

Modul: Technische Informatik II (T3INF 2005)

Kurs: TINF23B

Lehreinheiten:

- Rechnerarchitekturen 1 (T3INF 2005.1)
- Systemnahe Programmierung 1 (T3INF 2005.3)

Notenbildung:

- zusammen mit Betriebssystemtechnik (T3INF 2005.2) → eine Prüfungsleistung
- „Teile“:

Einheit	Prüfungsleistung	(geplante) Gewichtung
Betriebssystemtechnik	Klausur	ca. 40%
Rechnerarchitektur 1	Klausur	ca. 40% +Δ%
Systemnahe Programmierung 1	Klausur	ca. 20% -Δ%
$\Sigma > 50\% \rightarrow \text{bestanden}$		

Literatur:

- Th. Flik
Mikroprozessortechnik
Springer Verlag, verschiedene Auflagen
- Andrew S. Tanenbaum
Structured Computer Organization
Prentice Hall, 4. Auflage
- M. Morris Mano
Computer System Architecture
Prentice Hall, 3. Auflage
- Bernd Becker, Rolf Drechsler, Paul Molitor
Technische Informatik, Eine Einführung
Pearson Studium, 2005 Formaler Ansatz
- Steven W. Smith
The Scientist and Engineer's Guide to Digital Signal Processing
1999, Second Edition
www.analog.com { Auch für „Signale und Systeme“ }

Inhaltsverzeichnis

1 Einleitung	1.1	8 Steuerwerk	8.1
1.1 Allgemein	1.1	8.1 Allgemein	8.1
1.2 Digital-Rechner, geschichtliche Entwicklung	1.1	8.2 Strukturen	8.1
1.3 Zukunft	1.3	8.3 Darstellung der Steuersignale	8.5
1.4 Vergleich 8051 / Cortex M4	1.4	8.4 Beispielstruktur einer CPU	8.6
1.5 Quellen zu Kap. 1	1.5	8.5 Anhang zu Kap. 8	8.7
1.6 Quellen zu Kap. 1	1.5	8.6 Quellen zu Kap. 8	8.8
2 Minimalsystem	2.1	9 Speicher	9.1
2.1 Architektur	2.1	9.1 Adressräume	9.1
2.2 Beispiele	2.3	9.2 Adressierung	9.3
2.3 Quellen zu Kap. 2	2.4	9.3 Typ / Aufbau	9.9
3 CPU: Grobstruktur	3.1	9.4 Quellen zu Kap. 9	9.11
3.1 Funktionsblöcke	3.1	10 Speicherverwaltung	10.1
3.2 Schaubild	3.1	10.1 Allgemein	10.1
4 Zahlendarstellung	4.1	10.2 Segmentverwaltung	10.2
4.1 positive / negative Zahl	4.1	10.3 Quellen zu Kap. 10	10.10
4.2 Gleitkommazahl	4.7	11 Erweiterung des Minimalsystems	11.1
4.3 Beispiele	4.12	11.1 Interrupt	11.1
4.4 Quellen zu Kap. 4	4.15	11.2 DMA	11.11
5 Arithmetik	5.1	11.3 Fließband / Pipeline	11.15
5.1 Ganzzahl-Addition	5.1	11.4 Quellen zu Kap. 11	11.23
5.2 Ganzzahl-Subtraktion	5.8	12 Programmieren I	12.1
5.3 Ganzzahl-Multiplikation, Division	5.10	12.1 Befehlszyklus	12.1
5.4 Gleitkomma Arithmetik	5.18	12.2 Hierarchie von Sprachen	12.3
5.5 Arithmetik-Anhang	5.19	12.3 Klassifikation von Befehlen	12.4
5.6 Quellen zu Kap. 5	5.21	12.4 Quellen zu Kap. 12	12.8
6 Bussystem	6.1	13 Programmieren II	13.1
6.1 Allgemein	6.1	13.1 Schichten eines Computers	13.1
6.2 Einteilung	6.1	13.2 Programmerstellung	13.2
6.3 Bus-Anhang	6.10	13.3 Adressierungsarten	13.4
6.4 Quellen zu Kap. 6	6.11	13.4 Einteilung von Befehlen	13.7
7 Rechenwerk	7.1	13.5 Quellen zu Kap. 13	13.11
7.1 Allgemein	7.1	14 Programmieren Anhang	14.1
7.2 Register	7.1	14.1 Speicherorganisation	14.1
7.3 Beschreibung	7.2	14.2 Befehlssatz	14.2
7.4 Struktur	7.2	14.3 Assembler - Direktiven	14.4
7.5 Quellen zu Kap. 7	7.2	14.4 Datei Template	14.5
		14.5 Quellen zu Kap. 14	14.8

1 Einleitung

1.1 Allgemein

• Rechner, Computer

- {Eingabedaten}
 - [Rechner]
 - {Ausgabedaten}
- Gerät, welches automatisch Anweisungen ausführt

• Begriffe

- Anweisung, Instruktion, Befehl
- Sequenz von Anweisungen:
Programm

1.2 Digital-Rechner, geschichtliche Entwicklung

1.2.1 Mechanisch

~1100 v. C. China: Abakus

Stellenwertsystem

~1600 Schickardt, mech. Addiermaschine

B. Pascal

~1700 Leibnitz, Binärsystem

~1850 Charles Babbage

~1885 Hollerith: Lochkartenmaschine

1.2.2 Theorie

1936 Alan Turing

von Neuman

1952 Wilkes

1.2.3 Elektromechanisch (Relais)

1938 Zuse: Binärsystem

Aiken: Dezimalsystem

1.2.4 Elektronisch

• Röhren

1946 ENIAC: Dezimalsystem

• HINWEIS !

bisher keine Universalrechner

bisher nicht frei programmierbar

• Transistoren (1947/1948)

1951 UNIVAC; Firma: Remington Rand

★ Halbleiter-Prozessoren

(1959 1. IC; Firma Fairchild)

4 Bit Intel: 4004

8 Bit Intel: 8008

„ 8080, Motorola 6800

„ 8085

16 Bit TI: TMS 9900

Intel: 8086

Motorola: MC 68000

32 Bit Intel: i386

Motorola 68020

64 Bit Intel: Pentium

Motorola 68060

1986 RISC-Typ (MIPS)

→ weitere :

Zilog Z80, Z8000

SPARC

PowerPC601

Alpha

ARM

★ Halbleiter-Controller

Prozessor + I/O + Extras

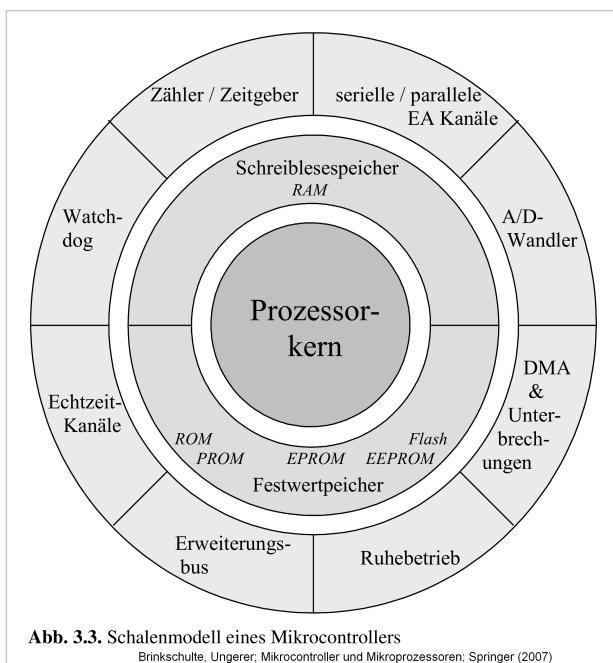


Abb. 3.3. Schalenmodell eines Mikrocontrollers
Brinkschulte, Ungerer; Mikrocontroller und Mikroprozessoren; Springer (2007)

[BU07]

8051- Familie: Intel, Infineon, Phillips, Analog Devices, Atmel, Maxim/Dalles, ...

68CXX: Motorola | PIC: Microchip | AVR: Atmel

ARM7: ARM | Cortex: ARM |

* SOC System on Chip

z.B. im Raspberry Pi, Tablett/Handy

1.2.5 Entwicklung der IC-Herstellung

SSI Small Scale Integration

MSI Medium Scale Integration

LSI Large ...

VLSI Very Large ...

¬ Probleme

Gate-Isolationsdicke \sim nm

Gate-Größe \sim nm

1.3 Zukunft

¬ Polymer

Kunststoff statt Si

¬ Optisch

- Integriert Optisch
- Photonische Gitter

¬ Biologisch

- Si + Neuronen
- DNA - Computer ?

¬ Quanten-Computer

- Quad-Bit

1.4 Vergleich 8051 / Cortex M4

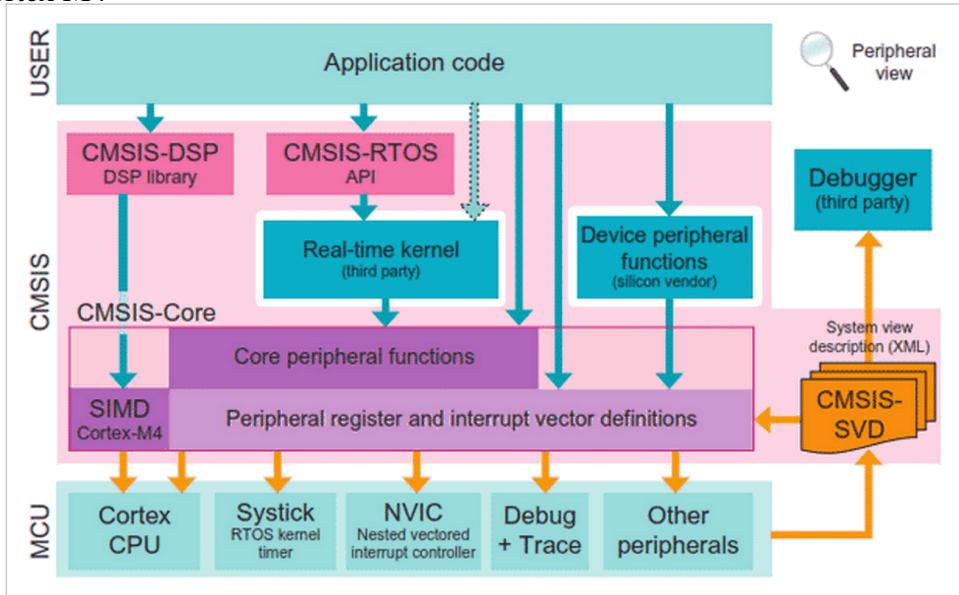
1.4.1 Übersicht

	8051-Familie	Cortex M4	
Register-Breite	8 Bit	32 Bit	nicht relevant
Architektur	Harvard	von Neumann ¹	beide wichtig
Speicher	64 KB + 64 KB + 256 B	4 GB ¹	64 KB übersichtlicher
Port-Controll-Register	1	10^2	8051 einfacher zu programmieren
Befehle	110	204	
Hardware zugriff	SFR (=MMR)	MMR (=SFR)	kein Unterschied
Bussystem	von Außen zugänglich	zwei interne (versteckte) Bussysteme	8051 besser für Unter-richt
Interrupt:	Auto (feste Tabelle)	"Vektoren" mit fester Ta-belle	
mit Prioritäten	wenigen "Auswahl Bits"	mit "Auswahl Register"	kaum Unterschied
nested	ja	ja	
Pipeline	"ja" (2 Stufen)	ja (5 Stufen)	
DMA	nein	ja	M4 moderner,
interne Busse	nein	ja	8051 wird dafür nicht verwendet
Rechte	nein	"ja"	

nested Interrupt : verschachtelte Interrupt

1.4.2 Softwarestruktur

* Cortex M4



[Emb15]

Quelle: <https://www.embedded.com/basics-of-the-cortex-mcu-software-interface-standard-part-1-cmsis-specification/>

Abrufdatum: 2020-08-24

* 8051 Auch RTOS möglich

¹2.2 Memory organization

Program memory, data memory, registers and I/O ports are organized within the same linear 4 Gbyte address space.

[ST13]

²8.1 GPIO introduction

Each general-purpose I/O port has four 32-bit configuration registers (GPIOx_MODER, GPIOx_OTYPER, GPIOx_OSPEEDR and GPIOx_PUPDR), two 32-bit data registers (GPIOx_IDR and GPIOx_ODR), a 32-bit set/reset register (GPIOx_BSRR), a 32-bit locking register (GPIOx_LCKR) and two 32-bit alternate function selection register (GPIOx_AFRL and GPIOx_AFRH).

[ST13]

1.5 Quellen zu Kap. 1

- [BU07] Uwe Brinkschulte und Theo Ungerer. “Mikrocontroller und Mikroprozessoren”. 2. Auflage. Springer-Verlag Berlin Heidelberg, 2007. ISBN: 978-3-540-46801-1.
- [Emb15] Embedded. “Basic of the Cortex MCU Software Interface Standart. Part 1 - CMSIS”. Embedded Staff, 17. Feb. 2015. URL: www.embedded.com/basics-of-the-cortex-mcu-software-interface-standard-part-1-cmsis-specification/ (besucht am 24.08.2020).
- [ST13] ST. “STM32F405xx/07xx, STM32F415xx/17xx, STM32F42xxx and STM32F43xxx advanced ARM-based 32-bit MCUs. Reference manual (RM0090)”. Rev 5. Doc ID 018909. STMicroelectronics, Sep. 2013.

2 Minimalsystem

minimaler Aufbau eines Computers

2.1 Architektur

2.1.1 Übersicht der Komponenten

⌘ Zentraleinheit, CPU

⌘ Speicher

Speichert Werte an Adressen in Zellen

¬ Programm Speicher

¬ Daten Speicher

⌘ Ein-/Ausgabe Einheit

⌘ Bussystem

⌘ Zusatzkomponenten

nicht im Minimalsystem !

¬ Interrupt - Controller

¬ DMA - Controller
(Direct Memory Access)

¬ Co - Prozessoren

bei Mikrocontrollern :

¬ Watch-Dog

¬ Timer, UART, ADU, DAU

2.1.2 Harvard - Architektur

⌘ Blockschaltbild

⌘ Eigenschaften

- getrennte Adress(-räume) für Daten u. Befehle
oder (schräfer):
getrennte Busse für Daten u. Befehle

⌘ Folgen (insb. getrennte Busse)

- ¬ Compilat im Datenspeicher
- ¬ meist gleichzeitige Kommunikation mit CPU
- ¬ Hohe Leistungsfähigkeit
- ¬ Aufwendig
- ¬ z.B. DSP

2.1.3 von Neuman - Architektur

Princeton - Architektur

* Blockschaltbild

* Eigenschaften

- ein Adressraum für Daten u. Befehle
 - mit
 - ein (System-) Bus (insb. nur ein Speicherbus)

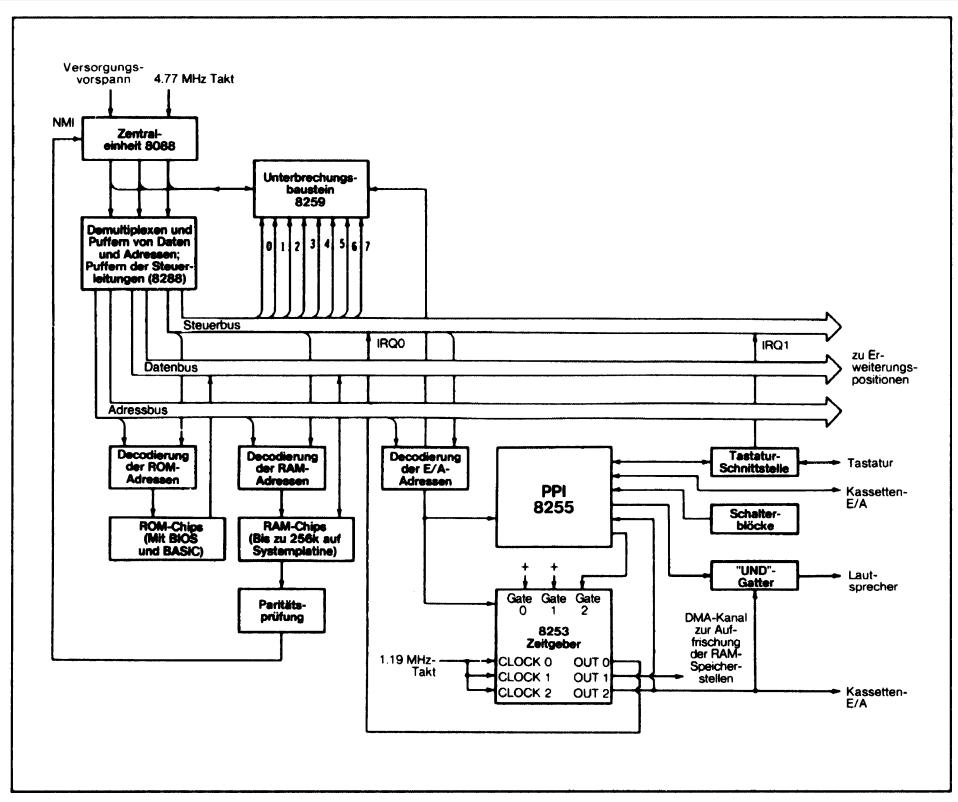
* Folgen

- ¬ Sequenzielle Kommunikation mit CPU
- ¬ allg. Anwendungen

2.2 Beispiele

2.2.1 Ur-PC

von Neumann



2.2.2 8051-System

Harvard (ein Bus-System)

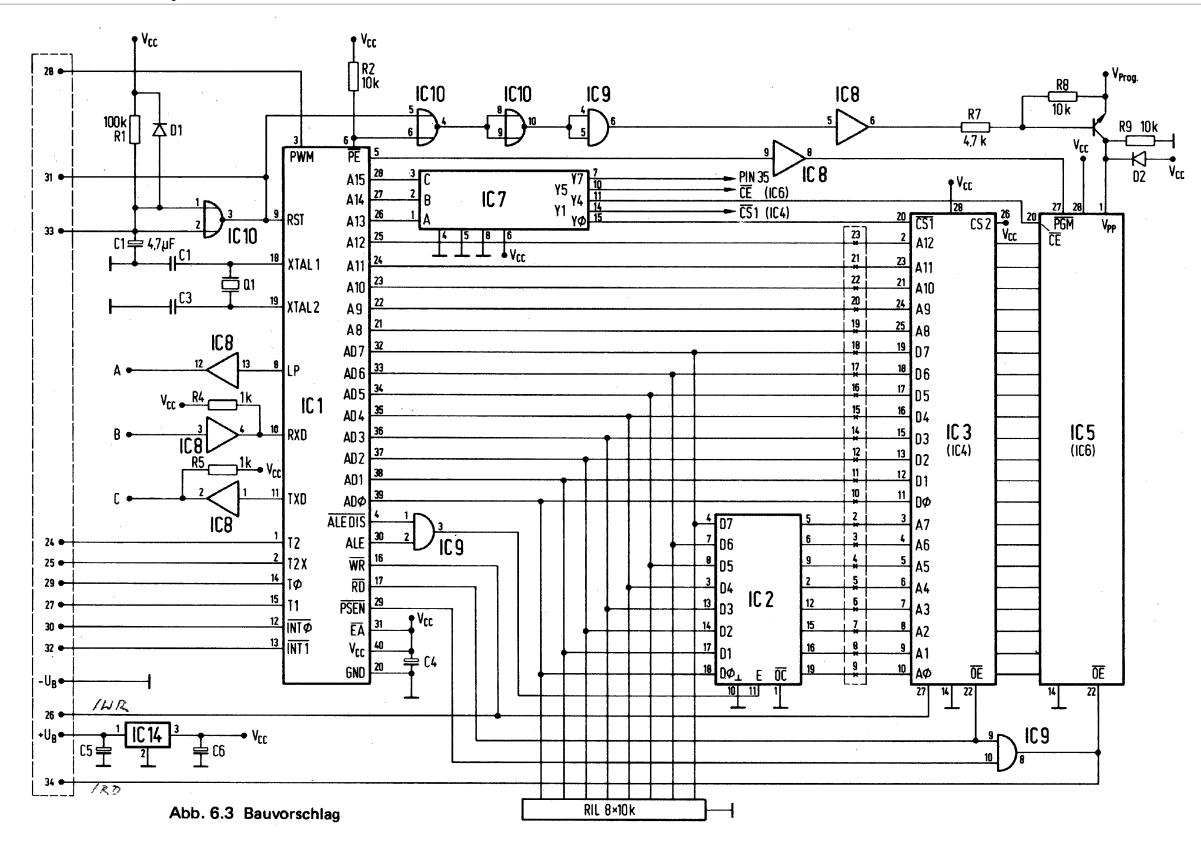
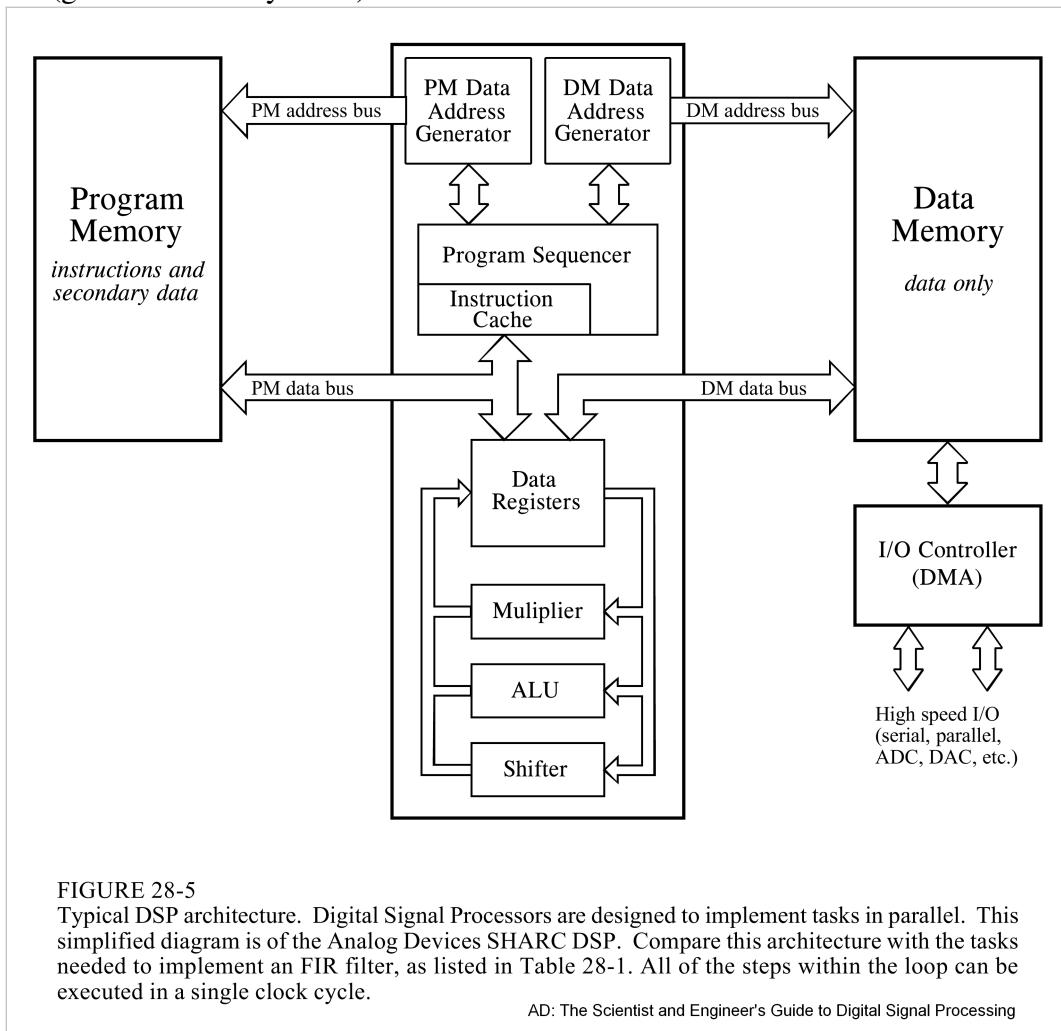


Abb. 6.3 Bauvorschlag

2.2.3 DSP

Harvard (getrennte Bus-Systeme)



[DSP99, Fig. 28.5]

2.3 Quellen zu Kap. 2

[DSP99]

Steven W. Smith. "The Scientist and Engineer's Guide to Digital Signal Processing".
2. Auflage. California Technical Publishing, 1999. ISBN: 0-9660176-6-8. URL: DSPguide.com.

3 CPU: Grobstruktur

3.1 Funktionsblöcke

✖ Steuerwerk

✖ Operationswerk:

○ Rechenwerk

○ Leitwerk

✖ Businterface

3.2 Schaubild

Alternative Zuordnungen

Steuerwerk + Leitwerk = Control Unit

oder

Businterface + Leitwerk = Businterface

4 Zahlendarstellung

4.1 positive / negative Zahl

4.1.1 Erhöhung der Stellenzahl

... ohne Änderung des Wertes

von n -Stellen auf m -Stellen

* pos. Binärzahl

$$\sum_{i=0}^{n-1} a_i 2^i = \sum_{i=0}^{m-1} b_i 2^i$$

\Leftrightarrow

Bezeichnung:

Unsigned Extension

* Vorzeichen Betrags Zahl

$$(-1)^{a_{n-1}} \cdot \sum_{i=0}^{n-2} a_i 2^i = (-1)^{b_{m-1}} \cdot \sum_{i=0}^{m-2} b_i 2^i$$

\Leftrightarrow

* Zweierkomplement Zahl

$$-a_{n-1} 2^{n-1} + \sum_{i=0}^{n-2} a_i 2^i = -b_{m-1} 2^{m-1} + \sum_{i=0}^{m-2} b_i 2^i$$

\Leftrightarrow

Bezeichnung:

Signed Extension

• Fixkommazahlen

- Spezialfall von Rationalzahl

mit Zeichenfolge

- Allgemein
Feste Position des Komma

- ¬ Vorteile
 - ★ schneller
 - ★ höhere 'Genauigkeit'
- ¬ Nachteil
 - ★ Berechnungen
Format "manuell" Überwachen
(s. Kap. 5.3.1)
- z.B. 16 Bit
- ★ Wertigkeit mit 13.3

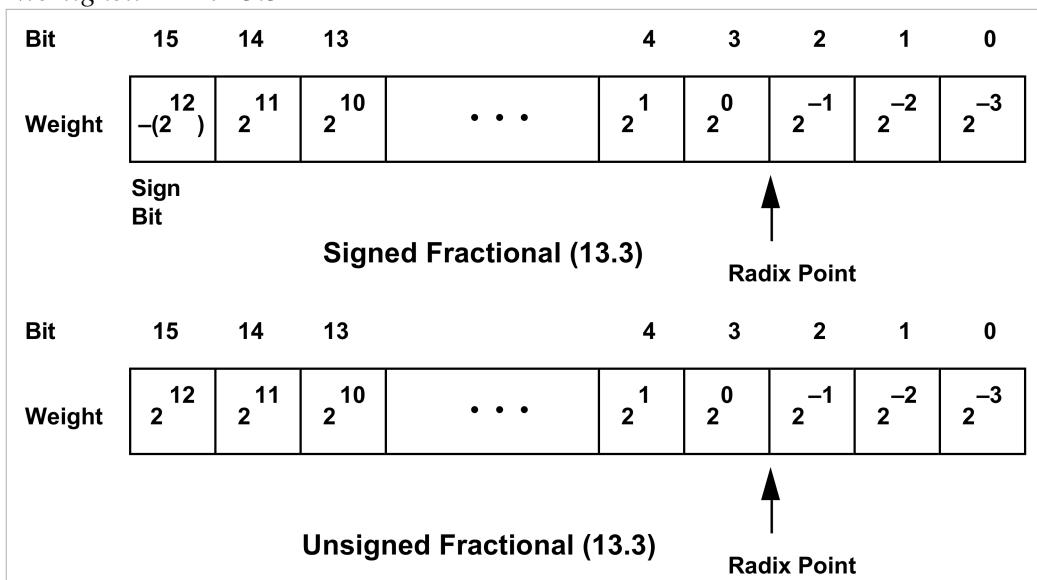


Figure C.2 Example Of Fractional Format

AD: ADSP 21xx Usermanual (2006)

[AD96, Fig. C.2]

* *Bereiche*
für Zweier-Komplement

Table C.1 shows the ranges of numbers representable in the fractional formats that are possible with 16 bits.

Format	Number of Integer Bits	Number of Fractional Bits	Largest Positive Value (0x7FFF) In Decimal	Largest Negative Value (0x8000) In Decimal	Value of 1 LSB (0x0001) In Decimal
1.15	1	15	0.999969482421875	-1.0	0.000030517578125
2.14	2	14	1.999938964843750	-2.0	0.000061035156250
3.13	3	13	3.999877929687500	-4.0	0.000122070312500
4.12	4	12	7.999755859375000	-8.0	0.000244140625000
5.11	5	11	15.999511718750000	-16.0	0.000488281250000
6.10	6	10	31.999023437500000	-32.0	0.000976562500000
7.9	7	9	63.998046875000000	-64.0	0.001953125000000
8.8	8	8	127.996093750000000	-128.0	0.003906250000000
9.7	9	7	255.992187500000000	-256.0	0.007812500000000
10.6	10	6	511.984375000000000	-512.0	0.015625000000000
11.5	11	5	1023.968750000000000	-1024.0	0.031250000000000
12.4	12	4	2047.937500000000000	-2048.0	0.062500000000000
13.3	13	3	4095.875000000000000	-4096.0	0.125000000000000
14.2	14	2	8191.750000000000000	-8192.0	0.250000000000000
15.1	15	1	16383.500000000000000	-16384.0	0.500000000000000
16.0	16	0	32767.000000000000000	-32768.0	1.000000000000000

Table C.1 Fractional Formats And Their Ranges

AD: ADSP 21xx Usermanual (2006)

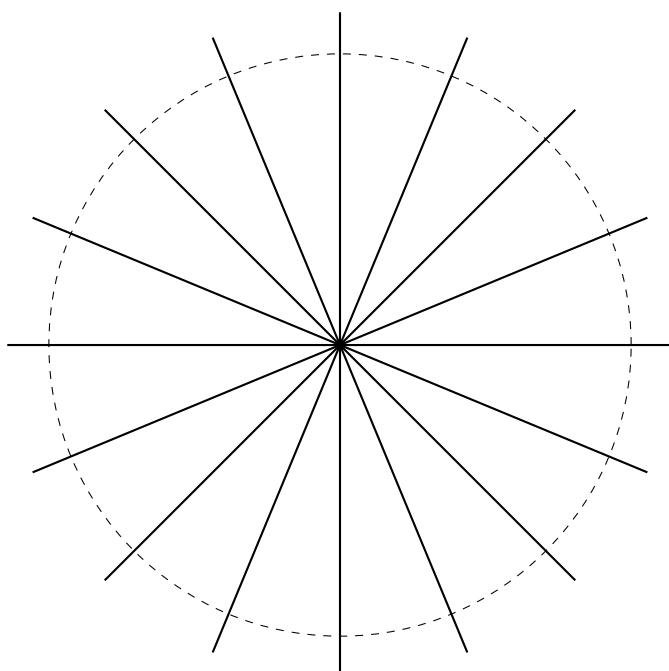
[AD96, Tab. C.1]

4.1.2 Zahlenkreis

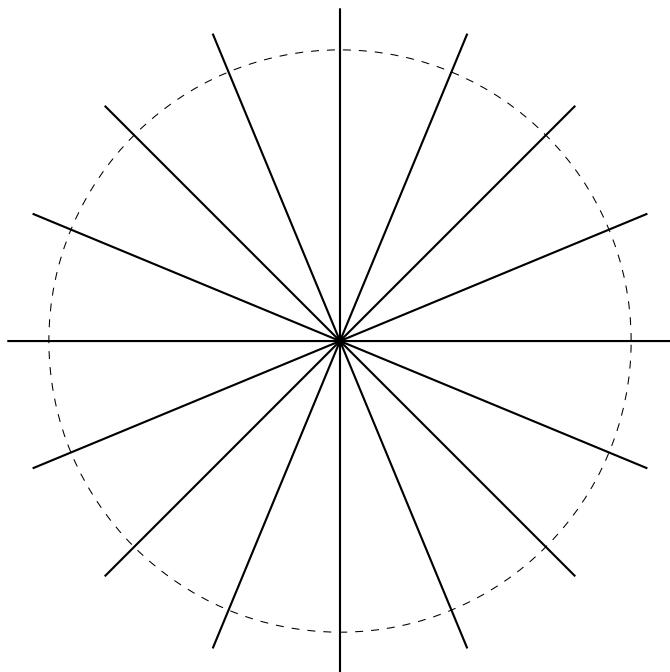
- ★ Allgemein
- Ganze Zahl
- ¬ Zahlenstrahl

- in Register
- ¬ endliche Stellenzahl
- ¬ Überlauf
- Zahlenkreis
- $n = 4 \rightarrow 16$ Segmente

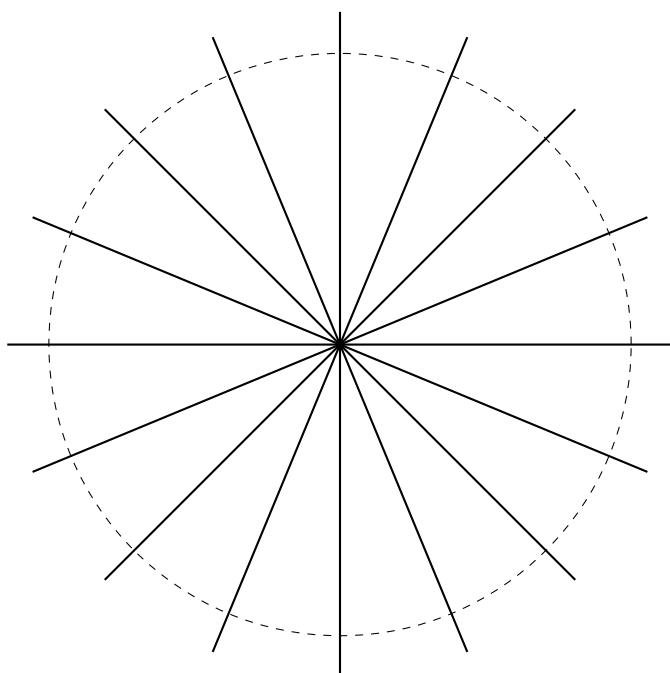
★ Binärzahl



★ Vorzeichenbetragszahl



★ Zweierkomplementzahl



★ Hinweis: Statusbits

Programm Status Wort (PSW)

z. B.: CY- Bit; OV- Bit

(andere Prozessoren: andere Bits)

4.1.3 Hinweise

✖ Genauigkeit:

Ganze Zahlen:

$$\Delta Z = 1$$

$$\Delta Z / Z \neq \text{konstant}$$

✖ Übersicht

UNSIGNED INTEGER		OFFSET BINARY		SIGN AND MAGNITUDE		TWO'S COMPLEMENT	
Decimal	Bit Pattern	Decimal	Bit Pattern	Decimal	Bit Pattern	Decimal	Bit Pattern
15	1111	8	1111	7	0111	7	0111
14	1110	7	1110	6	0110	6	0110
13	1101	6	1101	5	0101	5	0101
12	1100	5	1100	4	0100	4	0100
11	1011	4	1011	3	0011	3	0011
10	1010	3	1010	2	0010	2	0010
9	1001	2	1001	1	0001	1	0001
8	1000	1	1000	0	0000	0	0000
7	0111	0	0111	0	1000	-1	1111
6	0110	-1	0110	-1	1001	-2	1110
5	0101	-2	0101	-2	1010	-3	1101
4	0100	-3	0100	-3	1011	-4	1100
3	0011	-4	0011	-4	1100	-5	1011
2	0010	-5	0010	-5	1101	-6	1010
1	0001	-6	0001	-6	1110	-7	1001
0	0000	-7	0000	-7	1111	-8	1000

16 bit range:
0 to 65,535 16 bit range
-32,767 to 32,768 16 bit range
-32,767 to 32,767 16 bit range
-32,768 to 32,767

FIGURE 4-1

Common formats for fixed point (integer) representation. Unsigned integer is a simple binary format, but cannot represent negative numbers. Offset binary and sign & magnitude allow negative numbers, but they are difficult to implement in hardware. Two's complement is the easiest to design hardware for, and is the most common format for general purpose computing.

