

Rechnerarchitektur Zusammenfassung

Einleitung.....	4
Minimalsystem – minimaler Aufbau eines Computers	4
Architektur	4
Übersicht der Komponenten	4
Zusätzliche Komponenten (nicht im Minimalsystem enthalten)	4
Harvard vs. Von Neumann - Architektur	5
CPU: Grobstruktur	6
Funktionsblöcke	6
Zahlendarstellung.....	7
Binärzahlen.....	7
Vorzeichenbetragszahl/Einerkomplement.....	7
Zweierkomplement/Signed Extension	8
Hinweis: Statusbits	8
Genauigkeit	8
Gleitkommazahlen	8
Allgemein.....	8
IEEE 754	9
Arithmetik.....	11
Ganzzahl-Addition	11
Binärzahl.....	11
Vorzeichenbetragszahl/Einerkomplement.....	11
Zweierkomplementzahl.....	11
Assembler-Befehle	12
Addition	12
Addition mit Carry	12
Decimal Adjust.....	12
Realisierung	12
Halbleiter-Addierer.....	12
Voll-Addierer	13
N-Bit-Addierer	13
Carry-Look Ahead Addierer	14
Serien Addierer.....	14
Ganzzahl-Subtraktion	15

Formal	15
Binärzahl	15
Einerkomplement	15
Zweierkomplement	15
Realisierung	15
Hinweis: ALU	15
Ganzzahl Multiplikation	16
Binärzahlen	16
Zweierkomplementzahlen	16
Ganzzahl Division	17
Binär	17
Zweierkomplement	17
Gleitkomma Arithmetik	18
Addition/Subtraktion	18
Multiplikation/Division	18
Bussystem	19
Allgemein	19
Einteilung	19
Nach Funktion (im Minimalsystem)	19
Nach Richtung	20
Nach Übertragungsart	20
Nach Synchronisation	20
Nach Einsatzgebiet	21
Nach Medium	21
Nach Topologie	21
Rechenwerk	22
Register	22
Beschreibung	22
Struktur	22
Steuerwerk	23
Allgemein	23
Strukturen	23
Festverdrahtetes Steuerwerk	23
Steuerwerk mit Mikroprogrammierung	23
Automatenansatz	23
Wilkes-Stringer-Steuerwerk	25

Vergleich Mikro- vs. Maschinenprogrammierung.....	25
Darstellung der Steuersignale	25
Beispielstruktur einer CPU	26
Speicher.....	27
Adressräume	27
Adressierungen.....	27
Anschlüsse	27
Auswahl der Speicherzellen	28
Externe Dekodierung.....	29
Beispiel	30
Interne Adressierung.....	30
Ein Decoder	31
Zwei Decoder.....	31
Ein Decoder und Spaltenlogik	32
Multiplexer von Zeilen- und Spaltenadressen	32
Erweiterung des Minimalsystems	33
Interrupt	33

Einleitung

Funktionsweise eines Computers:

Eingabedaten -> Verarbeitung -> Ausgabedaten
(analog/digital) (elektromechanisch) (analog/digital)

Abarbeitung eines Programms (= Sequenz von Anweisungen):

Anweisung 1, Anweisung 2, , Anweisung n -> sequenzielle Abarbeitung

Minimalsystem – minimaler Aufbau eines Computers

Architektur

Übersicht der Komponenten

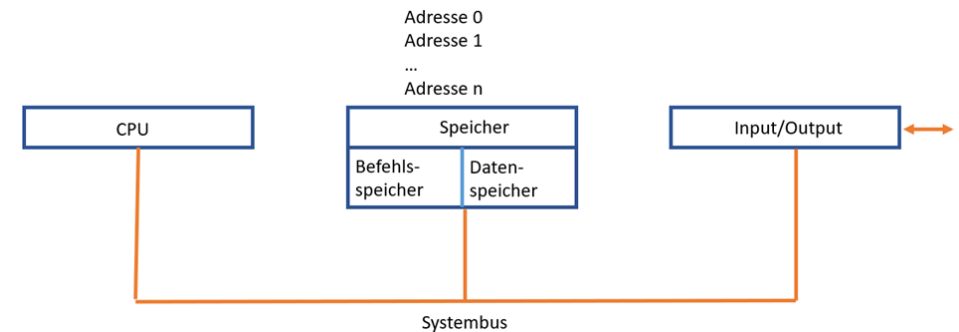
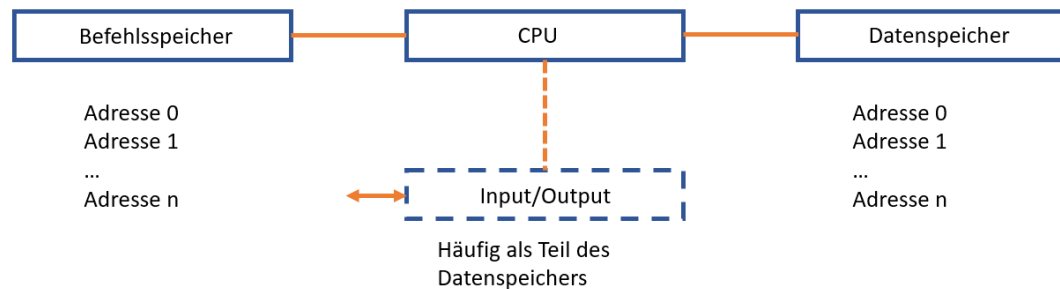
Komponente	Aufgabe
Zentraleinheit (CPU)	Lenkt und steuert alle Aufgaben des PCs
Speicher	Speichert Werte an Adressen in Zellen Programm- und Datenspeicher
Ein-/Ausgabeeinheit (Input/Output)	Verbindung nach Außen Input: Tastatur, Maus, Scanner Output: Grafikkarte, Monitor, Drucker
Bussystem (keine konkrete Komponente)	Menge aller Verbindungen (logisch und physikalisch)

Zusätzliche Komponenten (nicht im Minimalsystem enthalten)

Interrupts	Unterbrechen den normalen Programmablauf Alternativ: Polling -> bestimmter Zeitpunkt um Befehle abzuarbeiten (stetiges Abfragen)
Direct Memory Access (DMA)	Speicherzugriff vom Bussystem direkt auf Speicher, CPU wird dadurch nicht belastet Normaler Ablauf: Festplatte -> CPU -> Programmspeicher Hier: Festplatte -> Programmspeicher
Co Prozessoren	FPU (Floating Point Unit): schnellere Option auf Gleitkommazahlen umzurechnen GPU(Grafikprozessor): Wiedergabe von Bildern/Grafiken ➔ Um CPU zu entlasten
Sonstige HW bei Mikrocontroller	Watch-Dog: überwacht die Funktion andere Komponenten UART: dient zur Realisierung digitaler Schnittstellen Analog-Digital-Umwandler (ADU); Digital-Analog-Umwandler (DAU) Timer

Harvard vs. Von Neumann - Architektur

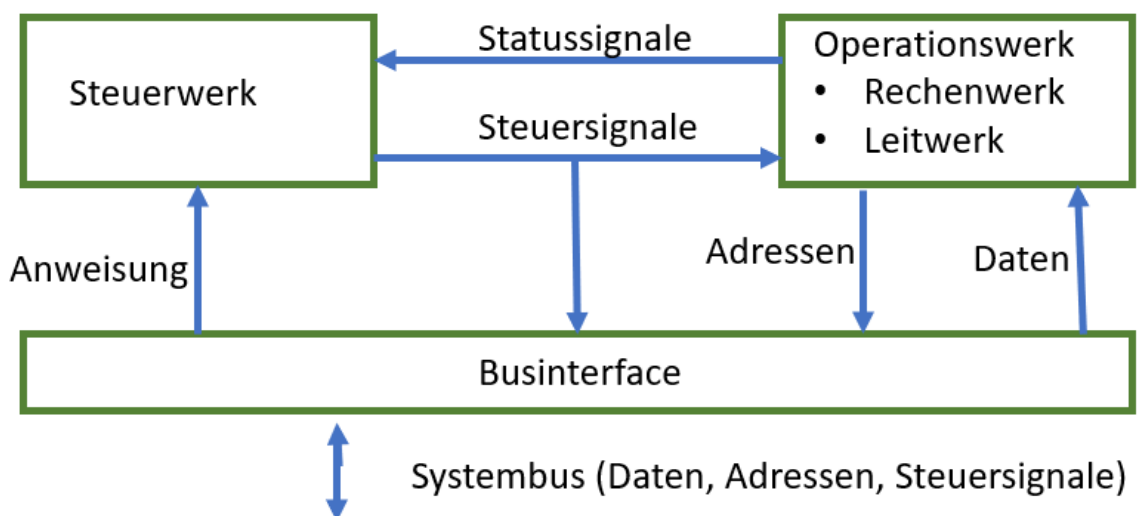
	Harvard	Von Neumann
Vorgang	kompiliertes Programm kann nicht direkt gestartet werden erzeugtes Programm sind Daten und werden beim Datenspeicher (statt Befehlsspeicher) geladen per DMA in Befehlsspeicher kopieren	CPU holt sich über Systembus Speicher Über Systembus zurück und Daten laden
Eigenschaften	Getrennter Adressraum für Befehle und Daten Getrennte Busleitungen	Gemeinsamen Adressraum Ein Bussystem
Vorteile	Befehle und Daten können gleichzeitig geladen werden -> schnell Trennung hilft, dass bei fehlerhafter Software kein Code (nur Daten) überschrieben werden können	Flexible Aufteilung des Speichers zwischen Programmen und Daten durch ein Bussystem
Nachteile	Freier Programmspeicher kann nicht für Daten genutzt werden Freier Datenspeicher kann nicht für Programme genutzt werden Aufwendig Adressraum ist doppelt vorhanden	Programme und Daten sich nicht unterscheidbar (gemeinsamer Speicher), bei falscher Adressierung -> Speicherinhalte könnten verändert werden



CPU: Grobstruktur

Funktionsblöcke

Steuerwerk	Steuert die Abläufe Input: Anweisungen, Statusmeldungen Output: Steuersignale, Gating-Signale(Trigger)
Operationswerk	Führt Abläufe aus Rechenwerk ALU (Arithmetisch-logische-Einheit) Register (Speicher in CPU) Leitwerk (MMU) Adressberechnungen (virtuell in physisch) Liefert Adressen
Businterface	Kontakt nach Außen, steuert die Buszugriffe Input Von innen: Daten, Adressen Von außen: Daten, Adressen, Steuersignale Output Nach innen: Daten, Anweisungen Nach außen: Daten, Adressen Steuersignale



