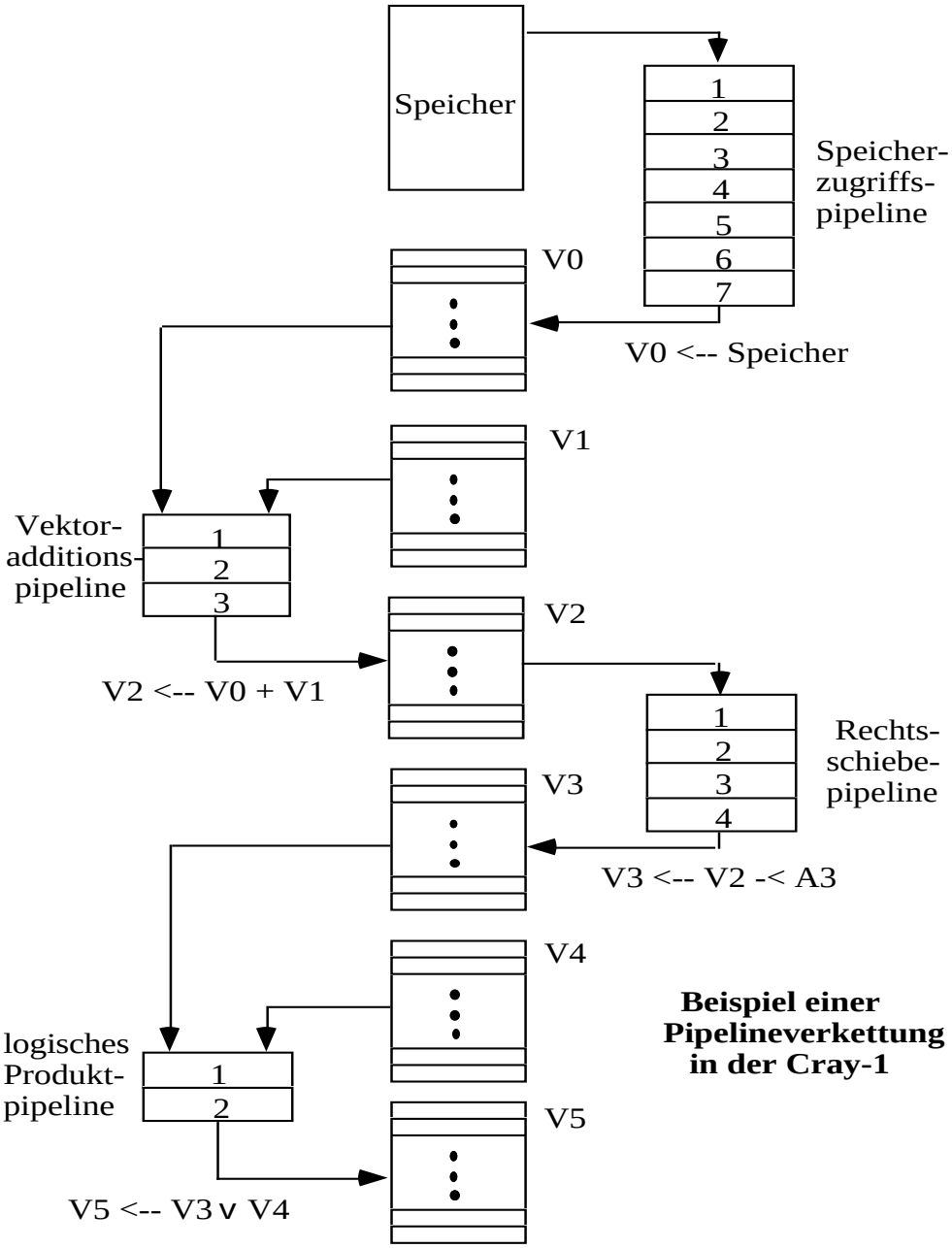
Die funktionalen Einheiten sind

Pipeline	Register	Stufen = Clockperioden
Adresspipeline		
ADD	A	2
MPY	A	6
skalare Pipeline		
ADD	S	3
SHIFT	S	2 oder 3
Logik	S	1
# führende Nullen	S	3
Vektorpipeline		
ADD	V oder S	3
SHIFT	V oder S	4
Logik	V oder S	2
GK-Pipeline		
ADD	S oder V	6
MPY	S oder V	7
1/x Approx.	S oder V	14
Speicherzugriff	V	7

Die Instruktionen werden in 4 Insrtruktionspuffern mit 64 x 16 Bit gehalten.