[DSP99]

4.2 Gleitkommazahl

Gleitpunktzahl,
Floating Point Number

4.2.1 Allgemein

- ¬ pos. u. neg. Zahlen
- ¬ Exponentialdarstellung
- ¬ Bestandteile
 - Vorzeichen: V'
 - Basis: B
 - Exponent: E
 - Mantisse: M'
- ¬ Wert

$$Z = V' \cdot M' \cdot B^E$$
- ¬ Eigenschaft
 - pro Wert mehrere $\{V', M', B, E\}$ möglich
 - Normierung

4.2.2 IEEE 754

* Allgemein

- ¬ Bestandteile
 - Basis: $B := 2$
 - Vorzeichen: $V' =: (-1)^V$
 - Exponent: $E =: C - S$
 - C : Charakteristik
 - S : Offset
 - Mantisse: $M' =: (1.M)_B$
- ¬ Bez.: normierte Darstellung
- ¬ Wert:

$$Z = (-1)^V \cdot (1.M) \cdot 2^{C-S}$$

- ¬ Hinweis:
- gespeichert wird: V, C, M

* IEEE 754 normal, short

Gesamt: 32 Bit (4 Byte)

$$S = 127$$

C mit 8 Bit

M mit 23 Bit

V mit 1 Bit

Bitfolge: [V | C | M]

★ IEEE 754 long, real

Gesamt: 64 Bit (8 Byte)

$S = 1023$

C mit 11 Bit

M mit 52 Bit

V mit 1 Bit

Bitfolge: [$V | C | M$]

★ Sonderwerte

Null : $C = 0 \wedge M = 0$

nicht normiert: $C = 0 \wedge M \neq 0$

unendlich: $C = \text{max} \wedge M = 0$

Not a Number: $C = \text{max} \wedge M \neq 0$

★ Eigenschaften

$\neg |$ kleinster Wert $| \neq 0$

$\rightarrow C = 1 \wedge M' = 1.000\dots$

$\neg |$ größter Wert $| < \infty$

$\rightarrow C = \text{max} - 1 \wedge M' = 1.111\dots$

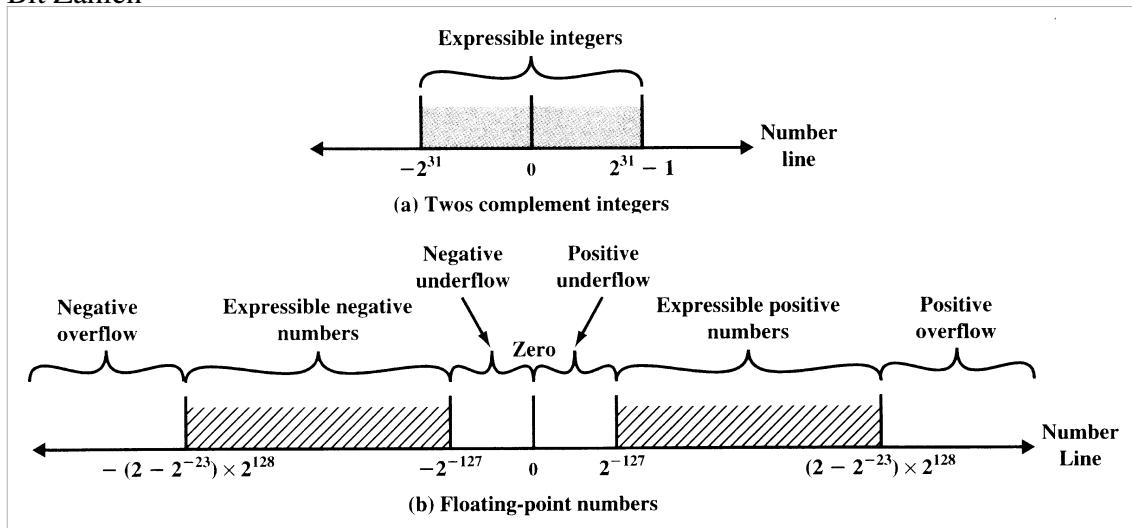
\neg Fehler / Genauigkeit

$\Delta Z \neq \text{const}$

$\Delta Z/Z \approx 10E-6$

★ Bild (mit Fehler)

32 Bit Zahlen



4 Zahlendarstellung

- ✖ *Bsp.: Umrechnungen mit short*
- ★ 52D

★ 1001 1001 1001 1001 1001 1001 1001 1001

✖ Rechenfehler

○ Bsp 1.

★ Programm

```
const long d=200000001;
long i;
float sum_f;
long sum_l;
printf("Summe +1 von 0 bis %d \n", d);
sum_f=0;
sum_l=0;
for (i=0; i<d; i++)
{
    sum_f = sum_f + 1;
    sum_l = sum_l + 1;
}
printf(" Ergebnise float:  %f  long %d\n", sum_f, sum_l);
```

★ Ausgabe

```
Summe +1 von 0 bis 200000001
Ergebnise float:  16777216.000000  long 200000001
```

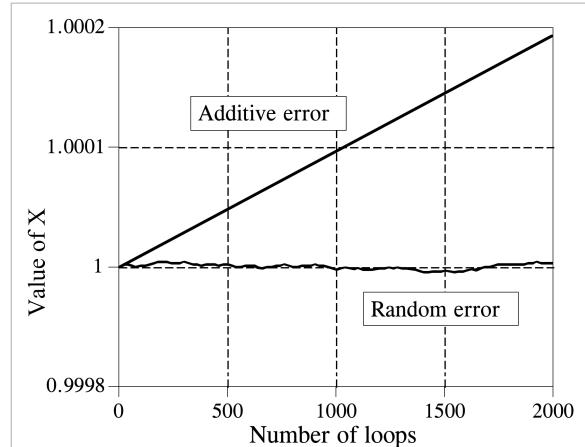
○ Bsp 2.

★ $(a-b)+(x-y) \neq (a+x)-(b+y)$
möglich!

★ Akkumulierter Rechenfehler:

Bsp. Programm

```
100 X = 1 ' initialize X
110 '
120 FOR I\% = 0 TO 2000
130 A = RND 'load random numbers
140 B = RND 'into A and B
150 '
160 X = X + A 'add A and B to X
170 X = X + B
180 X = X - A 'undo the additions
190 X = X - B
200 '
210 PRINT X 'ideally, X should be 1
220 NEXT I\%
230 END
```



[DSP99, Tab. 4.1, Fig. 4.4]

¬ Bsp. 3.

- ★ Numerische Integration von Differentialgleichungen:
„Der erzeugte Rundungsfehler hängt nicht nur von der verwendeten Computerarithmetik, sondern auch von der Implementierung ab.“ [SWP12, Seite 31 f]
- ★ Rechenfehler [SWP12, Beispiel 2.3.1]
 - horizontal: kleinere Rechenschritte
 - vertikal: steigender Fehler

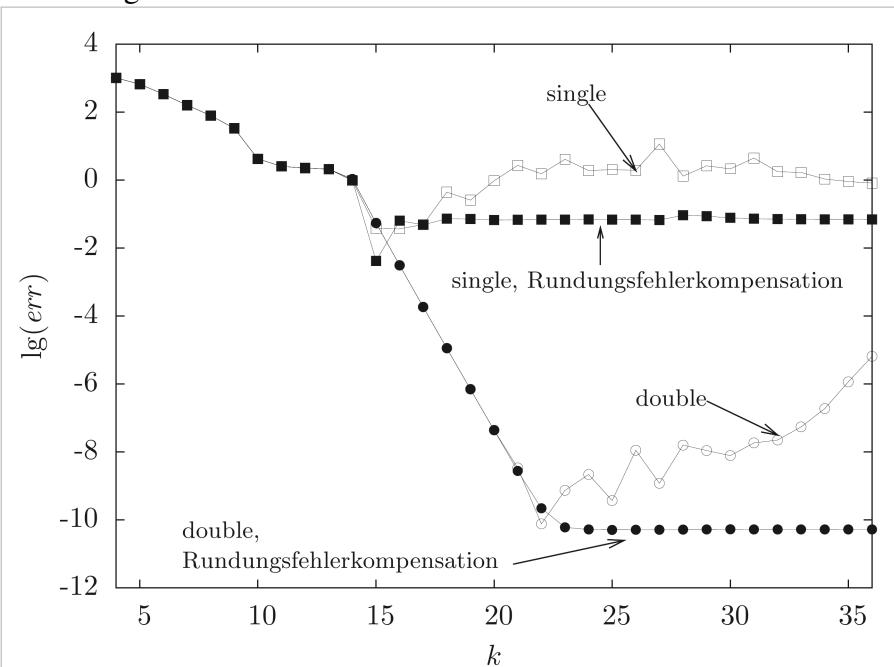


Abbildung 2.3.2: Gesamtfehler err bei Rechnung mit 2^k Schritten

Strehmel et.al.; Numerik gewöhnlicher Differentialgleichungen; Springer Spektrum (2012)

[SWP12, Abb. 2.3.2]

Rundungsfehlerkompensation [SWP12, Gl. (2.3.2)]

2

MSB LSE

LSB

○ Stellenwertsystem

- Binär
10_B
$$1 \cdot 2^{31} + 0 \cdot 2^{30} + 1 \cdot 2^{29} + 0 \cdot 2^{28} + \dots + 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0$$

$$= 2863311530_{\text{D}}$$
 - Oktal
(0)10 101 010 101 010 101 010 101 010 101 010 101 010 101 010_O
$$= 25252525252_{\text{O}}$$

$$= 2 \cdot 8^{10} + 5 \cdot 8^9 + 2 \cdot 8^8 + 5 \cdot 8^7 + 2 \cdot 8^6 + 5 \cdot 8^5 + 2 \cdot 8^4 + 5 \cdot 8^3 + 2 \cdot 8^2 + 5 \cdot 8^1 + 2 \cdot 8^0$$

$$= 2863311530_{\text{D}}$$
 - Hexadezimal (Sedenzial)
1010 1010 1010 1010 1010 1010 1010 1010 1010 1010 1010 1010 1010 1010 1010 1010_B
$$= AAAAAAAA_{\text{H}}$$

$$= 10 \cdot 16^7 + 10 \cdot 16^6 + 10 \cdot 16^5 + 10 \cdot 16^4 + 10 \cdot 16^3 + 10 \cdot 16^2 + 10 \cdot 16^1 + 10 \cdot 16^0$$

$$= 2863311530_{\text{D}}$$

- (Gepacktes-) BCD , Binär kodierte Dezimalzahl

Geht nicht !

- Vorzeichen Betrag

Vorzeichen negativ

$$\begin{aligned} \text{Betrag} &= 010_B \\ &= 0 \cdot 2^{30} + 1 \cdot 2^{29} + 0 \cdot 2^{28} + \dots + 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 \\ &= -(715827882_D) \end{aligned}$$

o 2-Komplement

Vorzeichen negativ \rightarrow 2-komplement Bildung

$$\begin{aligned}
 & -(010101010101010101010101010110_B) \\
 & = -(0101\ 0101\ 0101\ 0101\ 0101\ 0101\ 0110_B) \\
 & = -(55555556_H) \\
 & = -(1431655766_D)
 \end{aligned}$$

oder $-1 \cdot 2^{31} + 0 \cdot 2^{30} + \dots + 0 \cdot 2^0$

- Gleitkomma (IEEE short)

1 01010101 01010101010101010101010101010101

$V = (-1)^l \rightarrow$ negativ

$$C = 01010101_B = 85_D \rightarrow E = 85_D - 127_D = -42_D$$

$$M = (1.)0101010101010101010101010_B$$

$$Z = -1 \cdot 1.01010101010101010101010101010_B \cdot 2^{-42}$$

$$= -1 \cdot 1010101010101010101010101010.0_B \cdot 2^{-42} -$$

$$= -1 \cdot 11184810_D \cdot 2,7_{::D} \cdot 10^{-20}$$

$$= -3.031 \cdot 10^{-13}$$

ASCII

- 7-Bit

- ASCII

4 Zahlendarstellung

☒ 3

MSB LSB

1001100110011001100110011001100110011001

- Stellenwertsystem

- Binär

$$\begin{aligned}
 & 100110011001100110011001100110011001_B \\
 & = 1 \cdot 2^{31} + 0 \cdot 2^{30} + 0 \cdot 2^{29} + 1 \cdot 2^{28} + \dots + 1 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 \\
 & = 2576980377_D
 \end{aligned}$$

- Oktal

$$\begin{aligned}
 & (0)10\ 011\ 001\ 100\ 110\ 011\ 001\ 100\ 110\ 011\ 001_B \\
 & = 23146314631_O \\
 & = 2 \cdot 8^{10} + 4 \cdot 8^9 + 1 \cdot 8^8 + 4 \cdot 8^7 + 6 \cdot 8^6 + 3 \cdot 8^5 + 1 \cdot 8^4 + 4 \cdot 8^3 + 6 \cdot 8^2 + 3 \cdot 8^1 + 1 \cdot 8^0 \\
 & = 2576980377_D
 \end{aligned}$$

- Hexadezimal (Sedezimal)

$$\begin{aligned}
 & 1001\ 1001\ 1001\ 1001\ 1001\ 1001\ 1001\ 1001_B \\
 & = 99999999_H \\
 & = 9 \cdot 16^7 + 9 \cdot 16^6 + 9 \cdot 16^5 + 9 \cdot 16^4 + 9 \cdot 16^3 + 9 \cdot 16^2 + 9 \cdot 16^1 + 9 \cdot 16^0 \\
 & = 2576980377_D
 \end{aligned}$$

- (Gepacktes-) BCD, Binär kodierte Dezimalzahl

$$\begin{aligned}
 & 1001\ 1001\ 1001\ 1001\ 1001\ 1001\ 1001\ 1001_{BCD} \\
 & = 9 \cdot 10^7 + 9 \cdot 10^6 + 9 \cdot 10^5 + 9 \cdot 10^4 + 9 \cdot 10^3 + 9 \cdot 10^2 + 9 \cdot 10^1 + 9 \cdot 10^0 \\
 & = 99999999_D
 \end{aligned}$$

- Vorzeichen Betrag

1 001100110011001100110011001VBZ

Vorzeichen negativ

$$\begin{aligned}
 \text{Betrag} &= 0011001100110011001100110011001_B \\
 &= 0 \cdot 2^{30} + 0 \cdot 2^{29} + 1 \cdot 2^{28} + \dots + 1 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 \\
 &= -(429496729_D)
 \end{aligned}$$

- 2-Komplement

1001100110011001100110011001₂

Vorzeichen negativ → 2-komplement Bildung

$$\begin{aligned}
 & -(0110011001100110011001100111_B) \\
 & = -(0110\ 0110\ 0110\ 0110\ 0110\ 0110\ 0111_B) \\
 & = -(6666667_H) \\
 & = -(1717986919_D) \\
 & \text{oder } -1 \cdot 2^{31} + 0 \cdot 2^{30} + \dots + 1 \cdot 2^0
 \end{aligned}$$

- Gleitkomma (IEEE short)

1 00110011 00110011001100110011001

$$V = (-1)^1 \rightarrow \text{negativ}$$

$$C = 00110011_B = 51_D \rightarrow E = 51_D - 127_D = -76_D$$

$$M = (1.)00110011001100110011001_B$$

$$\begin{aligned}
 Z &= -1 \cdot 1.0011001100110011001_B \cdot 2^{-76} \\
 &= -1 \cdot 10011001100110011001.0_B \cdot 2^{-76-23} \\
 &= -1 \cdot 10066329_D \cdot 1,57.._D \cdot 10^{-30} \\
 &= -1,588.. \cdot 10^{-23}
 \end{aligned}$$

- ASCII

- 7-Bit

1001100 1100110 0110011 0011001 1001 ? 10011001_B 10011001_B 10011001_B 10011001_B

4C_H 66_H 33_H 19_H ?

String = Lf3 [EM]

- 8-Bit

99_H 99_H 99_H 99_H

String = öööö

? 1001 1001100 1100110 0110011 0011001

? 4C_H 66_H 33_H 19_H

String = Lf3 [EM]

4.4 Quellen zu Kap. 4

- [AD96] Analog Devices. “ADSP-2100 Family User’s Manual”. Rev 3.0. Analog Devices Inc., 1996. URL: www.analog.com.
- [DSP99] Steven W. Smith. “The Scientist and Engineer’s Guide to Digital Signal Processing”. 2. Auflage. California Technical Publishing, 1999. ISBN: 0-9660176-6-8. URL: DSPguide.com.
- [SWP12] Karl Strehmel, Rüdiger Weiner und Helmut Podhaisky. “Numerik gewöhnlicher Differentialgleichungen. Nichtsteife, steife und differential-algebraische Gleichungen”. 2. Auflage. Springer Spektrum, 2012. ISBN: 978-3-8348-2263-5.

5 Arithmetik

5.1 Ganzzahl-Addition

5.1.1 Formal

* Binärzahl

- ¬ i. Stelle
- ★ Übertrag von $i-1$ beachten
- ★ wenn $c_{n-1} = 1$
→ Bereichsüberschreitung
- ★ Ergebnis:
max. n+1 Stellen

* Vorzeichen Betrags Zahl

Regeln je nach Vorzeichen

* Zweierkomplement Zahl

- ★ „Regeln“ wie Binärzahl
- ★ Bereichsüberschreitung
 $c_{n-1} \neq c_{n-2}$

○ Beispiele / Proben

$n = 4$

A) keine Bereichsüberschreitung

- ★ positiv + positiv

$$\begin{array}{r} 2 \\ +3 \\ \hline 5 \end{array}$$

- ★ negativ + positiv

$$\begin{array}{r} -2 \\ +3 \\ \hline 1 \end{array}$$

B) mit Bereichsüberschreitung

- ★ positiv + positiv

$$\begin{array}{r} 4 \\ +5 \\ \hline 9 \end{array}$$

- ★ negativ + negativ

$$\begin{array}{r} -6 \\ +(-5) \\ \hline -11 \end{array}$$

5.1.2 Assemblerbefehle

(8051- Befehlssatz)

* Addition

ADD A, Rn; $(A) \leftarrow (A) + (Rn)$

beeinflusst: C, OV \Rightarrow

C set if carry out of Bit 7

cleared otherwise

OV set if

Carry out of Bit 7 and not out of Bit 6

or Carry not out of Bit 7 and out of Bit 6

cleared otherwise

* Addition mit Carry

ADDC A, Rn; $(A) \leftarrow (A) + (Rn) + (C)$

beeinflusst: C, OV

* Dezimal Adjust

DA; Korrektur des Akkumulators für BCD

beeinflusst: C

5.1.3 Realisierung

5.1.3.1 1 - Bit

* Halb-Addierer

- Formel

$$x + y = c | s$$

- Wahrheitstafel

- Boolesche Gleichung

- Schaltung / Symbol

- Laufzeit

★ Voll-Addierer

○ Formel

$$x + y + u = c | s$$

○ Wahrheitstafel

	x	y	u	c	s
0	0	0	0	0	0
0	0	1	0	0	0
0	1	0	0	0	0
0	1	1	0	0	0
1	0	0	0	0	0
1	0	1	0	0	0
1	1	0	0	0	0
1	1	1	0	0	0
1	0	0	1	0	1
1	0	1	1	0	1
1	1	0	1	0	1
1	1	1	1	0	1

○ Boolesche Gleichungen

○ Symbol

○ Schaltung(s-Varianten)

¬ zweistufiges Netz

(selber zeichnen)

★ Laufzeit

¬ aus HA

★ Schaltung

★ Laufzeit

5.1.3.2 N-Bit Addierer

★ Allg.

○ Formel

$$\vec{x} + \vec{y} + u = c \mid \vec{s}$$

○ Symbol

★ 2-stufiges Netz

○ Wahrheitstafel

→ Gleichungen

→ Schaltnetz

○ Laufzeit

★ Ripple-Carry Addierer

○ Aufbau

Reihenschaltung von VA

¬ Laufzeit

abhängig von N

• Carry-Look Ahead Addierer

Addierer mit vorab berechnetem Übertrag

¬ Vorberachtung

- ★ Problem: Laufzeit des Carry
- ★ Lösung: Carry c_i früher bereitstellen durch einsetzen von c_{i-1} in c_i
- ★ Hilfsgrößen G_i, P_i
- ★ G_i, P_i hängen nur von Stelle i ab
- ★ Laufzeit für $G_i, P_i : 1 \Delta T$

¬ Laufzeiten

ab x_i, y_i, u_0 gültig:
 bis G_i, P_i gültig
 + bis c_i gültig
 + bis s_i gültig
 = Gesamtzeit

¬ Symbole, Teilblöcke

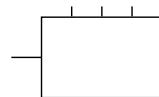
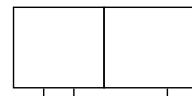
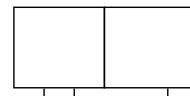
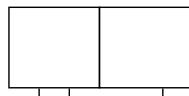
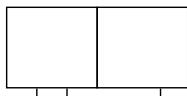
a) für G_i, P_i, s_i

b) für c_i

- ★ Bsp.: $n = 4$

$$\begin{aligned}
 c_0 &= G_0 \vee (P_0 \wedge u_0) \\
 c_1 &= G_1 \vee (P_1 \wedge G_0) \vee (P_1 \wedge P_0 \wedge u_0) \\
 c_2 &= G_2 \vee (P_2 \wedge G_1) \vee (P_2 \wedge P_1 \wedge G_0) \vee (P_2 \wedge P_1 \wedge P_0 \wedge u_0) \\
 c_3 &= G_3 \vee (P_3 \wedge G_2) \vee (P_3 \wedge P_2 \wedge G_1) \vee (P_3 \wedge P_2 \wedge P_1 \wedge G_0) \vee (P_3 \wedge P_2 \wedge P_1 \wedge P_0 \wedge u_0) \\
 c_4 &= G_4 \vee (P_4 \wedge G_3) \vee (P_4 \wedge P_3 \wedge G_2) \vee (P_4 \wedge P_3 \wedge P_2 \wedge G_1) \vee (P_4 \wedge P_3 \wedge P_2 \wedge P_1 \wedge G_0) \\
 &\quad \vee (P_4 \wedge P_3 \wedge P_2 \wedge P_1 \wedge P_0 \wedge u_0)
 \end{aligned}$$

¬ Schaltbild



¬ Hinweise

Anzahl der Eingänge pro UND :

$$N + 1$$

ab $N > 4$ „nicht herstellbar“

→ Abhilfe: Blockbildung

• BLOCKBILDUNG

z.B. Block mir 4-Bit Binär Addierer

$$X_i^{(4)} := x_{4 \cdot i+3} \cdots x_{4 \cdot i+0}$$

¬ Symbol

¬ „Ripple Carry Addierer aus 4-Bit Addierern“

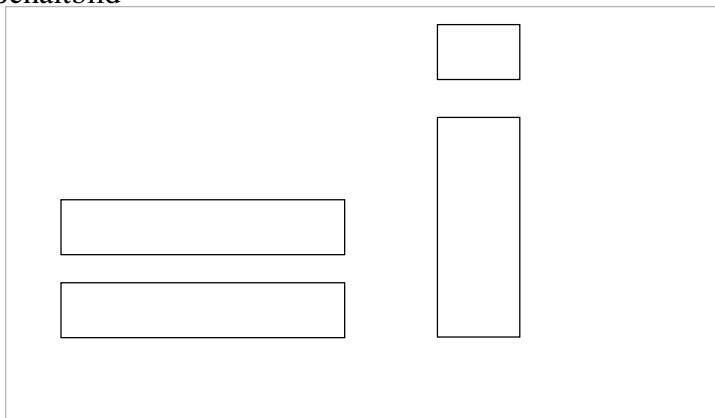
¬ „Carry Look Ahead Addierer mit 4-Bit Addierern“

$$c_i^{(4)} \text{ vorab berechnen}$$

- ¬ RTL / Assembler
- Block mit 8 Bit
- wie „Ripple Carry“ mit einem Addierer

❖ Serien Addierer

- Addition mit Speicher
- z.B. 1 Ein-Bit-Addierer für N-Bit Zahl
 $\vec{a} + \vec{b} = c \mid \vec{e}$
- Schaltbild



- Ablauf / Flussdiagramm

- Zeitbedarf
 $N \cdot (t_{VA} + t_{shift})$
- Hinweise
 - ¬ Hier: Startwert a wird überschrieben
 - ¬ Zeichnung ohne Ablaufsteuerung

5.2 Ganzzahl-Subtraktion

5.2.1 Formal

* Vorzeichen Betrags Zahl

je nach Vorzeichen andere Regeln

* Binärzahl

- i. Stelle

Differenz

Übertrag/Boorow

¬ Carry von i-1 beachten

¬ Regeln unterschiedlich
für Addition und Subtraktion

¬ Hinweis:
Binäre Subtraktion durch Addition der
Zweierkomplement Darstellung

* Zweierkomplement Zahl

$$a - b$$

- Fallunterscheidung

¬ $a > 0, b > 0$

¬ $a < 0, b > 0$

¬ $a < 0, b < 0$

- Regeln
wie Addition

5.2.2 Assemblerbefehle

(8051- Befehlssatz)

* Subtraktion

SUBB A, Rn; $(A) \leftarrow (A) - (Rn) - (C)$

beeinflusst: C, OV

5.2.3 Realisierung

* Binärzahl

- -stufiges Netz

Regeln : Tafel → ...

- Subtraktion durch Addition

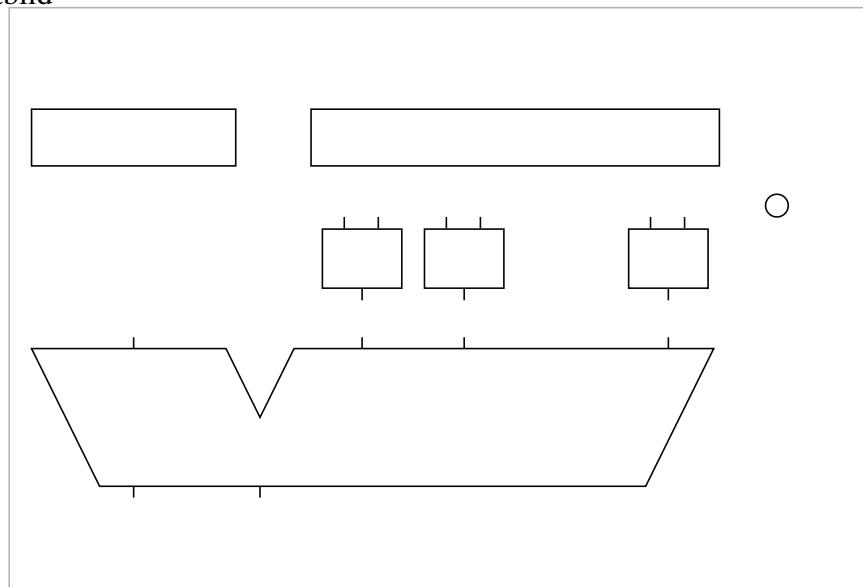
* 2-komplement Darstellung

a, b als N Bit Binär

mit $a, b < 2^{N-1}$

$$a - b \rightarrow a + \bar{b} + 1$$

* Schaltbild



* Zweierkomplement Zahl

Addition „gleich“ Subtraktion

Hinweis: ALU

* Enthält

- kombinatorische Logik
- Arithmetische Funktionen

* Teil der CPU

* Einzel IC

z.B. 74 382 s.S. 5.19 [TI88]

5.3 Ganzzahl- Multiplikation, Division

5.3.1 Multiplikation von Binärzahlen

* Formal

$$u \cdot v = p$$

○ je Stelle

4 Fälle

→

○ Formal

¬ Allgemein n Bit $\cdot m$ Bit

¬ Stellenanzahl n Bit $\cdot m$ Bit ($n, m \neq 1$)

5 Arithmetik

↪ z.B. 2 Bit · 2 Bit

→ $k =$

- ↪ Alternative Schreibweise
wie dezimale schriftliche
Multiplikation
- * z.B. $6_D \cdot 5_D = \dots$

”Stellen aus multiplizieren und gewichtet addieren“

- * z.B. Fixkommazahlen

General Rule:	4-Bit Example:	16-Bit Examples:																			
$ \begin{array}{r} M.N \\ \times P.Q \\ \hline (M+P) . (N+Q) \end{array} $	<table style="margin-left: auto; margin-right: auto;"> <tr> <td style="text-align: center;">1.111</td> <td style="text-align: center;">1.3 format</td> </tr> <tr> <td style="text-align: center;">\times</td> <td style="text-align: center;">11.11</td> </tr> <tr> <td colspan="2" style="text-align: center;"><hr/></td> </tr> <tr> <td colspan="2" style="text-align: center;">1111</td> </tr> <tr> <td colspan="2" style="text-align: center;"><hr/></td> </tr> <tr> <td colspan="2" style="text-align: center;">111.00001</td> </tr> </table>	1.111	1.3 format	\times	11.11	<hr/>		1111		1111		1111		1111		<hr/>		111.00001		5.3 \times 5.3 \hline 10.6	1.15 \times 1.15 \hline 2.30
1.111	1.3 format																				
\times	11.11																				
<hr/>																					
1111																					
1111																					
1111																					
1111																					
<hr/>																					
111.00001																					

3.5 format = (1+2) . (2+3)

Figure C.3 Format Of Multiplier Result

AD: ADSP 21xx Usermanual (2006)

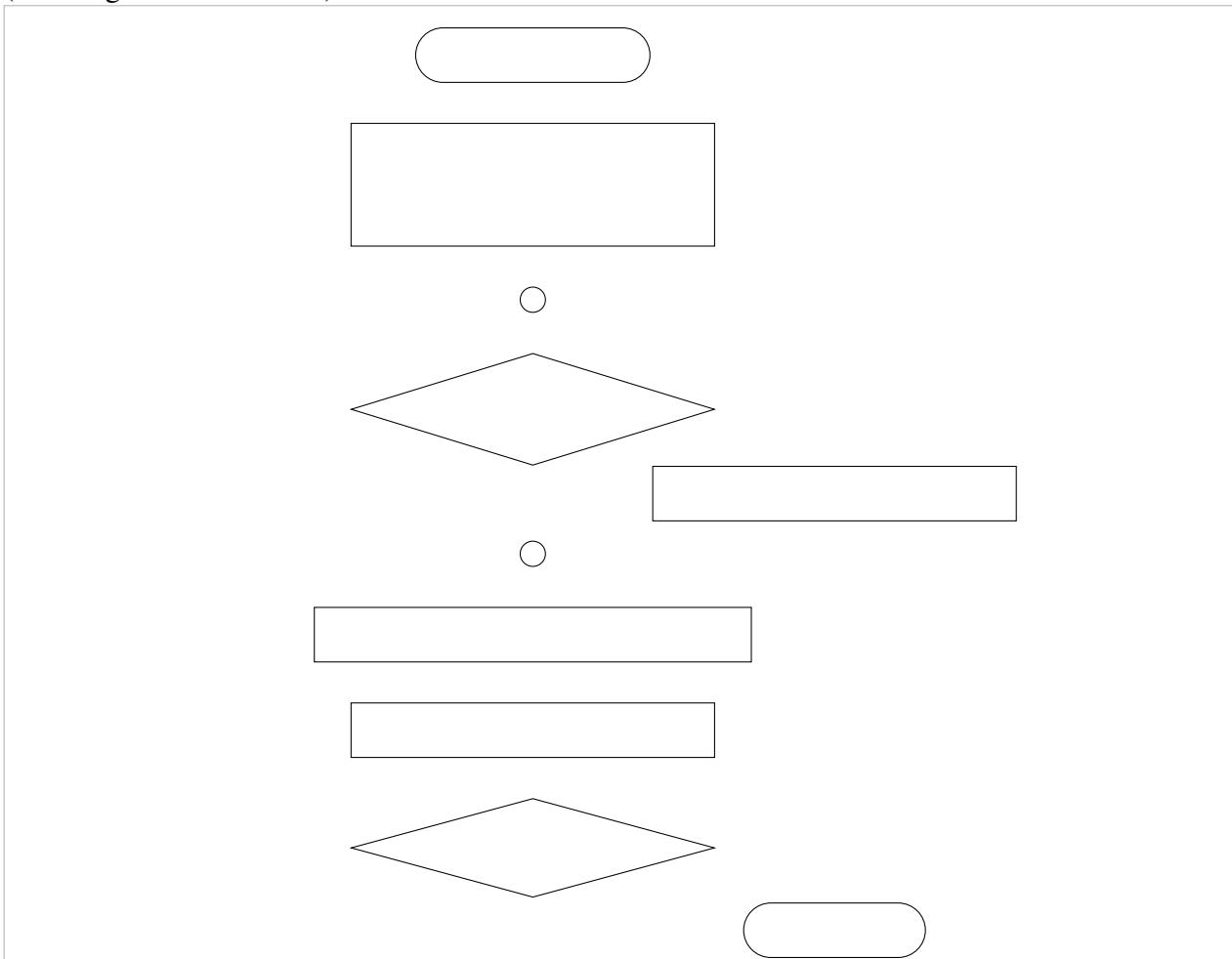
[AD96, Fig. C.3]