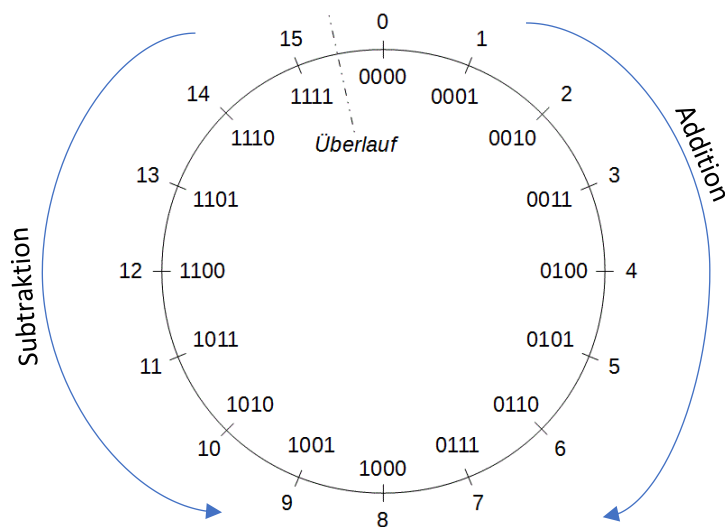
Alternativ: Steuerwerk + Leitwerk = CPU oder Businterface + Leitwerk = Businterface

Zahlendarstellung

- Allgemein: unendlich viele Zahlen, in CPU: endliche Stellenanzahl
- 4 Bits -> 16 Segmente
- Bei Überlauf: Carry-Bit = 1

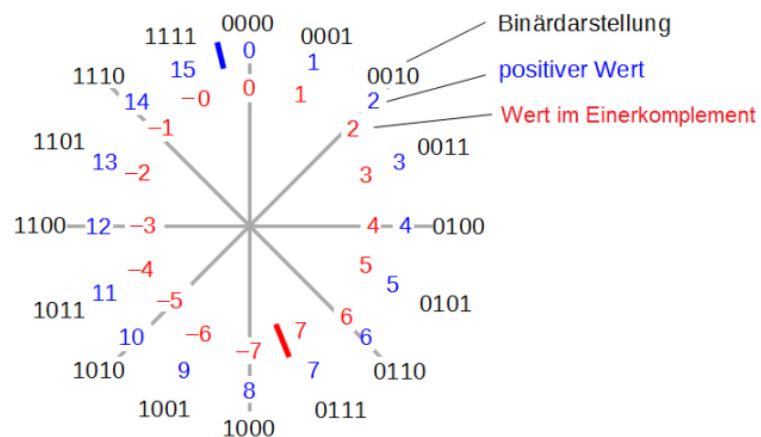
Binärzahlen

- $15+1=0$
- Falls 2^n nicht ausreicht 2^{n+1} Bits nötig
- Darstellung: 0010 = 00000010



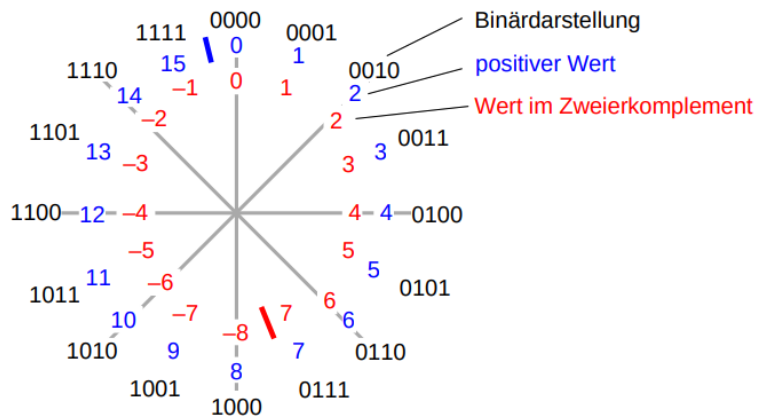
Vorzeichenbetragszahl/Einerkomplement

- erstes Bit gesetzt -> negative Zahl
- Darstellung: alle Bits negieren
- Falls 2^n nicht ausreicht 2^{n+1} Bits nötig
- Darstellung: 1010 = 10000010



Zweierkomplement/Signed Extension

- Bits negieren, 1 addieren
- Bei Bereichsüberschreitung (z.B. $7+1$) -> Overflow-Bit = 1
- Falls 2^n nicht ausreicht 2^{n+1} Bits nötig
- Darstellung: $1010 = 1111010$ -> Mit most significant Bit ergänzen



Hinweis: Statusbits

Programm Status Wort (PSW)

CY	AC F0 R51 R50	OV F1 P (Parität)
Carry	auxiliary Carry	Overflow flag
C ₇	C ₃ (BCD)	C ₇ ≠ C ₆

Genauigkeit

Abstand zwischen zwei Werten: $(z+1)-z = 1 = \Delta z$

$\Delta z/z \rightarrow$ nicht konstant

Gleitkommazahlen

Allgemein

- Exponentialdarstellung
 - Vorzeichen V
 - Basis B
 - Exponent E
 - Mantisse M
 - Wert ergibt sich aus $Z = V * M * B^E$
 - Problem: pro Wert mehrere V, M, B, E möglich
 - Lösung: Normierung durch IEEE 754

IEEE 754

Bestandteile

- $B = 2$
- $V = (-1)^V$
- $E = C$ (Charakteristik) – S (Offset)
- $M = (1.M)_B$
- Wert: $Z = (-1)^V * (1.M)_B * 2^{C-S}$

	Normal, short	Long, real
S	127	1023
C	8 Bit	11 Bit (max. 2047)
M	23 Bit	52 Bit
V	1 Bit	1 Bit

Bitfolge |V|C|M| (Hinweis: es wird auch nur V, C, M gespeichert)

Sonderwerte

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

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

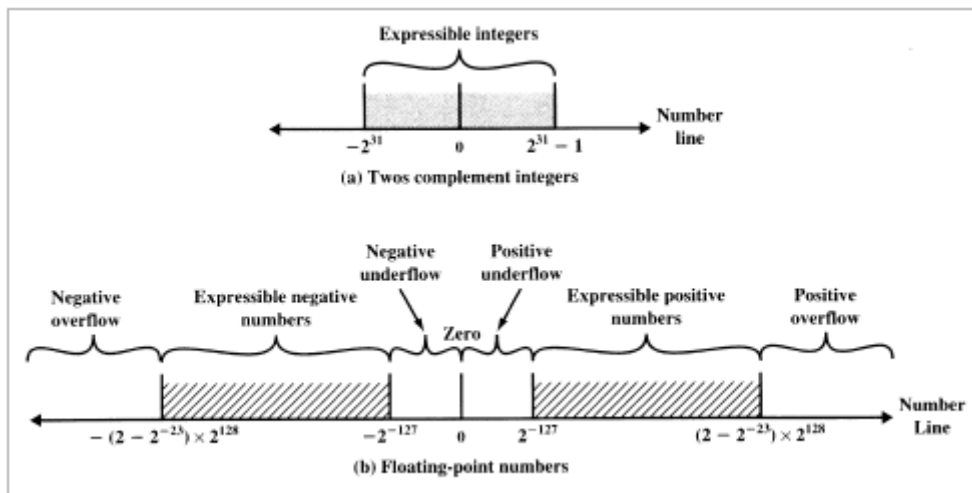
Unendlich: $C = \text{max} \wedge M = 0$

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

Eigenschaften

- Kleinster Wert
 - $C = 1 \wedge M = 1.000....$
 - Short:
 - $E = 1-127 = -126$
 - $Z = 1.00_B * 2^{-126} \approx 2,2_D * 10^{-38}$
 - Long:
 - $E = 1-1023 = -1022$
 - $Z = 1.00_B * 2^{-1023} \approx 2,2_D * 10^{-308}$
- Größter Wert
 - $C = \text{max} - 1 \wedge M = 1.111....$
 - Short:
 - $E = 254 - 127 = 127$
 - $Z = (2-2^{-23}) * 2^{127} \approx 3,4_D * 10^{38}$
 - Long:
 - $E = 2046-1023 = 1023$
 - $Z = (2-2^{-52}) * 2^{1023} \approx 1,8_D * 10^{308}$
- Fehler/Genauigkeit
 - $\Delta Z \neq \text{konstant}$
 - $\Delta Z/Z \approx$ Änderung des LSB (Lowest significant Bit) in M
 - Short $\approx 2^{-23} \approx \frac{1}{10^6}$
 - Long $\approx 2^{-52} \approx \frac{1}{10}$

Schaubild und Beispiele



Umrechnungen mit short

- $52D = 110100_B \cdot 2^0 = 1.10100_B \cdot 2^5$
 32 Bit, S=127, V=0, M= 1.10100...0 (32 Bit), C= 5 + 127 = 132_D = 10000100_B
- $1001\ 1001\ 1001\ 1001\ 1001\ 1001\ 1001\ 1001$
 V C M
 $C = 00110011_B = 51_D$
 $E = 51 - 127 = -76$
 $M = 1001\ 1001\ 1001\ 1001\ 1001\ 1001\ 1001\ 1001 \cdot 2^{-23}$
 $\quad\quad\quad 9\quad 9\quad 9\quad 9\quad 9\quad 9\quad 9\quad 9 \cdot 2^{-23}$
 $= 10066329_D \cdot 2^{-23}$
 $Z = (-1)^1 \cdot 10066329_D \cdot 2^{-23} \cdot 2^{-76} \approx -1,59 \cdot 10^{-23}$

Rechenfehler

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

Arithmetik

Ganzzahl-Addition

- 1. Stelle \triangleq Stellenwertzahl \triangleq wie Dezimalzahl

Binärzahl

- Keine Bereichsüberschreitung

$$\begin{array}{r} 0010 \\ + 0011 \\ \hline 001 \\ \hline 0101 \end{array} \quad \text{mit 4 Bit darstellbar}$$

- Bereichsüberschreitung

$$\begin{array}{r} 00100 \\ + 00101 \\ \hline 01 \\ \hline 01001 = 9 \end{array} \quad \text{mit 5 Bit darstellbar}$$

➔ Bereichsüberschreitung -> n+1 Bit nötig

Vorzeichenbetragszahl/Einerkomplement

- Fallunterscheidung nötig, vom Vorzeichen abhängig
- Negative Zahlen werden bitweise invertiert
- 1. Bit gesetzt \triangleq negative Zahl, ansonsten normal addieren

$$14 - 7 = 7$$

$$\begin{array}{r} 01110 \\ + - 11000 \\ \hline 1 \\ \hline 100110 \\ \hline 1 \\ \hline 00111 \end{array}$$

➔ Überlauf -> 1 Bit addieren

Zweierkomplementzahl

- Falls $c_{n-1} \neq c_{n-2}$ -> nicht mit n Bit darstellbar
- Keine Bereichsüberschreitung

$$\begin{array}{r} 2+3=5 \qquad -2+3=5 \\ 0010 \qquad 1110 \\ + 0011 \qquad + 0011 \\ \hline 001 \qquad 111 \\ \hline 0101 \qquad 0001 \end{array}$$