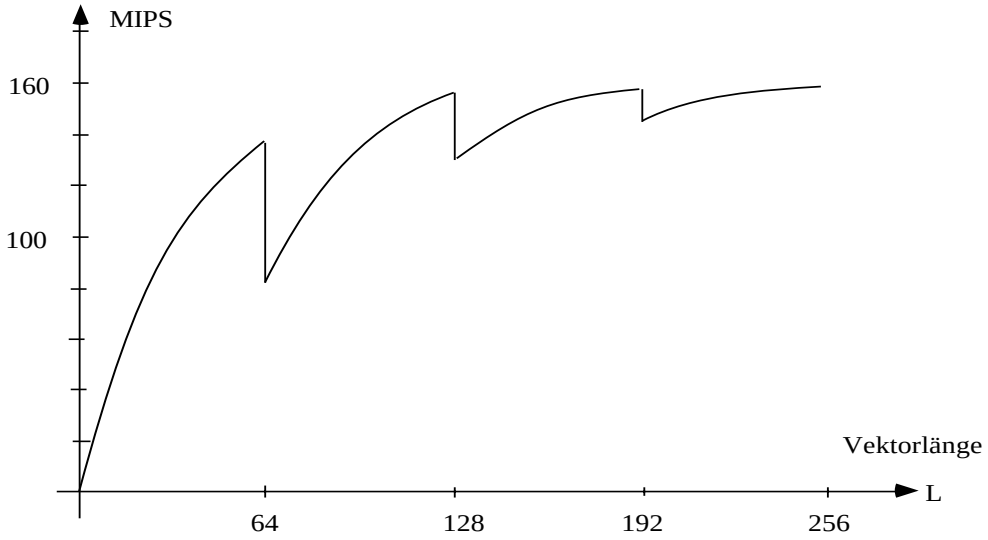
Ein Befehl hat eine Länge von 16 Bit: 7 Bit Opcode, 3 x 3 Bit für 2 Operandenregister und das Zielregister oder 7 Bit Opcode und 3 Bit für das Zielregister und 6 Bit für eine Konstante.

Einen Teil ihrer Leistungsfähigkeit erhielt die CRAY-1 durch die mögliche Verkettung von Operationen, die zusammen eine tiefe Pipeline aufbauen.



Ohne Umweg über den Speicher werden Vektorregister und Pipelines hintereinandergeschaltet.

Die Begrenzung auf 64 Komponenten macht sich bemerkbar bei Rechnungen mit langen Vektoren: bei 65, 129 und 193 Komponenten zeigt die Leistung einen Einbruch durch die dann notwendig werdende Umorganisation der Rechnung.

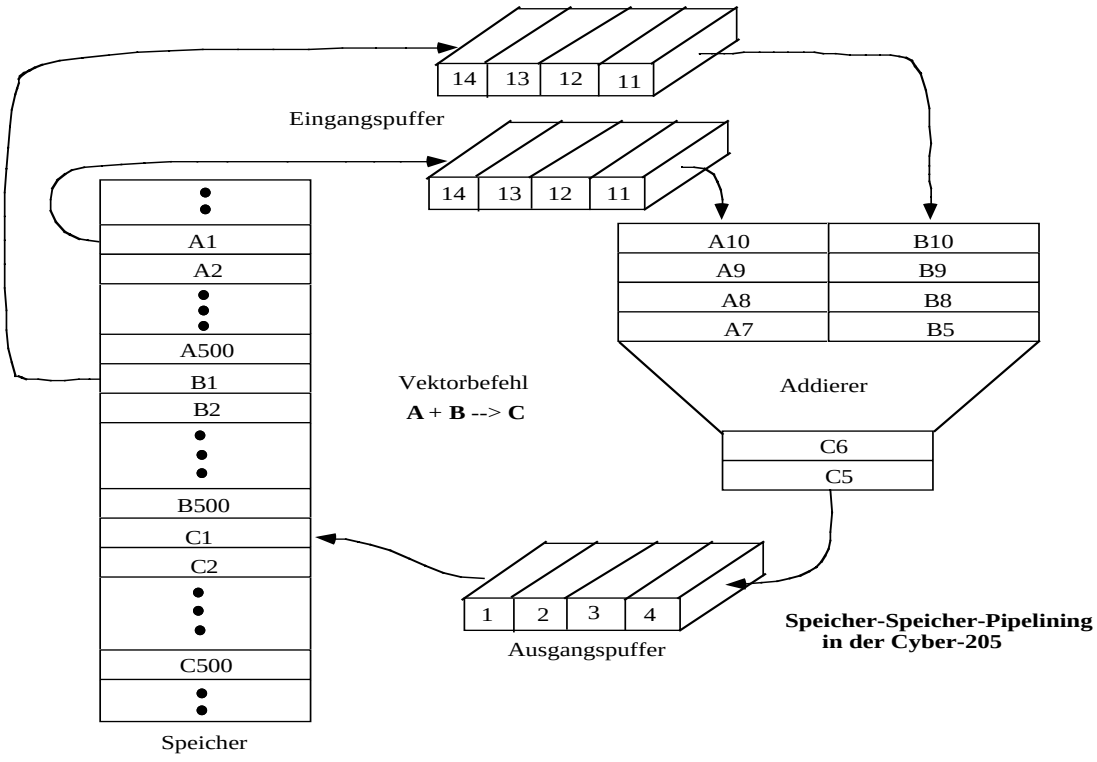


Die Geschwindigkeit der Maschine wird festgelegt durch die Taktperiode von 12,5ns entsprechend 80 MHz und die Möglichkeiten interner Parallelarbeit (Vektorpipeline, skalare Pipeline, Adresspipeline)

Weiterentwicklungen der CRAY-1 bezogen sich auf den Einsatz mehrerer paralleler CPU's, ohne zunächst das grundlegende Konzept zu ändern (z. B. CRAY-XMP mit 2 - 4 Prozessoren).

3.2.3.2. Cyber-205

Wenige Jahre nach der CRAY-1 baute Firma Control Data einen Vektorrechner in der Speicher-Speicher-Architektur.



Die Pipelines werden über einen schnellen Pufferspeicher vom Hauptspeicher mit Daten versorgt und die Resultate im Speicher wieder aufgesammelt.

Mit 4 Pipelines erreichte die Maschine bei einem Pipelinetakt von 20 ns eine Leistung von bis zu 200 MFlop für 64-Bit Vektoroperationen.