⌘ Algorithmus

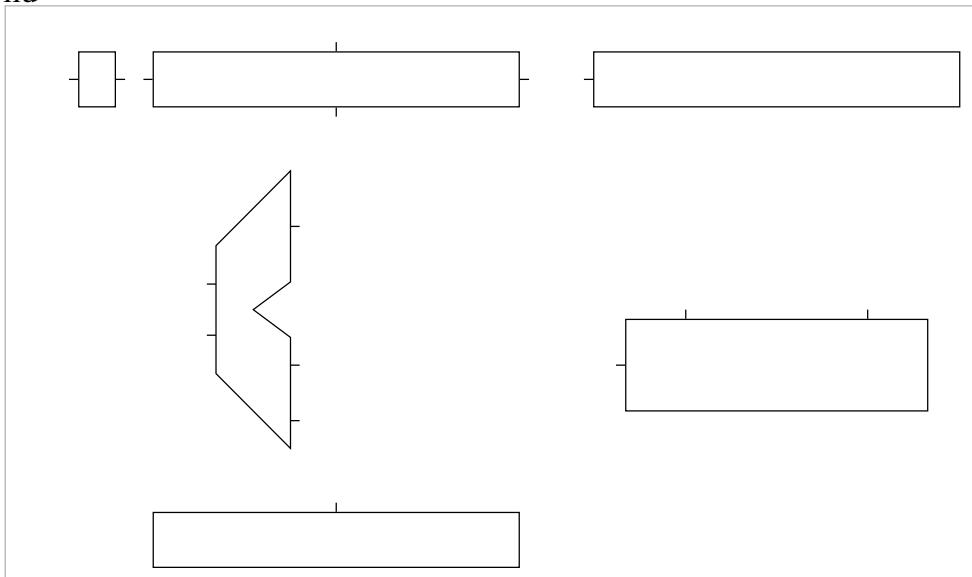
- „Aufgabe“
 $\vec{u} \cdot \vec{v} = \vec{p}$
- Variablen
 A, U, Q, C
- Hilfsvariable
 j

* Flussdiagramm

(Struktogramm s. S. 5.20)



* Schaubild



★ Weitere Möglichkeiten

- Addition mit 1-Bit Serien Addierer
 - mit $2N$ -Bit Addierer
 - 2 stufiges Netz
 - Array, Speicher
- jeweils unterschiedliche Hardware

5.3.2 Multiplikation von Zweierkomplement Zahlen

a) Algorithmus anpassen

mit negative Zahlen

Sei $v > 0, u < 0$

★ Formal

z.B.: $-6_D \cdot 5_D$

★ Algorithmus

Algorithmus anpassen

★ Schaubild

Steuerung modifizieren

b) Korrekturen

- gegeben: Binär Multiplikation
- Ansatz:
 - Vergleich von Binär- u. Zweier Komplement Multiplikation
 - Unterschiede → Korrekturen feststellen
- Umsetzung
 - Binär-Multiplikation mit anschließenden Korrekturen

Multiplikation

Positive Binärzahlen im Stellenwertsystem

$$A = a_{N-1} \cdot 2^{N-1} + \sum_{i=0}^{N-2} a_i \cdot 2^i \text{ und } B = b_{M-1} \cdot 2^{M-1} + \sum_{i=0}^{M-2} b_i \cdot 2^i$$

Multiplikation:

$$A \cdot B = a_{N-1} 2^{N-1} \cdot b_{M-1} 2^{M-1} + a_{N-1} \cdot 2^{N-1} \cdot \sum_{i=0}^{M-2} b_i \cdot 2^i + b_{M-1} \cdot 2^{M-1} \cdot \sum_{i=0}^{N-2} a_i \cdot 2^i + \sum_{i=0}^{N-2} a_i \cdot 2^i \cdot \sum_{i=0}^{M-2} b_i \cdot 2^i$$

Zweierkomplement Zahlen

$$C = -c_{N-1} \cdot 2^{N-1} + \sum_{i=0}^{N-2} c_i \cdot 2^i \text{ und } D = -d_{M-1} \cdot 2^{M-1} + \sum_{i=0}^{M-2} d_i \cdot 2^i$$

Multiplikation:

$$C \cdot D = c_{N-1} 2^{N-1} \cdot d_{M-1} 2^{M-1} - c_{N-1} \cdot 2^{N-1} \cdot \sum_{i=0}^{M-2} d_i \cdot 2^i - d_{M-1} \cdot 2^{M-1} \cdot \sum_{i=0}^{N-2} c_i \cdot 2^i + \sum_{i=0}^{N-2} c_i \cdot 2^i \cdot \sum_{i=0}^{M-2} d_i \cdot 2^i$$

Vergleich von $A \cdot B$ mit $C \cdot D$

Werden Zweierkomplement Zahlen mit Rechenwerken für positive Zahlen multipliziert, so ergibt sich für

- a) C, D positiv, d.h. $c_{N-1} = 0$ und $d_{M-1} = 0$:

Richtige Ergebnis

- b) C negativ und D positiv, d.h. $c_{N-1} = 1$ und $d_{M-1} = 0$:

Ergebnis um $2 \cdot 2^{N-1} \cdot \sum_{i=0}^{M-2} d_i \cdot 2^i$ zu groß d.h. Fehler von $2^N \cdot D$

- c) C positiv und D negativ, d.h. $c_{N-1} = 0$ und $d_{M-1} = 1$:

Ergebnis um $2 \cdot 2^{M-1} \cdot \sum_{i=0}^{N-2} c_i \cdot 2^i$ zu groß d.h. Fehler von $2^M \cdot C$

- d) C, D negativ, d.h. $c_{N-1} = 1$ und $d_{M-1} = 1$:

Ergebnis um Fehler von b) plus Fehler von c) zu groß

Folgerung für den Ablauf:

1.) Zweierkomplement Zahlen wie positive Binärzahlen multiplizieren,

2.) wenn Multiplikand negativ,

dann Multiplikator um Stellenzahl des Multiplikanden verschieben und vom Ergebnis abziehen
bzw. dann Multiplikator linksbündig vom Ergebnis abziehen

3.) wenn Multiplikator negativ,

dann Multiplikand um Stellenzahl des Multiplikators verschieben und vom Ergebnis abziehen
bzw. dann Multiplikanden linksbündig vom Ergebnis abziehen

5.3.3 Division ganzer positiver Zahlen

★ Formal

$$u/v = q ; r$$

Beispiel Algorithmen:

Restoring

→ Vorgehen:

Division durch wiederholte gewichtete

Subtraktion

z.B.: 7 / 3

★ Flussdiagramm siehe Seite 5.16
(Struktogramm s. S. 5.20)

non Restoring

★ Flussdiagramm siehe Seite 5.17
(Struktogramm s. S. 5.20)

5.3.4 Assemblerbefehle

(8051)

★ Multiplikation

MUL AB

(A) ← LowByte (A·B)

(B) ← HighByte (A·B)

beeinflusst: C, OV

★ Division

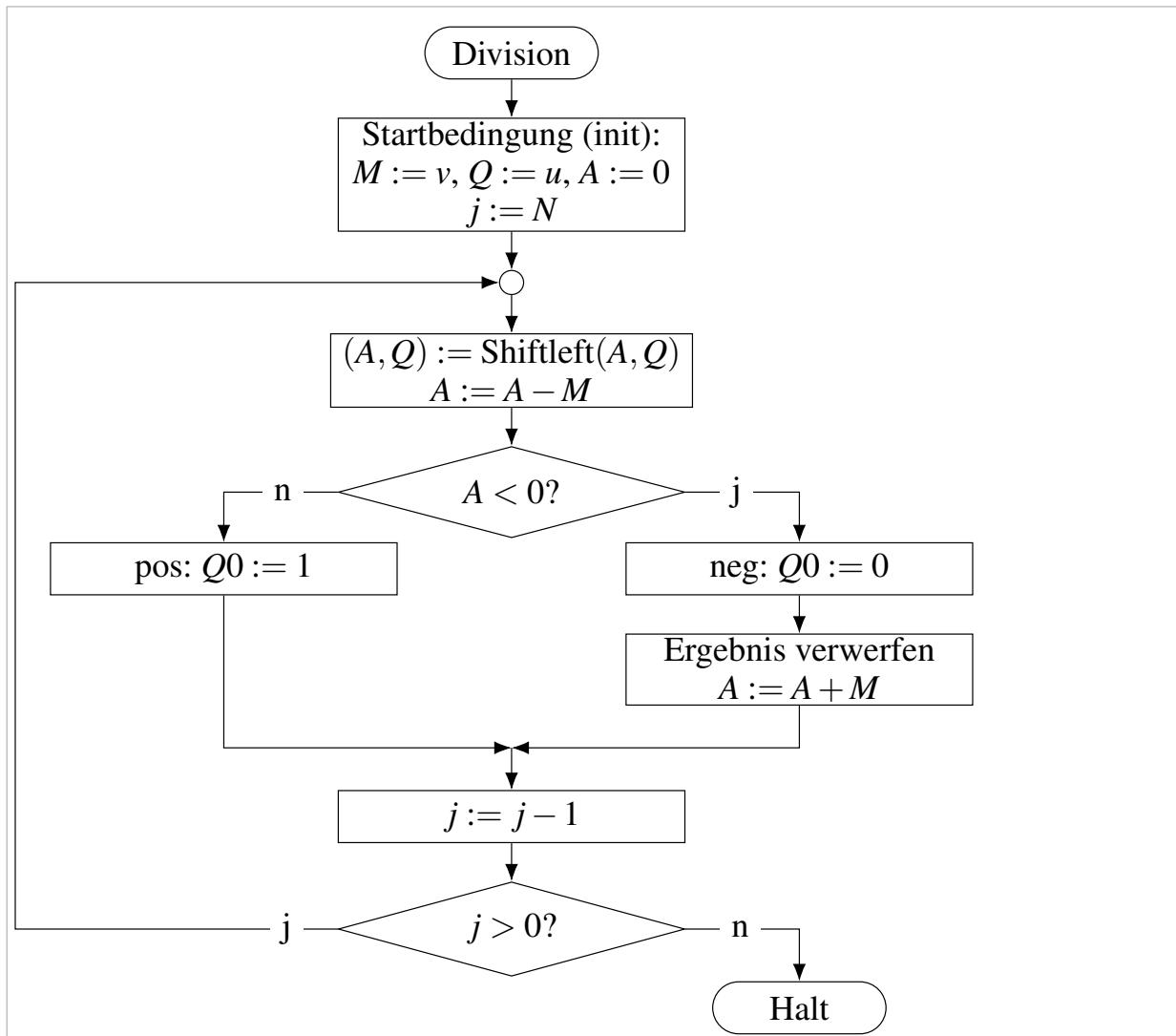
DIV AB

(A) ← Quotient(A / B)

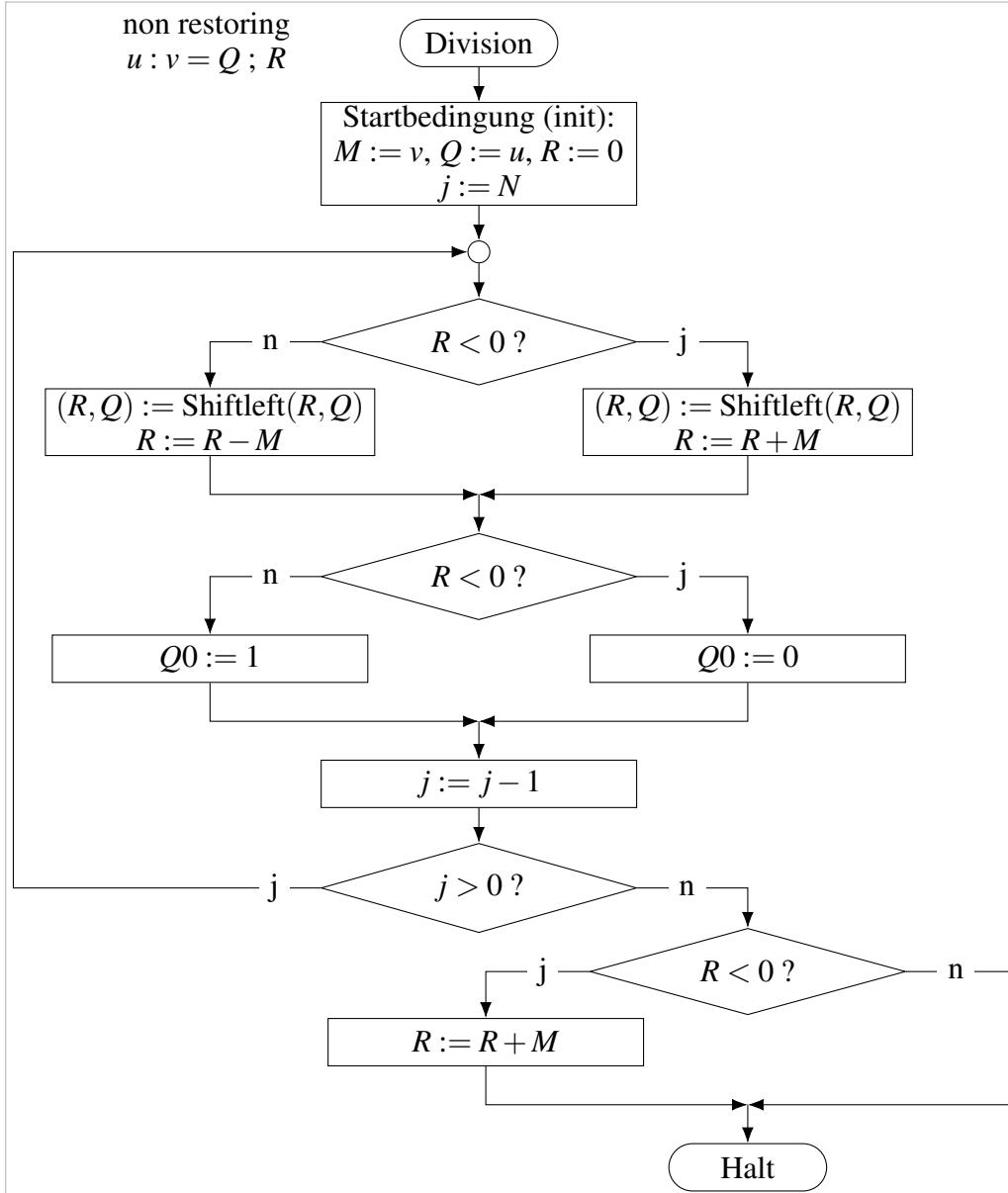
(B) ← Rest (A / B)

beeinflusst: C, OV

Flussdiagramm: Restoring Algorithmus



Flussdiagramm: Non Restoring Algorithmus



5.4 Gleitkomma Arithmetik

5.4.1 Gleitkomma Addition (Subtraktion)

* Allgemein

Exponentielle Darstellung

$$A = M_A \cdot 2^{E_A}$$

$$A \stackrel{+}{(-)} B$$

* Vorgehen

- Exponenten gleichsetzen
- Mantissen verarbeiten
- Normieren

5.4.2 Gleitkomma Multiplikation (Division)

* Allgemein

Exponentielle Darstellung

$$U = M_U \cdot 2^{E_U}; V = \dots$$

$$U \cdot V = P(Q)$$

* Vorgehen

- Exponenten verrechnen
Mantissen verrechnen
 - Normieren
- * Hinweise:
Blöcke zu Funktionen

5.5 Arithmetik-Anhang

5.5.1 ALU-IC

[TI88, Seite 1]

SN54LS381A, SN54S381, SN74LS381A, SN54LS382A, SN74LS382A, SN74S381 ARITHMETIC LOGIC UNITS/FUNCTION GENERATORS

SDLS168 – JANUARY 1981 – REVISED MARCH 1988

PIN DESIGNATIONS		
DESIGNATION	PIN NOS.	FUNCTION
A3, A2, A1, A0	17, 19, 1, 3	WORD A INPUTS
B3, B2, B1, B0	16, 18, 2, 4	WORD B INPUTS
S2, S1, S0	7, 6, 5	FUNCTION-SELECT INPUTS
C _n	15	CARRY INPUT FOR ADDITION, INVERTED CARRY INPUT FOR SUBTRACTION
F3, F2, F1, F0	12, 11, 9, 8	FUNCTION OUTPUTS
\bar{P} ('LS381A 'S381 ONLY)	14	ACTIVE-LOW CARRY PROPAGATE OUTPUT
\bar{G} ('LS381A 'S381 ONLY)	13	ACTIVE-LOW CARRY GENERATE OUTPUT
C _{n+4} ('LS382A ONLY)	14	RIPPLE-CARRY OUTPUT
OVR ('LS382A ONLY)	13	OVERFLOW OUTPUT
VCC	20	SUPPLY VOLTAGE
GND	10	GROUND

- Fully Parallel 4-Bit ALUs in 20-Pin Package for 0.300-Inch Row Spacing
- Ideally Suited for High-Density Economical Processors
- 'LS381A and 'S381 Feature \bar{G} and \bar{P} Outputs for Look-Ahead Carry Cascading
- 'LS382A Features Ripple Carry (C_{n+4}) and Overflow (OVR) Outputs
- Arithmetic and Logic Operations Selected Specifically to Simplify System Implementation:
 - A Minus B
 - B Minus A
 - A Plus B
 - and Five Other Functions

description

The 'LS381A, 'S381 and 'LS382A are low-power Schottky and Schottky TTL arithmetic logic units (ALUs)/function generators that perform eight binary arithmetic/logic operations on two 4-bit words as shown in the function table. The exclusive-OR, AND, or OR function of the two Boolean variables is provided without the use of external circuitry. Also, the outputs can be cleared (low) or preset (high) as desired. The 'LS381A and 'S381 provide two cascade outputs (\bar{P} and \bar{G}) for expansion utilizing SN54S182/SN74S182 look-ahead carry generators. The 'LS382 provides a C_{n+4} output to ripple the carry to the C_n input of the next stage. The 'LS382A detects and indicates two's complement overflow condition via the OVR output. The overflow output is logically equivalent to C_{n+3} ⊕ C_{n+4}. When the 'LS382A is cascaded to handle word lengths longer than four bits in length, only the most significant overflow (OVR) output is used.

The SN54' family is characterized for operation over the full military temperature range of -55°C to 125°C. The SN74' family is characterized for operation from 0°C to 70°C.

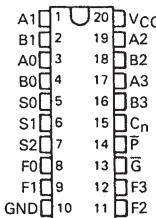
SN54LS381A, SN54S381

... J OR W PACKAGE

SN74LS381A, SN74S381

... DW OR N PACKAGE

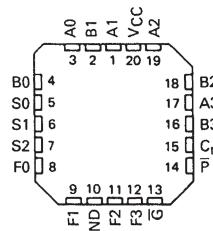
(TOP VIEW)



SN54LS381A, SN54S381

... FK PACKAGE

(TOP VIEW)



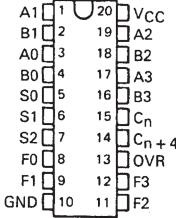
SN54LS382A ...

J OR W PACKAGE

SN74LS382A ...

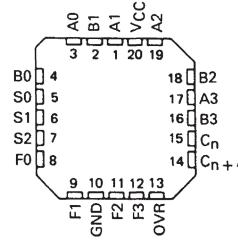
DW OR N PACKAGE

(TOP VIEW)



SN54LS382A ... FK PACKAGE

(TOP VIEW)



FUNCTION TABLE

SELECTION	ARITHMETIC/LOGIC OPERATION
S2 S1 S0	
L L L	CLEAR
L L H	B MINUS A
L H L	A MINUS B
L H H	A PLUS B
H L L	A \oplus B
H L H	A + B
H H L	AB
H H H	PRESET

H = high level, L = low level

PRODUCTION DATA information is current as of publication date. Products conform to specifications per the terms of Texas Instruments standard warranty. Production processing does not necessarily include testing of all parameters.

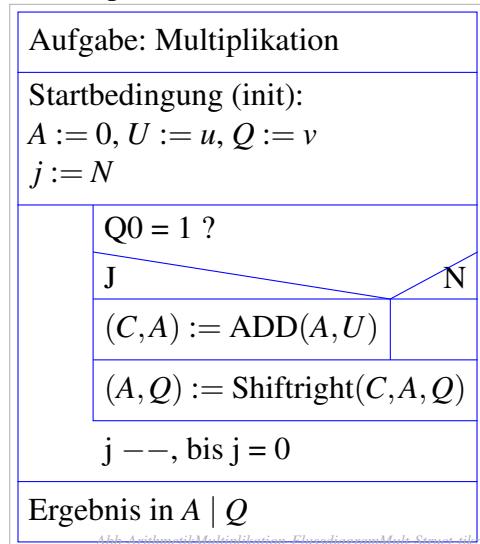
Copyright © 1988, Texas Instruments Incorporated



5.5.2 Multiplikation, Division

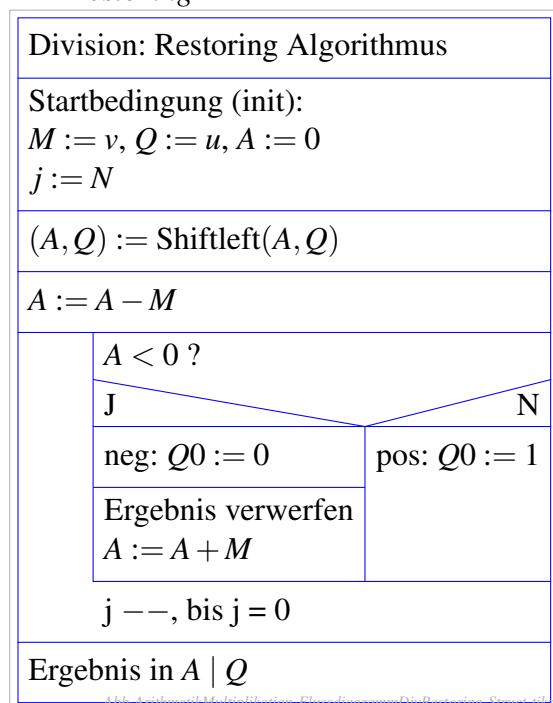
Struktogramme der Algorithmen

* Multiplikation

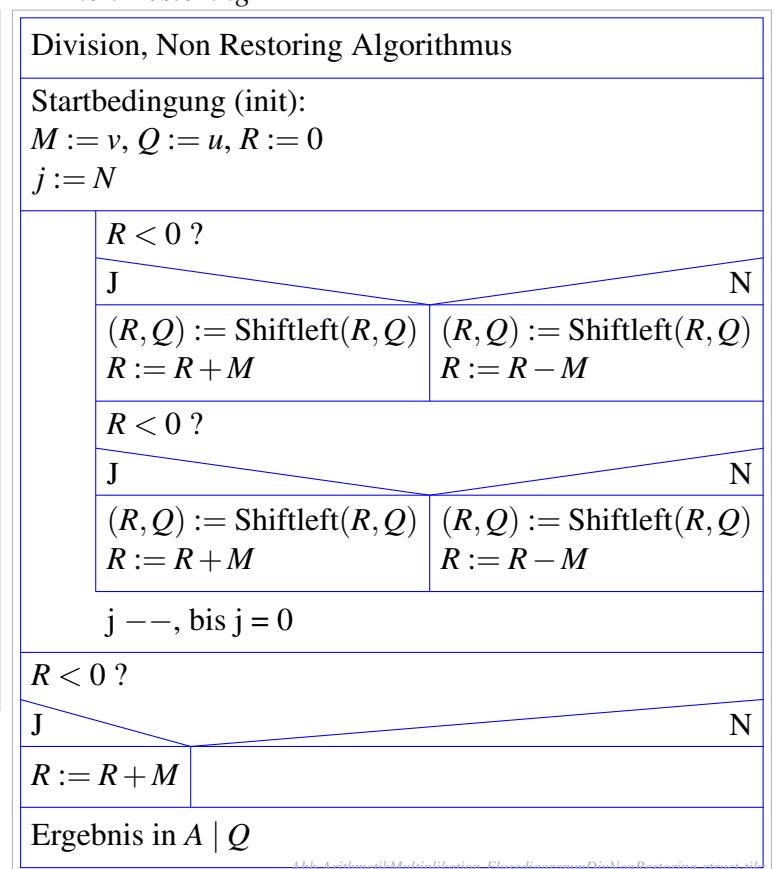


* Division

- Restoring



- non Restoring



5.6 Quellen zu Kap. 5

- [AD96] Analog Devices. “ADSP-2100 Family User’s Manual”. Rev 3.0. Analog Devices Inc., 1996. URL: www.analog.com.
- [TI88] TI. “SN54LS381A, SN54S381, SN74LS381A, SN54LS382A, SN74LS382A, SN74S381. Arithmetic Logic Units/Function Generators”. SDLS168. Texas Instruments Incorporated, 1988.

6 Bussystem

- Einordnung: Minimalsystem → Bus

6.1 Allgemein

- *Aufgabe:*
 - Verbindung von/zwischen Geräten
 - Zusammenfassung von Leitungen/Signalen
- *Darstellung:*
 - ¬ Schematisch [DIN60617-3]

- ¬ Zeitdiagramm

6.2 Einteilung

6.2.1 nach Funktion

im Minimalsystem

✳ Datenbus

- Übertragung von Werten
- Bitmuster

✳ Adressbus

- Nummer der Speicherstelle
(Adresse)
- Auswahl
 - ¬ "Adresse" (Gruppe von Ltg./Bits)
 - ¬ "direkt" (eine Ltg. pro Gerät)

✳ Steuerbus

- Steuerung des Ablaufes
 - ¬ Datenflussrichtung (read, write)
 - ¬ Daten-/ Adressgültigkeit (enable)

✳ Systembus

Daten- + Adr.- + Strg.- Bus

+ MULTIPLEX

(hier) Information im Zeit-Multiplex

Bsp. Adressen ↔ Daten

* Ablauf:

6.2.2 nach Richtung

- * *unidirektional* (simplex)
 - Informationsfluss in eine Richtung
- * *bidirektional* (duplex)
 - Informationsfluss in beide Richt.
 - Ausführung
 - halbduplex
 - vollduplex

* BSP.

- Systembus
 - Daten- B.: bidirekt., halbduplex
 - Adress- B.: unidirekt
 - Strg.- B.: (meist) unidirekt
- RS232: bidirektional, vollduplex
- RS485: bidirektional, halbduplex
- RS422: bidirektional, vollduplex

6.2.3 nach Übertragungsart

- * *seriell*
 - eine Leitung
 - N-Bit pro Ltg. und Zeitschritt (T)
 - (meist $N = 1$)

* *parallel*

- M-Leitungen parallel
- meist 1 Bit pro Ltg.

* BSP.

- Systembus
 - Daten, Adressen : parallel
 - Strg. : parallel
- RS232, RS485, RS422: seriell, $N = 1$
- Ethernet: seriell, $N = 64$

* HINWEISE Schlagworte:

- Seriell ist schneller als Parallel (?)
- Parallel ist auch Seriell
 - (Byte-seriell, Bit-parallel)

6.2.4 nach Synchronisation

zeitliche Absprache

zwischen Sender u. Empfänger

Wann sind Werte gültig?

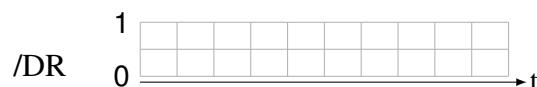
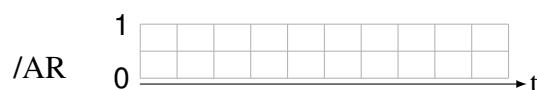
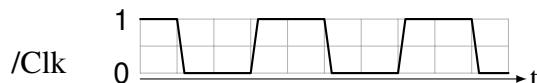
Wann beginnt Übertragung/Datenpaket ?

• Unterscheidungen

- *Takt*
- ¬ *synchron*
gemeinsamer Takt/Clock
- ¬ *asynchron*
ohne Takt
- *Erkennung*
- ¬ *Handshake*
 - * *ohne Rückmeldung*
 - * *mit Rückmeldung*
 - ¬ *Bit-Synchronisation*

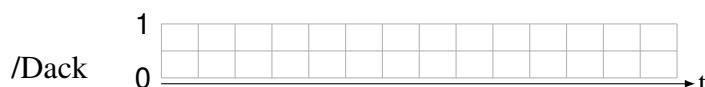
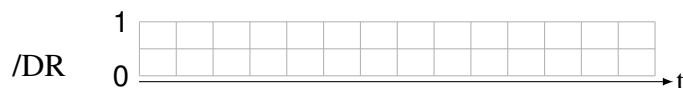
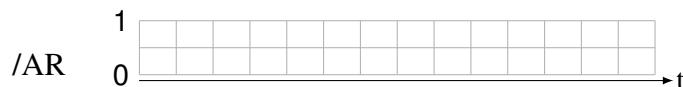
• Beispiele

- "Speicherzugriff"
(z.B. [Fli05, S.317])
- ¬ *asynchron*
jedoch synchronisiert
- * *ohne Rückmeldung*



z.B. 8051: Bilder: siehe Seite 6.7

- * *mit Rückmeldung*



variable Zeit:

Wait-States, Wartezyklen

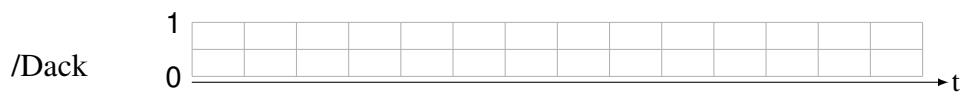
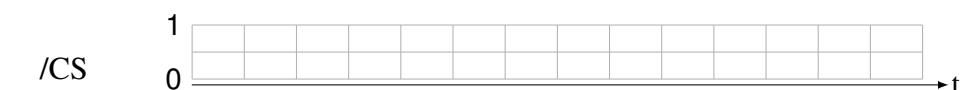
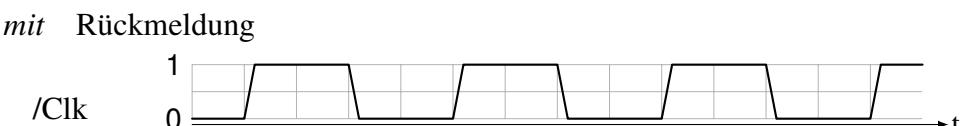
¬ *synchron*

jedoch mit Chip-Selekt (Handshake)

* *ohne Rückmeldung*



* *mit Rückmeldung*



○ *IEC, HPIB, GPIB, IEEE 488*

asynchron,

Handshake mit Rückmeldung

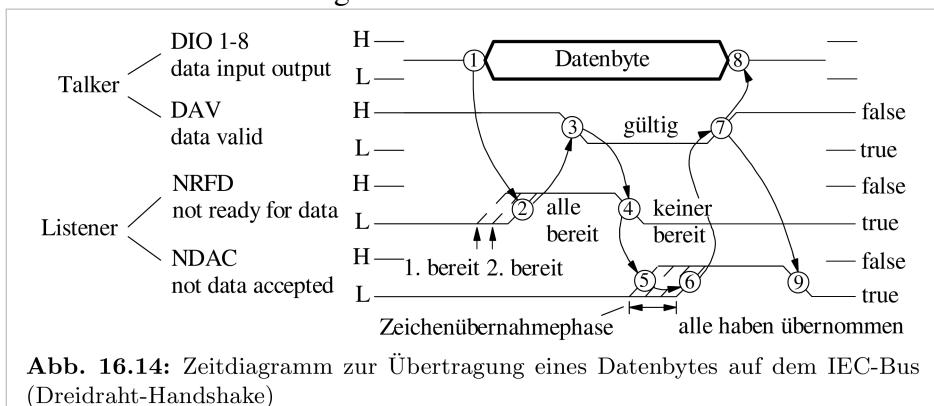


Abb. 16.14: Zeitdiagramm zur Übertragung eines Datenbytes auf dem IEC-Bus (Dreidraht-Handshake)

[Ler07]

6 Bussystem

- RS232
 - asynchron
 - *Hardware-Handshake*

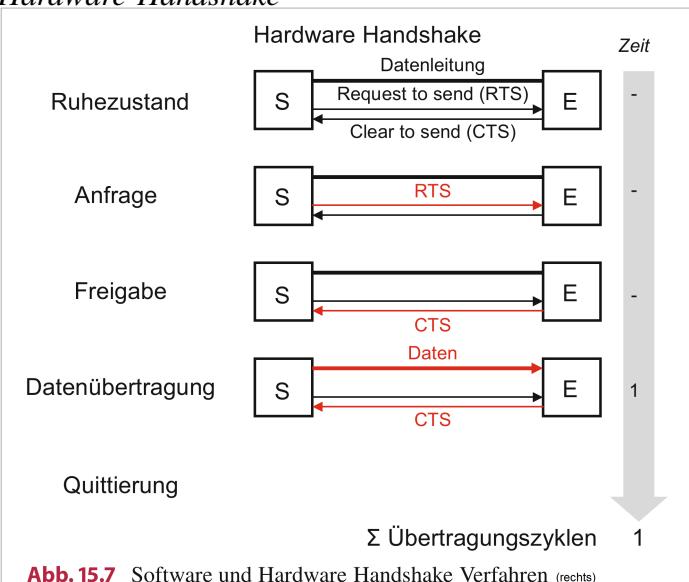


Abb. 15.7 Software und Hardware Handshake Verfahren (rechts)

[HBG17]

- | | | |
|------|----------------------|-------------------|
| DTR: | Data Terminal Ready, | Sender bereit |
| DSR: | Data Set Ready, | Empfänger bereit |
| RTS: | Request To Send, | Sende Anforderung |
| CTS: | Clear To Send, | Sende Bereit |

- *Software-Handshake*

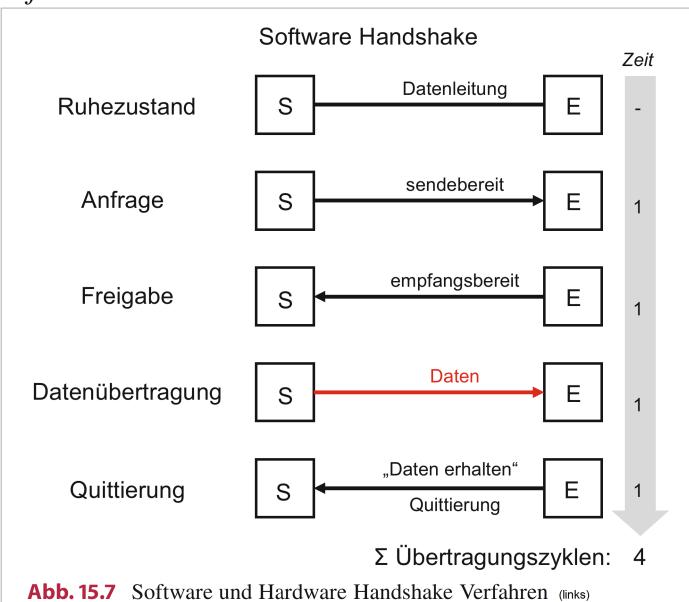


Abb. 15.7 Software und Hardware Handshake Verfahren (links)