- Bereichsüberschreitung

$$\begin{array}{r} 4+5=9 \qquad -6+(-5)=-11 \\ 00100 \qquad 11010 \\ + 00101 \qquad + 11011 \\ \hline 0010 \qquad 11010 \\ \hline 01001 \qquad 10101 \end{array} \quad \left. \vphantom{\begin{array}{r} 00100 \\ + 00101 \\ \hline 0010 \\ \hline 01001 \end{array}} \right\} \text{Mit 1 erweitern, da negative Zahl}$$

Assembler-Befehle

Addition

- ADD A,Rn ; $A \leftarrow A + Rn$
- Beeinflusst C, OV ->
 - C set if carry out of Bit 7
Cleared otherwise
 - OV set if carry out of Bit 7 and not out of 6
Or not out of 7 and out of 6
cleared otherwise

Addition mit Carry

- ADDC A,Rn; $A \leftarrow A + Rn + C$
- Beeinflusst C, OV

Decimal Adjust

- Für BCD Zahl
- DA; Korrektur des Akkumulators für BCD
- Beeinflusst C

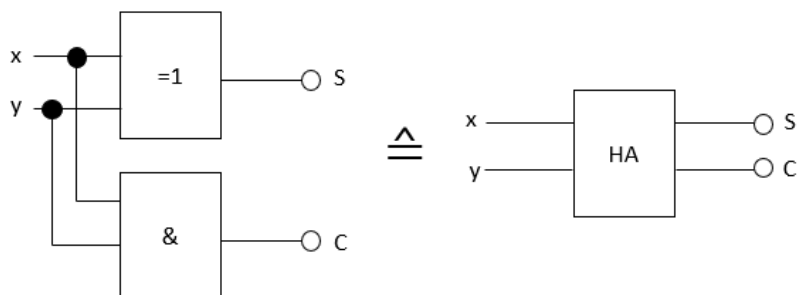
Realisierung

Halbleiter-Addierer

- Formel: $x+y = \text{carry} | \text{summe}$ (Pipe steht für einzelne Bits); mit $x,y,c,s \in \{0,1\}$
- Wahrheitstafel

X	Y	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

- Gleichung
 - $S = (\neg X \wedge Y) \vee (X \wedge \neg Y) = X \oplus Y$
 - $C = X \wedge Y$
- Schaltung



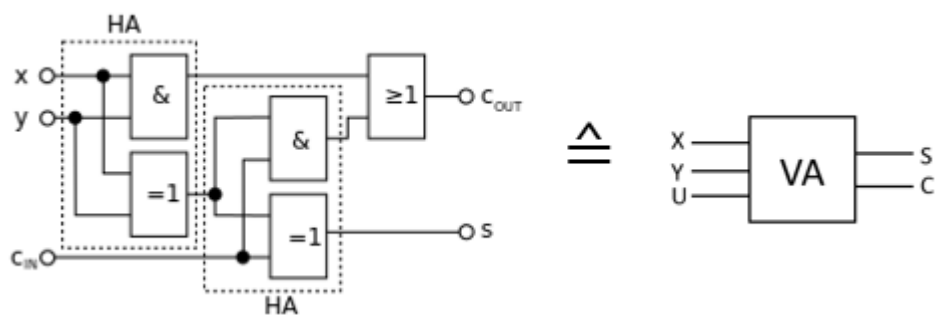
- Laufzeit
 - Sei ΔT Laufzeit eines UND/ODER Gatters
 - $t_{HA} = 2\Delta T$

Voll-Addierer

- Formel: $x+y+u=c \mid s \rightarrow u = \text{Übertrag}$
- Wahrheitstafel

	X	Y	U	C	S
0	0	0	0	0	0
1	0	1	0	0	1
2	1	0	0	0	1
2	1	1	0	1	0
1	0	0	1	0	1
2	0	1	1	1	0
2	1	0	1	1	0
3	1	1	1	1	1

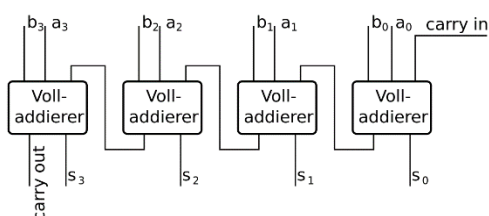
- Gleichung
 - $S = (\neg x \wedge y \wedge \neg z) \vee (x \wedge \neg y \wedge \neg u) \vee (\neg x \wedge \neg y \wedge u) \vee (x \wedge y \wedge u)$
 - $C = (x \wedge y) \vee (x \wedge u) \vee (y \wedge u) = (x \wedge y) \vee (x \wedge y) \wedge u$
- Schaltung



- Laufzeit
 - $T_{VA, \text{ aus HA}} = 5\Delta T$

N-Bit-Addierer

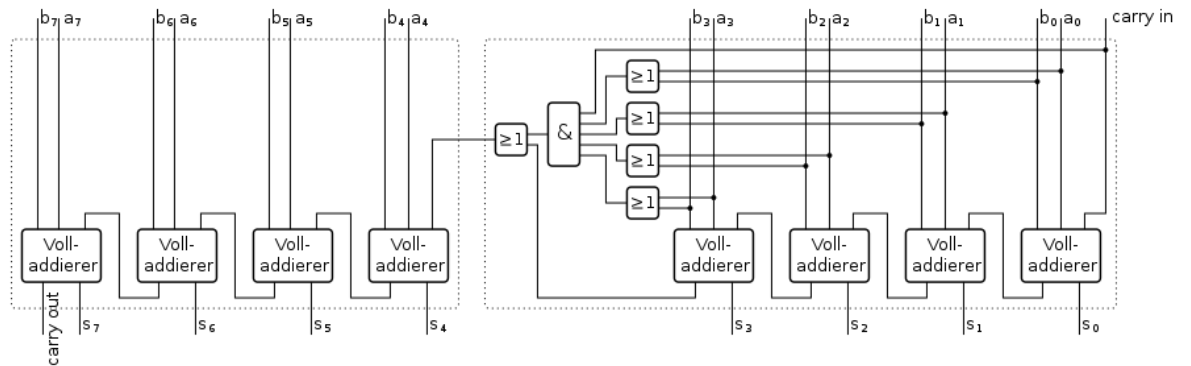
- Formel: $\xrightarrow{b} \xrightarrow{a} + u = c \mid \xrightarrow{s}$
- Schaltung



- Laufzeit
 - Von N (weil auf Vorgänger gewartet werden muss) abhängig
 - $t_{RC} = N \cdot t_{VA} = 2N \cdot \Delta t$

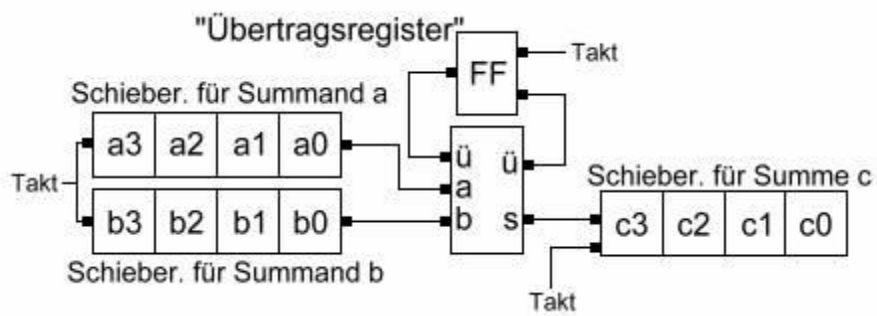
Carry-Look Ahead Addierer

- Addierer mit vorab berechnetem Übertrag
- Problem: Laufzeit des Carrys, Lösung: Carry früher bereitstellen
- Laufzeit $5\Delta T$



Serien Addierer

- Addition mit Speicher
- Schaltbild



Ganzzahl-Subtraktion

Formal

Binärzahl

- Binäre Subtraktion durch Addition der Zweierkomplementdarstellung
 - $5-2 = 5 + (-2)$

$$\begin{array}{r} 0101 \\ 1110 \\ \hline 0011 \end{array}$$

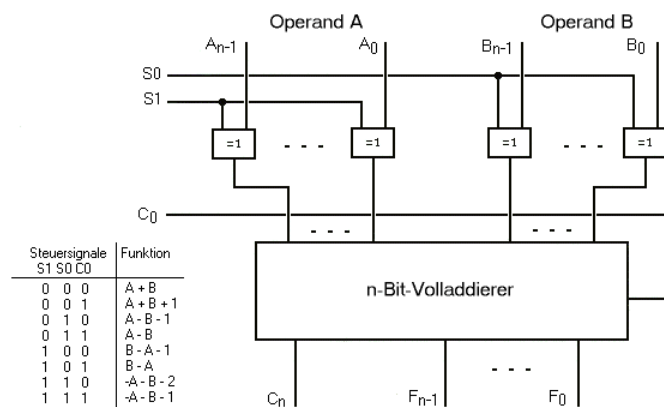
Einerkomplement

- Je nach Vorzeichen andere Regeln (Fallunterscheidungen)

Zweierkomplement

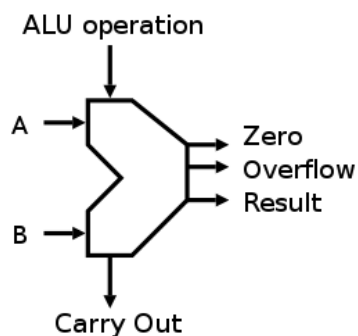
- $a > 0, b > 0 \rightarrow$ Regeln wie Binär
- $a < 0, b > 0 \rightarrow -2-4 = (-2) + (-4) \rightarrow$ binäre Addition
- $a < 0, b < 0 \rightarrow (-2)-(-4) = -2+4 \rightarrow$ binäre Addition

Realisierung



Hinweis: ALU

- Enthält kombinatorische Logik + arithmetische Funktionen
- Teil der CPU



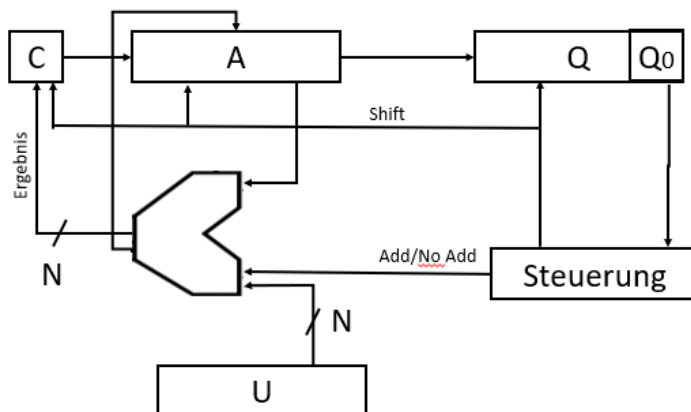
Ganzzahl Multiplikation

Binärzahlen

- Wie dezimale schriftliche Multiplikation (Stellenzahl erhöht sich)

Multiplikation dual	
1 1 0 0	* 1 1 1 0
<hr/>	
1 1 0 0	
1 1 0 0	
1 1 0 0	
0 0 0 0	
<hr/>	
1 0 1 0	1 0 0 0
<hr/>	

