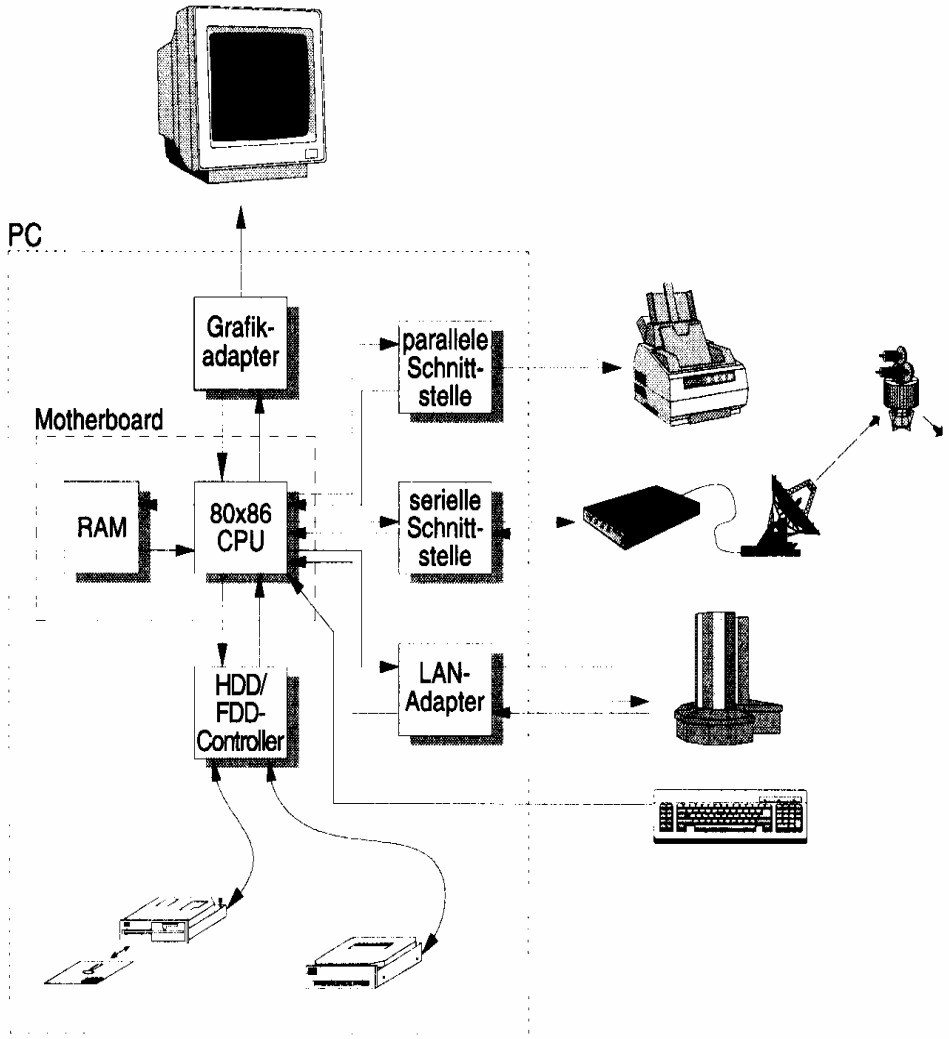