[HBG17]

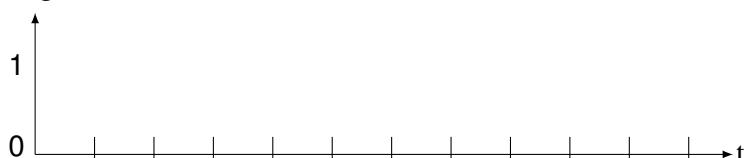
- * Software-Protokoll "Daten anhalten"
 - Xoff: Senden unterbrechen
 - Xon: Senden fortsetzen

- ¬ Bit-Synchronisation
Start der Übertragung,
Startbedingung: Fallende Flanke

Parameter:

- Startbit : '0', Anzahl: 1
- Datenbits : Anzahl Bits: 7; 8
- Stoppbit : '1', Anzahl: 1; 1,5; 2
- Baudrate: 4800 Bd, 9600 Bd, ...
- Parität: keine, gerade, ungerade

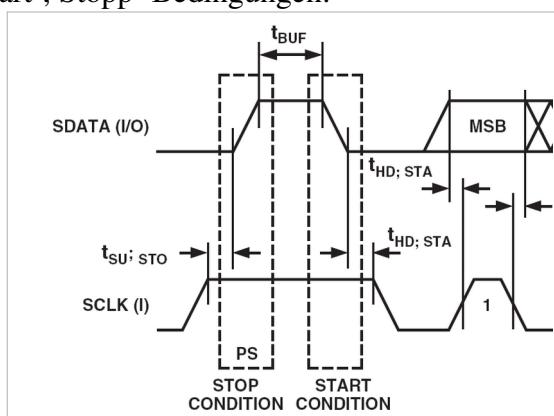
Zeitdiagramm



○ I2C

- Bitübertragung : synchron
- Paketanfang: asynchron
- Leitungen: Daten: SDA
- Takt: SCL

Start-, Stopp- Bedingungen:



Speicher Zugriff 8051

[Inf00]

Programm Speicher

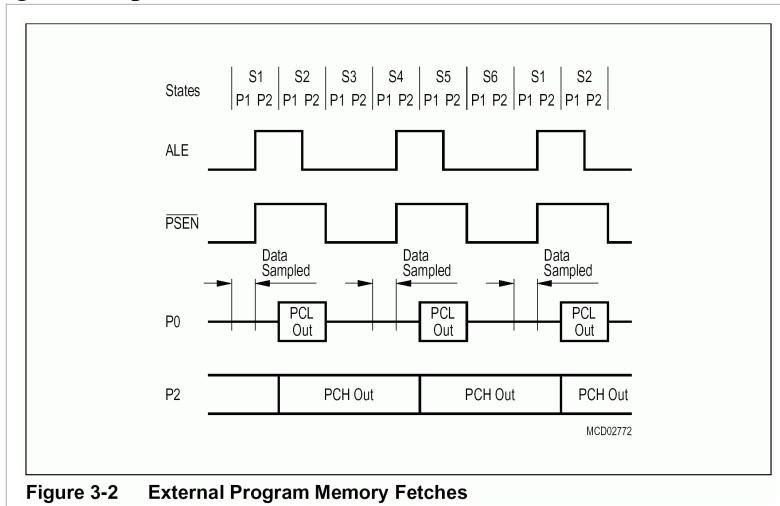


Figure 3-2 External Program Memory Fetches

Daten Speicher

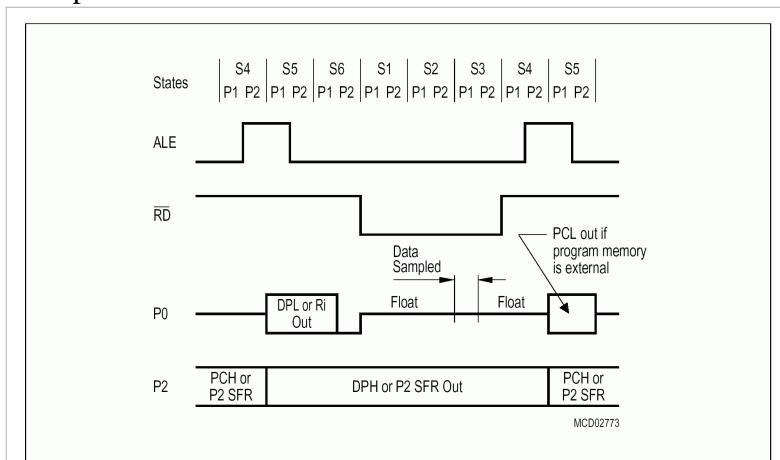


Figure 3-3 External Data Memory Read Cycle

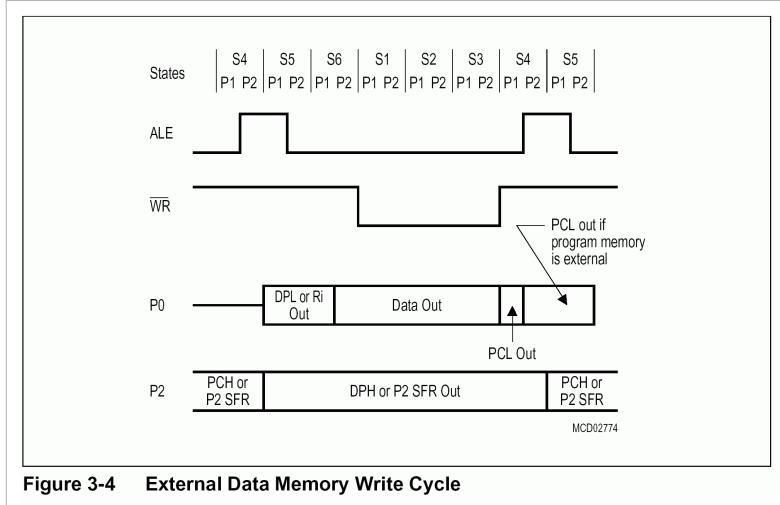


Figure 3-4 External Data Memory Write Cycle

Legende:

- State: Phase (P) entspricht Oszillator Takt
 PSEN: Program Select Enable
 WR: Write Data
 PCL/H: Program Counter Low- bzw. High-Byte

- ALE: Address Latch Enable
 RD: Read Data
 P0 / 2: Port 0 bzw. 2

6.2.5 nach Einsatzgebiet

• Feldbusse für Industrie-Automatisierung

CAN-Bus

Profi-Bus

Inter-Bus

• PC-Bus

- ISA Daten: 8 Bit
16 Bit

Adr.: 24 Bit

Takt: 8 MHz

- EISA Daten/Adr.: 32 Bit

Takt: 8 / 33 MHz

- MCA Daten/Adr.: 32 Bit

Takt: 33 MHz

- PCI Daten/Adr.: 32 Ltg.

+ optional 64 Daten/Adr. Ltg.

Takt: 33 MHz (66 MHz)

Strg.: über Befehle

- PCI-e Daten/Adr.: serielle, volldublex

Takt: 1250 MHz

• PC-Grafik

- VESA-Local Bus

EISA + VESA Rechner

- AGP

PCI + AGP Rechner

1.0: 66 MHz, 32 Bit

2.0: 2x, 4x Modus

• Industrie PC

- Compact PCI

wie PCI für 19"

- VME-Bus Daten: 32 Bit

Adr.: 64 Bit

Multiplex

- PC 104

Module

6.2.6 nach Medium

- * *elektrisch*
 - leitungsgebunden
 - freiraum
- * *optisch*
 - leitungsgebunden
 - freiraum

6.2.7 nach Topologie

- * *physikalisch*
 - Bus
 - Punkt zu Punkt

* *logisch*

- Bus
- Stern
- Baum
- Maschen
- Gitter, Cube, Hyper Cube

6.3 Bus-Anhang

6.3.1 8051- Buszyklen

- Übersicht

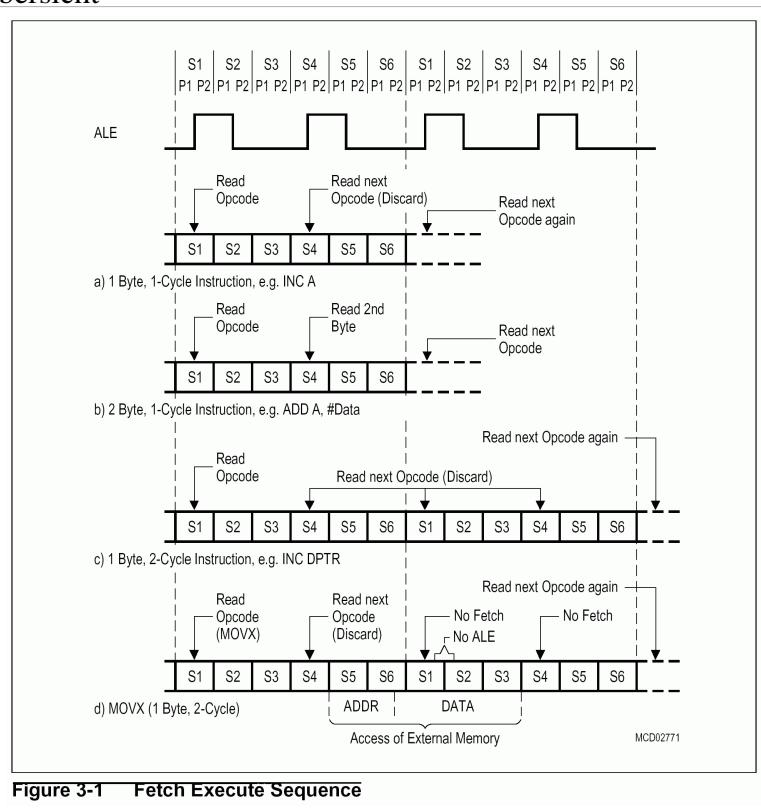
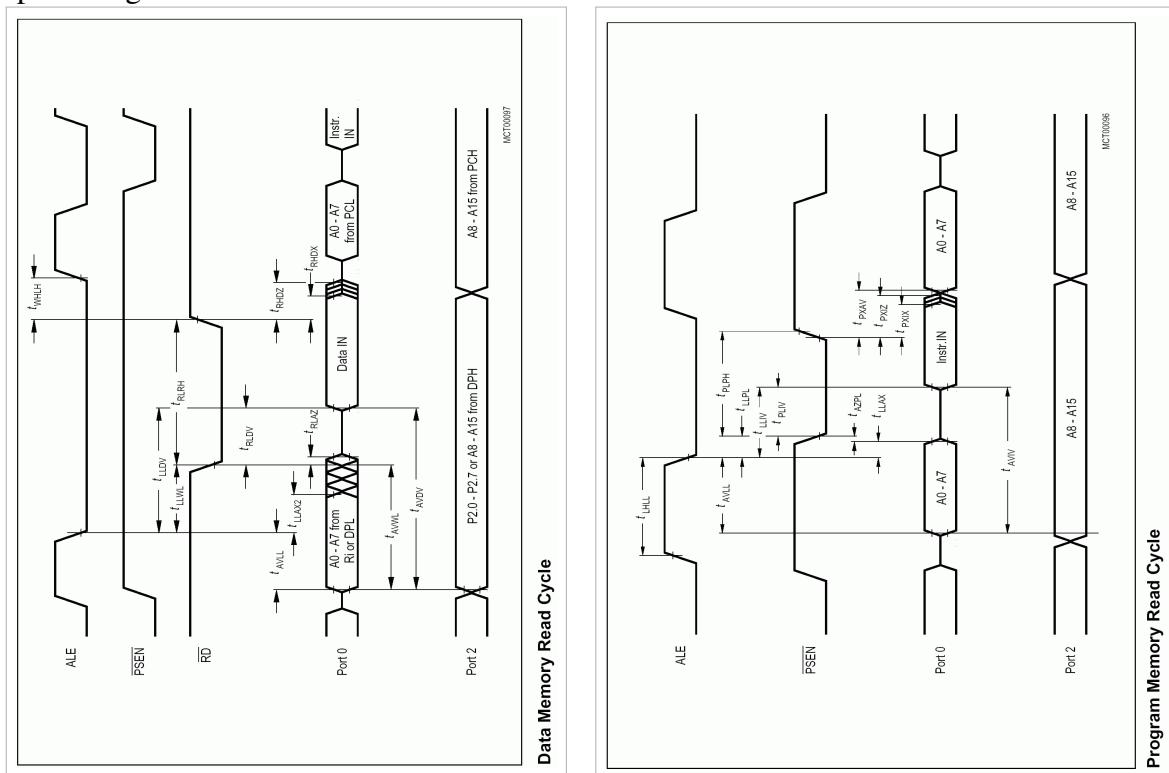


Figure 3-1 Fetch Execute Sequence

[Inf00]

- Bsp. Timing



6.4 Quellen zu Kap. 6

- [DIN60617-3] DIN. “Graphische Symbole für Schaltpläne. Teil 3: Schaltzeichen für Leiter und Verbinder”. DIN EN 60 617-3 : 1997. Ersatz für DIN 40900-3:1988-03; Übersetzung von IEC 617-3. DIN Deutsches Institut für Normung e.V., Berlin, Aug. 1997.
- [Fli05] Thomas Flik. “Mikroprozessortechnik und Rechnertechnik”. 7. Auflage. Springer, 2005. ISBN: 3-540-22270-7.
- [HBG17] Ekbert Hering, Klaus Bressler und Jürgen Gutekunst. “Elektronik für Ingenieure und Naturwissenschaftler”. 7. Auflage. Springer Vieweg, 2017. ISBN: 978-3-662-54213-2. DOI: 10.1007/978-3-662-54214-9.
- [Inf00] Infineon. “C500; Architecture and Instruction Set. User’s Manual”. 2000-07. Infineon Technologies AG, Juli 2000.
- [Ler07] Reinhard Lerch. “Elektrische Messtechnik. Analoge, digitale und computergestützte Verfahren”. 4. Auflage. Springer Berlin Heidelberg New York, 2007. ISBN: 978-3-540-73610-3.
- [Sim97] Simens. “C515A, 8-Bit CMOS Microcontroller. User’s Manual”. 08.97. Siemens AG, Bereich Halbleiter, Marketing-Kommunikation, 1997.

7 Rechenwerk

7.1 Allgemein

Rechenwerk = ALU + Register

7.2 Register

* Arbeitsregister

Akkumulator

* Allgemeine Register

Rn

* Spezial Register

- ¬ Status
- ¬ Stack
- ¬ Data Pointer
- ¬ Programm Zähler
- ¬ (Adress- Register)
- ¬ (General Purpose Register)

* Spezial Funktion Register

'51: im upper internal RAM

s. Seite [9.2](#)

○ Bezeichnung und Adresse

Table 2
Special Function Registers - Functional Blocks

Block	Symbol	Name	Address	Contents after Reset
CPU	ACC	Accumulator	E0H ¹⁾	00H
	B	B-Register	F0H ¹⁾	00H
	DPH	Data Pointer, High Byte	83H	00H
	DPL	Data Pointer, Low Byte	82H	00H
	PSW	Program Status Word Register	D0H ¹⁾	00H
	SP	Stack Pointer	81H	07H
Interrupt System	IE	Interrupt Enable Register	A8H ¹⁾	0X000000B ³⁾
	IP	Interrupt Priority Register	B8H ¹⁾	XX000000B ³⁾
Ports	P0	Port 0	80H ¹⁾	FFH
	P1	Port 1	90H ¹⁾	FFH
	P2	Port 2	A0H ¹⁾	FFH
	P3	Port 3	B0H ¹⁾	FFH
Serial Channel	PCON ²⁾	Power Control Register	87H	0XXX0000B ³⁾
	SBUF	Serial Channel Buffer Register	99H	XXH ³⁾
	SCON	Serial Channel Control Register	98H ¹⁾	00H
Timer 0 / Timer 1	TCON	Timer 0/1 Control Register	88H ¹⁾	00H
	TH0	Timer 0, High Byte	8CH	00H
	TH1	Timer 1, High Byte	8DH	00H
	TL0	Timer 0, Low Byte	8AH	00H
	TL1	Timer 1, Low Byte	8BH	00H
	TMOD	Timer Mode Register	89H	00H
Timer 2	T2CON	Timer 2 Control Register	C8H ¹⁾	00H
	T2MOD	Timer 2 Mode Register	C9H	XXXXXXXX0B ³⁾
	RC2H	Timer 2 Reload/Capture Register, High Byte	CBH	00H
	RC2L	Timer 2 Reload/Capture Register, Low Byte	CAH	00H
	TH2	Timer 2 High Byte	CDH	00H
	TL2	Timer 2 Low Byte	CCH	00H
	Pow. Sav. Modes	Power Control Register	87H	0XXX0000B ³⁾

1) Bit-addressable special function registers

2) This special function register is listed repeatedly since some bits of it also belong to other functional blocks.

3) "X" means that the value is undefined and the location is reserved

7.3 Beschreibung

durch

* *Hardware Beschreibung*

VHDL, Verilog

* *Funktions- Beschreibung*

RTL

ISA

7.4 Struktur

7.4.1 Direkt verdrahtet

Verbindung durch Schaltnetz

7.4.2 Bussystem

* *zwei Busse z.B.*

Ablauf: $(R1) \leftarrow (R1) + (R2)$

* *Varianten*

1, 2, 3 Busse

* *Hinweise*

nicht vermerkt:

Steuerung durch Steuerleitungen

7.5 Quellen zu Kap. 7

[Sie97]

Siemens. "Microcomputer Components, 8-Bit CMOS Microcontroller, C501. Data Sheet". 1997-04-01. Siemens AG, Bereich Halbleiter, Marketing Kommunikation, 1997.

8 Steuerwerk

8.1 Allgemein

* Aufgabe

- Steuerung der Steuerleitungen je nach Befehl

* Eingangssignale

- Instruktion | Befehl | Op-Code
- Statusflags | Statusmeldungen | Condition Codes
- (Takt)

* Ausgangssignale

- Steuersignale, Gating- Signale

8.2 Strukturen

8.2.1 fest-verdrahtetes Steuerwerk

* Allgemein

- Wahrheitstafel → Gl. → Schaltung
- ohne "Struktur"
- Eigenschaften
 - schnell („Echtzeit“)
 - unflexibel
 - Relais Steuerung ↔ direkt verdrahtet
 - SPS ↔ ?

8.2.2 Steuerwerk mit Mikroprogrammierung

- als endlicher Automat

- mit Schaltwerk

s. z.B. [PD80]

8.2.2.1 Programm Ansatz

* Allgemein

○ Abläufe in CPU zerlegen

→ μ -Operationen

○ μ -Operationen durchnummerieren

→ μ -Adressen

○ Analyse von Befehlsabfolgen

Sequenzen von μ -Operationen

viele Sprünge zw. Sequenzen

¬ Unterprogramm Struktur

○ μ -Op. mit μ -Adr. ergänzen

→ μ -Befehl :=

○ μ -Befehle bilden μ -Programm

○ μ -Programm steht im

μ -Programmspeicher

○ Adressierung durch:

μ -Adresse u. Ext. Signale

¬ Bild

8.2.2.2 Automaten Ansatz

★ Allgemein: Endlicher Automat

Finite State Maschine

★ Struktur

- Signale

Eingangsvariable: $\vec{x} \in X$

Ausgangsvariable: $\vec{y} \in Y$

Innerer Zustand: $\vec{q}, \vec{q}' \in Q$

- Aufbau

Schaltnetz

Zustands- Variablen- Speicher

- Schaltnetze

¬ für Zustände \vec{q}'

$$\lambda : X + Q \rightarrow Q \text{ bzw. } \vec{q}' = \lambda(\vec{x}, \vec{q})$$

¬ für Ausgangsvariable \vec{y}

★ Mealy:

$$\beta : X + Q \rightarrow Y \text{ bzw. } \vec{y} = \beta(\vec{x}, \vec{q})$$

oder (!)

★ Moore:

$$\beta : Q \rightarrow Y \text{ bzw. } \vec{y} = \beta(\vec{q})$$

★ Übergangsdiagramme

sei z.B. $X = \{0, 1\}$

$Y = \{\vec{y}_1, \vec{y}_2\}$

$Q = \{\vec{q}_1, \vec{q}_2\}$

★ Mealy

Übergangspfeile mit Ausgangswert

★ Moore

Zustand mit Ausgangswert

¬ Hinweis

- Zustände sind von Mealy / Moore abhängig
- bei Prozessoren kein Endzustand

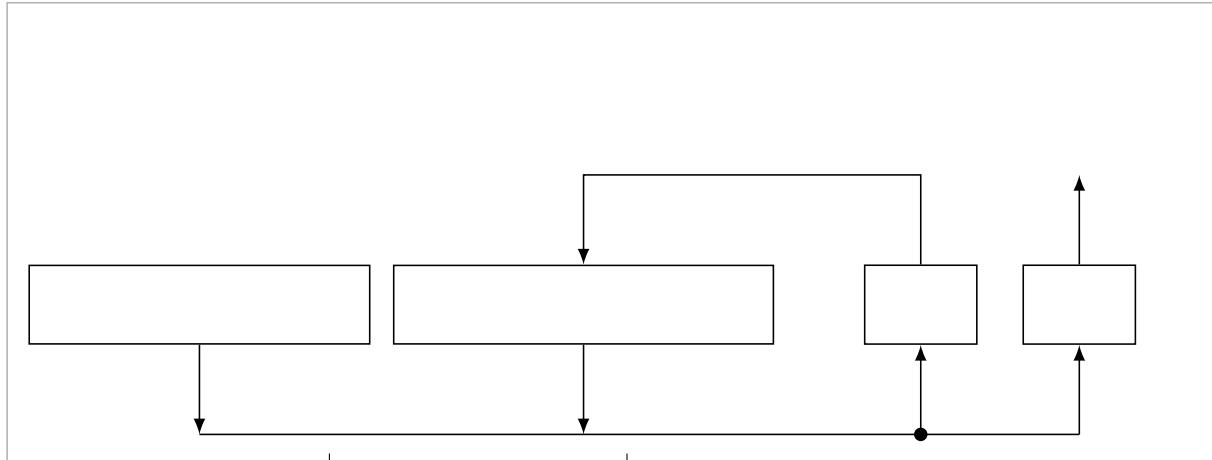
8.2.2.3 Wilkes-Stringer Steuerwerk

(1953¹) siehe auch S. 8.7

* Aufbau

- Instruktions- Register + Statusflags
- μ -Adress- Register
- Schaltnetze: β, λ

* Bild



* Bezeichnungen:

	Automat	Prozessor
\vec{q}	Innerer Zustand	
\vec{x}	Eingangs- Variable	
\vec{y}	Ausgangs- Variable	
	Zustands Variablen Speicher	
	β, λ	
$\vec{q}' + \vec{y}$		

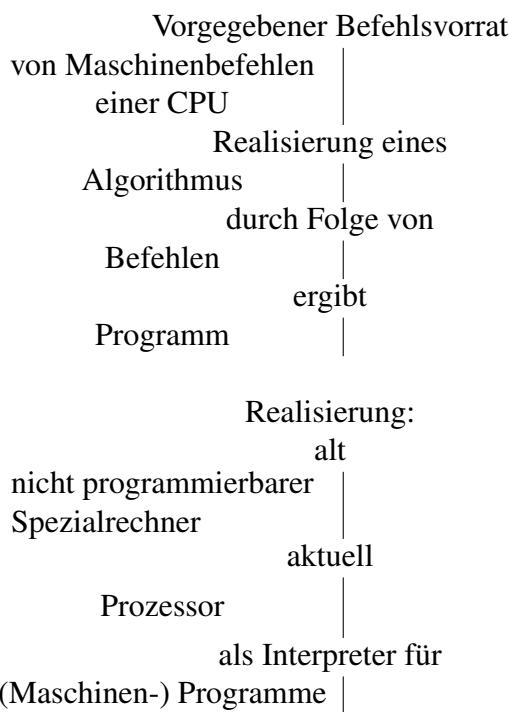
* Hinweise

- Sichtweise: Instruktion entspricht Einsprungs-/ Start- Adresse eines μ -Unterprogramms
- Austausch des μ -Befehlsspeichers
d.h. ändern des μ -Programms
Folge: ändern des (Assembler-) Befehlssatzes
- Hardware ändert sich,
jedoch gleiche (Maschinen-) Befehle
- nur μ -Befehls Satz austauschen
- μ -Befehle (i. A.) nicht vom Anwender zugänglich
- „Einfacher“ Entwurf von Steuerwerken
Folge: immer mehr und komplexere Befehle
- Original Bild von Wilkes-Stringer: s. Seite 8.7

¹Wilkes, M.V., Stringer, J. B.: Microprogramming and the design of the control circuits in an electronic digital computer.
Proc. Cambridge Phil. Soc., Vol. 49, pp. 230 - 238, April 1953

8.2.2.4 Vergleich zwischen Mikro- u. Maschinen- Programmierung

Maschinen Programm | μ -Programm



8.3 Darstellung der Steuersignale

* nicht codiert

- d.h. ein Gating Signal kontrolliert eine Funktion
- Bez.: horizontale μ -Programmierung

* (voll) codiert

- d.h. Gruppe von Signalen steuert Gruppe von Funktionen
- Bez.: vertikale μ -Programmierung
Nanocode

* andere Bezeichnungen

- teilweise codiert : Nanocode
- vollständig codiert : Picocode

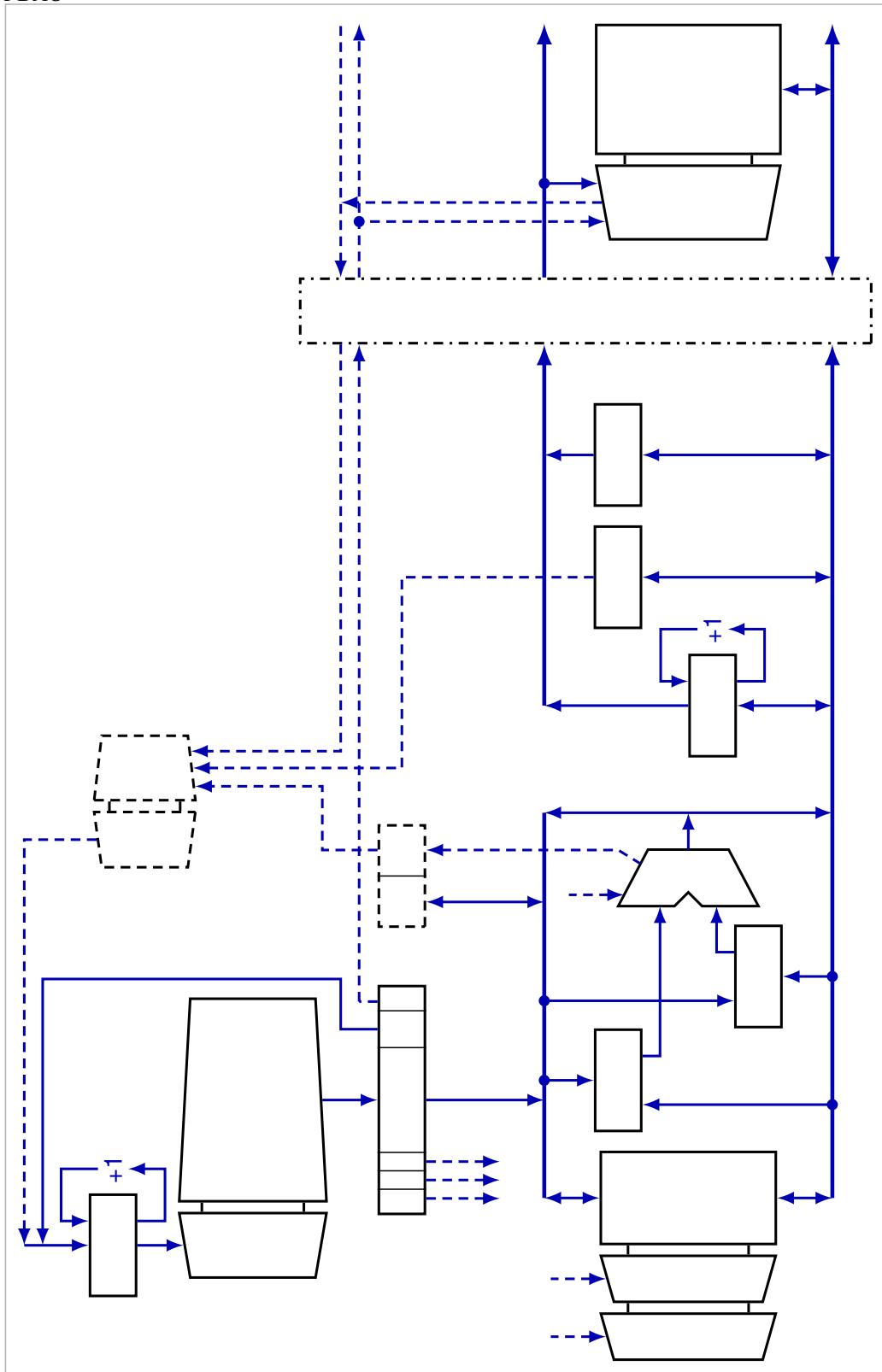
8.4 Beispielstruktur einer CPU

nach: Th. Flik; Mikroprozessortechnik

Springer Verlag, 6. Aufl., 2001

[Fli01]

- AUFBAU



8.5 Anhang zu Kap. 8

* M. V. Wilkes und J. B. Stringer:

[WS53]

Microprogramming and the design of the control circuits in an electronic digital computer. Proc. Cambridge Phil. Soc., Vol. 49, pp. 230 - 238, April 1953

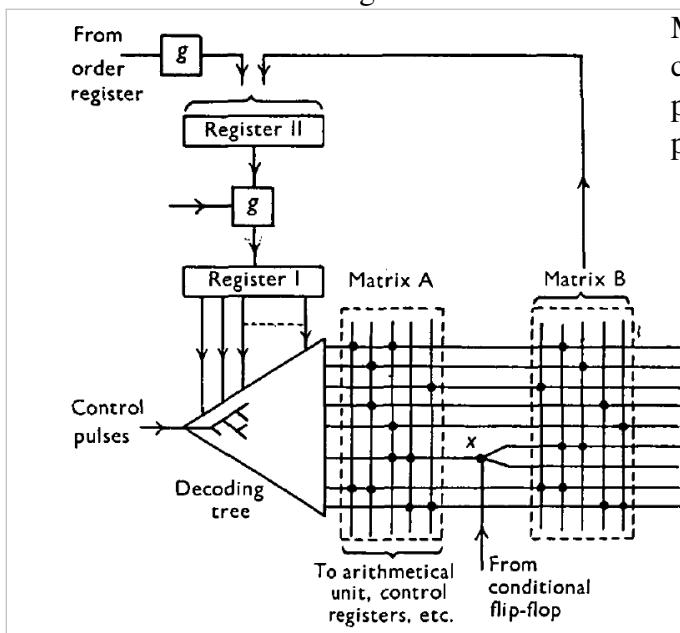


Fig. 1. Micro-control unit.

* ARM 7 (ARM7TDMI; Technical Reference Manual; Revision: r4p1)

[ARM04]

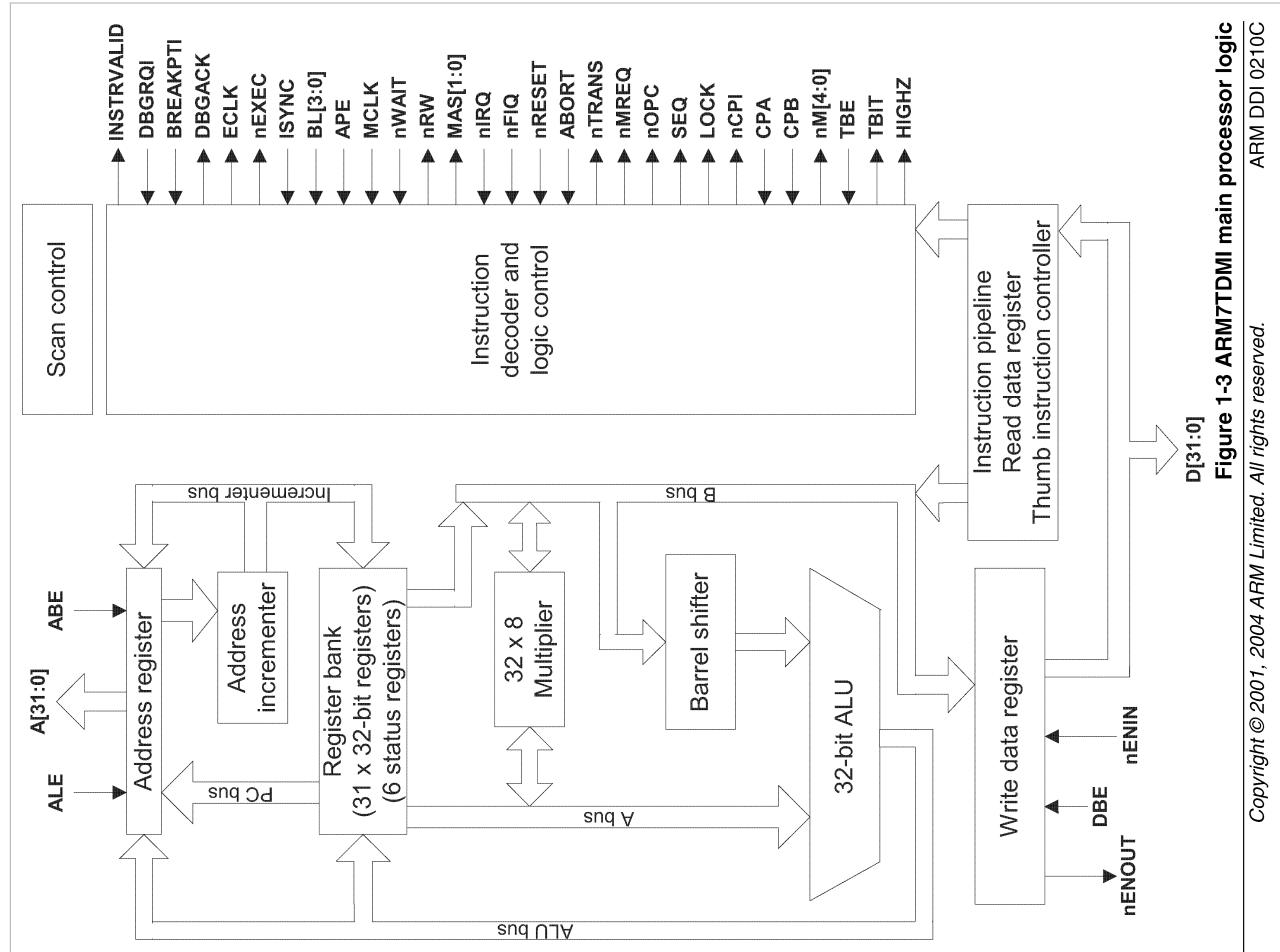


Figure 1-3 ARM7TDMI main processor logic
Copyright © 2001, 2004 ARM Limited. All rights reserved.
ARM DDI 0210C

8.6 Quellen zu Kap. 8

- [ARM04] ARM. “ARM7TDMI. Technical Reference Manual”. r4p1. [ARM DDI 0210C]. ARM Limited, 26. Nov. 2004.
- [Fli01] Thomas Flik. “Mikroprozessortechnik und Rechnertechnik”. 6. Auflage. Springer, 2001. ISBN: 3-540-42042-8.
- [PD80] D. A. Patterson und D. R. Ditzel. “The Case for the Reduced Instruction Set Computer”. In: “ComputerArchitecture News” 8 (6 15. Okt. 1980), S. 25–33.
- [WS53] M. V. Wilkes und J. B. Stringer. “Microprogramming and the design of the control circuits in an electronic digital computer”. In: “Mathematical Proceedings of the Cambridge Philosophical Society” 49 (2 Apr. 1953), S. 230–238. DOI: 10.1017/S0305004100028322.