- Schaltung



Zweierkomplementzahlen

- Fallunterscheidung
 - $u > 0, v < 0 \rightarrow$ Zahlen tauschen
 - $u < 0, v < 0 \rightarrow$ beide negativ \rightarrow wie binär
 - $u > 0, v > 0 \rightarrow$ wie binär
- Algorithmus
 - Negative Zahl? \rightarrow merken
 - Negative Zahl in positive umwandeln
 - Zahlen multiplizieren
 - Ergebnis gegebenenfalls negieren

Ganzzahl Division

Binär

- $u/v = y; r$

$$\begin{array}{r}
 1001\textcolor{red}{0}\textcolor{blue}{1}\textcolor{orange}{1}\textcolor{blue}{0} : 110 = 1\textcolor{red}{1}\textcolor{orange}{0}\textcolor{blue}{0}\textcolor{blue}{1}\textcolor{blue}{0} \\
 \hline
 - 110 \\
 \hline
 0011\textcolor{red}{0} \\
 - 110 \\
 \hline
 000\textcolor{orange}{1}\textcolor{orange}{1}\textcolor{orange}{0} \\
 - 110 \\
 \hline
 000\textcolor{blue}{0}
 \end{array}$$

- Shift-Operation, wenn Dividend und Divisor Zweierpotenzzahl

$$\begin{aligned}
 4:2=2 &\rightarrow 2^2 : 2^1 = 2^1 \\
 100 : 010 &= 010
 \end{aligned}$$

Zweierkomplement

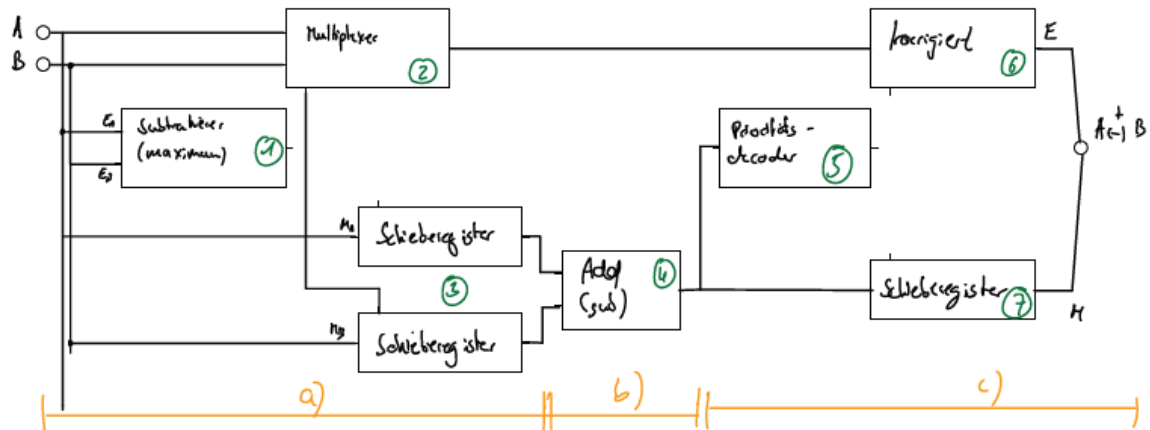
$$\begin{array}{r}
 \text{Zweierkomplement} \\
 \hline
 1100\textcolor{red}{1}\textcolor{red}{1}\textcolor{red}{1} / 10010 = 1011 \text{ Ergebnis} \\
 \hline
 + \textcolor{purple}{10111} \\
 \hline
 \cancel{4}00011\textcolor{red}{1}\textcolor{red}{1} \\
 + \textcolor{purple}{10111} \\
 \hline
 \cancel{4}00110\textcolor{orange}{1} \\
 + \textcolor{purple}{10111} \\
 \hline
 \cancel{4}00100 \text{ Rest}
 \end{array}$$

10111

Gleitkomma Arithmetik

Addition/Subtraktion

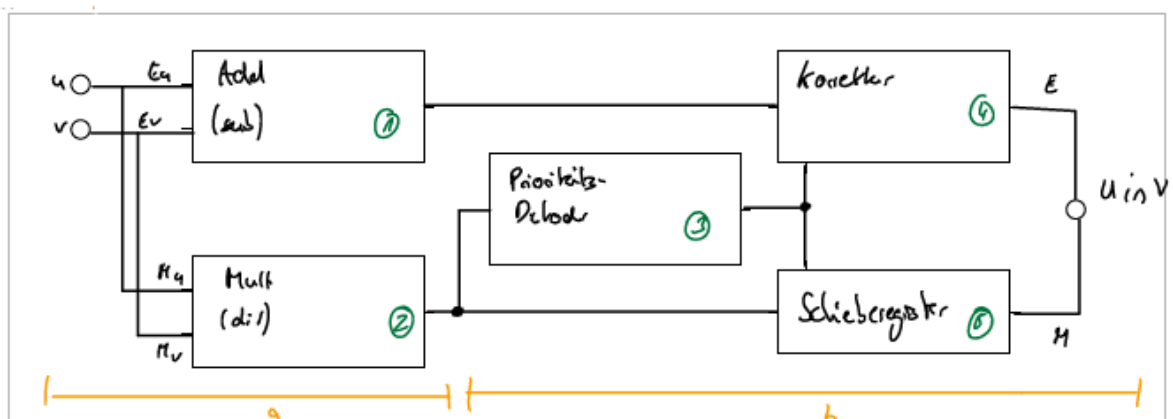
- Vorgehen
 - Exponenten angleichen (Komma verschieben, sodass $E = \max(E_A, E_B)$)
 - Mantisse verarbeiten (addieren/subtrahieren)
 - Normieren, sodass Mantisse = 1.XXX



- a) 1. Max (EA,EB) bestimmen, Differenz (EA,EB)
2. Exponent auswählen
3. Mantisse/Komma verschieben
- b) 4. Mantisse addieren/subtrahieren
- c) 5. Höchste „1“ suchen/höchste Zahl
6. Exponent korrigieren
7. Mantisse korrigieren

Multiplikation/Division

- Vorgehen
 - Exponenten verrechnen (addieren/subtrahieren)
 - Mantisse verrechnen (multiplizieren/dividieren)
 - Normieren, sodass Mantisse = 1.XXX

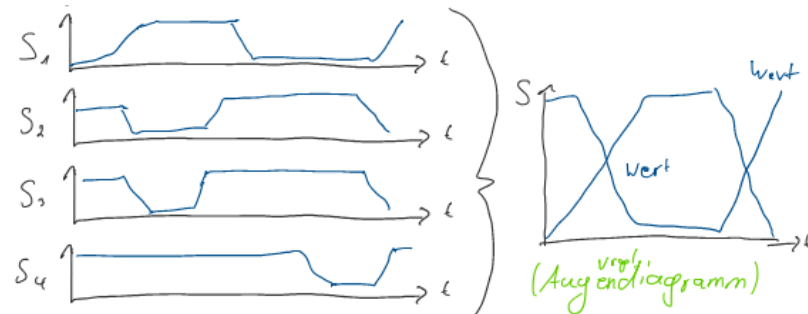


- a)
 1. Exponent: Add/Sub
 2. Mantisse: Mult/Div
- b)
 3. Führende „1“ suchen
 4. Exponent korrigieren
 5. Mantisse korrigieren

Bussystem

Allgemein

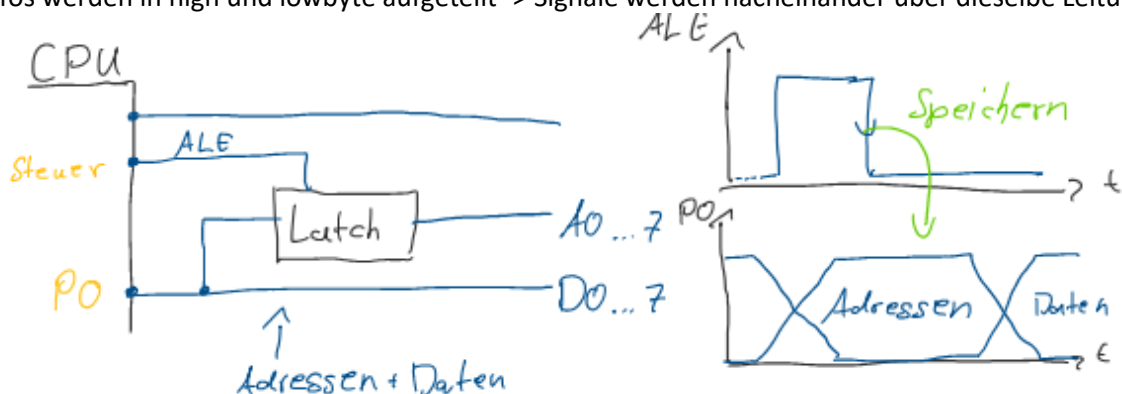
- Aufgabe: Verbindung von/zwischen Geräten
- Zusammenfassung von Leitungen/Signalen
- Darstellung
 - Schematisch
 - Zeitdiagramm





Einteilung

Nach Funktion (im Minimalsystem)

Datenbus	Übertragung von Daten/Befehle zwischen HW-Komponenten (z.B. Prozessor, RAM)
Adressbus	Teil des Systembusses Rechenaufgaben und Position im Arbeitsspeicher werden transportiert Anzahl Adressleitungen = Anzahl der Adressiermöglichkeiten 2 Typen: <ul style="list-style-type: none"> • Decoder = Gruppe von Leitungen • Direkt = eine Leitung pro Gerät
Steuerbus	Steuerung des Ablaufs Lese-/Schreib-Steuerung (=Datenflussrichtung) Daten-Adressgültigkeit (enable)
Systembus	Bei von Neumann-Architektur Daten-, Adress-, und Steuerbus in einem
Multiplexer	Busse teilen sich Infos Infos werden in high und lowbyte aufgeteilt -> Signale werden nacheinander über dieselbe Leitung gesendet



Nach Richtung

Unidirektional (simplex)	Didirektional (duplex)
Informationsfluss in eine Richtung	Informationsfluss in beide Richtungen Ausführung <ul style="list-style-type: none"> Halbduplex (senden/empfangen abwechselnd) Vollduplex (gleichzeitig)
	

Beispiel

Datenbus	Bidirekt, halbduplex	CPU ↔ Speicher
Adressbus	Unidirekt	Adr. → Speicher
Steuerbus	Meist unidirekt	CPU → Speicher

Nach Übertragungsart

Seriell	Parallel
Ein Signal Takt: <ul style="list-style-type: none"> Baud-Rate = $1/T$ Bit-Rate = Bit/T 	Mehrere Signale gleichzeitig
N -Bit pro Leitung	Meist 1 Bit pro Leitung/Signal