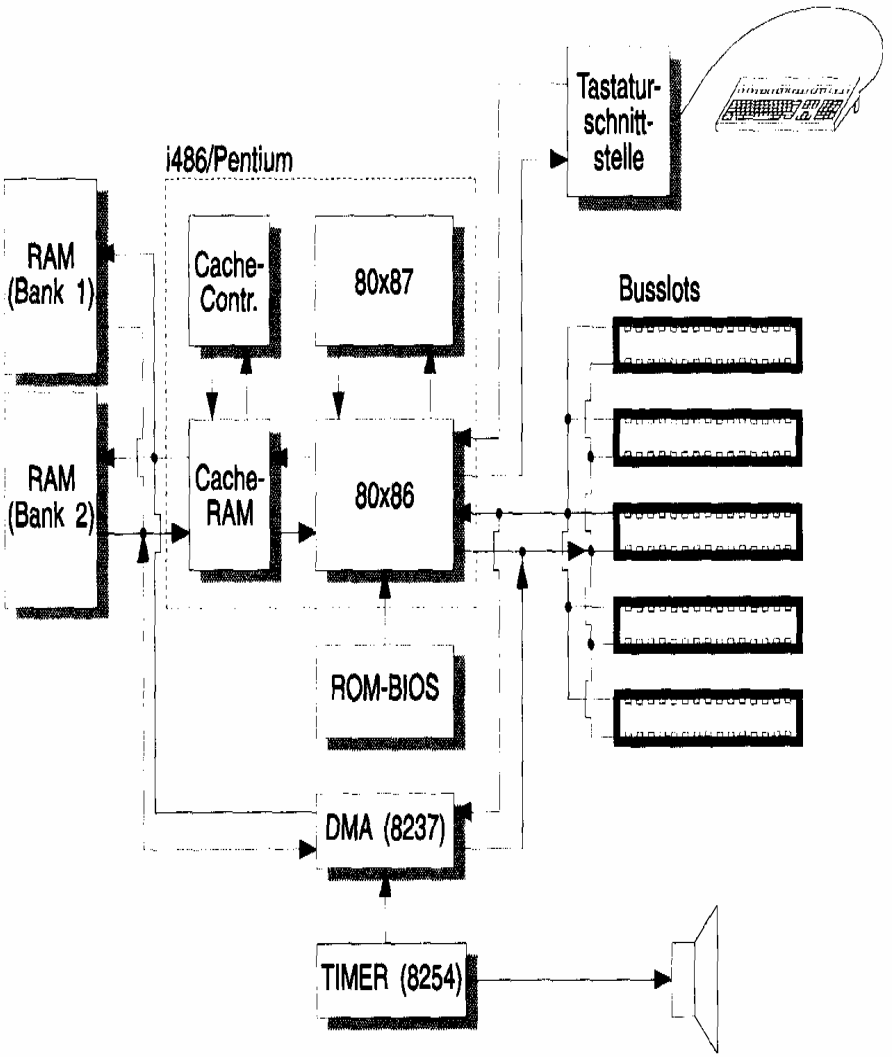
6 Aufbau von Rechnersystemen