9 Speicher

9.1 Adressräume

✖ *Allgemein*

„Speicher“ liefert Werte

✖ *Arten:*

Progr. Sp.

Daten Sp.

I/O Bereich

✖ *Unterscheidung*

per Software

per Hardware

✖ *Aufteilung, Adressierung*

- alles getrennt

- Daten u. Prog. gemeinsam,
I/O getrennt

- Daten u. I/O gemeinsam,
Prog. getrennt

Bez. :

- alles gemeinsam

✖ *Hinweis*

¬ *Sichtweise*

* *Hardware* Adressräume

* *Software* Segmente

Bsp. 8051

✖ *Arten*

- Code Speicher

- Externer Datenspeicher

- Interner Datenspeicher
incl. I/O (Ports)

✖ *Architektur*

- Code- u. Daten-Adressen getrennt

→ Harvard-Architektur

9 Speicher

• Darstellung Speicher Plan

- Übersicht

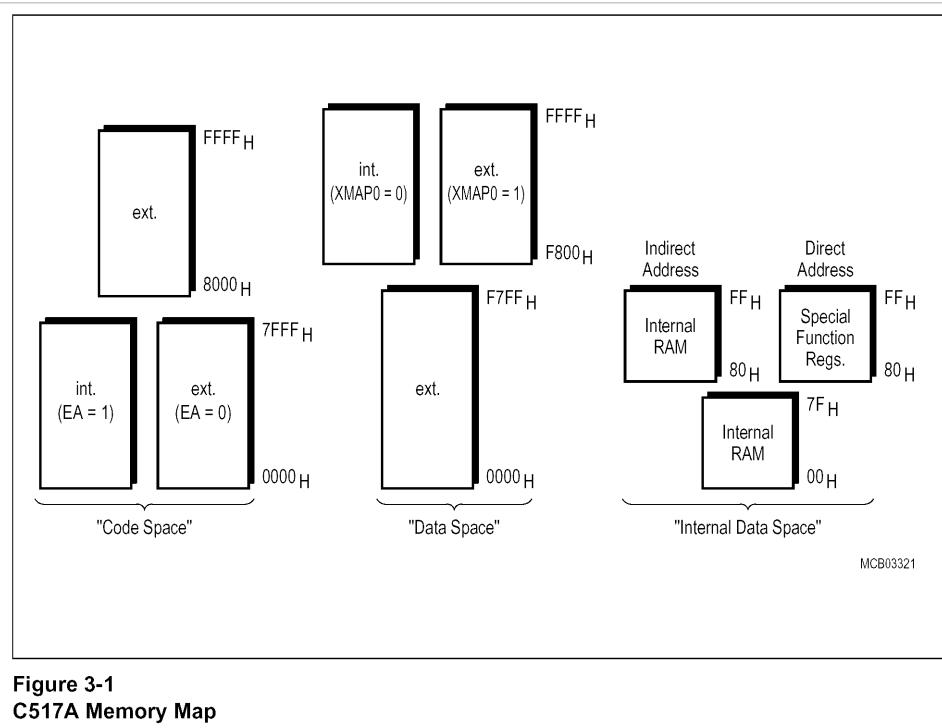


Figure 3-1
C517A Memory Map

[Sim99]

- Stuktur interner Datenspeicher

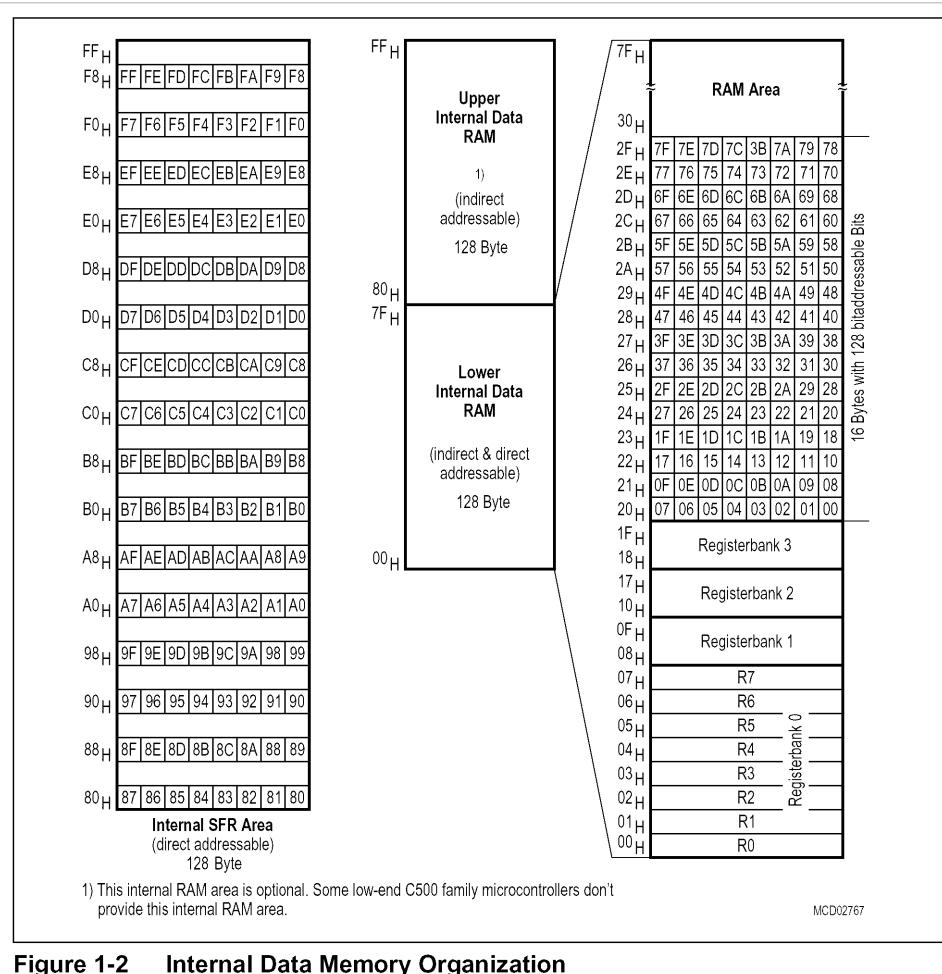


Figure 1-2 Internal Data Memory Organization

[Inf00]

9.2 Adressierung

* Einordnung

Minimalsystem

→ Speicher

→ Adressierung

9.2.1 Allgemein

* Zuordnung zwischen

CPU/Programm und Speicher

logischen Adressen und physikalischen Adr.

log. Speicher Bereichen und Speicher IC

CPU, Programmwerten und Speicherzellen

CPU-Anschlüsse und Speicheranschlüssen

* Anschlüsse

- *CPU*
 - Daten - Leitungen
 - Adress - Leitungen
 - Strg. - Leitungen: RD/WR
- *Speicher*
 - Daten - Leitungen
 - Adress - Leitungen
 - Strg. - Leitungen : RD/WR, CS

* Auswahl der Speicherzelle 2. stufig

- 1. Auswahl des IC durch CS
 - Bez.: Decoder außerhalb von CPU / Sp. IC
 - externe Adressierung
 - CS - Adressteil
- 2. Auswahl der Speicherzelle
 - Bez.: Decoder innerhalb von Sp. IC
 - interne Adressierung
 - direkter Adressteil
- *Zusammen* Aufteilung

Zwei Adressteile
CS- u. direkter Teil

* Bsp.

CPU: 16 Adr. Ltg.
Sp. IC: 12 Adr. Ltg.

9.2.2 Externe Dekodierung

9.2.2.1 vollständige Dekodierung

Alle Bits des CS-Teils werden ausgewertet

* Realisierung

○ *I aus N Decoder*

○ *CS-Decoder*

* Speicherzuordnung

○ *Startadresse*

Adr. des 1. Byte im IC
direkter Adressteil = 0

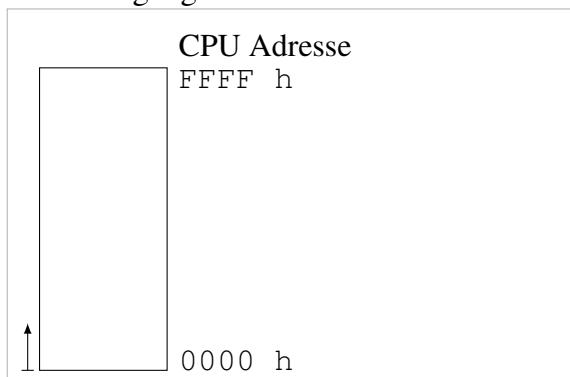
○ *Endadresse*

Adr. des letzten Byte im IC
direkter Adressteil = maximal

○ *Test*

Speichergröße = End Adr. - Start Adr. + 1

* Speicherbelegung



9.2.2.2 unvollständige Dekodierung

nicht alle Bits des CS-Teils werden ausgewertet

* Realisierung

- CS-Decoder

* Speicherzuordnung

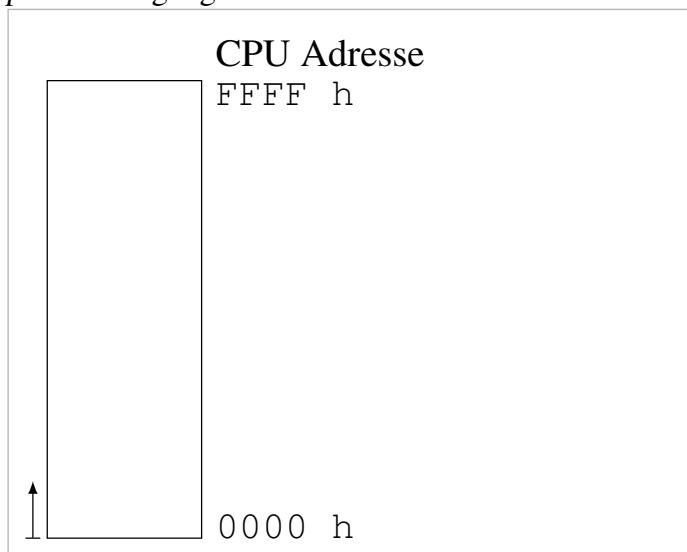
- Startadresse

direkter Adressteil = 0

- Endadresse

direkter Adressteil = Maximum

* Speicherbelegung



* Hinweis Folge:

- * keine eindeutige Zuordnung zw. log. u. phys. Adr.
- * vgl.
Assoziativer-Cache,
Hash-Algorithmen

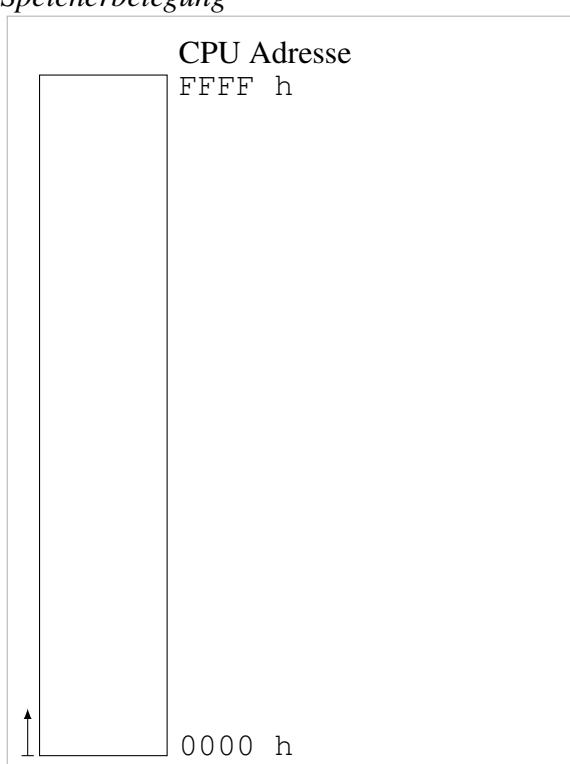
9.2.2.3 Beispiel

✖ Geg. :

- CPU:
64 K Byte adressierbar
- Speicher
 - IC1, IC2: 2 K Byte
mit Startadresse IC1 = 1000 h
mit Startadresse IC2 = 9000 h
 - IC3: 4 K Byte
mit $CS3 = \overline{A15} \wedge A14 \wedge \overline{A13}$

✖ Ges. :

- Anzahl der Leitungen
- letzte Speicher Adr.
- Decoder für IC1, IC2
- Start- u. End-Adresse(n) für IC3
- Speicherbelegung



9.2.3 Interne Adressierung

9.2.3.1 Allgemein

Auswahl der Speicherzelle im Speicherbaustein

¬ *Speicherzelle*:

- ★ 1 Bit (oder 1 Byte, 1 Wort)

★ Anschlüsse:

Selekt, R/W, Daten In, Daten Out

¬ *Speicherbaustein*

- ★ N Bit (oder N Byte)

★ Anschlüsse:

- Steuerleitungen: Chip Selekt, R/W
- Adressleitungen: $\text{ld}(N)$
- Datenleitungen: je Wortbreite

9.2.3.2 interner Aufbau

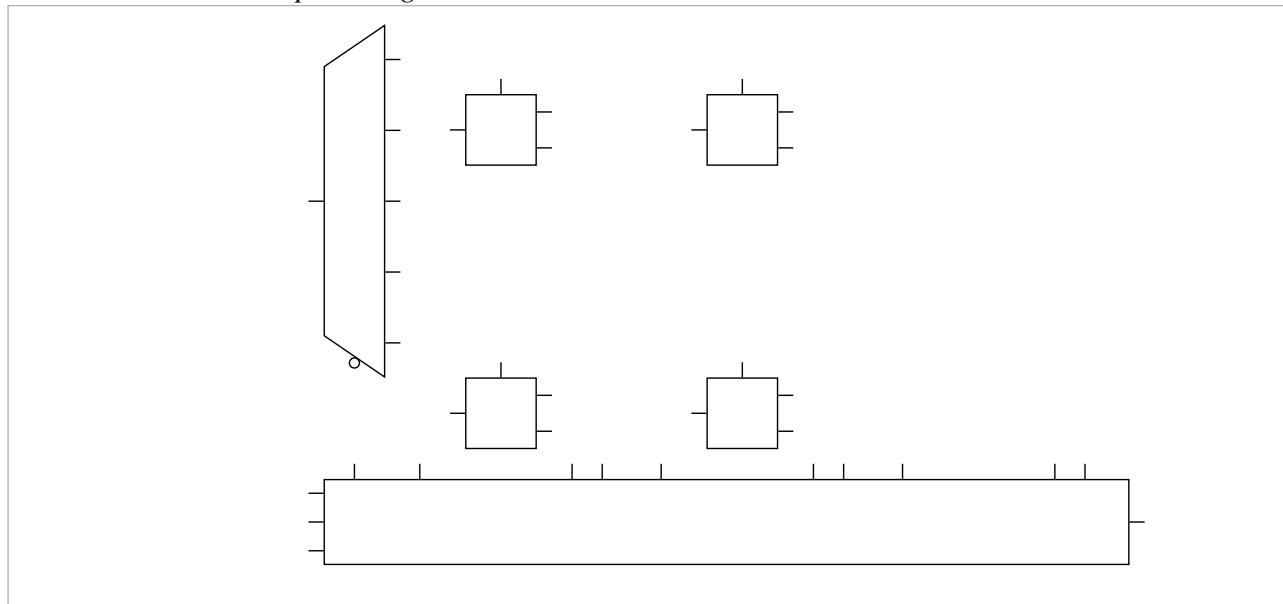
✖ Aufteilung (meist)

N Zellen auf k Zeilen u. l Spalten

so dass: $k \cdot l = N$

✖ Aufbau

- ★ ein Decoder und Spaltenlogik



✖ Ablauf

Spaltenlogik speichert Werte zwischen

✖ Hinweise

- ★ bei Dynamisches RAM:
 - automatisches auffrischen
- ★ große $N \rightarrow$ viele Adr. Leitungen
 - Zeitmultiplex

★ Weiterentwicklung Varianten

¬ *Zeitmultiplex*

Zeilen- u. Spaltenadresse

Row Address Strobe für Zeilenauswahl

Column Address Strobe für Spaltenauswahl

¬ *Evolution*

★ *Normal*

★ *Fast Page Mode*

★ *Extended Data Output*

★ *Burst EDO*

★ *Synchronous Dynamic RAM*

★ *Double Data Rate SDRAM*

★ *Rambus-DRAM*

9.3 Typ / Aufbau

9.3.1 nach Aufgabe

- Programm Speicher
- Daten Speicher

9.3.2 nach Zugriffsart

- wahlfrei
RAM
- seriell
SAM

9.3.3 nach Richtung

- nur lesen
ROM
- nur schreiben
WOM

9.3.4 nach Haltbarkeit

- flüchtig
- nicht flüchtig

9.3.5 nach Aufbau

9.3.5.1 Statisches RAM

★ Aufbau Bi-stabile Kippstufe, Flip-Flop
(2 Inverter)+Selekt

★ Eigenschaften

- ohne Spannung : Datenverlust
- Auslesen : kein Datenverlust
- Aufwand : hoch
- gegenüber DRAM :
groß, schnell, teuer

9.3.5.2 Dynamisches RAM

★ Aufbau Kondensator als Speicher

★ Ablauf

★ Eigenschaften

- ohne Spannung: Datenverlust
- Auslesen: Datenverlust
→ Rückschreiben der Daten
- Selbstentladung
→ zyklisches (dynamisches) auffrischen
- Aufwand: gering
- gegenüber SRAM: klein, langsam, preiswert

9.3.5.3 Masken programmiertes ROM

★ Aufbau Leiterbahn als „Speicher-C“

★ Programmierung

beim Hersteller

★ Eigenschaften

- ohne Spannung: Datenerhalt
- hohe Stückzahl → preiswert

9.3.5.4 (einmal) programmierbares ROM

★ Aufbau „Sicherungen“ als Speicher

★ Programmieren

Durchbrennen der Sicherungen

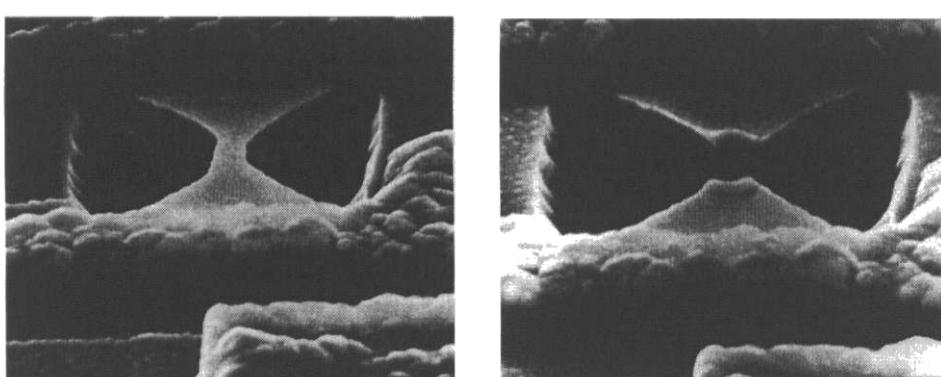


Fig. 10.35 REM-Foto eines Schmelzpfades bei etwa 15 000-facher Vergrößerung im intakten Zustand (a) und nach der Programmierung (b) (Werkbild AMD)

F. Kohlrauch; Praktische Physik (1996)

[Koh96, Fig. 10.35]

Programmiergerät

★ Eigenschaften

- nicht Änderbar
- ohne Spannung: Datenerhalt
- hohe Stückzahl → preiswert

9.3.5.5 (UV) löschbares PROM

- ✖ Aufbau MOS-FET als Speicher
- ✖ Programmieren
 - Schreiben: Ladung auf Gate
Programmiergerät
 - Löschen: Ladung vom Gate
UV-Lampe

9.3.5.6 elektrisch löschbares PROM

- ✖ Aufbau wie EPROM
- ✖ Programmieren
 - lesen u. schreiben in der Schaltung
 - Bit- / Bytewiese
 - Zeitbedarf
schreiben >ms
lesen langsamer als DRAM

9.3.5.7 Sonderformen

- ✖ Flash
 - Blockweise löschbares EEPROM
- ✖ Ringkernspeicher
 - Magnetisierung eines Ringes
 - nicht flüchtiger Speicher
 - Datenverlust beim lesen
- ✖ NVRAM, non volatile RAM
 - SRAM mit Batterie
 - SRAM mit Flash
- ✖ VRAM, Video RAM
 - schnelles SRAM
- ✖ Dual ported RAM
 - Speicher mit 2 Businterfaces
 - Grafikkarte, Messwerterfassung
- ✖ serielle EEPROM
 - serielles Businterface
- ✖ MRAM
 - Magnetoresistives RAM
- ✖ FRAM
 - Ferroelectric RAM

9.4 Quellen zu Kap. 9

- [Inf00] Infinion. "C500; Architecture and Instruction Set. User's Manual". 2000-07. Infineon Technologies AG, Juli 2000.
- [Koh96] Friedrich Kohlrausch. "Praktische Physik. Band 3". B. G. Teubner Stuttgart, 1996.
ISBN: 3-519-23000-3.
- [Sim99] Simens. "C517A, 8-Bit CMOS Microcontroller. User's Manual". 01.99. Siemens AG, Bereich Halbleiter, Marketing-Kommunikation, 1999.

10 Speicherverwaltung

10.1 Allgemein

10.1.1 Aufgabe

hier:

- Zuordnung von Adressen zu Speicherzellen
 - ¬ Adressen:
 - in Programm verwendete Werte/Bezüge
 - ¬ Speicherzellen:
 - Signalleitungen zum Speicher IC

nicht (!):

- Zuordnung von Speicher zu Prozess/-Programm
- Auslagerungstechniken

10.1.2 Bisher

10.1.2.1 Adressdekodierer

Zuordnung durch (hardware) Dekoder

- vollständig → Zuordnung : eindeutig
- unvollständig → Zuordnung : mehrdeutig
- beide → Anzahl der Adressleitungen:
 $\log. \geq \text{phy.}$

10.1.2.2 Adressräume

Einteilung in
Daten-, Prog- u. E/A- Bereich

Zuordnung:

Befehl \leftrightarrow Strg. Ltg.

10.2 Segmentverwaltung

10.2.1 Allgemein

- *Segment*
 - Speicherbereich
- *Hardware* unterstützt Software
 - ¬ Segmenttypen
 - ¬ Adressberechnung
 - ¬ Zugriffsrechte / Status
- *Bild*
- Bezeichnung
 - 2-Dimensionaler Speicher

Umsetzung

- * *durch Assembler*
Pseudo-Assemblerbefehle (s. Kap. 14.3)
- * *mit Registern*
s. Kap. 10.2.2
- * *mit Tabellen*
s. Kap. 10.2.3

10.2.2 Segmentverwaltung mit Registern

10.2.2.1 Historische Entwicklung 8086

Ziel: mehr Speicher (1M Byte)

Problem: 16 Bit Adressregister

Lösung: Segment Register
 phys. Adresse berechnen

phys. Adr = ...

10.2.2.2 Umsetzung (8086)

★ Allgemein

- Adressleitungen : 20
- Segment-, Offset-Register : 16 Bit
 - ¬ Segment-Register:
 - CS
 - SS
 - DS
 - ES
 - ¬ Offset-Register:
 - IP
 - BX
 - SI, DI, BI
 - SP
- Darstellung
 - (Seg. Reg : Offs. Reg)
- Verknüpfung
 - phys. Adr. = (Seg.Reg) · 16 + (Offs. Reg.)

★ Speicherplan

Grafische Darstellung der Speicherzuordnung

★ Eigenschaften

- Startadresse: 16 Byte Granular
- Segmentanfang beliebig
- Aufteilung zw. Seg.Reg u. Offs. reg nicht eindeutig
 - Segmente überschneiden sich !

★ Hinweise

- Speicher Modelle bei C-Compilern
- Verschiedene Sprünge
 - FAR
 - NEAR
- vgl. AJMP, LJMP

10.2.2.3 Beispiel: i386, Real Mode

★ andere Bezeichnungen

- | | | |
|---------------|---|----------------|
| Segment Reg | → | Index Reg |
| Offset Reg | → | Base Reg |
| Direkt | → | Displacement |
| phys. Adresse | → | Effektive Adr. |

★ Bild:

[int95]

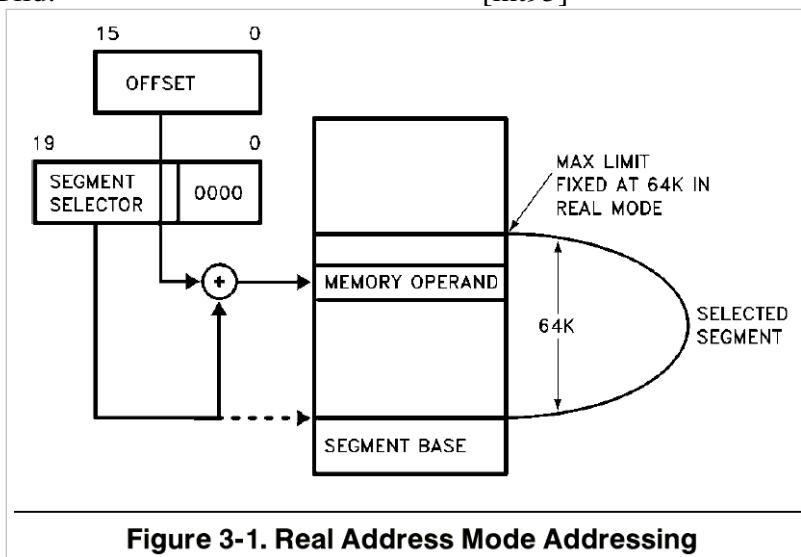


Figure 3-1. Real Address Mode Addressing

Hinweise

- Adressberechnung → MMU
- Segmente können überlappen
- zwischen Segmenten Lücken im Speicher
- Segment zusammenhängender Bereich
- keine Status, Rechte für Segmente

10.2.3 Segmentverwaltung mit Tabelle

10.2.3.1 Allgemein

(am i386 im Protected Mode)

- Segmentanfang steht in Tabelle
- Segmentregister als Index
- Tabelleneinträge:
 - Segment Base Adresse
 - Segment Limit
 - Status, Rechte
- Tabelle im Hauptspeicher
- Adressumsetzung
 - von logischer Adresse (Seg:Offs)
 - nach physikalischer Adresse

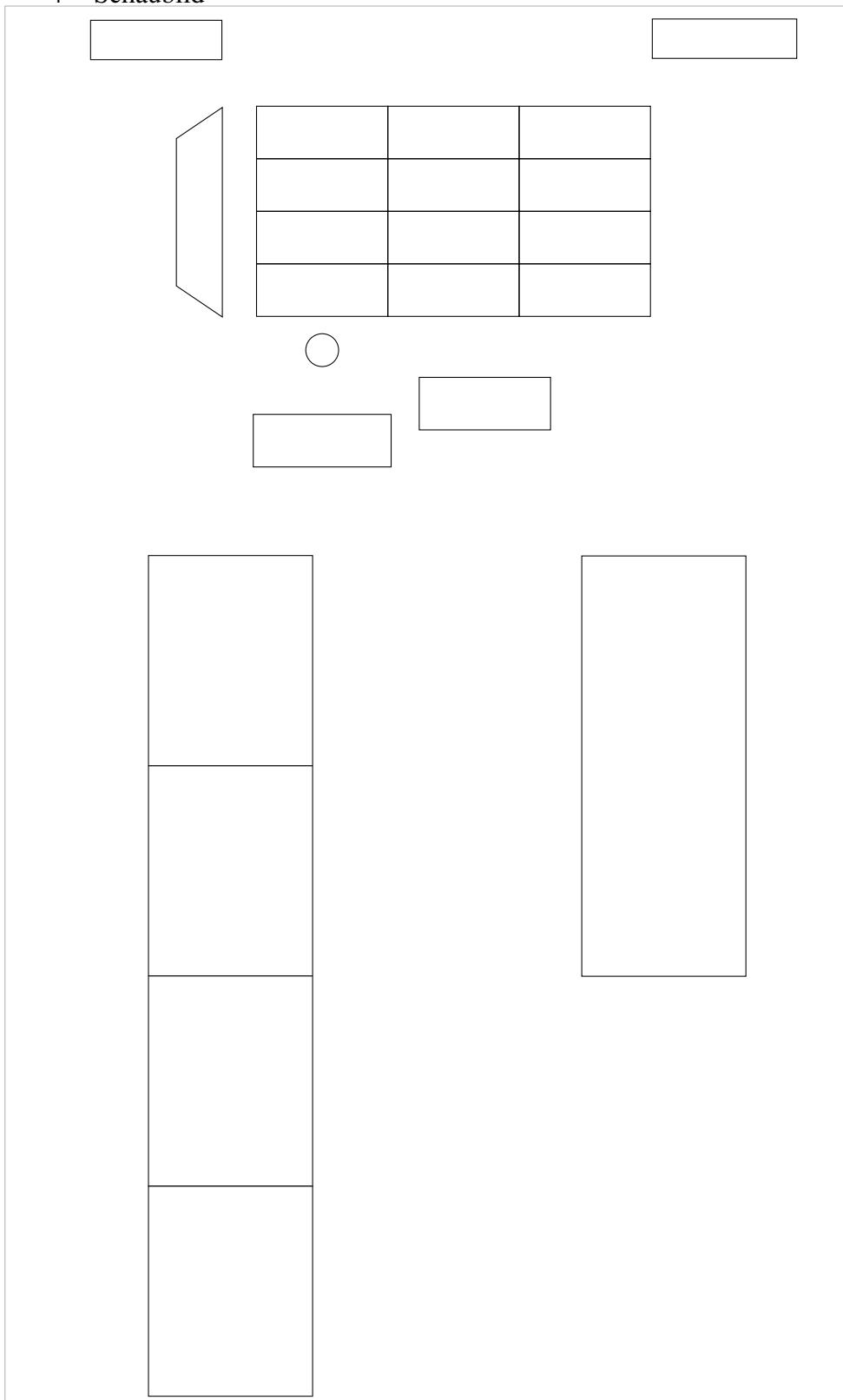
* Schaubild

siehe Seite [10.6](#)

* Hinweise

- Ein Segment =
ein zusammenhängender Speicherbereich
- Eigenschaften
 - wenig Segmente
kleine Tabelle, große Segmente, (grobe Struktur)
 - viele Segmente
große Tabelle, kleine Segmente, (feine Struktur)
- Segmente können überlappen
- Speicher kann Lücken enthalten
- Adressumsetzung → MMU

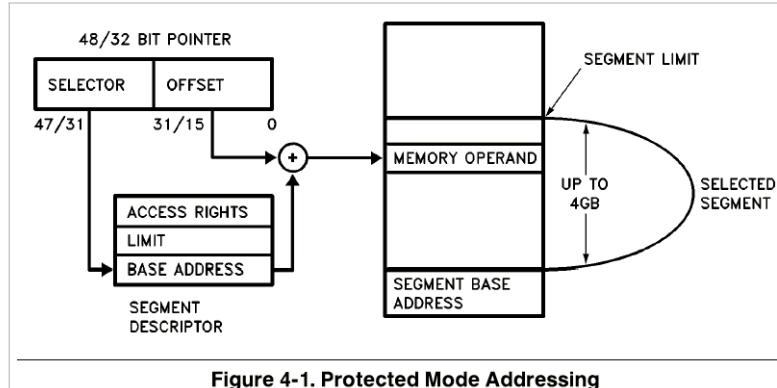
+ Schaubild



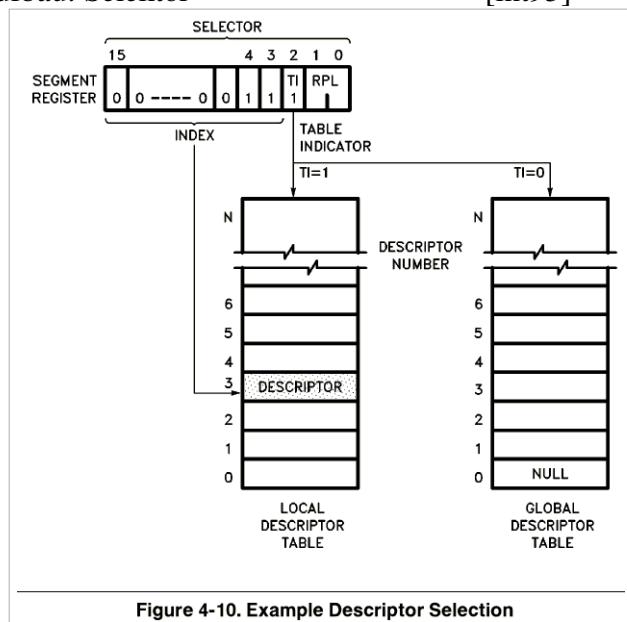
10.2.3.2 Beispiel: i386, Protected Mode

* Adressumsetzung [int95]