Hinweis

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

Nach Synchronisation

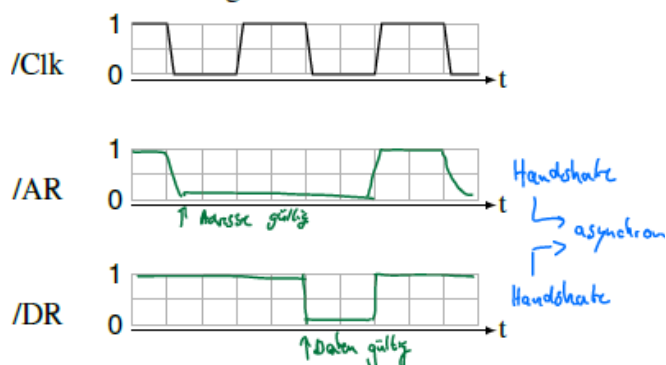
- Zeitliche Absprache zwischen Senden und Empfangen

Takt	Erkennung
Synchron (z.B. bei steigender Flanke, Clock) Asynchron (ohne Takt)	Handshake (asynchron) <ul style="list-style-type: none"> Ohne Rückmeldung Mit Rückmeldung Bit-Synchronisation

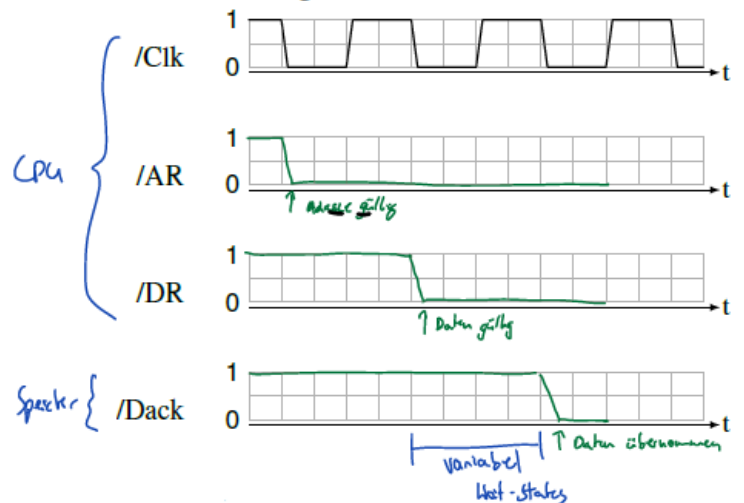
Beispiel

Asynchron (aber synchronisiert)

★ ohne Rückmeldung



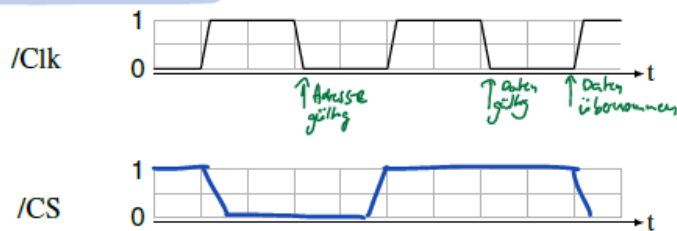
★ mit Rückmeldung



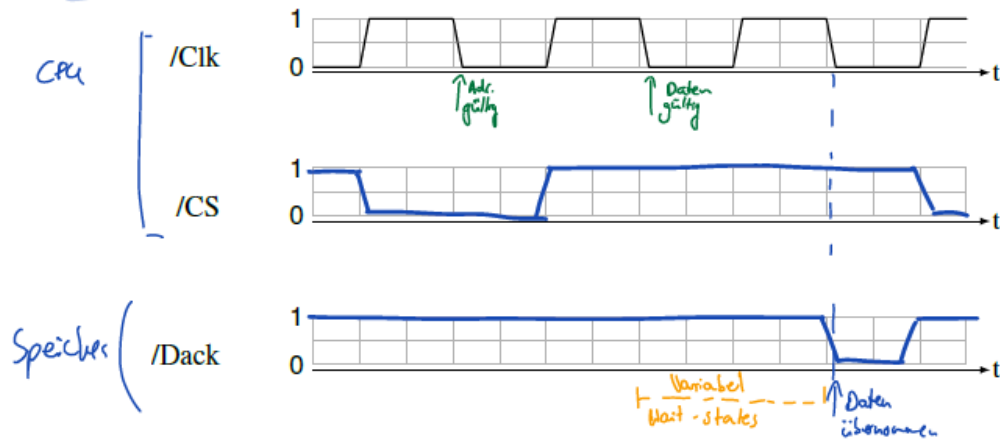
Synchron

jedoch mit Chip-Selekt (Handshake)

★ ohne Rückmeldung



★ mit Rückmeldung



Nach Einsatzgebiet

Feldbusse für Industrie-Automatisierung	PC-Bus	PC-Grafik	Industrie PC
CAN-Bus Profi-Bus	ISA EISA MCA PCI PCI-e	VESA-Local Bus AGP	Compact PCI VME PC

Nach Medium

Elektrisch	Optisch
Leistungsgebunden	Leitungsgebunden
Freiraum	Freiraum

Nach Topologie

Physikalisch	Logisch
Bus Punkt zu Punkt	Bus Stern Baum Maschen (Gitter, Cube, Hyper Cube)

Rechenwerk

Rechenwerk = ALU + Register

Register

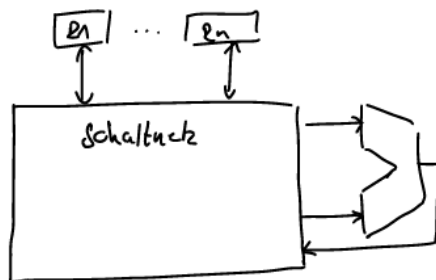
Arbeitsregister	Akkumulator
Allgemeine Register	R ₀ -R ₇ in 8051
Spezielle Register	Status (Programstatus) Stack (SP) Data Pointer (DPRT) Programmzähler Adressregister General Purpose Register

Beschreibung

Hardware Beschreibung	Funktions-Beschreibung
Auf Gatterebene	Verarbeitung der Daten
VHDL (HW-Beschreibungssprache)	RTL (Register-Transfer-Language) ISA (Instruction-Set)

Struktur

- Direkt verdrahtet -> Verbindung durch Schaltnetz



- Bussystem

Ein Bus, zeitaufwändig, nicht parallel	Zwei Busse

Mehr Busse -> schneller und aufwändiger

Steuerwerk

Allgemein

- Aufgabe: Steuerung der Steuerleitungen je nach Befehl
- Eingangssignale
 - Instruktionen/Befehle (ans Register)
 - Statusflags (von ALU)
 - Evtl Takt
- Ausgangssignale
 - Steuersignale/Gating-Signale

Strukturen

Festverdrahtetes Steuerwerk

Allgemein

- Wahrheitstafel -> Gleichung -> Schaltung
- Ohne Struktur
- Eigenschaften: Schnell, unflexibel (bei Änderung alles neu machen)

Steuerwerk mit Mikroprogrammierung

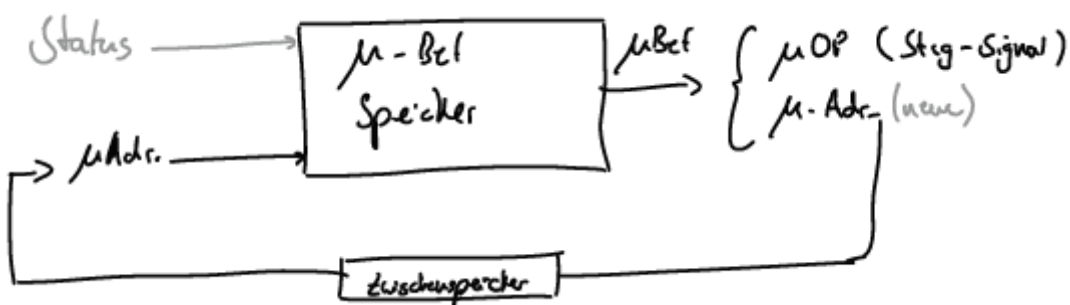
- Als endlicher Automat mit Schaltwerk

Programmansatz

1. Abläufe in der CPU in μ -Operationen (= Sequenz von Steuersignalen) zerlegen
2. μ -Operationen mit μ -Adressen durchnummerieren
3. Analyse von Befehlsabfolgen -> Wiederholt sich μ -Programm?

Unterprogrammstruktur

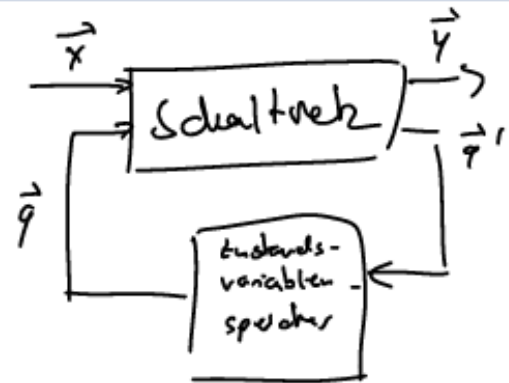
- μ -Operation mit μ Adresse ergänzen -> μ -Befehl = [μ Operation | μ Zieladresse]
- μ Befehle bilden μ Programm
- μ Programm steht im μ Programmspeicher
- Adressierung durch μ Adresse und externem Signal



Automatenansatz

Endlicher Automat (=endliche Anzahl von inneren Zuständen)

Eingangsvariable: $\vec{x} \in X$
 Ausgangsvariable: $\vec{y} \in Y$
 Innerer Zustand: $\vec{q}, \vec{q}' \in Q$



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

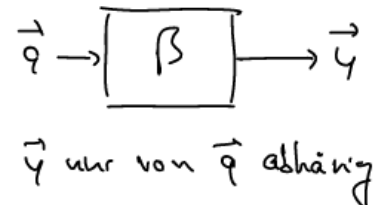
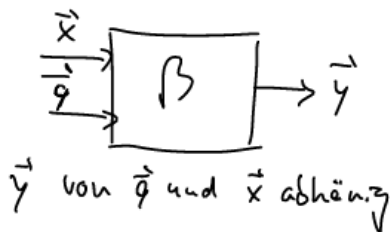


- für Ausgangsvariable \vec{y}

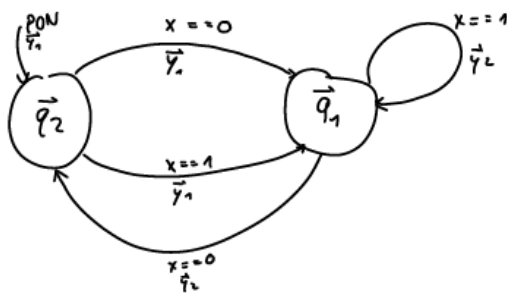
★ Mealy:
 $\beta : X + Q \rightarrow Y$ bzw. $\vec{y} = \beta(\vec{x}, \vec{q})$

oder (!)