- Speicher
  - ⇨ RAM, ROM, Cache
- Prozessor
  - ⇨ Integer
  - ⇨ Gleitkommaarithmetik
  - ⇨ Cachecontroller
- E/A
  - ⇨ Tastatur
  - ⇨ Grafikkarte
  - ⇨ Diskettencontroller
  - ⇨ Festplattencontroller
  - ⇨ Netzwerkkarte



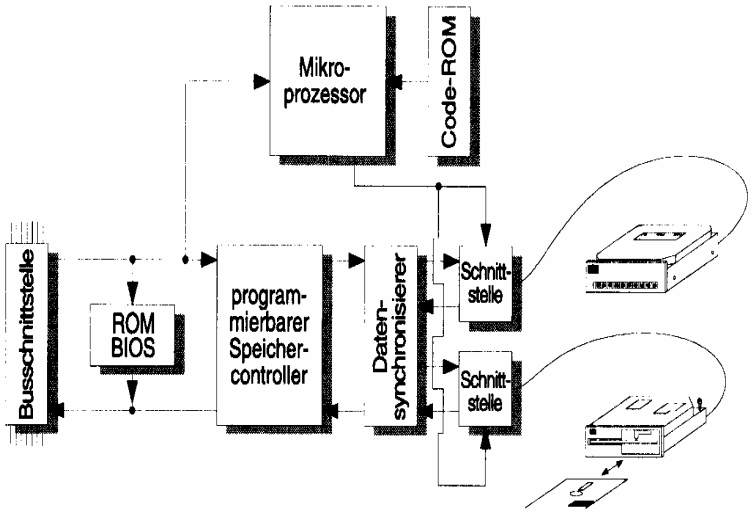
Hauptkomponenten der Zentraleinheit

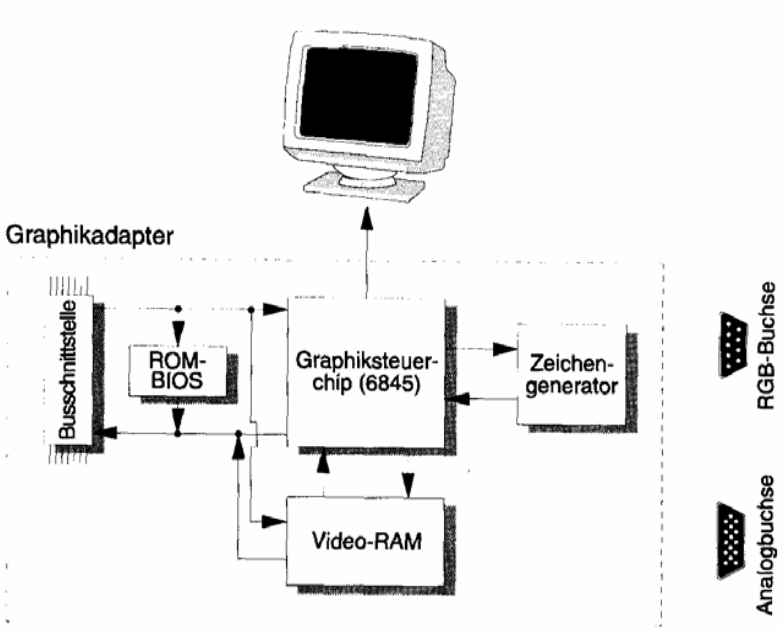
- Speicher
  - ⇨ RAM
  - ⇨ ROM
- Prozessor
  - ⇨ Integer-CPU
  - ⇨ Gleitkomma-Prozessor
  - ⇨ Cache
  - ⇨ Cachecontroller
- Bus
- Peripherie
  - ⇨ Schnittstellen
  - ⇨ Timer
  - ⇨ DMA