- aus Seg:Offs wird [Selektor | Offset]
- Offset : 32 Bit
- Selektor : 16 Bit



* Aufbau: Selektor [int95]



* Adressierbar

- log. Adressraum
→ 64 T Byte
- phys. Adressraum
→ 4 G Byte

10 Speicherverwaltung

★ Aufbau : Descriptor

[int95]

Basis, Limit, Status / Rechte

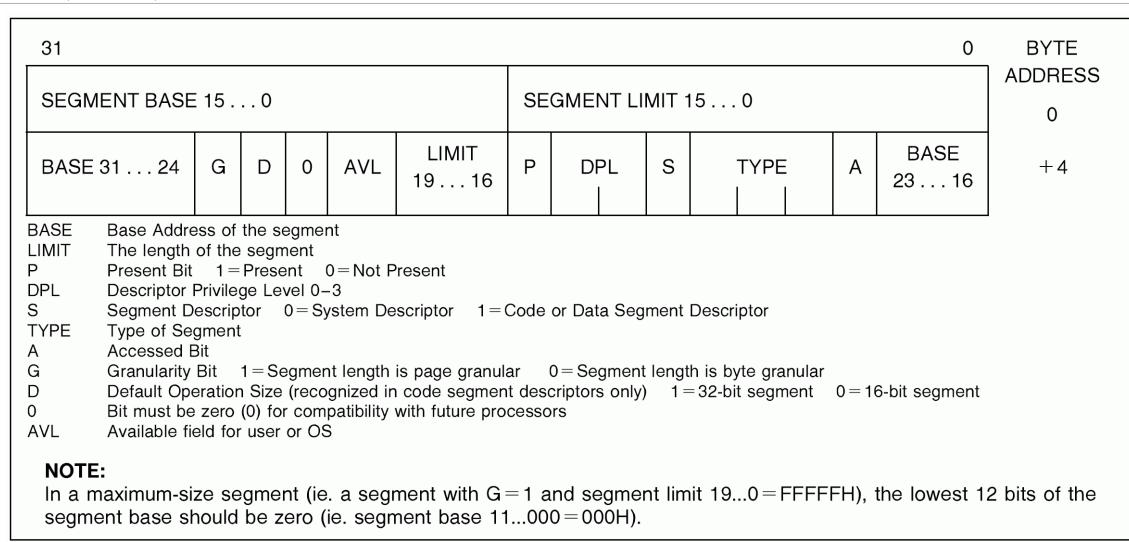


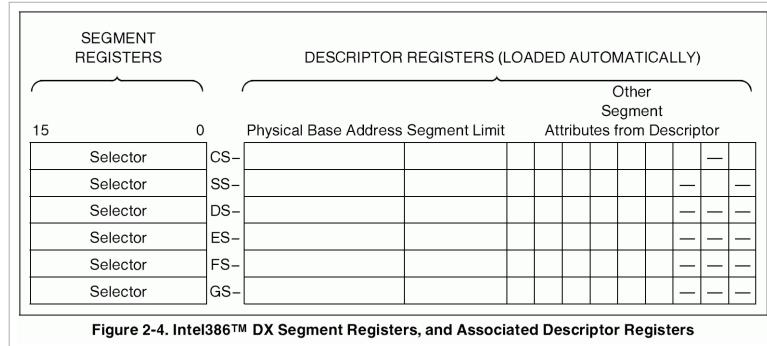
Figure 4-5. Segment Descriptors

★ Realisierung

- für jede Adressumsetzung mehrere Speicherzugriffe
- Abhilfe

„Tabelle“ in CPU

[int95]



❖ Typen u. Startadresse der Tabellen

- Typ: Global/Local/(Interrupt) Descriptor Table
- Auswahl: Selektor Bit Nr. 2
- Startadressen : LDTR, GDTR, (IDTR)

[int95]

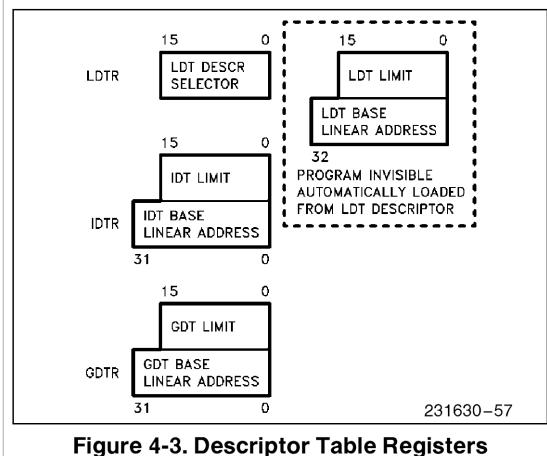


Figure 4-3. Descriptor Table Registers

¬ GDT

¬ LDT

LDTR zeigt auf GDT

GDT enthält LDT- Basis u. LDT- Limit

¬ Sonstiges

Spezial Befehle

LDT nicht von Windows genutzt

Hinweise

- Ersetzungsstrategien
Betriebssystem
- Auslagern von Segmenten
Speicher Lücken
Abhilfe: Garbage Collection

Kombination

Segment- und Seiten-Verwaltung

[int95]

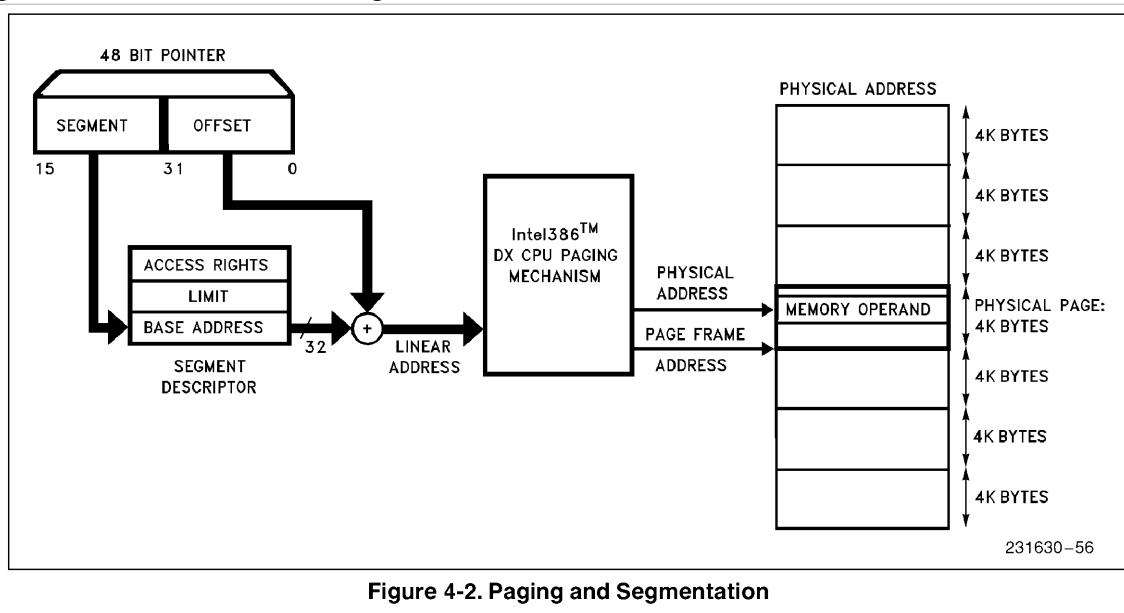


Figure 4-2. Paging and Segmentation

10.3 Quellen zu Kap. 10

[int95]

intel. "Intel386TM DX microprocessor. 32-bit chmos microprocessor with integrated memory management". [Order Number: 231630-011]. Intel Corporation, Dez. 1995.

11 Erweiterung des Minimalsystems

11.1 Interrupt

11.1.1 Allgemein

❖ Aufgabe

Reaktion auf Äußere Ereignisse

- mit Polling
(Software)
- mit Interrupt
(Hardware)

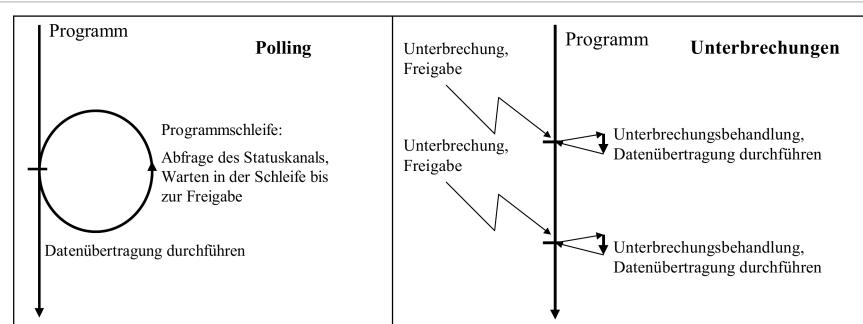


Abb. 4.8. Auswertung der Synchronisation im Prozessorkern mit Polling und mit Unterbrechungen
Brinkschulte, Ungerer; Mikrocontroller und Mikroprozessoren; Springer (2007)

[BU07]

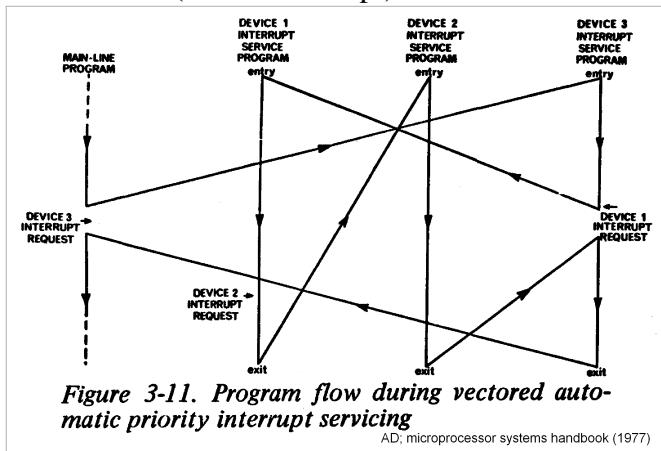
❖ Bezeichnungen (nicht eindeutig)

- ¬ Interrupt
- ¬ Trap
- ¬ Exception

❖ Eigenschaften

- benötigt
 - Hardware
 - Leitungen
- Reaktionszeit (maximale)

- Prioritäten (nested Interrupt)



[BD79]

- Rechte (evtl.) höhere Rechte

11.1.2 Quellen

- ¬ eigenständige, externe Geräte
→ Leitung
- ¬ interne Komponente
→ Flag
- ¬ „unzulässiger“ Befehl
→ Flag
- ¬ „fehlerhafter“ Befehl
→ Flag
- ¬ geplant/gewollt
→ Flag

11.1.3 Ablauf

- (Anfrage)
- Feststellen ob Interrupt vorliegt
- feststellen woher / welche Quelle
- entscheiden ob stattgegeben wird
- Prioritäten neu festlegen
- ermitteln der UP- Adresse
- Aufruf der Interrupt Routine
- (UP ausführen)
- UP beenden
- Prioritäten zurücksetzen
- Rücksprung

* Hinweis:

Flag löschen durch Hard- oder Software

Quelle: Analog Devices; microprocessor systems handbook; D. P. Burton and A. L. Dexter; Analog Devices Inc.; 1979; ISBN: 0-916550-04-4

11.1.4 Typen

oder

Ermittlung der Adresse der Interrupt-Routine

- Auto- Interrupt
- Interrupt- Nummern
- Vektor- Interrupt

11.1.4.1 Auto- Interrupt

z.B. 80517, C517A

* Quellen

- 10 interne
- 7 externe

* Prioritäts- Struktur

- Globale Freigabe (EAL)
- Prioritätsstufen (4)
- Rangfolge in Stufen
- individuelle Freigabe
(siehe Datenblatt, Seite 11.8)

* Zieladresse

fest zugeordnet

* Ablauf

- (Anfrage)
Flag setzen
- Feststellen ob Interrupt vorliegt
- feststellen woher / welche Quelle
Flag lesen
- entscheiden ob stattgegeben wird
Fallunterscheidung
- Prioritäten neu festlegen
in Interrupt - Logik
- ermitteln der UP- Adresse
Fest verbunden
- Aufruf der Interrupt Routine
Hardware-Call
- UP ausführen
Flag löschen
- UP beenden
RETI
- Prioritäten zurücksetzen
in Interrupt - Logik
- Rücksprung
PC Umsetzen

* Hinweis

ARM7 (Bilder s. u.)

11.1.4.2 mit Vektor- Nummern

z.B. 8080

* Quellen

ein Interrupt- Eingang

* Prioritätsstruktur

Globale Freigabe Flag

* Zieladresse

einer (Interrupt-) Nummer zugeordnet

* Sonstiges

Interrupt- Bestätigungs- Signal

* Ablauf

- (Anfrage)
Signal
- Feststellen ob Interrupt vorliegt
einmal pro Befehl
- feststellen woher / welche Quelle
eine Quelle
- entscheiden ob stattgegeben wird
Freigabe testen
- Prioritäten neu festlegen
Freigabe Sperren
- ermitteln der UP- Adresse
Nummer ermitteln
Adresse Berechnen
- Aufruf der Interrupt Routine
Hardware-Call
- UP ausführen
- Prioritäten zurücksetzen
Enable Interrupt
- UP beenden
Return
- Rücksprung
PC Umsetzen

11.1.4.3 Vektor- Interrupt

z.B. 8086

✖ Quellen

- Eingang: NMI
- Eingang: INTR
- Software

✖ Prioritätsstruktur

- NMI: immer
- Software: Immer
- INTR: Freigabe Flag

✖ Sonstiges

Bestätigungsausgang

✖ Zieladresse

Adresse aus Tabelle (s. 11.1.6.3)

✖ Ablauf

- (Anfrage)
 - Signal, Flag
- Feststellen ob Interrupt vorliegt
 - einmal pro Befehl
- feststellen woher / welche Quelle
- entscheiden ob stattgegeben wird
 - NMI, Software immer
 - INTR wenn IF == 0
- Prioritäten neu festlegen
 - Freigabe sperren
- ermitteln der UP- Adresse
 - Nummer ermitteln
 - Tabelle lesen
 - Adresse berechnen
- Aufruf der Interrupt Routine
 - Hardware-Call
- UP ausführen
- UP beenden
 - IRET
- Prioritäten zurücksetzen
 - in Interrupt - Logik
- Rücksprung
 - PC Umsetzen

✖ Hinweise

- Tabelle im Hauptspeicher
- Tabellenaufbau durch OS
- Umbiegen eines Interrupts möglich

11.1.5 Interrupt-Controller

11.1.5.1 Vorbemerkung

8086 hat nur einen INTR- Eingang
viele Quellen ?

Anschluss- „Ver-Odern“

⌘ Aufbau

⌘ Ablauf

- alle haben Zugriff
- nur einer antwortet

Kollisionskontrolle ?

Priorität ?

Daisy-Chain

⌘ Aufbau

⌘ Ablauf

- Anfrage wird durchgereicht
- Antwort wird durchgereicht

Kollisionskontrolle: Lage

Priorität: Lage

Immer

Zusätzliche (CPU abhängige) Logik

→ Aufwand

Besser :

Trennung von CPU u. Geräte abhängiger Logik

→ (Programmierbarer-) Interrupt Controller

Bus arbitrierung

auch für Buszuteilung bei

- DMA (s.u.)
- Multiprozessor-Systemen

11.1.5.2 Interrupt- Controller

PC: 8259

* Aufgabe

- Vermittlung zw. Gerät u. CPU
- Erzeugung Interrupt Nr.
- Zuordnung Gerät ↔ Interrupt Nr.
- Zuordnung Gerät ↔ Priorität

* Grundschaltung

- Aufbau

- Ablauf

- Konfigurieren
- Gerät an Kontroller
- Kontroller mit CPU
- CPU durch Programm an Gerät

* Kaskade

- Aufbau

- Ablauf

- Konfigurieren
- Anfragen von Kontroller 2
über Kontroller 1
an CPU
- Antwort von CPU
direkt an Kontroller 2

- PC

Zuordnung: Gerät ↔ Anschluss vorgegeben
PIC im Chipsatz bzw. South-Bridge enthalten

11.1.6 Anhang

11.1.6.1 Interrupt-Stuktur des c517a

[Sim99]

SIEMENS

Interrupt System
C517A

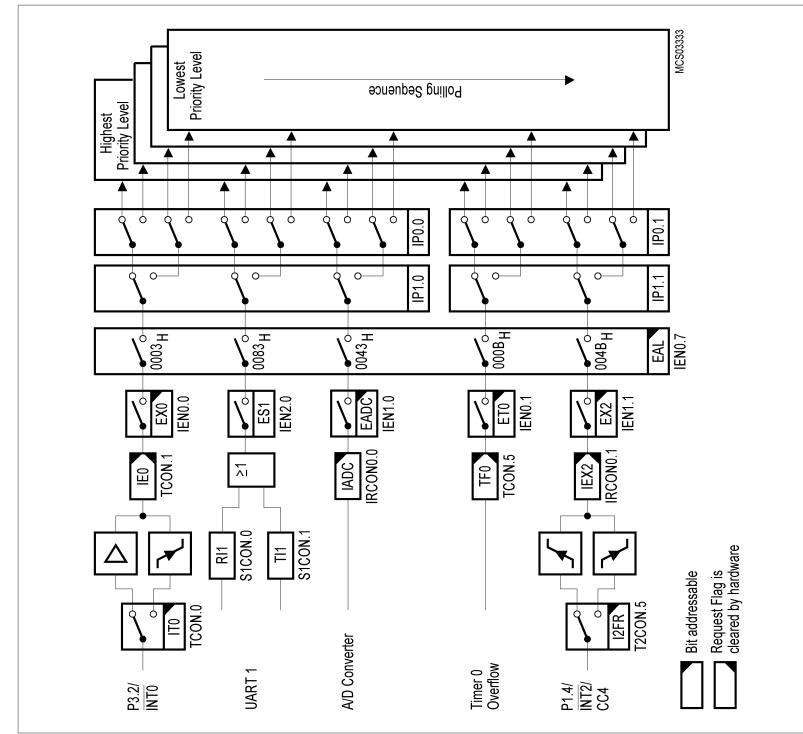


Figure 7-1
Interrupt Structure, Overview Part 1

Semiconductor Group

7-2

SIEMENS

Interrupt System
C517A

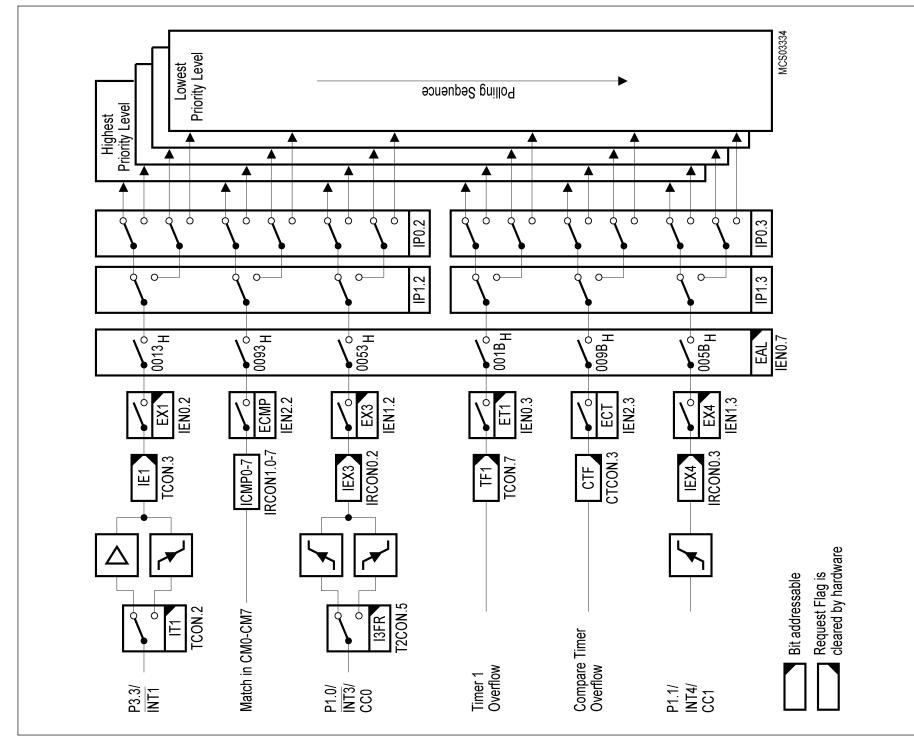


Figure 7-2:
Interrupt Structure, Overview Part 2

Semiconductor Group

7-3

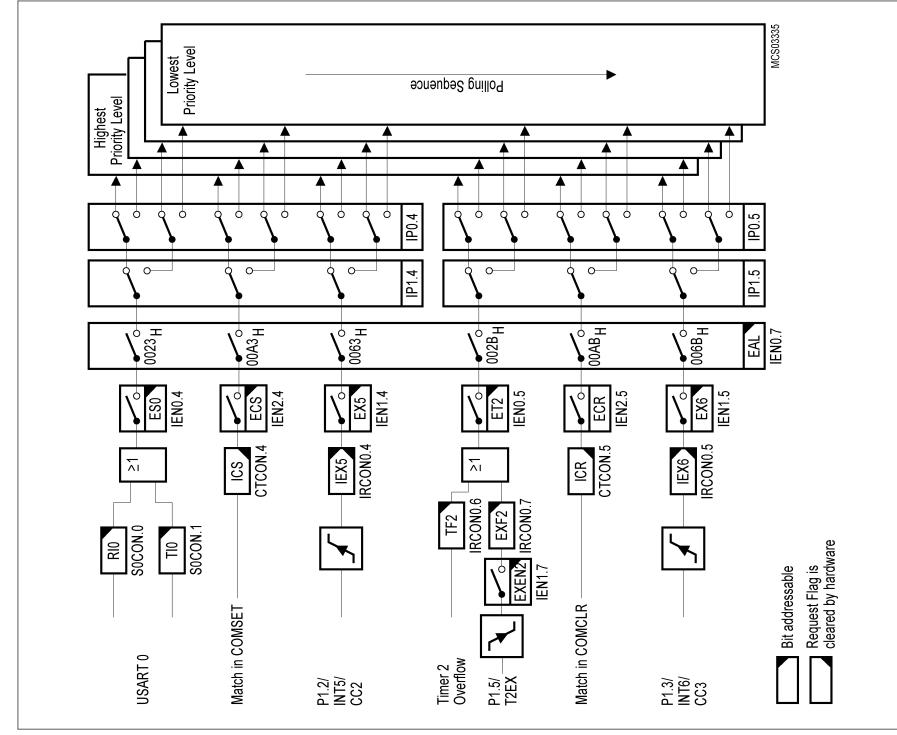


Figure 7-3:
Interrupt Structure, Overview Part 3

Note that if an interrupt of a higher priority level goes active prior to S5P2 in the machine cycle labeled C3 in figure 7-4 then, in accordance with the above rules, it will be vectored to during C5 and C6 without any instruction for the lower priority routine to be executed.

Thus, the processor acknowledges an interrupt request by executing a hardware-generated LCALL to the appropriate servicing routine. In some cases it also clears the flag that generated the interrupt, while in other cases it does not. Then this has to be done by the user's software. The hardware clears the external interrupt flags IE0 and IE1 only if they were transition-activated. The hardware-generated LCALL pushes the contents of the program counter onto the stack (but it does not save the PSW) and reloads the program counter with an address that depends on the source of the interrupt being vectored to, as shown in table 7-2.

Table 7-2
Interrupt Source and Vectors

Interrupt Source	Interrupt Vector Address	Interrupt Request Flags
External Interrupt 0	0003H	IE0
Timer 0 Overflow	000B _H	TF0
External Interrupt 1	0013H	IE1
Timer 1 Overflow	001B _H	TF1
Serial Channel 0	0023H	RIO / TIO
Timer 2 Overflow / Ext. Reload	002B _H	TF2 / EXF2
A/D Converter	0043H	ADC
External Interrupt 2	004B _H	IE2
External Interrupt 3	0053H	IE3
External Interrupt 4	005B _H	IE4
External Interrupt 5	0063H	IE5
External Interrupt 6	006BH	IE6
Serial Channel 1	0083H	R11 / T11
Compare Match Interrupt of Compare Registers CM0-CM7 assigned to Timer 2	0093H	ICMP0 - ICMP7
Compare Timer Overflow	009BH	CTF
Compare Match Interrupt of Compare Register COMSET	00A3H	ICS
Compare Match Interrupt of Compare Register COMCLR	00ABH	ICR

11.1.6.2 ARM7 Quellen u. Vektor-Adressen

Address	Exception	Mode on entry
0x00000000	Reset	Supervisor
0x00000004	Undefined instruction	Undefined
0x00000008	Software interrupt	Supervisor
0x0000000C	Abort (prefetch)	Abort
0x00000010	Abort (data)	Abort
0x00000014	Reserved	Reserved
0x00000018	IRQ	IRQ
0x0000001C	FIQ	FIQ

Table 3-4: Exception vector addresses

[ARM98]

Table 72. IRQ/FIQ MMRs Bit Description

Bit	Description
0	All Interrupts OR'ed
1	SWI
2	Timer0
3	Timer1
4	Wake-Up Timer – Timer2
5	Watchdog Timer – Timer3
6	Flash Control
7	ADC Channel
8	PLL Lock
9	I2C0 Slave
10	I2C0 Master
11	I2C1 Master
12	SPI Slave
13	SPI Master
14	UART
15	External IRQ0
16	Comparator
17	PSM
18	External IRQ1
19	PLA IRQ0
20	PLA IRQ1
21	External IRQ2
22	External IRQ3
23	PWM Trip (IRQ only)/ PWM Sync (FIQ only)

[Dev06]

11.1.6.3 Intel 386 Interrupt Liste

Table 2-5. Interrupt Vector Assignments

Function	Interrupt Number	Instruction Which Can Cause Exception	Return Address Points to Faulting Instruction	Type
Divide Error	0	DIV, IDIV	YES	FAULT
Debug Exception	1	any instruction	YES	TRAP*
NMI Interrupt	2	INT 2 or NMI	NO	NMI
One Byte Interrupt	3	INT	NO	TRAP
Interrupt on Overflow	4	INTO	NO	TRAP
Array Bounds Check	5	BOUND	YES	FAULT
Invalid OP-Code	6	Any Illegal Instruction	YES	FAULT
Device Not Available	7	ESC, WAIT	YES	FAULT
Double Fault	8	Any Instruction That Can Generate an Exception		ABORT
Coprocessor Segment Overrun	9	ESC	NO	ABORT
Invalid TSS	10	JMP, CALL, IRET, INT	YES	FAULT
Segment Not Present	11	Segment Register Instructions	YES	FAULT
Stack Fault	12	Stack References	YES	FAULT
General Protection Fault	13	Any Memory Reference	YES	FAULT
Intel Reserved	15			
Page Fault	14	Any Memory Access or Code Fetch	YES	FAULT
Coprocessor Error	16	ESC, WAIT	YES	FAULT
Intel Reserved	17–31			
Two Byte Interrupt	0–255	INT n	NO	TRAP

* Some debug exceptions may report both traps on the previous instruction, and faults on the next instruction.

[int95]

11.2 Direkt Memory Access

11.2.1 Aufgabe

Datentransfer mit Buskontrolle

Schnellerer Transfer

11.2.2 Ablauf ohne DMA

z. B. Daten von Gerät nach Speicher:

Gerät → CPU → Speicher

↪ *Programmgesteuert*

- Schleife

↪ *Eigenschaften*

- belegt Ressourcen (Register)
- braucht Rechenzeit

11.2.3 Aufbau mit DMA

⌘ *Schaubild*

11.2.4 Teile eines DMA-Controller

* Aufzählung

- Steuerung

- Register

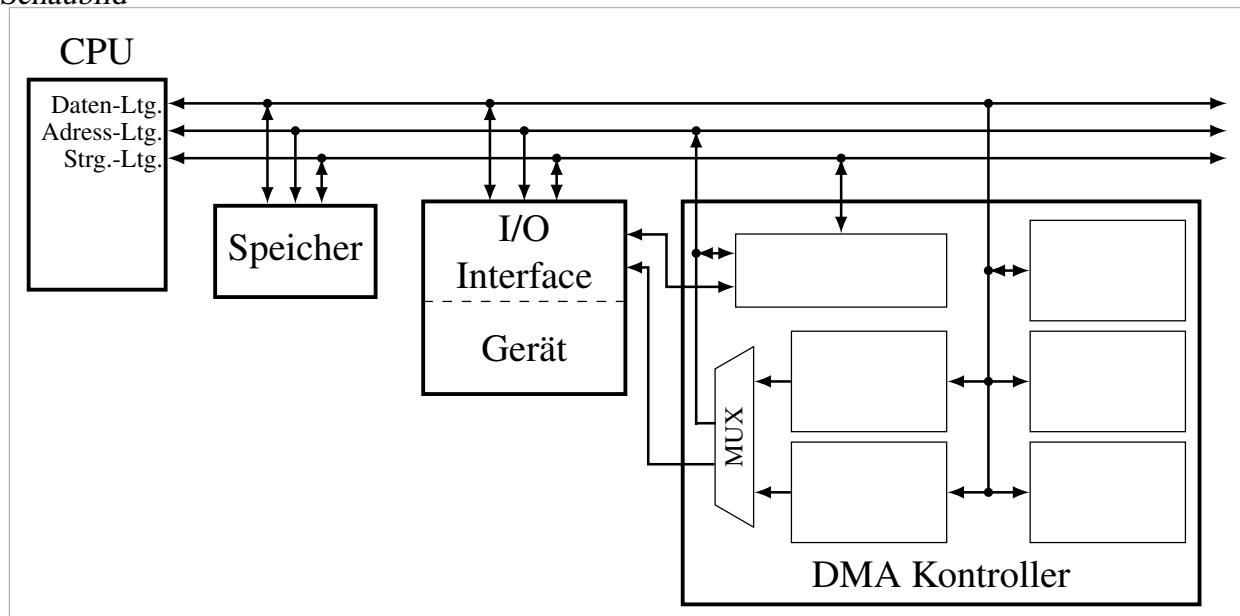
- Anschlüsse

- zum Systembus
- zum I/O-Interface

- Optional

- Speicher (Datenregister)

* Schaubild



nach [BU07, Abb. 4.53]

11.2.5 Ablauf mit DMA

* Schritte

- Kontroller initialisieren
 - * CPU → DMA

- Transfer anstoßen
 - ¬ Programmgesteuert
 - ¬ Bedarfsgesteuert
- Übergabe Kontroller übernimmt
- Transfer Kontroller steuert
 - ¬ Modi: Übertragungs-Art
 - * Direkt
 - ohne Zwischenspeicher
 - Speicher ↔ Gerät
 - * Indirekt
 - mit Zwischenspeicher
 - Speicher ↔ DMA ↔ Speicher/Gerät

- * Indirekt
 - mit Zwischenspeicher
 - Speicher ↔ DMA ↔ Speicher/Gerät
- ¬ Modi: Zugriffs-Art
 - * Blockmodus
 - viele Byte/Block
 - * Vorrang-Modus
 - ein Byte

- Kontroller gibt ab
CPU übernimmt Buskontrolle
- Vorteile
 - schneller

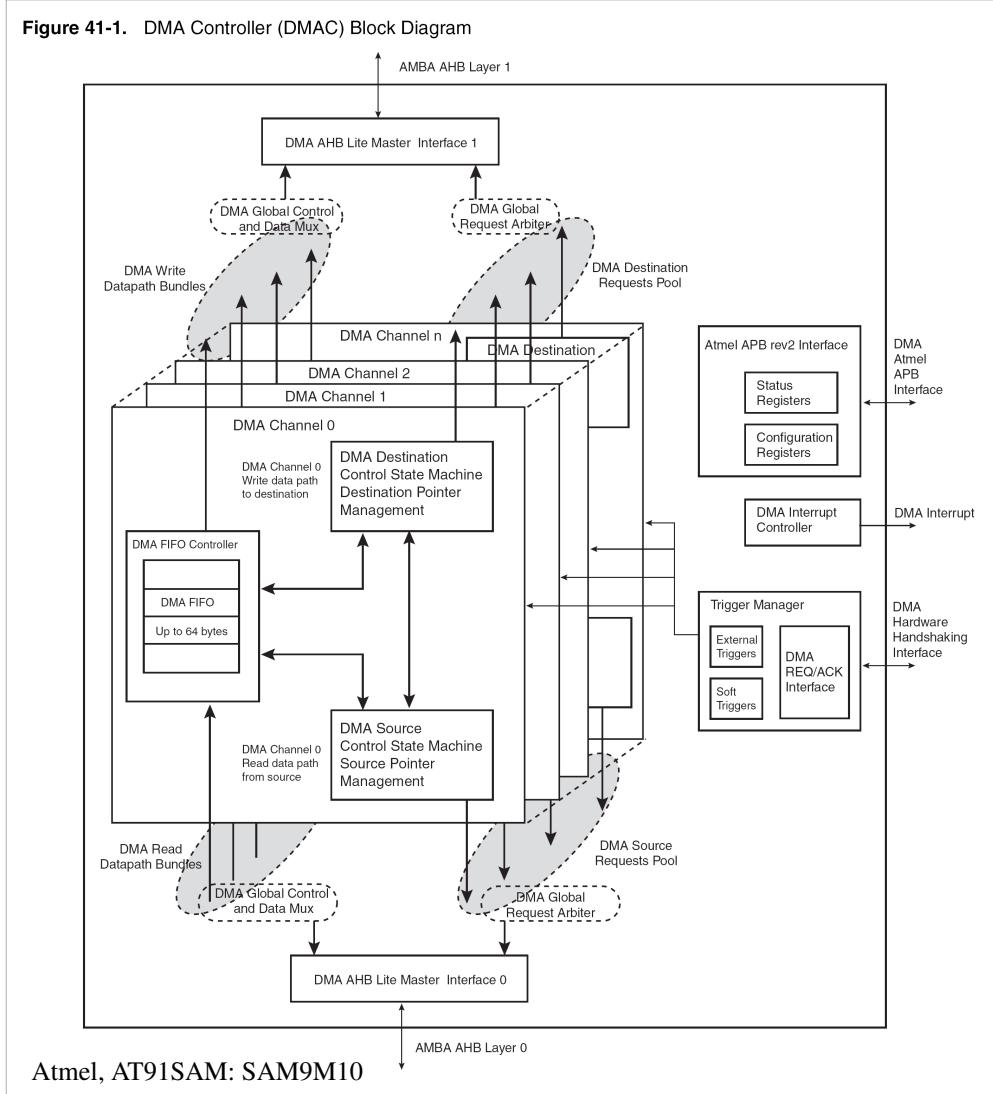
11.2.6 DMA- Kanal

Kanal ≈ Ein Transferauftrag
Ein Kontroller - mehrere Kanäle

11.2.7 Beispiel

Atmel, ARM9

Figure 41-1. DMA Controller (DMAC) Block Diagram

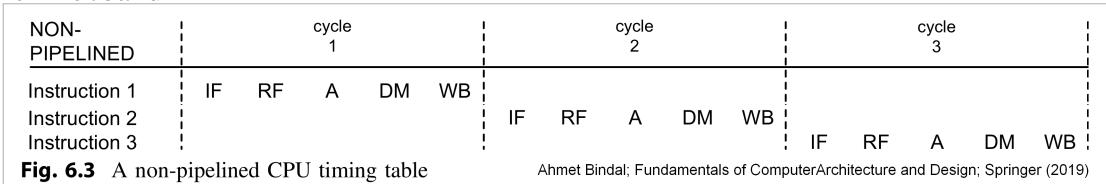


[Atm11]

11.3 Fließband / Pipeline

vgl. Kap. 12.1 Befehlszyklus

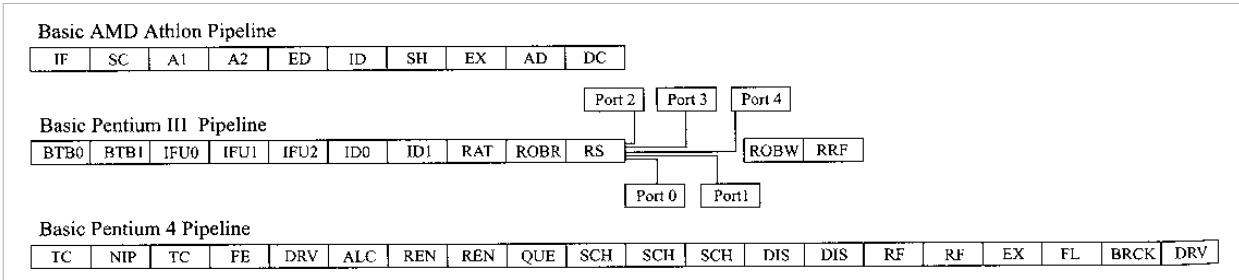
* ohne Fließband



[Bin19]

11.3.1 Allgemein

- Befehlsabarbeitung in Teilschritte zerlegen
- Teilschritte gleichzeitig ausführen
- pro Teilschritt eine Bearbeitungsstufe
- ¬ Ziel: Mehr Befehle pro Zeit
(nicht weniger Zeit pro Befehl)
- ¬ Ideal: pro Maschinencyklus ein Befehl fertig
- ¬ Optimal: jede Stufe ausgelastet
- ¬ Bsp.
- ¬ '51: Fetch u. Execute Gleichzeitig
→ 2-stufige Pipeline
- ¬ Befehlszyklus
 - Befehl holen Code Fetch
 - Befehl dekodieren Decode
 - Operand holen Operand Fetch
 - Befehl ausführen Execute
 - Werte schreiben Write → 5-stufige Pipeline
- ¬ PI: 5-stufig; P4: 20-stufig; Athlon: 10-stufig



- Bezeichnung: Superpipeline
- a) pro Stufe $\frac{1}{2}$ Takt → 2 Bef. in einer Stufe ?
- b) Anzahl der Stufen · 2 und Takt $\cdot \frac{1}{2}$?