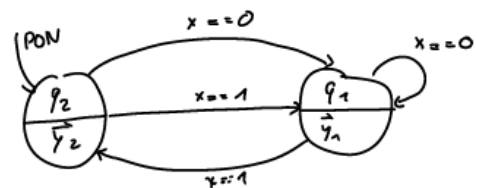
★ Moore:
 $\beta : Q \rightarrow Y$ bzw. $\vec{y} = \beta(\vec{q})$



★ Mealy
 Übergangspfeile mit Ausgangswert



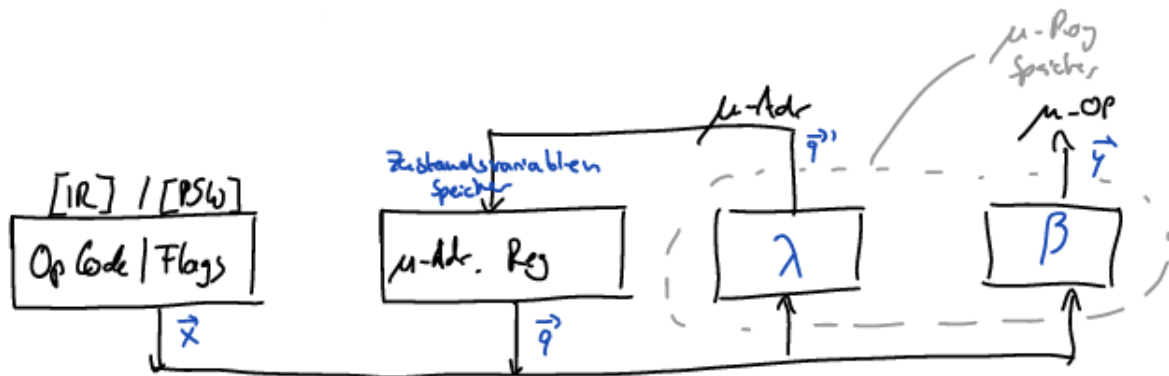
★ Moore
 Zustand mit Ausgangswert



Wilkes-Stringer-Steuerwerk

Aufbau

- Instuktionsregister + Statusflags
- μ -Adressenregister
- Schaltnetze β, λ



	Automat	Prozessor
\rightarrow q	Innerer Zustand	μ -Adresse
\rightarrow x	Eingangsvariable	Befehl+Flag
\rightarrow y	Ausgangsvariable	μ Operation
	Zustandsvariablenspeicher	μ Adress-Register
	β, λ , Netz	μ Programmspeicher
$\rightarrow + \rightarrow$ $q' \quad y$		μ -Befehl

Hinweis

- Wenn HW sich ändert \rightarrow μ -Programm ändert sich
- μ -Befehle sind nicht vom Anwender zugänglich

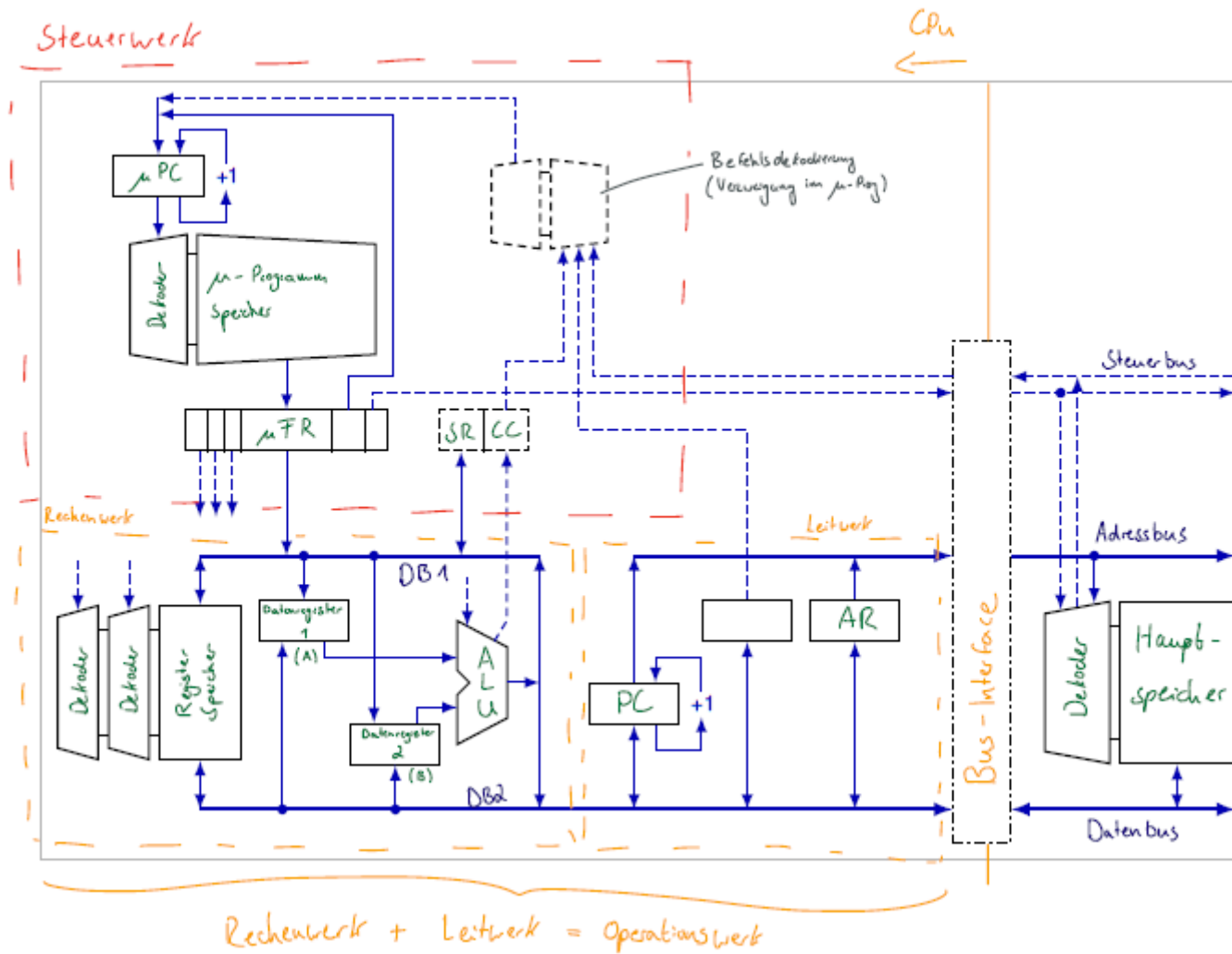
Vergleich Mirko- vs. Maschinenprogrammierung

	Maschinenprogramm	μ -Programm
Vorgegebener Befehlsvorrat	Von Maschinenbefehlen in einer CPU	Eines Steuerwerks von μ -Operationen
Realisierung eines	Algorithmus	Befehl
Durch Folge von	Befehlen ergibt Programm	μ Operationen ergibt μ Programm
Realisierung alt	Nicht programmierbarer Speicherrechner	Fest verdrahtetes Steuerwerk
Aktuelle	Prozessor	μ -programmierbare Steuerwerke
Als Interpreter für	Programme	μ -Programme

Darstellung der Steuersignale

Nicht codiert	Voll codiert	Andere Bezeichnungen
Jedes Gating-Signal kontrolliert eine Funktion Beziehung: horizontale μ -Programmierung	Gruppe von Signalen steuern Gruppe von Funktionen Beziehung: vertikale μ Programmierung, Nanocode	Teilweise codiert: Nanocode Vollständig codiert: Picocode

Beispielstruktur einer CPU



Speicher

Adressräume

- Speicher liefert Werte
- Arten
 - Programmspeicher
 - Datenspeicher
 - I/O-Bereich
- Unterscheidung
 - Per SW: Befehl im Programm
 - Per HW: Steuerleitung
- Aufteilung der Adressierung
 - Alles getrennt
 - Daten- und Programmspeicher, I/O getrennt
 - Datenspeicher und I/O, Programmspeicher getrennt
 - Alles gemeinsam

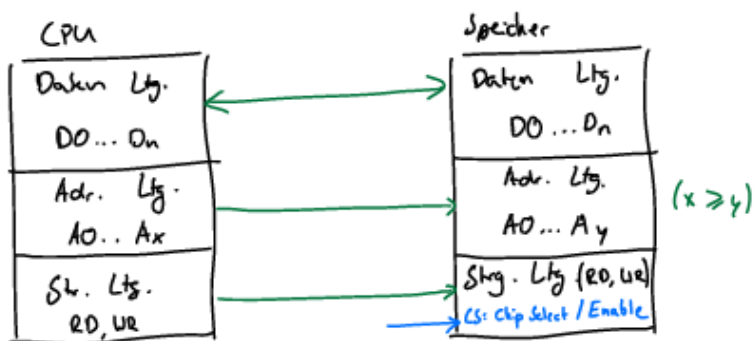
Adressierungen

... sind Zuordnungen zwischen

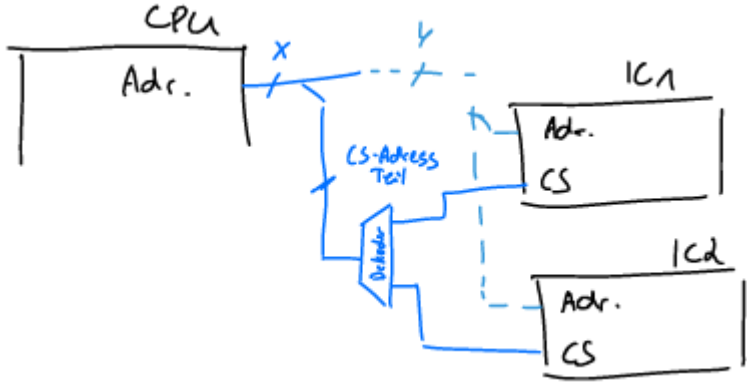
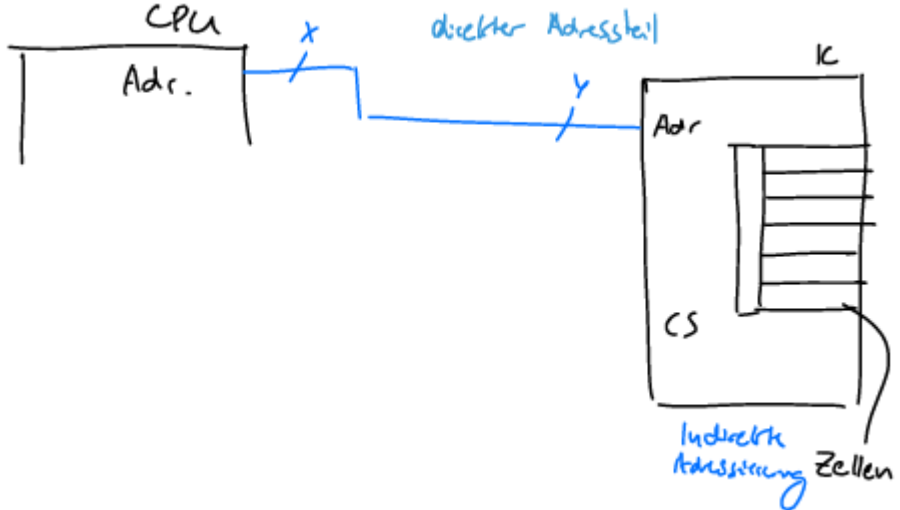
Logischen Adressen	Physikalischen Adressen
Logischen Speicherbereichen	Speicher IC
CPU, Programmworten	Speicherzellen
CPU-Anschlüssen	Speicheranschlüssen