Peripherie

Festplatten- und Diskettencontroller

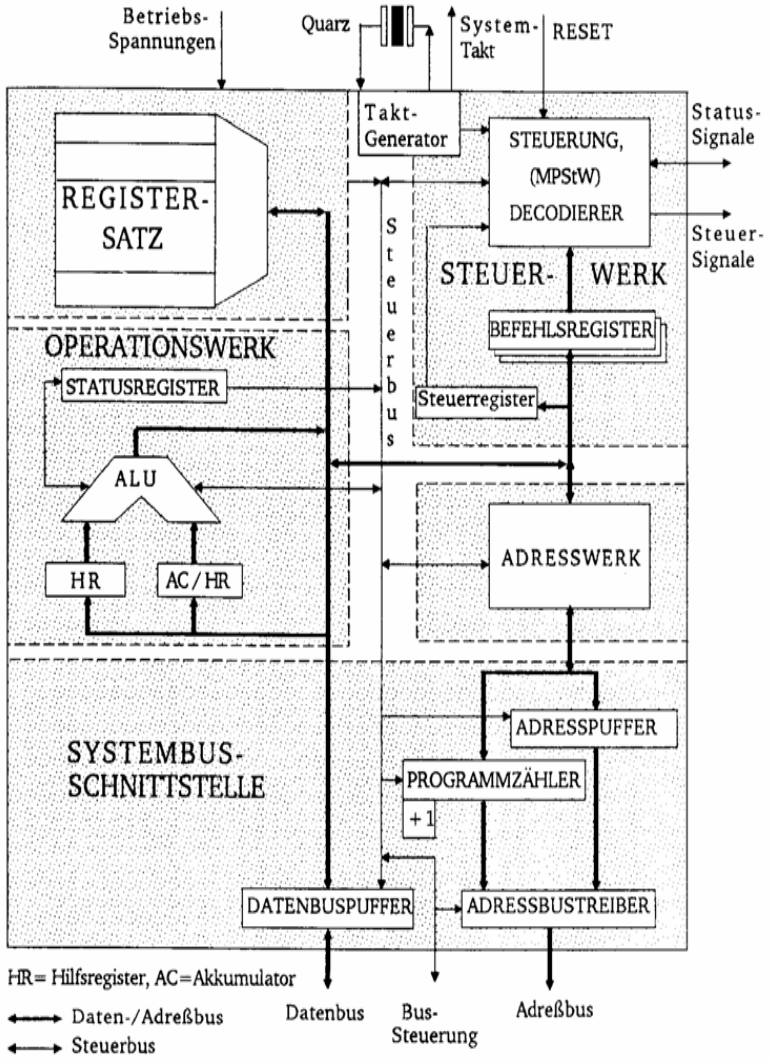




Grafikadapter

Prinzipieller Aufbau eines typischen Mikroprozessors

- **Steuerwerk**
  - ⇨ **Liefert die Steuersignale für das Rechenwerk**
  - ⇨ **Steuert den Ablauf der Operationen**
- **Rechenwerk (Operationswerk)**
  - ⇨ **führt die arithmetischen und logischen Operationen aus**
- **Registersatz**
  - ⇨ **speichert die Operanden für das Rechenwerk**
- **Adresswerk**
  - ⇨ **Berechnet die Adressen für die Befehle oder die Operanden**
- **Systembus-Schnittstelle**
  - ⇨ **Treiber**
  - ⇨ **Zwischenspeicher**
  - ⇨ **Adresszähler**