11.3.2 Ablauf

z.B. 5-stufige Pipeline

Befehlssequenz: Bef1, Bef2,

¬ Stufenbelegung

Takt	Stufe				
	CF	D	OF	E	W
1					
2					
3					
4					
5					

¬ Ablauf → alternative zur Stufenbelegung

¬ Zeiten

ohne P.: in 5 Takten : 1 Befehl

mit P.: in 5 Takten : „3“Befehle

je länger desto mehr Befehle

11.3.3 Beispiele

- ARM 7 TDMI

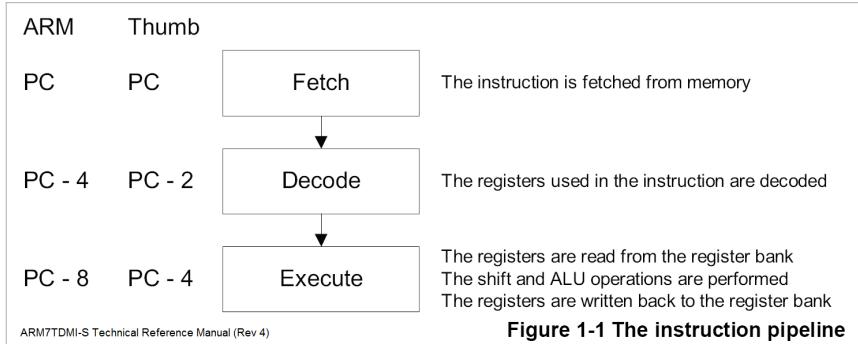


Figure 1-1 The instruction pipeline

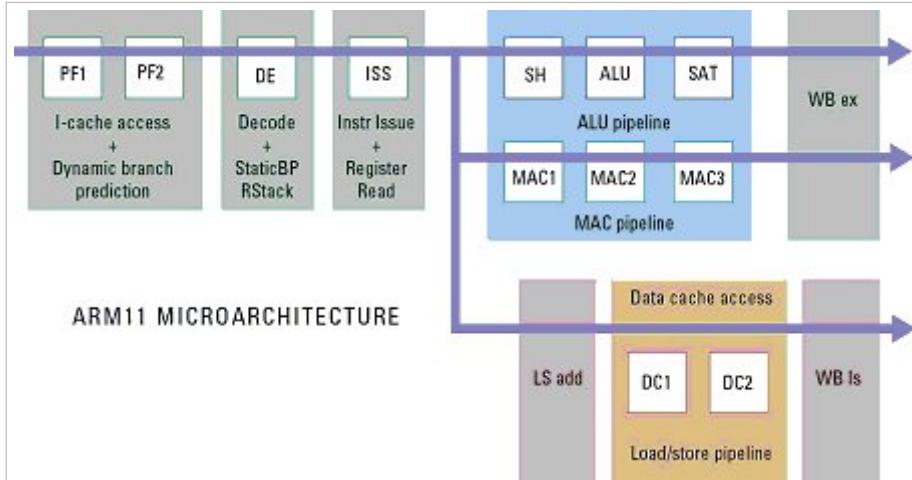
- Intel Strong ARM (bei PDA's)

[ARM01]



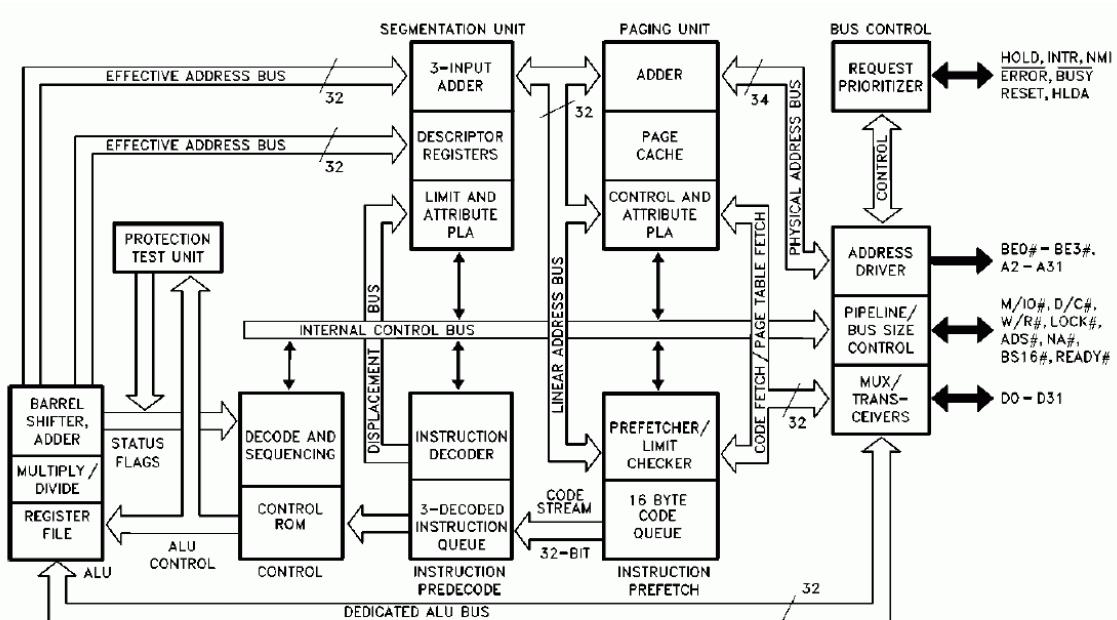
Figure 2. The seven stages of the Intel® PXA250 applications processor's pipeline.

- ARM11



- i386

[int95]



11.3.4 Probleme

11.3.4.1 Hardware

✖ Gleichtakt

- Bearbeitungszeit jeder Stufe gleich
 - ¬ sonst warten (stall) von zu schneller Stufe oder allen Stufen, auf langsamste Stufe
 - ¬ Abhilfe: langsamste Stufe
 - ... beschleunigen
 - ... in Teile aufteilen

✖ Speicherzugriff

- mehrere Stufen : gleichzeitig lesen / schreiben
 - ¬ Abhilfe: Dualport Speicher/Register/Cache (Speicher: „intelligentes“ Businterface)

✖ Speicherbandbreite

- (ideal) pro Takt ein Wert (Befehl / Daten)
 - ¬ Abhilfe: Cache

✖ „Parameterübergabe zw. den Stufen“

- ¬ Abhilfe: Zwischenspeicher

11.3.4.2 Software

Datenabhängigkeit (Data Hazard)

- ¬ Befehl benötigt Wert, welcher vom vorherigem Befehl berechnet wird
 - z.B. CLR R1
ADD A, R1

- Abhilfe
 - ¬ Hardware
 - Stufen teilweise verzögern (stall)
→ Zeitverlust
 - Ergebnis direkt an Stufeneingang
Register-Bypassing
Feed-Forwarding
 - ¬ Software (durch Compiler)
 - NOP einfügen
 - CLR R1
 - NOP
 - NOP
 - ADD A, R1
 - Programm langsamer
(u. länger)
 - Befehle umordnen
 - CLR R1
 - Bef x
 - Bef y
 - ADD A, R1

Sprungabhängigkeit

Sprung ändert Befehlsreihenfolge,
Pipeline jedoch schon
mit falschen Befehlen gefüllt

z.B. : Bef a
 JZ x ; (A==0)
 Bef 1
 Bef 2
 Bef 3
 :
X: Bef b

- ★ Bef 1, Bef 2, Bef 3 falsch geladen

- *Abhilfe*
 - ¬ *Hardware*
 - ★ wenn Sprung erkannt,
dann keine neuen Befehle laden
Folge:
 - weniger Buszugriffe
 - Pipeline verzögert
 - nix/wenig verwerfen
 - ★ wenn Sprung-entscheiden,
dann falsche Befehle verwerfen
Folge:
 - unnütze Buszugriffe
 - Pipeline verzögert
 - evtl. Ergebnisse verwerfen

¬ Software

- ★ Befehle nach Sprungbef. ausführen
und Compiler sortiert Befehle um
/ fügt NOP ein

JZ X

Bef a

NOP

NOP

Bef 1

Bef 2

:

X: Bef b

Folge:

- nur wenn Compiler ↔ CPU
ein Team
- keine Verzögerung
- keine unnötigen Buszugriffe
- nix verwerfen
- bei NOP: langsamer, länger
- Bez.: Befehlspositionen nach Sprungbefehl
von denen geladen wird:
Delay Slot

- Sprungvorhersage, Branch Prediction

Steuerwerk „rät“ Zieladresse

bzw. springen: Branch taken

weiter: Branch not taken

und lädt Befehle spekulativ von dort

wenn

richtig geraten: keine Verzögerung

falsch geraten: spekulativ Geladene verwerfen

- ¬ Vorhersage-Taktiken

- statisch (durch Compiler)

- spezielles Bit im Sprung-Befehl

- je nach Befehl

- nach Zieladresse

- dynamisch (zur Laufzeit, durch Strg. W.)

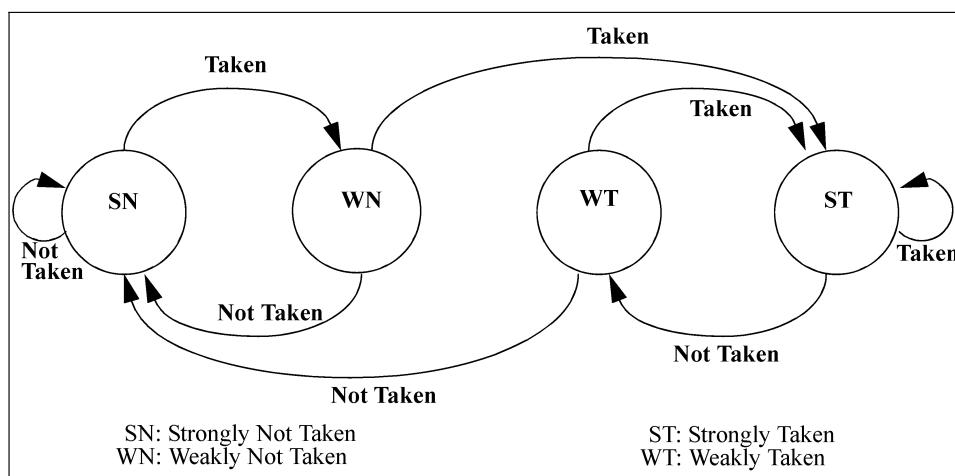
nach Vorgeschiechte

merken wie letztes Mal, so auch diesmal

Ausführung: Branch Prediction Table /

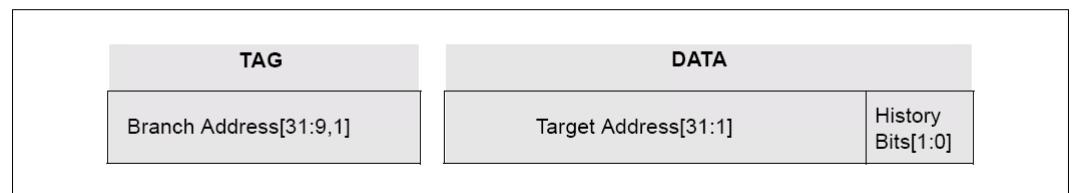
Branch Target Buffer (z.B. Strong ARM)

Figure 5-2. Branch History



[int02]

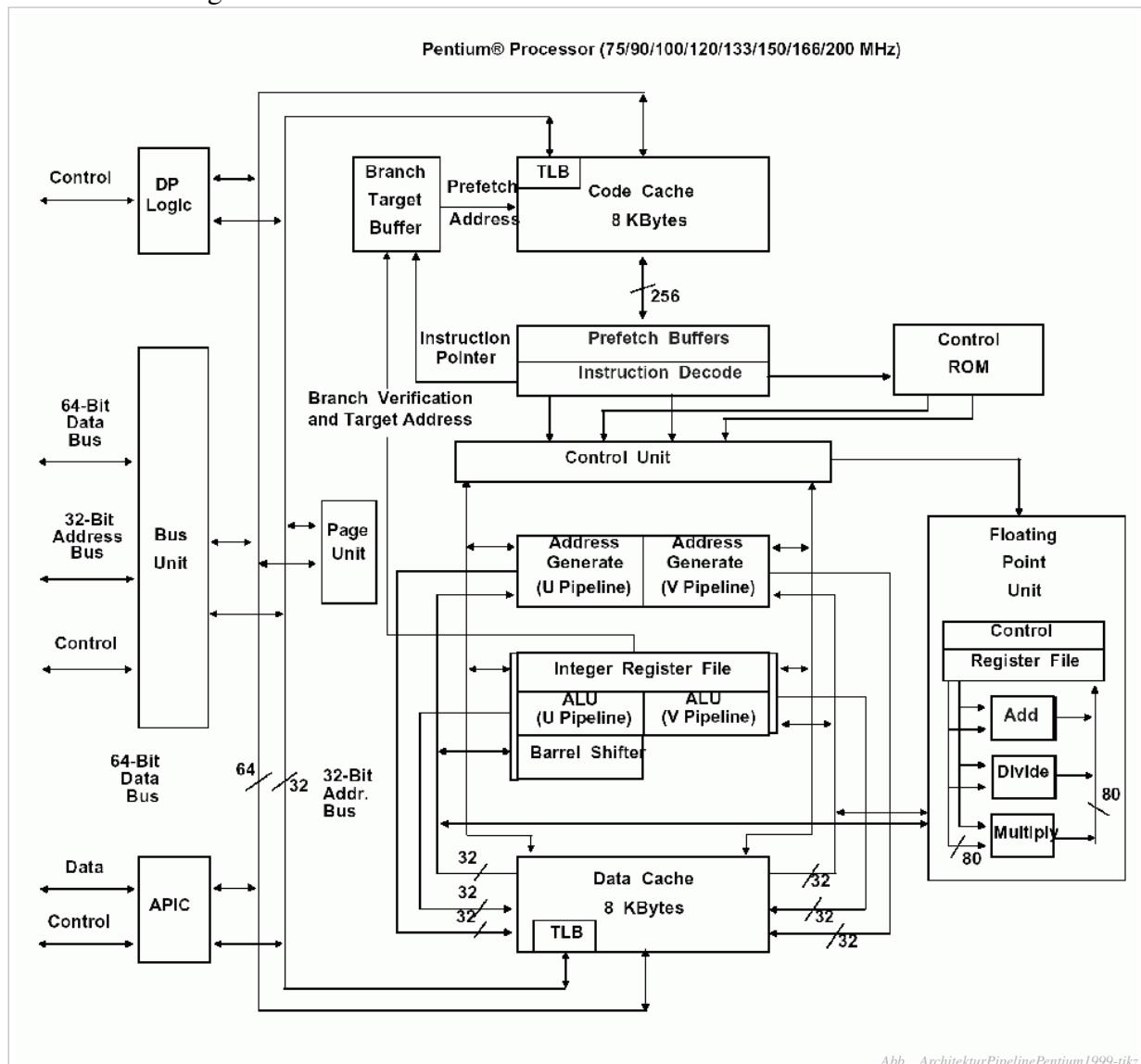
Figure 5-1. BTB Entry



[int02]

○ Bsp.: Pentium 1993

- Extra Einheit zur Berechnung der Zieladresse mit BTB
- Sprungabhängigkeit auch bei Interrupt
- 4-wege assoziativ, 512 Zeilen, Adresse d. Bef. als Tag
- Werte: Zieladresse, 4 Bit Historie
- Befehl ohne Vorgeschichte: statisch



Abb_ArchitekturPipelinePentium1993-tikz

[int97, Figure 1]

11.4 Quellen zu Kap. 11

- [ARM01] ARM. “ARM7TDMI-S. Technical Reference Manual”. Rev 4. [ARM DDI 0234A]. ARM Limited, 28. Sep. 2001.
- [ARM98] ARM. “ARM710T. Datasheet”. B. [ARM DDI 0086B]. ARM Limited, Juli 1998.
- [Atm11] Atmel. “AT91SAM, ARM-based Embedded MPU. SAM9M10”. [6355C- ATARM?19 - Apr-11]. Atmel Corporation., 19. Apr. 2011.
- [BD79] D. P. Burton und A. L. Dexter. “microprocessor systems handbook”. Second Impression. Analog Devices, Inc., 1979. ISBN: 0-916550-04-4.
- [Bin19] Ahmet Bindal. “Fundamentals of Computer Architecture and Design”. 2. Auflage. Springer Nature Switzerland AG, 2019. ISBN: 978-3-030-00223-7.
- [BU07] Uwe Brinkschulte und Theo Ungerer. “Mikrocontroller und Mikroprozessoren”. 2. Auflage. Springer-Verlag Berlin Heidelberg, 2007. ISBN: 978-3-540-46801-1.
- [Dev06] Analog Devices. “ADuC7019/20/21/22/24/25/26/27”. Rev A. [D04955-0-1/06(A)]. Analog Devices, Inc., 2006.
- [int02] intel. “Intel XScale Microarchitecture for the PXA250 and PXA210 Applications Processors. User’s Manual”. [Order Number: 278525-001]. Intel Corporation, Feb. 2002.
- [int95] intel. “Intel386TM DX microprocessor. 32-bit chmos micropocessor with integrated memory management”. [Order Number: 231630-011]. Intel Corporation, Dez. 1995.
- [int97] intel. “PENTIUM PROCESSOR”. [Order Number 241997-010]. Intel Corporation, Juni 1997.
- [Sim99] Simens. “C517A, 8-Bit CMOS Microcontroller. User’s Manual”. 01.99. Siemens AG, Bereich Halbleiter, Marketing-Kommunikation, 1999.

12 Programmieren I

12.1 Befehlszyklus

* grob

- Power On (Reset)
intern: CPU auf Grundeinstellungen
- Befehl holen
- Befehl ausführen
- Befehl holen
- Befehl ausführen
- ...

→ Endlosschleife

(kein Rücksprung, kein Programmende)

* feiner

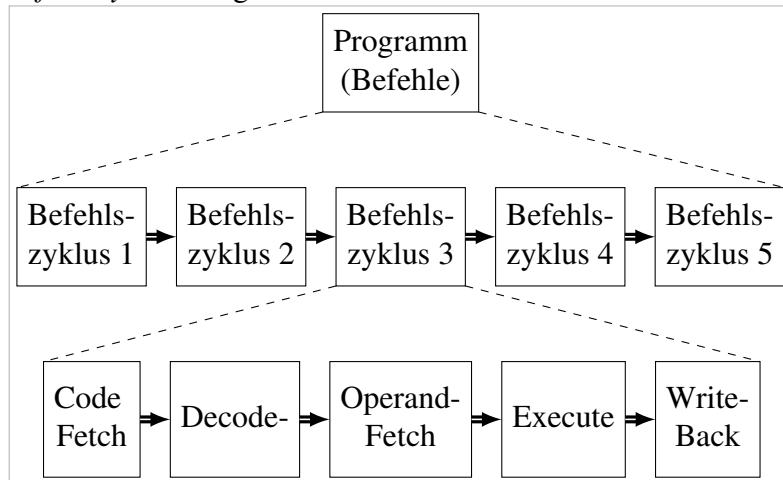
- Befehl holen
- Befehl dekodieren
- Operanden holen
- Ausführen
- Rückschreiben

* Hinweis: (s. Kap. 11.3)

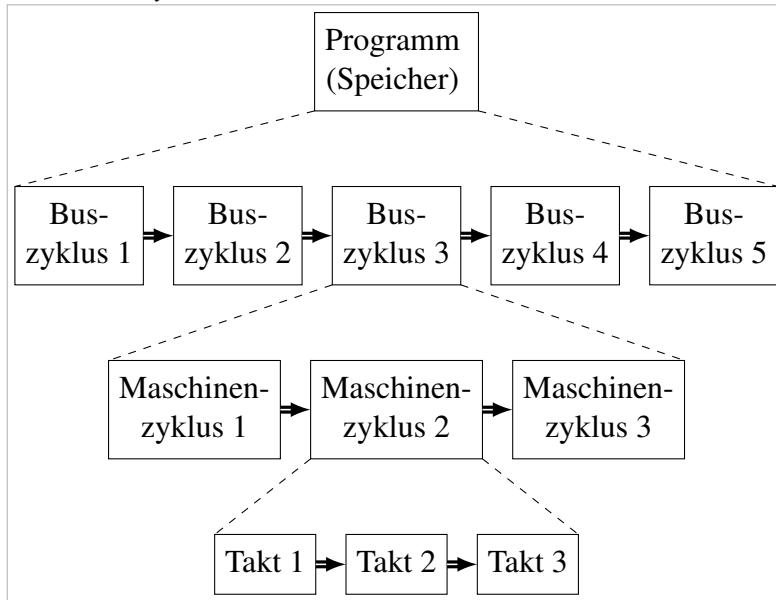
Vorstufe zur Fließbandverarbeitung (Pipeline)

* Vergleich

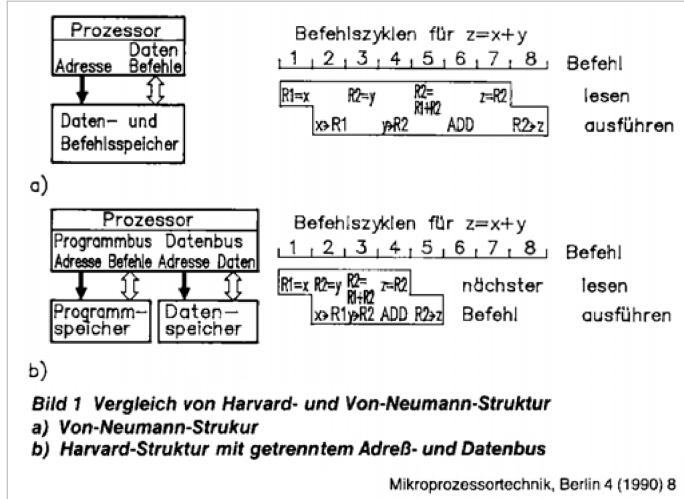
- Maschinen- vs. Befehls-Zyklus
- ¬ Befehlszyklus vgl. S. 11.15



⇒ *Maschinenzyklus* s. S.6.7



○ *v. Neuman vs. Harvard*



[Heu80, Bild 1]

12.2 Hierarchie von Sprachen

(vgl. Kap. 13.1)

- ✖ Maschinen-Code, Maschinen-Befehl
 - Binärzeichenfolge
 - wird von CPU direkt ausgeführt
 - CPU abhängig
- ✖ Assembler-Sprache
 - Symbole für Maschinen-Befehle
Mnemonic
 - wird vom Assembler-Programm (Compiler) in Maschinen-Code umgesetzt
 - 1 Assembler-Befehl = 1 Maschinen-Befehl
 - CPU abhängig
- ✖ höhere Sprache, Problem orientierte Sprache
 - Anweisungen
 - Compiler/Interpreter → Maschinen-Code
 - 1 Anweisung = viele Maschinen-Befehle
 - CPU unabhängig
- ✖ Register Transfer Language (RTL)
 - Assembler Niveau
 - $(A) \leftarrow (A) + (R1)$
 $\approx A := A + R1$
 - CPU unabhängig

12.2.1 Aufbau von Maschinen-Befehlen

- Befehl ist Bitfolge
Aufgabe, Zweck → Semantik
- Bestehend aus 2. Teilen
 - ¬ 1. Operations-Code, Op-Code, Befehls-Teil
 - ¬ 2. Operand
 - ¬ Reihenfolge (meist)

MSB	LSB
Op-Code	Operand
 - ¬ Länge von Op-Code variiert
 - ¬ Länge, Anzahl von Operand variiert
- z. B. : Bitfolge ('51)

00101	001	
-------	-----	--
- *Hinweis:*
 - mehrere Operanden möglich (Kap. 12.3.1)
 - mehrere Op-Codes möglich (Kap. 12.3.2)

12.3 Klassifikation von Befehlen

12.3.1 Klassifikation nach Anzahl von Operanden

⌘ 3-Adress Befehle

$(A3) \leftarrow (A2) \text{ op } (A1)$

⌘ 2-Adress Befehle

$(A2) \leftarrow (A2) \text{ op } (A1)$

⌘ 1-Adress Befehle

$(A) \leftarrow (A) \text{ op } (A1)$

Akkumulator Maschine

⌘ 0-Adress Befehle

op

Stack Maschine

Operation holt Operand vom Stack

Ergebnis wieder auf Stack

Stack besonderer Speicherbereich

UPN, JAVA intern

⌘ 4-Adress Befehle

$(A3) \leftarrow (A2) \text{ op } (A1)$

$(PC) \leftarrow (A4)$

vgl. μ - Befehle

⌘ Hinweis

○ meist:

Prozessoren nicht eindeutig zuzuordnen

○ bei modernen CPU (RISC)

meist 3-Adress Befehle

12.3.2 Klassifikation nach M. Flynn

[Fly72]

- ¬ Unterscheidung nach Op-Code/Steuerwerke und Daten/ALU pro Befehl

* SISD

Single Instruction Single Data

- ¬ ein Steuerwerk steuert eine ALU
- ¬ „Normale“ CPU

* SIMD

Single Instruction Multiple Data

- ¬ ein Steuerwerk steuert mehrere ALU
- ein Op-Code für mehrere Daten(-ströme)

¬ z.B.

- MMX (Multimedia Extension) (ab Pentium II)
- 64 Bit Register : MM0 ... MM7

ein Befehl steuert

8 gleichzeitige	1 Byte	Operationen
oder 4 gleichzeitige	2 Byte	Operationen
oder 2 gleichzeitige	4 Byte	Operationen
oder 1	8 Byte	Operation

ab PII: MMX-2: 128 Bit Register

- Vektorrechner mit
 - Vektorregister: Register mit N-Komponenten (Vektoren)
 - Vektorbefehlen: Befehl für Vektoren
- Anwendung:
 - N-Dim. Gleichungssystemen (Earth- Simulator)
 - 3-Dim. Grafik (Playstation2)

* MIMD

Multiple Instruktion Multiple Data

- ¬ mehrere Steuerwerke steuern mehrere ALUs
- mehrere Op-Codes + Operanden in einem Befehl
- mehrere Prozessoren parallel in einer Einheit

- ¬ Realisierung: VLIW
- Very Long Instruction Word
- Speicherwort = VLIW - Befehl

* MISD

Multiple Instruction Single Data

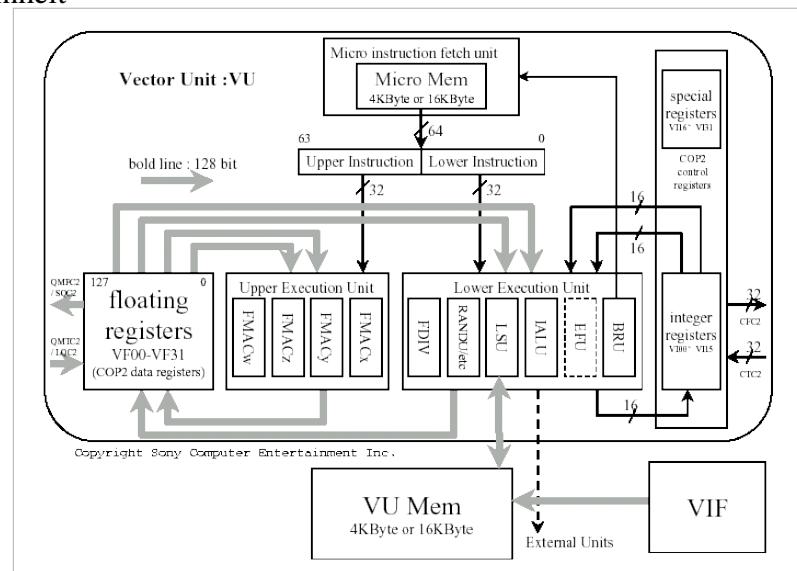
- ¬ mehrere Steuerwerke steuern eine ALU
- gibt es nicht

+ BSP: Playstation 2: Emotion Engine

mit RISC-Kern + 2 Vektor Units

- ¬ SIMD: Vektorprozessoren (VU)
- 4-Elemente

- ¬ VLIW:
- [Lower Inst. | Upper Inst.] pro VU



12.3.3 RISC / CISC

Reduced Instruction Set Computer
Complex Instruction Set Computer

12.3.3.1 historische Entwicklung

✿ einfache Prozessoren

- ¬ einfache Befehle
- ¬ wenige Befehle
- ¬ wenige Register

✿ größere Prozessoren → CISC

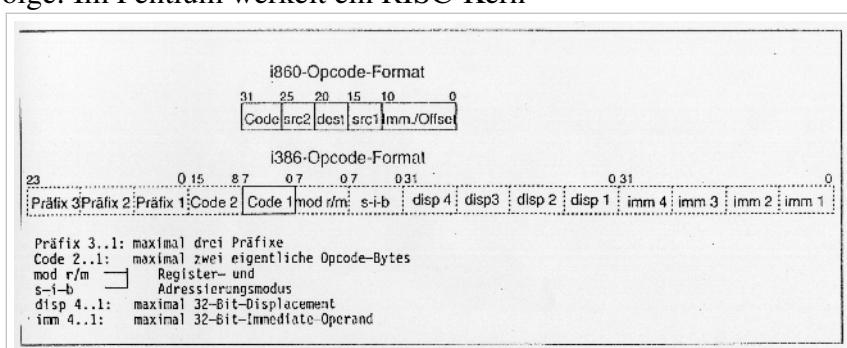
- ¬ komplexe Befehle
- ¬ viele Befehle
- ¬ viele Adressierungsarten
- ¬ Folge:
- ¬ einfache Compiler (+)
Hochsprachen Bef. ist ein Maschinen Befehl
- ¬ kurzer Maschinencode (+)
- ¬ unterschiedliche Befehlslängen
- ¬ aufwendige Dekodierung,
komplexes Steuerwerk (-)
- ★ unterschiedliche Buszugriffe (-)
- ¬ mehrere Maschinenzyklen pro Befehl (-)

✿ Untersuchungen ergeben

- ¬ wenige Befehle werden häufig gebraucht
Folge: Befehlssatz einschränken
- ¬ Speicher / Buszugriffe langsam
Folge: mehr Register
- ¬ Steuerwerk zu teuer
Folge: Steuerwerk einfacher durch
- ★ feste Befehlslänge (auch besser Buszugriffe)
- ★ einheitlicher Befehlsaufbau

✿ Ergebnis: RISC

- ¬ kleiner, schneller CPU-Kern
- ¬ jedoch: CISC Marktführer (8086 Nachfolger)
- Folge: Im Pentium werkelt ein RISC-Kern



12.3.3.2 RISC-typisch

✖ Reduzierter Befehlssatz

- ¬ nur einfache Befehle, wenige Befehle
 - Ziel: einfaches Steuerwerk
 - z.B.: nur OR mit 3 Operanden anstatt CLR, MOV
 - ¬ pro Maschinenzyklus ein Befehl

✖ viele Register

- Anzahl ≈ 32
 - Ziel: alle Werte in Registern
- General Purpose Register
 - Ziel: einfaches Steuerwerk, einfacher Compiler
- Register mit festen Werten
 - $(R0) \equiv 0; (R32) = \text{Rücksprungadresse}$
 - ¬ (und Registerbänke)

✖ Fester Befehlsaufbau mit fester Befehlslänge

- alle Befehle gleich Lang
 - ¬ optimierter Buszugriff
 - ¬ gute Cache Auslastung
 - ¬ lange Programme, schlechte Speicherausnutzung
- fester Befehlsaufbau
 - ¬ einfaches Steuerwerk
 - ¬ wenige Befehlstypen
 - Ziel: einfaches Steuerwerk
- ★ Operationen
 - nur 3 Adr. Befehle
 - $(\text{Adr}1) \leftarrow (\text{Adr}2) \text{ op } (\text{Adr}3)$
 - Register laden
 - feste Operandenlänge
 - Sprung
 - feste Sprungadressenlänge (evtl. aufteilen)

Bsp: ARM 7, (siehe Bild Seite 12.8)

✖ Load-Store Architektur

- ¬ Operationen nur mit Registerinhalten
- ¬ Speichertransfer nur von/nach Registern
- ¬ Zweck:
 - Trennung von Rechnung und Speicherzugriffen
 - Speicherzugriffe langsam
 - gleichzeitig rechnen und speichern

Beispiel

ARM 7

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0																						
Data processing and FSR transfer	Cond	0	0	1	Opcode			S	Rn	Rd		Operand 2										
	Multiply	0	0	0	0	0	0	A	S	Rd	Rn	Rs		1	0	0	1	Rm				
Multiply long	Cond	0	0	0	0	1	U	A	S	RdHi	RdLo	Rn		1	0	0	1	Rm				
Single data swap	Cond	0	0	0	1	0	B	0	0	Rn		Rd		0	0	0	0	1	0	0	1	Rm
Branch and exchange	Cond	0	0	0	1	0	0	1	0	1	1	1	1	1	1	1	1	1	0	0	0	1
Halfword data transfer, register offset	Cond	0	0	0	P	U	0	W	L	Rn		Rd		0	0	0	0	1	S	H	1	Rm
Halfword data transfer, immediate offset	Cond	0	0	0	P	U	1	W	L	Rn		Rd		Offset		1	S	H	1	Offset		
Single data transfer	Cond	0	1	1	P	U	B	W	L	Rn		Rd		Offset								
Undefined	Cond	0	1	1												1						
Block data transfer	Cond	1	0	0	P	U	S	W	L	Rn		Register list										
Branch	Cond	1	0	1	L												Offset					
Coprocessor data transfer	Cond	1	1	0	P	U	N	W	L	Rn		CRd		CP#		Offset						
Coprocessor data operation	Cond	1	1	1	0	CP Opc			CRn		CRd		CP#		CP	0	CRm					
Coprocessor register transfer	Cond	1	1	1	0	CP	Opc	L	CRn		Rd		CP#		CP	1	CRm					
Software interrupt	Cond	1	1	1	1												Ignored by processor					

Figure 1-5 ARM instruction set formats

[ARM98]

12.4 Quellen zu Kap. 12

- [ARM98] ARM. "ARM710T. Datasheet". B. [ARM DDI 0086B]. ARM Limited, Juli 1998.
- [Fly72] Michael J. Flynn. "Some Computer Organizations and Their Effectiveness". In: "IEEE Transaction on Computers" C-21 (9. Sep. 1972), S. 948–960.
- [Heu80] Gert Heuer. "Digitale Singalprozessoren". In: "Mikroprozessorteknik" 4 (8 1980), S. 4–9. DOI: 10.1515_9783112610787-003.
- [PD80] D. A. Patterson und D. R. Ditzel. "The Case for the Reduced Instruction Set Computer". In: "ComputerArchitecture News" 8 (6 15. Okt. 1980), S. 25–33.