Anschlüsse

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



Auswahl der Speicherzellen

Auswahl des IC durch CS	Auswahl der Speicherzelle
Decoder außerhalb von CPU/Sp. IC → Externe Adressierung → CS-Adressteil	Decoder innerhalb von Sp. IC → Interne Adressierung → Direkter Adressteil
	

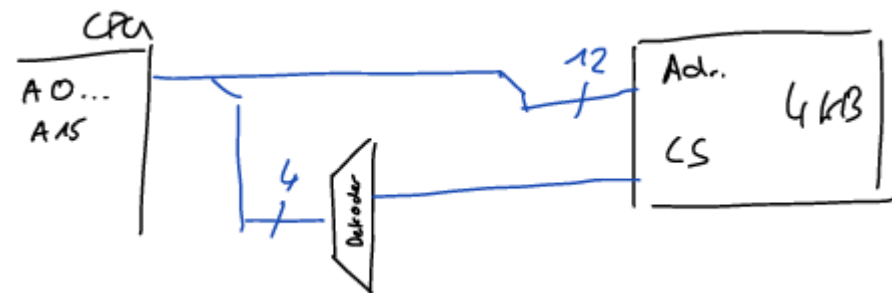
→ Zusammen: Zwei Adressteile, CS-Adressteil und direkter Adressteil

Beispiel


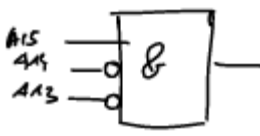
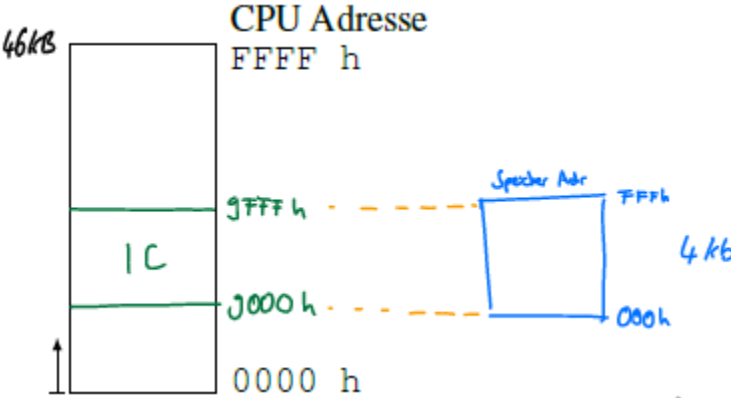
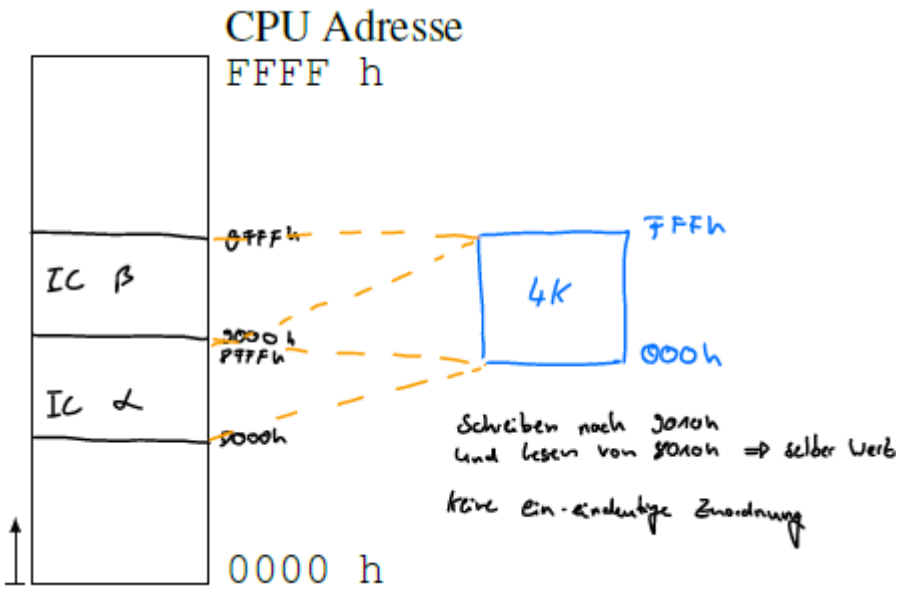
CPU: 16 Adressleitungen → 64 KB adressierbar

SP. IC: 12 Adressleitungen → 4 KB adressierbar (direkter Adressteil)

Rest (16-12= 4) → CS Adressteil



Externe Dekodierung

Art	Vollständige Dekodierung	Unvollständige Dekodierung
CS-Teil Auswertung	Vollständig	Unvollständig
Realisierung	<p><i>1 aus N Decoder</i></p> 	<p><i>CS-Decoder</i></p> 
Speicherzuordnung	<p>CPI Adressen \leftrightarrow Speicheradressen Startadresse: Adresse des 1. Byte im IC, direkter Adressteil = 0 Endadresse: Adresse des letzten Byte im IC, direkter Adressteil = maximal Test: Speichergröße = End + Startadresse + 1</p>	<p>Startadresse: direkter Adressteil = 0 Endadresse: direkter Adressteil = Maximum</p>
Speicherbelegung	<p>46kB CPU Adresse FFFF h</p> 	<p>CPU Adresse FFFF h</p>  <p>Schreiben nach 3000h und lesen von 3000h \Rightarrow selber Wert keine Ein-eindeutige Zuordnung</p>

Hinweis: Keine Eindeutige Zuordnung zwischen logischen und physikalischen Adressen

Beispiel

Gegeben:

- CPU: 64K Byte adressierbar

daraus folgt CPU 2^{16} -> 16 Leitungen

- Speicher

- IC1, IC2: 2K Byte
- Startadresse IC1 = 1000h
- Startadresse IC2 = 9000h
- IC3: 4K Byte

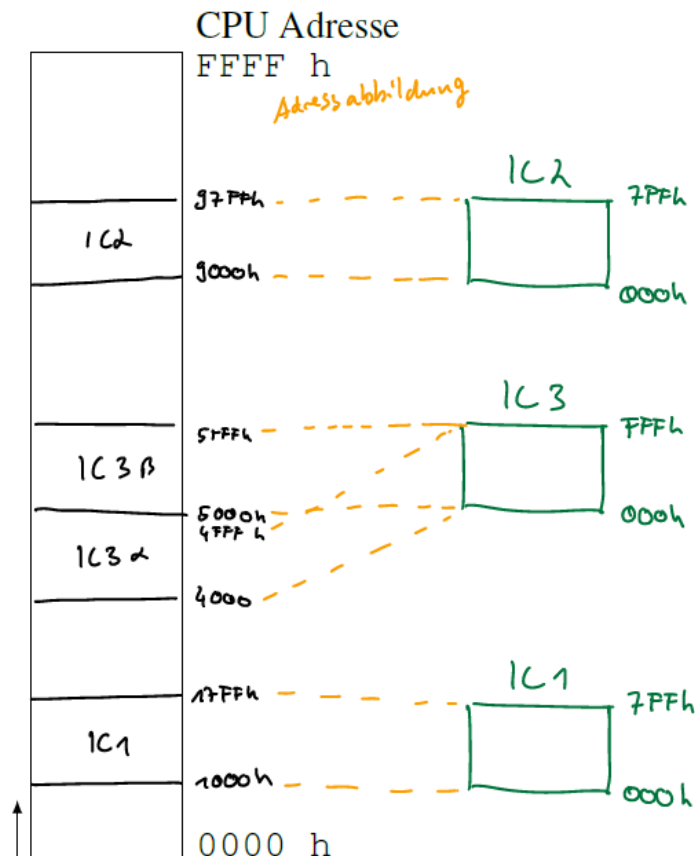
daraus folgt IC1, IC2: 2^{11} -> 11 Leitungen

daraus folgt IC3: 2^{12} -> 12 Leitungen

- mit $CS3 = \neg A15 \wedge A14 \wedge \neg A13$

Gesucht:

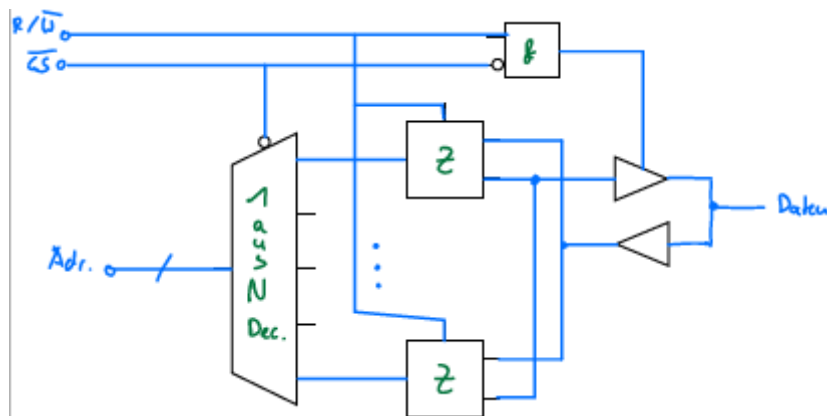
- Anzahl der Leitungen
 - CPU 16 Leitungen
 - IC1, IC2 11 Leitungen
 - IC3 12 Leitungen
- Letzte Speicheradresse
 - IC1, IC2: 2k (11Bit) 0111 1111 1111 = 7FF H
 - IC3: 4k (12 Bit) 1111 1111 1111 = FFF H
- Decoder für IC1, IC2 (CS-T, direkt)
 - IC1: 1000 H 0001 0000 0000 0000 CS1: $\neg A15 \wedge \neg A14 \wedge \neg A13 \wedge A12 \wedge \neg A11$
 - IC2: 9000 H 1001 0000 0000 0000 CS2: $A15 \wedge \neg A14 \wedge \neg A13 \wedge A12 \wedge \neg A11$
- Start- und Stoppadressen für IC3
 - CS3: $\neg A15 \wedge A14 \wedge \neg A13$
 - Start: 010X, 0000 000 0000, α : x= 0 = 4000H, β : x=1 = 5000H
 - Ende: 010X 1111 1111 1111 -> α : 4FFF H, β : 5FFF H



Interne Adressierung

- Auswahl der Speicherzellen im Speicherbaustein
- Speicherzelle
 - 1 Bit (oder 1 Byte, 1 Wort)
 - Anschlüsse: Selekt, R/W, Daten In, Daten Out
- Speicherbaustein
 - N Bit/Byte
 - Anschlüsse:
 - Steuerleitungen: Chip Selekt, R/W
 - Adressleitungen: $\lg(N)$
 - Datenleitungen: Je Wortbreite