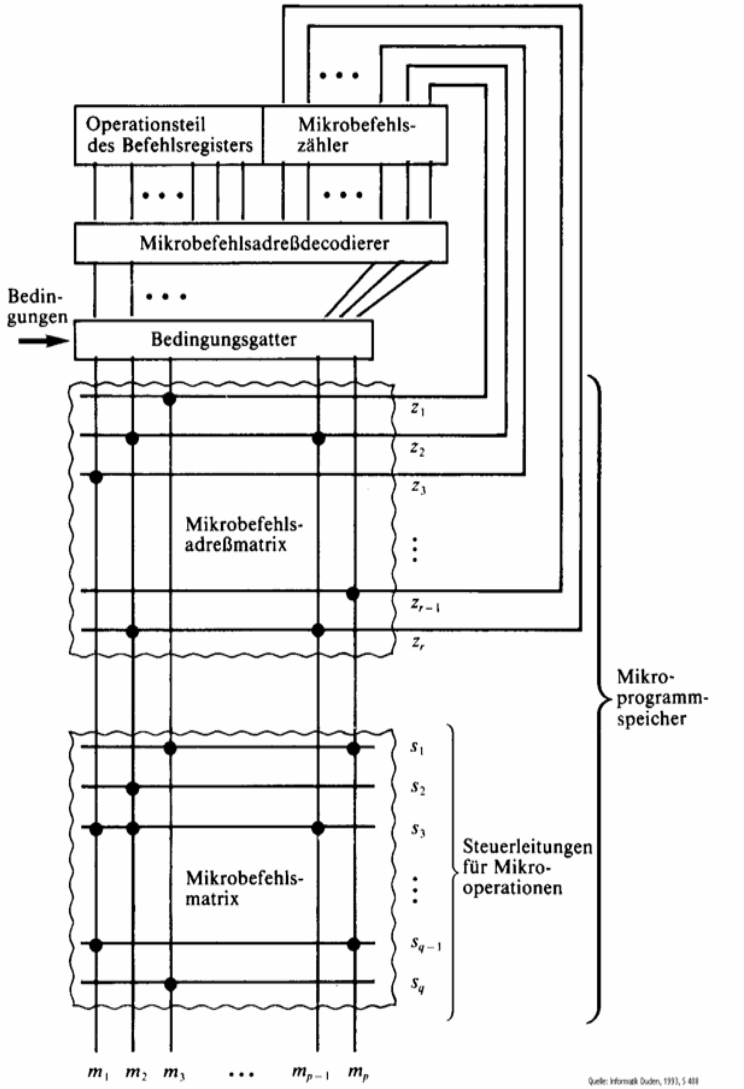


Das Steuerwerk

- Ablaufsteuerung der Befehlsbearbeitung im Operationswerk
- Synchrones Schaltwerk
- Komponenten eines typischen Steuerwerks
  - ⇨ Befehlsdekodierer: analysiert und entschlüsselt den aktuellen Befehl
  - ⇨ Steuerung: generiert die Signale für das Rechenwerk
  - ⇨ Befehlsregister: speichert den aktuellen Befehl
  - ⇨ Steuerregister: liefert Bedingungen zur Entscheidung des Befehlsablaufs
- Festverdrahtetes Steuerwerk
  - ⇨ das Steuerwerk wird als System mehrstufiger logischer Gleichungen implementiert und minimiert
- Mikroprogrammiertes Steuerwerk
  - ⇨ das Steuerwerk wird in einem ROM implementiert
- Mikroprogrammierbares Steuerwerk
  - ⇨ das Steuerwerk wird in einem RAM implementiert und wird beim Neustart des Prozessors geladen

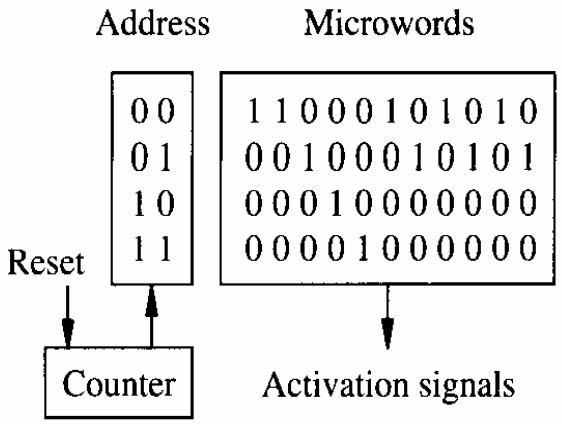
Mikroprogrammierung

- **Mikrooperationen**
  - ⇨ **elementare Operationen wie das Setzen eines Registers**
- **Mikrobefehle**
  - ⇨ **Zusammenfassung bestimmter Mikrooperationen, die zu einem Taktzeitpunkt gleichzeitig ausgeführt werden können**
- **Mikroprogrammierung**
  - ⇨ **Realisierung der Maschinenbefehle durch durch eine Folge von Elementaroperationen**



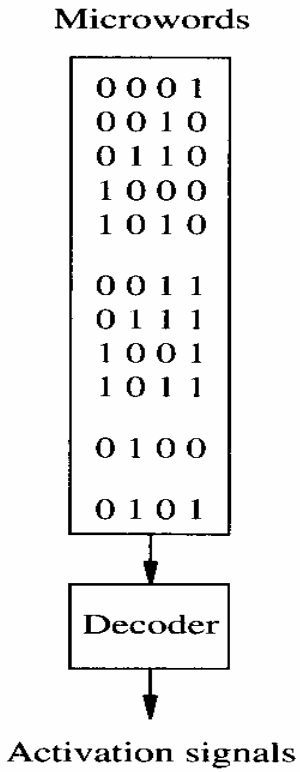
Vertikale und horizontale Mikroprogrammierung

- **Horizontale Mikroprogrammierung**
  - ⇨ **jedes Ausgangssignal erhält eine eigene Steuerleitung**