13 Programmieren II

13.1 Schichten eines Computers

nach [Tan99]

A.S. Tanenbaum

Structured Computer Organisation

Prentice Hall, 4. Aufl. 1999

Ebene 5

Ebene 4

Ebene 3

Ebene 2

Ebene 1

13.2 Programmerstellung

13.2.1 Begriff: Assembler

• Assembler-Sprache

• Assembler-Programm

13.2.2 Programm Erstellung

Quell-Code

•

•

•

•

Programm Speicher

13.2.3 Einfacher Aufbau einer Assemblerdatei

- ✖ Allgemein
 - Zeilen

- Befehle
 - ¬ Assemblerbefehle

- Liste s. S. 14.2
- ¬ Pseudo-Assemblerbefehle

Tabelle s. S.14.4

- Dateiabschluss
 - END

✖ Bsp.

- B1

- B2

13.3 Adressierungsarten

(Transportbefehle)

13.3.1 Allgemein

⊕ Befehl besteht aus

 ⊖ Operation

 ⊖ Operand

⊕ Adressierung gibt an:

Wo

○ Mögliche Orte:

 ⊖ im Befehl

 ⊖ in Register

 ⊖ in Speicher

⊕ im Folgenden

 ⊖ MOV A, ?

 ⊖ Schreibweise in Anlehnung an '51

 ⊖ graphische Darstellung

13.3.2 Arten

⊕ unmittelbar, immediate

Bez.: #...

★ Wert steht im Befehl

★ RTL: (A) ← #daten

Assembler: MOV A, #daten

Maschinen Code: 0111 0100 daten

★ Bild

* direkt

Bez: (...)

- z.B. (Register-) direkt
 - * Wert steht im Register
 - Registername (-adresse) steht im Befehl
- * $(A) \leftarrow (R_n)$
MOV A, Rn
1110 1rrr
- * *Bild*

* Varianten

- Adressen

* indirekt

Bez.: ((...))

- z.B. Register indirekt
 - * Wert steht im Datenspeicher
 - Adresse der Speicherzelle steht im Register
 - Registername steht im Befehl
- * $(A) \leftarrow ((R_i))$
MOV A, @Ri
1110 011i
- * *Bild*

* *Varianten*

- allgemeine Register
- spezial Register
 - Data Pointer
 - Stack Pointer
 - Speicher Zelle
- indizierte Adressierung
 - Register + Offset
 - Register + Register
 - PC + Offset

✖ *Hinweise*

- Adressierungsarten gelten (im Prinzip) für alle Operanden
 - ¬ jedoch meist nicht alle Adressierungsarten für alle Operanden implementiert
 - ¬ wenn alle Arten für alle Operanden bei allen Befehlen
- Bez.: Orthogonaler Befehlssatz
(gut für Compiler, aufwendig für CPU)

13.4 Einteilung von Befehlen

nach Funktion

13.4.1 NOP

no operation

NOP

13.4.2 Transportbefehle

$(A) \leftarrow (B)$

MOV A, B

13.4.3 Arithmetische Befehle

$(A) \leftarrow (A) + (R1)$

ADD A, R1

13.4.4 Logische Befehle

$(A) \leftarrow (A) \text{ AND } (R1)$

ANL A, R1

13.4.5 Verzweigungs- Befehle

Programmsteuerungsbefehle

* Allgemein

- für Sprünge, Verzweigungen, Schleifen
- ändern des linearen Programmablaufs
- (PC) enthält Adresse
des nächsten Befehls
- ★ ohne Verzweigung
 $(PC) ++$

- ★ mit Verzweigung
 $(PC) \leftarrow \text{neue Adr.}$

- Arten
 - ¬ Sprung
 - ★ unbedingter
 - ★ bedingter
 - ¬ Unterprogramm Aufruf
(Rücksprung Adresse ?)

★ unbedingter Sprung

○ verzweigt immer

○ $(PC) \leftarrow \#Adresse$

○ 8051

¬ LJMP adr16

$(PC) \leftarrow adr15-0$

¬ AJMP adr11

$(PC) \leftarrow (PC) + \#2$

$(PC10-0) \leftarrow adr10-0$

¬ SJMP rel

$(PC) \leftarrow (PC) + \#2$

$(PC) \leftarrow (PC) + \#rel$

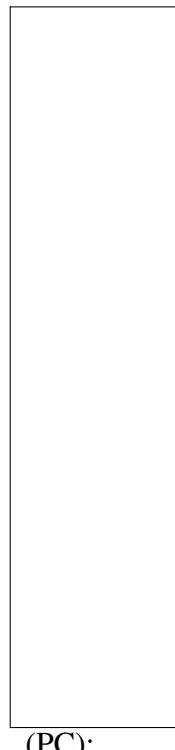
¬ Sprungzielbereiche

gesamter Programmspeicher: 64 K Byte

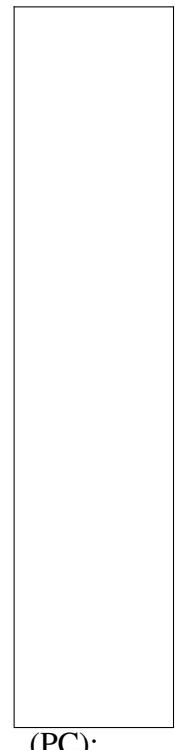
LJMP



AJMP



SJMP



★ Hinweis

AJMP-Bereich vgl. Seitenverwaltung

❖ bedingter Sprung

Verzweigung nur wenn Bedingung WAHR

z.B. JZ rel

$(PC) \leftarrow (PC) + \#2$

if (A)==0 then $(PC) \leftarrow (PC) + \#rel$

❖ Unterprogrammaufrufe

○ Allgemein

- ¬ ein Programmteil (UP) wird mehrfach genutzt
- ¬ UP steht nur einmal im Speicher
- ¬ Aufruf durch CALL- Befehl
- ¬ Anfang durch Adresse (label) festgelegt
- ¬ Ende durch Rücksprung-Befehl festgelegt
(Wohin zurückspringen?)

○ Ablauf

Programm (PC)

Adr.	Bef.
1	HP1
2	HP2
3	CALL X
4	HP3
5	HP4
6	CALL X
7	HP5
...	...
20	X:UP1
21	UP2
22	RET

○ Schritte in [] nicht notwendig

* [Parameterübergabe]

¬ Aufruf
mit Rücksprungadresse merken

* [Retten des Status]

¬ Verzweigen
durch umsetzen des PC

¬ UP abarbeiten

* [mit
Anfang: Retten, Parameterlesen
Ende: Parameterschreiben, Restaurieren]

¬ Rücksprung
durch $(PC) \leftarrow$ Rücksprungadresse

* [Restaurieren des Status]

¬ HP weiter

* [Parameterübergabe]

★ Befehle des 8051

- ACALL addr11
 - $(PC) \leftarrow (PC) + \#2$
 - (PC) sichern
 - $(PC10-0) \leftarrow \#addr11$
- LCALL addr16
 - $(PC) \leftarrow (PC) + \#3$
 - (PC) sichern
 - $(PC) \leftarrow \#addr16$
 - Sichern genauer
 - $(SP) \leftarrow (SP) + \#1$
 - $((SP)) \leftarrow (PC7-0)$
 - $(SP) \leftarrow (SP) + \#1$
 - $((SP)) \leftarrow (PC15-8)$
- RET
 - $(PC) \leftarrow$ Rücksprungadresse
 - genauer
 - $(PC15-8) \leftarrow ((SP))$
 - $(SP) \leftarrow (SP) - \#1$
 - $(PC7-0) \leftarrow ((SP))$
 - $(SP) \leftarrow (SP) - \#1$

★ Hinweise

- Andere Darstellung

HP	UP	X:
\vdash	\vdash	\vdash
Call X	Call X	Ret
\vdash	\vdash	\vdash
- Verwendung
 - immer: CALL - RET
 - nie: CALL - JMP
 - nie: JMP - RET
 - möglich: CALL - JMP - RET

13.5 Quellen zu Kap. 13

- [Tan99] Andrew S. Tanenbaum. "Structured Computer Organization. Funktionsweise und Einsatzgebiete". 4. Auflage. Prentice Hall, 1999. ISBN: 0-13-095990-1.

14 Programmieren Anhang

14.1 Speicherorganisation

[Sim99]

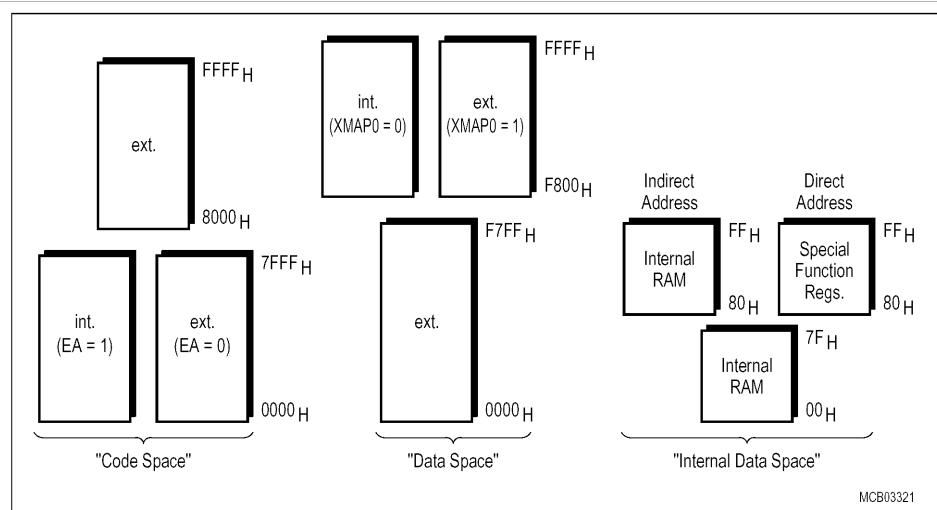


Figure 3-1
C517A Memory Map

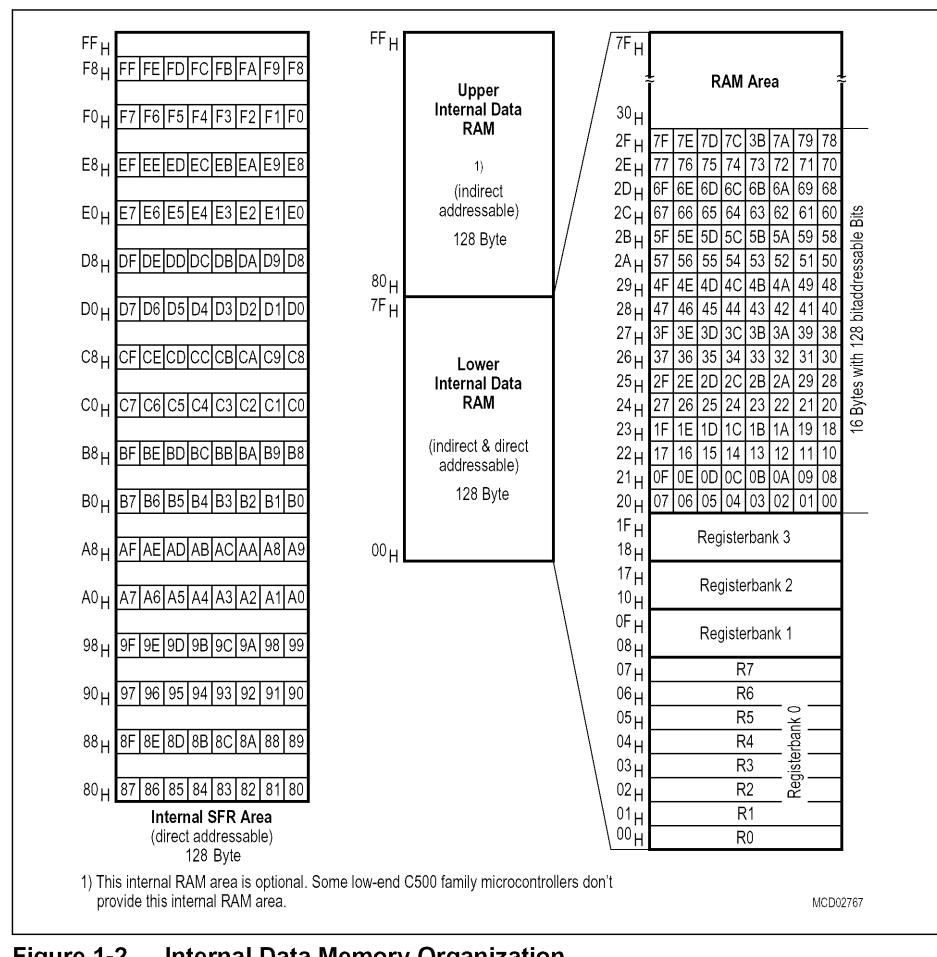


Figure 1-2 Internal Data Memory Organization

14.2 Befehlssatz

[Inf00]

4.4 Instruction Set Summary Tables

The following two tables give a survey about the instruction set of the C500 family microcontrollers. In **Table 4-3** the instructions are ordered in functional groups. In **Table 4-4** the instructions are ordered in the hexdecimal order of their opcode.

4.4.1 Functional Groups of Instructions

Table 4-3 Instruction Set Summary

Mnemonic	Description	Byte	Cycle
Arithmetic Operations			
ADD A,Rn	Add register to accumulator	1	1
ADD A,direct	Add direct byte to accumulator	2	1
ADD A,@Ri	Add indirect RAM to accumulator	1	1
ADD A,#data	Add immediate data to accumulator	2	1
ADDC A,Rn	Add register to accumulator with carry flag	1	1
ADDC A,direct	Add direct byte to A with carry flag	2	1
ADDC A,@Ri	Add indirect RAM to A with carry flag	1	1
ADDC A,#data	Add immediate data to A with carry flag	2	1
SUBB A,Rn	Subtract register from A with borrow	1	1
SUBB A,direct	Subtract direct byte from A with borrow	2	1
SUBB A,@Ri	Subtract indirect RAM from A with borrow	1	1
SUBB A,#data	Subtract immediate data from A with borrow	2	1
INC A	Increment accumulator	1	1
INC Rn	Increment register	1	1
INC direct	Increment direct byte	2	1
INC @Ri	Increment indirect RAM	1	1
DEC A	Decrement accumulator	1	1
DEC Rn	Decrement register	1	1
DEC direct	Decrement direct byte	2	1
DEC @Ri	Decrement indirect RAM	1	1
INC DPTR	Increment data pointer	1	2
MUL AB	Multiply A and B	1	4
DIV AB	Divide A by B	1	4
DA A	Decimal adjust accumulator	1	1

Table 4-3 Instruction Set Summary (cont'd)

Mnemonic	Description	Byte	Cycle
Logic Operations			
ANL A,Rn	AND register to accumulator	1	1
ANL A,direct	AND direct byte to accumulator	2	1
ANL A,@Ri	AND indirect RAM to accumulator	1	1
ANL A,#data	AND immediate data to accumulator	2	1
ANL direct,A	AND accumulator to direct byte	2	1
ANL direct,#data	AND immediate data to direct byte	3	2
ORL A,Rn	OR register to accumulator	1	1
ORL A,direct	OR direct byte to accumulator	2	1
ORL A,@Ri	OR indirect RAM to accumulator	1	1
ORL A,#data	OR immediate data to accumulator	2	1
ORL direct,A	OR accumulator to direct byte	2	1
ORL direct,#data	OR immediate data to direct byte	3	2
XRL A,Rn	Exclusive OR register to accumulator	1	1
XRL A,direct	Exclusive OR direct byte to accumulator	2	1
XRL A,@Ri	Exclusive OR indirect RAM to accumulator	1	1
XRL A,#data	Exclusive OR immediate data to accumulator	2	1
XRL direct,A	Exclusive OR accumulator to direct byte	2	1
XRL direct,#data	Exclusive OR immediate data to direct byte	3	2
CLR A	Clear accumulator	1	1
CPL A	Complement accumulator	1	1
RL A	Rotate accumulator left	1	1
RLC A	Rotate accumulator left through carry	1	1
RR A	Rotate accumulator right	1	1
RRC A	Rotate accumulator right through carry	1	1
SWAP A	Swap nibbles within the accumulator	1	1
Data Transfer¹⁾			
MOV A,Rn	Move register to accumulator	1	1
MOV A,direct	Move direct byte to accumulator	2	1
MOV A,@Ri	Move indirect RAM to accumulator	1	1

Table 4-3 Instruction Set Summary (cont'd)

Mnemonic	Description	Byte	Cycle
MOV A,#data	Move immediate data to accumulator	2	1
MOV Rn,A	Move accumulator to register	1	1
MOV Rn,direct	Move direct byte to register	2	2
MOV Rn,#data	Move immediate data to register	2	1
MOV direct,A	Move accumulator to direct byte	2	1
MOV direct,Rn	Move register to direct byte	2	2
MOV direct,direct	Move direct byte to direct byte	3	2
MOV direct,@Ri	Move indirect RAM to direct byte	2	2
MOV direct,#data	Move immediate data to direct byte	3	2
MOV @Ri,A	Move accumulator to indirect RAM	1	1
MOV @Ri,direct	Move direct byte to indirect RAM	2	2
MOV @Ri,#data	Move immediate data to indirect RAM	2	1
MOV DPTR,#data16	Load data pointer with a 16-bit constant	3	2
MOVC A,@A + DPTR	Move code byte relative to DPTR to accumulator	1	2
MOVC A,@A + PC	Move code byte relative to PC to accumulator	1	2
MOVX A,@Ri	Move external RAM (8-bit addr.) to A	1	2
MOVX A,@DPTR	Move external RAM (16-bit addr.) to A	1	2
MOVX @Ri,A	Move A to external RAM (8-bit addr.)	1	2
MOVX @DPTR,A	Move A to external RAM (16-bit addr.)	1	2
PUSH direct	Push direct byte onto stack	2	2
POP direct	Pop direct byte from stack	2	2
XCH A,Rn	Exchange register with accumulator	1	1
XCH A,direct	Exchange direct byte with accumulator	2	1
XCH A,@Ri	Exchange indirect RAM with accumulator	1	1
XCHD A,@Ri	Exchange low-order nibble indir. RAM with A	1	1
Boolean Variable Manipulation			
CLR C	Clear carry flag	1	1
CLR bit	Clear direct bit	2	1
SETB C	Set carry flag	1	1
SETB bit	Set direct bit	2	1
CPL C	Complement carry flag	1	1
CPL bit	Complement direct bit	2	1

Table 4-3 Instruction Set Summary (cont'd)

Mnemonic	Description	Byte	Cycle
ANL C	AND direct bit to carry flag	2	2
ANL C,bit	AND complement of direct bit to carry	2	2
ORL C,bit	OR direct bit to carry flag	2	2
ORL C,/bit	OR complement of direct bit to carry	2	2
MOV C,bit	Move direct bit to carry flag	2	1
MOV bit,C	Move carry flag to direct bit	2	2
Program and Machine Control			
ACALL addr11	Absolute subroutine call	2	2
LCALL addr16	Long subroutine call	3	2
RET	Return from subroutine	1	2
RETI	Return from interrupt	1	2
AJMP addr11	Absolute jump	2	2
LJMP addr16	Long jump	3	2
SJMP rel	Short jump (relative addr.)	2	2
JMP @A + DPTR	Jump indirect relative to the DPTR	1	2
JZ rel	Jump if accumulator is zero	2	2
JNZ rel	Jump if accumulator is not zero	2	2
JC rel	Jump if carry flag is set	2	2
JNC rel	Jump if carry flag is not set	2	2
JB bit,rel	Jump if direct bit is set	3	2
JNB bit,rel	Jump if direct bit is not set	3	2
JBC bit,rel	Jump if direct bit is set and clear bit	3	2
CJNE A,direct,rel	Compare immediate to A and jump if not equal	3	2
CJNE A,#data,rel	Compare immediate to A and jump if not equal	3	2
CJNE Rn,#data,rel	Compare immedi. to reg. and jump if not equal	3	2
CJNE @Ri,#data,rel	Compare immedi. to ind. and jump if not equal	3	2
DJNZ Rn,rel	Decrement register and jump if not zero	2	2
DJNZ direct,rel	Decrement direct byte and jump if not zero	3	2
NOP	No operation	1	1
1) MOVA,ACC is not a valid instruction.			

14.3 Assembler - Direktiven

Programm Linie				
Symbol, Adress or Name	Directive	Operand	Function	Beschreibung
Start- und Memory Location Definitions	ORG	<expression>	Set location counter value	Setze den (lokalen) Adresszähler auf <expression>
	END		End of program	Übersetzungsende
	DB	{ <expression> <string> <list> }	Define byte in program memory	Definiere Byte im Programmspeicher
	DW	{ <expression> <string> <list> }	Define word in program memory	Definiere Word (16 Bit) im Programmspeicher
	DS	<expression>	Advance active location counter	Reserviere <expression> Byte im Speicher
	DBIT	<expression>	Advance bit location counter	Reserviere <expression> Bit im adressierbaren Bereich
	EQU	<expression>	Create new symbol	Definition eines neuen Symbols (nicht redefinierbar)
	SET	<expression>	Set symbol value temporarily	Definition eines Symbols (redefinierbar)
	USING	<expression>	Select register bank	Definition der aktuellen Registerbank
	CODE	<expression>	Define code address symbol	Definition eines Symbols im Programmspeicher
Symbol Definitions	DATA	<expression>	Define data address symbol	Definition eines Symbols im internen Datenspeicher
	IDATA	<expression>	Define indirect data address symbol	Definition eines Symbols im indirekt adressierbaren internen Datenspeicher
	XDATA	<expression>	Define an off chip data address symbol	Definition eines Symbols im externen Datenspeicher
	BIT	<expression>	Define a bit address symbol	Definition eines Symbols im bitadressierbaren Bereich
	SEGMENT	{ CODE XDATA DATA IDATA BIT } [-{PAGE INPAGE INBLOCK BITADDRESSABLE UNIT}]	Declare relocatable segment [Assign attributes]	Definition eines Segmentnamens mit einer Typangabe sowie ggf. Speicher- grenzen
	RSEG	<segment-name>	Select relocatable segment	Aktivieren des vorher definierten relativen Segmentes <segment-name>
	CSEG	[AT <absolute address>]	Select code segment	Die folgenden Befehle und Definitionen werden auf absolute Adressen im CODE-Segment bezogen
	DSEG	[AT <absolute address>]	Select internal data segment	Die folgenden Definitionen werden auf absolute Adressen im DATA (interner Datenspeicher)- Segment bezogen
	ISEG	[AT <absolute address>]	Select indirect internal segment	Die folgenden Definitionen werden auf absolute Adressen im IDATA (indirekt adressierbarer interner Datenspeicher)- Segment bezogen
	XSEG	[AT <absolute address>]	Select external data segment	Die folgenden Definitionen werden auf absolute Adressen im XSEG (externer Datenspeicher)- Segment bezogen
Absolute Segmentation	BSEG	[AT <absolute address>]	Select bit address segment	Die folgenden Definitionen werden auf absolute Adressen im BSEG (bitadressierbarer Bereich)- Segment bezogen
	PUBLIC	{<name> <list-of-names>}	Extend Symbol definition outside current module	Erweitern des Gültigkeitsbereiches von <name> bzw. <namensliste> über die Modulgrenzen hinaus
	EXTRN	{ CODE XDATA DATA IDATA BIT NUMBER } ({ <name> <list-of-names> }) [...]	Refer to an external symbol	Bezignahme auf eine Symboldefinition von <name> bzw. <namensliste> in einem anderen Modul (durch PUBLIC)
	NAME	<modul name>	Define module name	Definition des Modul-Namens (Ohne Name-Anweisung wird der Name der Datei ohne Erweiterung ausgewählt)

14.4 Datei Template

Datei: TEMPLATE.A51

This template file TEMPLATE.A51 is provided in the folder \C51\ASM

```

;-----
; Source code template for A251/A51 assembler modules.
; Copyright (c) 1995-2000 KEIL Software, Inc.
;-----
$NOMOD51           ; disable predefined 8051 registers
#include <reg52.h>    // include CPU definition file (for example, 8052)

;-----
; Change names in lowercase to suit your needs.
;
; This assembly template gives you an idea of how to use the A251/A51
; Assembler. You are not required to build each module this way-this is only
; an example.
;
; All entries except the END statement at the End Of File are optional.
;
; If you use this template, make sure you remove any unused segment declarations,
; as well as unused variable space and assembly instructions.
;
; This file cannot provide for every possible use of the A251/A51 Assembler.
; Refer to the A51/A251 User's Guide for more information.
;-----

;-----
; Module name (optional)
;-----
NAME      module_name

;-----
; Here, you may import symbols from other modules.
;-----
EXTRN  CODE   (code_symbol)    ; May be a subroutine entry declared in
                                ; CODE segments or with CODE directive.

EXTRN  DATA   (data_symbol)    ; May be any symbol declared in DATA segments
                                ; or with DATA directive.

EXTRN  BIT    (bit_symbol)     ; May be any symbol declared in BIT segments
                                ; or with BIT directive.

EXTRN  XDATA  (xdata_symbol)   ; May be any symbol declared in XDATA segments
                                ; or with XDATA directive.

EXTRN  NUMBER (typeless_symbol); May be any symbol declared with EQU or SET
                                ; directive

;-----
; You may include more than one symbol in an EXTRN statement.
;-----
EXTRN  CODE  (sub_routine1, sub_routine2), DATA (variable_1)

;-----
; Force a page break in the listing file.
;-----
$EJECT

;-----
```

14 Programmieren Anhang

```
; Here, you may export symbols to other modules.  
-----  
PUBLIC  data_variable  
PUBLIC  code_entry  
PUBLIC  typeless_number  
PUBLIC  xdata_variable  
PUBLIC  bit_variable  
  
-----  
; You may include more than one symbol in a PUBLIC statement.  
-----  
PUBLIC  data_variable1, code_table, typeless_num1, xdata_variable1  
  
-----  
; Put the STACK segment in the main module.  
-----  
?STACK      SEGMENT IDATA           ; ?STACK goes into IDATA RAM.  
             RSEG    ?STACK           ; switch to ?STACK segment.  
             DS     5                ; reserve your stack space  
                           ; 5 bytes in this example.  
  
$EJECT  
  
-----  
; Put segment and variable declarations here.  
-----  
  
-----  
; DATA SEGMENT--Reserves space in DATA RAM--Delete this segment if not used.  
;  
data_seg_name   SEGMENT DATA          ; segment for DATA RAM.  
                 RSEG    data_seg_name  ; switch to this data segment  
data_variable:  DS     1              ; reserve 1 Bytes for data_variable  
data_variable1: DS     2              ; reserve 2 Bytes for data_variable1  
  
-----  
; XDATA SEGMENT--Reserves space in XDATA RAM--Delete this segment if not used.  
;  
xdata_seg_name  SEGMENT XDATA         ; segment for XDATA RAM  
                 RSEG    xdata_seg_name ; switch to this xdata segment  
xdata_variable: DS     1              ; reserve 1 Bytes for xdata_variable  
xdata_array:    DS     500             ; reserve 500 Bytes for xdata_array  
  
-----  
; INPAGE XDATA SEGMENT--Reserves space in XDATA RAM page (page size: 256 Bytes)  
; INPAGE segments are useful for @R0 addressing methodes.  
; Delete this segment if not used.  
;  
page_xdata_seg  SEGMENT XDATA INPAGE ; INPAGE segment for XDATA RAM  
                 RSEG    xdata_seg_name ; switch to this xdata segment  
xdata_variable1:DS     1              ; reserve 1 Bytes for xdata_variable1  
  
-----  
; ABSOLUTE XDATA SEGMENT--Reserves space in XDATA RAM at absolute addresses.  
; ABSOLUTE segments are useful for memory mapped I/O.  
; Delete this segment if not used.  
;  
                 XSEG    AT 8000H       ; switch absolute XDATA segment @ 8000H  
XIO:          DS     1              ; reserve 1 Bytes for XIO port  
XCONFIG:      DS     1              ; reserve 1 Bytes for XCONFIG port
```

```

;-----
; BIT SEGMENT--Reserves space in BIT RAM--Delete segment if not used.
;-----
bit_seg_name SEGMENT BIT ; segment for BIT RAM.
    RSEG bit_seg_name ; switch to this bit segment
bit_variable: DBIT 1 ; reserve 1 Bit for bit_variable
bit_variable1: DBIT 4 ; reserve 4 Bits for bit_variable1

;-----
; Add constant (typeless) numbers here.
;-----
typeless_number EQU 0DH ; assign 0D hex
typeless_num1 EQU typeless_number-8 ; evaluate typeless_num1

$EJECT

;-----
; Provide an LJMP to start at the reset address (address 0) in the main module.
; You may use this style for interrupt service routines.
;-----
        CSEG AT 0 ; absolute Segment at Address 0
        LJMP start ; reset location (jump to start)

;-----
; CODE SEGMENT--Reserves space in CODE ROM for assembler instructions.
;-----
code_seg_name SEGMENT CODE

        RSEG code_seg_name ; switch to this code segment

        USING 0 ; state register_bank used
                  ; for the following program code.

start:      MOV SP, #?STACK-1 ; assign stack at beginning

;-----
; Insert your assembly program here. Note, the code below is non-functional.
;-----
repeat_label: ORL IE, #82H ; enable interrupt system (timer 0)
              SETB TR0 ; enable timer 0
        MOV A, data_symbol
        ADD A, #typeless_symbol
        CALL code_symbol
        MOV DPTR, #xdata_symbol
        MOVX A, @DPTR
        MOV R1, A
        PUSH AR1
        CALL sub_routine1
        POP AR1
        ADD A, R1
        JMP repeat_label

code_entry:   CALL code_symbol
              RET

code_table:  DW repeat_label
              DW code_entry
              DB typeless_number
              DB 0

$EJECT

```

```

;-----  

; To include an interrupt service routines, provide an LJMP to the ISR at the  

; interrupt vector address.  

;-----  

        CSEG    AT 0BH           ; 0BH is address for Timer 0 interrupt  

        LJMP    timer0int  

;  

;-----  

; Give each interrupt function its own code segment.  

;-----  

int0_code_seg SEGMENT CODE      ; segment for interrupt function  

        RSEG    int0_code_seg   ; switch to this code segment  

        USING   1               ; register bank for interrupt routine  

;  

timer0int:    PUSH   PSW  

                MOV    PSW, #08H       ; register bank 1  

                PUSH   ACC  

                MOV    R1, data_variable  

                MOV    DPTR, #xdata_variable  

                MOVX  A, @DPTR  

                ADD   A, R1  

                MOV   data_variable1, A  

                CLR   A  

                ADD   A, #0  

                MOV   data_variable1+1, A  

                POP   ACC  

                POP   PSW  

                RETI  

;  

;-----  

; The END directive is ALWAYS required.  

;-----  

        END      ; End Of File

```

Hinweis:

”\$EJECT” : steuert den Seitenumbruch beim Ausdrucken

14.5 Quellen zu Kap. 14

- [Inf00] Infinion. “C500; Architecture and Instruction Set. User’s Manual”. 2000-07. Infineon Technologies AG, Juli 2000.
- [Sim99] Simens. “C517A, 8-Bit CMOS Microcontroller. User’s Manual”. 01.99. Siemens AG, Bereich Halbleiter, Marketing-Kommunikation, 1999.