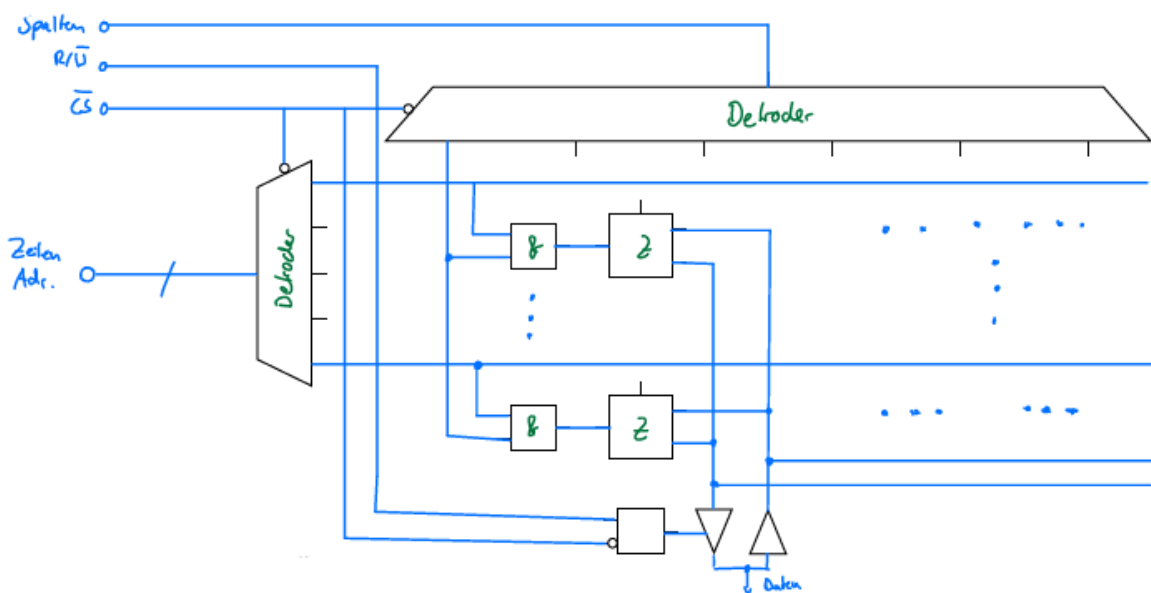
Ein Decoder



Eigenschaften:

Große N -> komplexer/großer Decoder

Zwei Decoder



Eigenschaften:

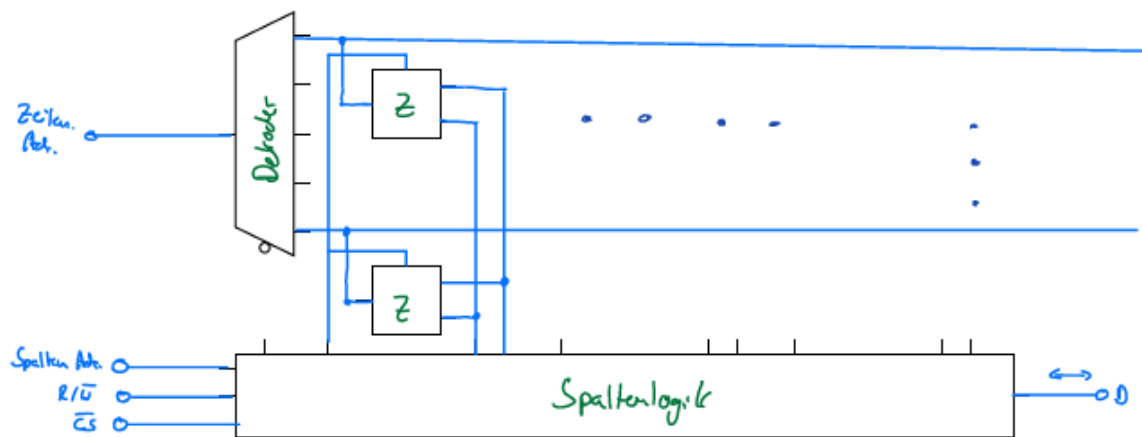
Große N -> Viele Ein-/Ausgänge

Parallel geschaltet

Aufteilung:

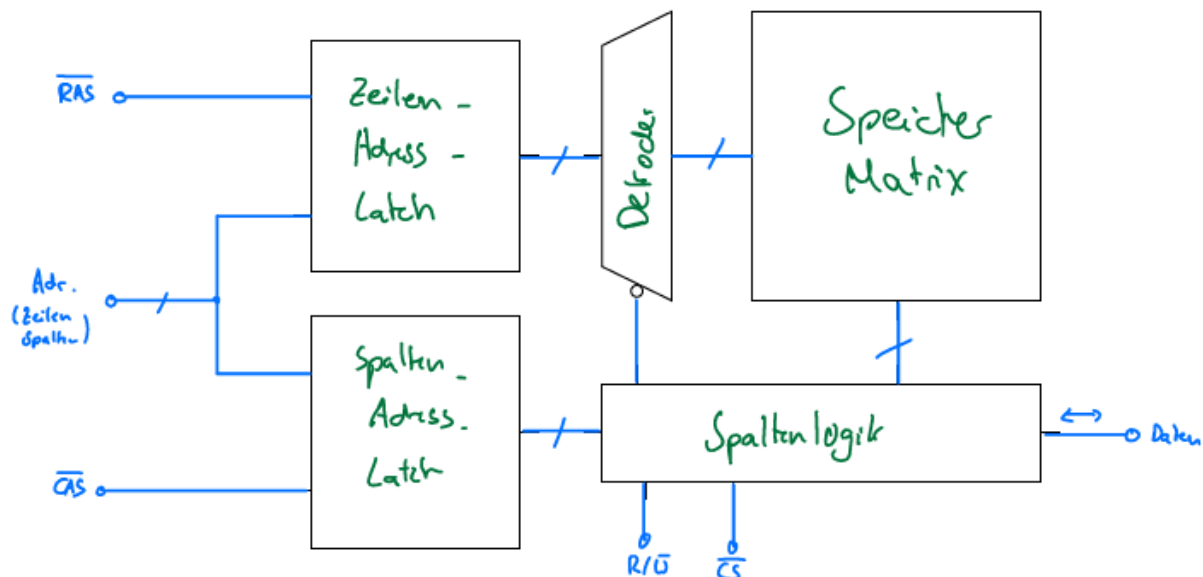
N Zellen auf k Zeilen und l Spalten

Ein Decoder und Spaltenlogik



Aufteilung	Ablauf	Eigenschaften
N Zellen auf k Zeilen und l Spalten $k * l = N$	<p>Speichert Wörter zwischen</p> <p>Lesen:</p> <ol style="list-style-type: none"> 1. Lesen komplette Zeile in Spaltenlogik 2. Spaltenlogik wählt Spalte <p>Schreiben:</p> <ol style="list-style-type: none"> 1. Lesen komplette Zeile in Spaltenlogik 2. Spaltenlogik verändert Spaltenwert 3. Rückschreiben der kompletten Zeile 	<p>Bei wiederholten lesen aus gleicher Zeile kein neues lesen aus der Zeile</p> <p>Für Dynamisches RAM: automatisches auffrischen</p> <p>Für große N -> viele Adressleitungen</p>

Multiplexer von Zeilen- und Spaltenadressen



Aufteilung: N Zellen auf k Zeilen und l Spalten, sodass $k * l = N$

Erweiterung des Minimalsystems

Interrupt

Allgemein

- Aufgabe: Reaktion auf äußere Ereignisse

mit Polling	mit Interrupt
Software zyklischer Aufruf Endlosschleife mit Abruf wird von Programm gesteuert synchron	Hardware zusätzlich, außerhalb des Minimalsystems Reaktion vom Programmablauf unabhängig asynchron

- Bezeichnungen

Interrupt	HW generiert Unterprogrammaufruf bzw. asynchron zum Hauptprogramm
Trap	Software generierter Interrupt, bzw. Befehl löst Interrupt aus synchron zum Programmablauf
Exception	Übergriff

- Eigenschaften
 - benötigt Leistung und HW (Interrupt-Controller)
 - Reaktionszeit -> kalkulierbar
- Quellen

Welche Quelle	Ablauf
eigenständige, externe Geräte -> Leitungen	Tastatur -> Leitung -> CPU -> Flag
Interne Komponente -> Flag	UART -> Flag
„unzulässiger“ Befehl	FDIV ohne FPU -> Flag
„fehlerhafter“ Befehl	DIV durch 0 -> Flag
geplant/gewollt	set Bit -> Flag

- Ablauf allgemein

	Ablauf	Reaktion
1	Anfrage von Quelle	Flag setzten
2	Feststellen, ob Interrupt vorliegt	Flag gesetzt?
3	Feststellen woher/welche Quelle:	Woher, welche Flag? Flag lesen
4	entscheiden, ob stattgegeben wird (Prioritätsprüfung)	aktuell laufendes Programm α, β wenn $\beta > \alpha$, dann Interrupt stattgefunden sonst läuft Hauptprogramm weiter Flag bleibt aber gesetzt
5	ermitteln der Unterprogramm (UP)-Adresse	Zieladresse bestimmen (Startadresse der Interruptroutine)
6	Aufruf der Interruptroutine	Rücksprungadresse, Programm Counter umsetzen
7	UP ausführen	UP/Interrupt –Routine ausführen
8	UP beenden	(meist Sonderbefehl RETI)
9	Prioritäten zurücksetzten	um α
10	Rücksprung	PC auf Rücksprungadresse

Hinweis: Flag wird durch HW oder SW gelöscht

- Typen, zu Ermittlung der Adresse der Interrupt-Routine (Zieladresse)

	Auto Interrupt	Interrupt-Nummer	Vektor-Interrupt
Beschreibung	feste Verknüpfung mit Quelle	Verknüpfung über Rechnung/Nummer	Verknüpfung über Tabelle
Quellen	10 interne, 7 externe	Ein Interrupt-Eingang	Eingang: NMI, NTR oder Software
Prioritätsstruktur	Globale Freigabe Prioritätsstufen (4) Rangfolge in Stufen (durch Polling) individuelle Freigabe	Globale Freigabe Flag	NMI: immer Software: immer INTP: Freigabe durch Flag
Zieladresse	fest zugeordnet, mit Quellen verbunden	Einer Interrupt-Nummer zugeordnet (von Architektur abhängig)	Adresse aus der Tabelle
Sonstiges/ Hinweis		Interrupt- Bestätigungssignal (INTA)	Tabelle im RAM, Tabellenaufbau durch OS, Umbiegen eines Interrupts möglich

➔ Ablauf ist im Großen und Ganzen gleich (auf Sonstiges/Hinweise achten)