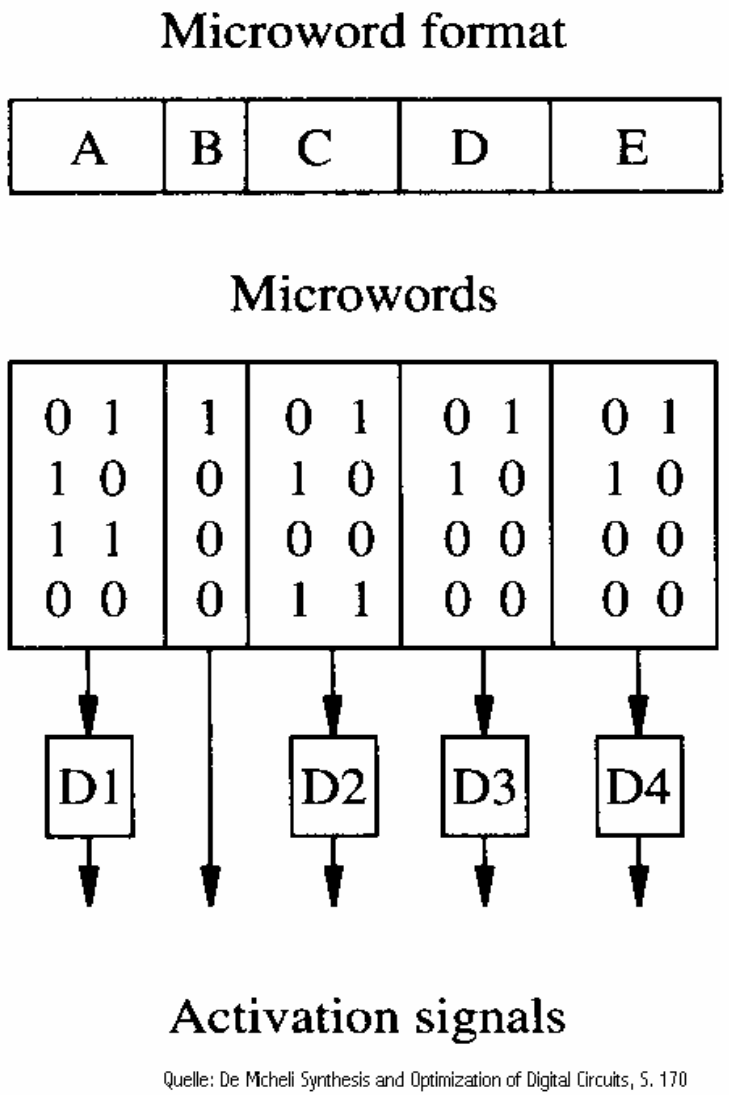
Quelle: De Michell Synthesis and Optimization of Digital Circuits, S. 169

- **Vertikale Mikroprogrammierung**
  - ⇨ **Die Ausgangssignale werden über einen Multiplexer angesteuert**



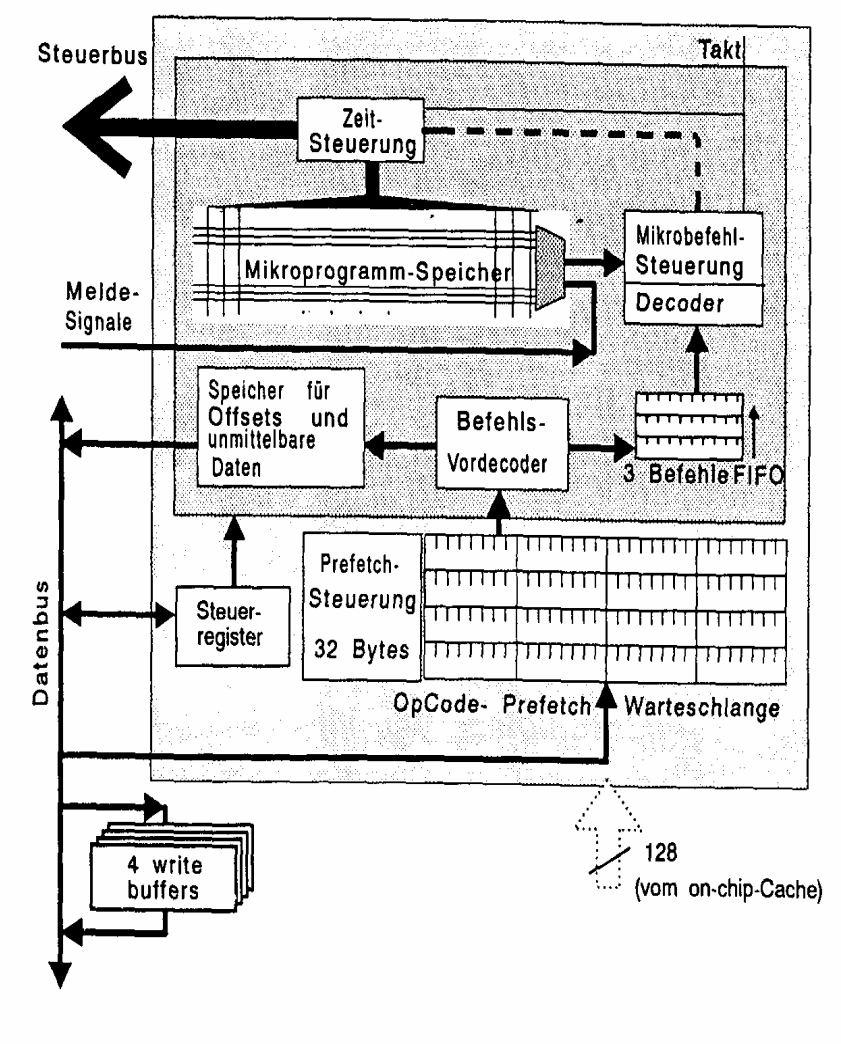
Mischformen

- **Unabhängige Teile des horizontalen Mikrobefehlsworts werden zusammengefaßt und vertikal kodiert**



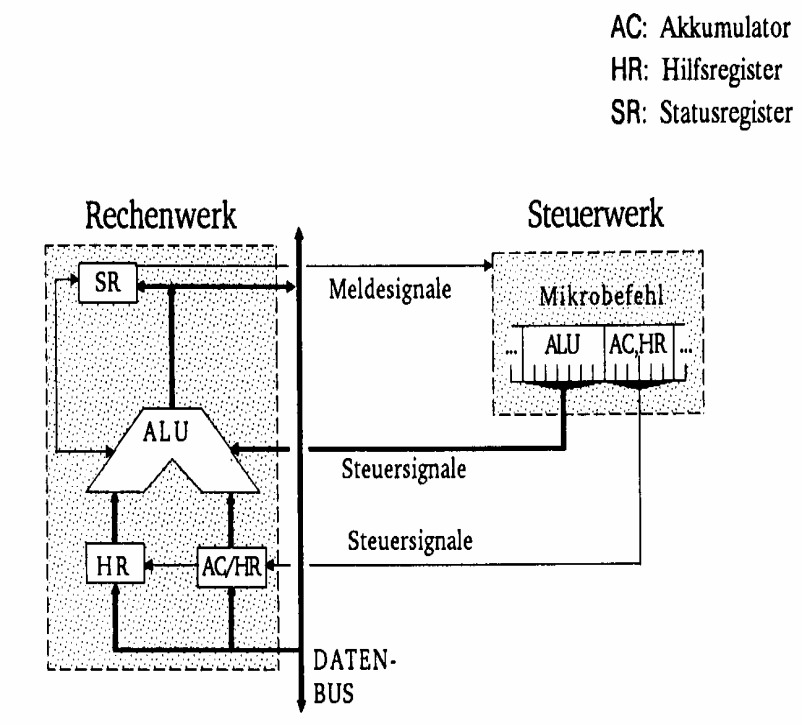
Quelle: De Micheli Synthesis and Optimization of Digital Circuits, S. 170

Das Steuerwerk des Intel 486



Das Rechenwerk

- **ALU**
  - ⇨ **berechnet alle Operationen**
- **Akkumulator**
  - ⇨ **speichert das Ergebnis einer Operation**
  - ⇨ **stellt einen Operanden zur Verfügung**
- **Hilfsregister**
  - ⇨ **stellt den zweiten Operanden zur Verfügung**
- **Statusregister**
  - ⇨ **Speichert besondere Ergebnisse**



Das Statusregister

- **Einzelne logisch unabhängige Bits**
  - **CF (Carry Flag)** — **Übertrag**
  - **ZF (Zero Flag)** — **Ergebnis der letzten Operation ist 0**
  - **SF (Sign Flag)** — **negatives Ergebnis bei der letzten Operation**
  - **OF (Overflow Flag)** — **Überlauf bei der letzten Operation**
  - **EF (Even Flag)** — **Gerades Ergebnis bei der letzten Operation**
  - **PF (Parity Flag)** — **ungerade Anzahl der ´1´-Bits**
- **Einsatz: Diese Flags werden bei bedingten Sprüngen ausgewertet**

Transfer- und Ein-/Ausgabebefehle

Mnemonic	Bedeutung
LD	Laden eines Register (*load*)
LEA	Laden eines Registers mit der Adresse eines Operanden (*load effective address*)
ST	Speichern des Inhalts eines Registers (*store*)
MOVE	Übertragen eines Datums (in beliebiger Richtung)
EXC	Vertauschen der Inhalte zweier Register bzw. eines Registers und eines Speicherwortes (*exchange*)
TFR	Übertragen eines Registerinhalts in ein anderes Register (*transfer*)
PUSH	Ablegen des Inhalts eines oder mehrerer Register im Stack
PULL (POP)	Laden eines Registers bzw. mehrerer Register aus dem Stack
STcc	Speichern eines Registerinhaltes, falls die Bedingung cc (nach Tabelle 1.14-11) erfüllt ist

Mnemonic	Bedeutung
IN, READ	Laden eines Registers aus einem Peripheriebaustein
OUT, WRITE	Übertragen eines Registerinhalts in einen Peripheriebaustein

Arithmetische und Logische Befehle

Mnemonic	Bedeutung
ABS	Absolutbetrag bilden (*absolute*)
ADD	Addition ohne Berücksichtigung des Übertrags (*add*)
ADC	Addition mit Berücksichtigung des Übertrags (*add with carry*)
CLR	Löschen eines Registers oder Speicherwortes (*clear*)
CMP	Vergleich zweier Operanden (*compare*)
COM	bitweises Invertieren eines Operanden (Einerkomplement) (*complement*)
DAA	Umwandlung eines dualen Operanden in eine Dezimalzahl (*decimal adjust A*)
DEC	Register oder Speicherwort dekrementieren (*decrement*)
DIV	Division (*divide*)
INC	Register oder Speicherwort inkrementieren (*increment*)
MUL	Multiplikation (*multiply*)
NEG	Vorzeichenwechsel im Zweierkomplement (*negate*)
SUB	Subtraktion ohne Berücksichtigung des Übertrags (*subtract*)
SBC	Subtraktion mit Berücksichtigung des Übertrags (*subtract with carry*)

Mnemonic	Bedeutung
AND	UND-Verknüpfung zweier Operanden
OR	ODER-Verknüpfung zweier Operanden
EOR	Antivalenz-Verknüpfung zweier Operanden (*exclusive or*)
NOT	Invertierung eines (Booleschen) Operanden

Flag- und Bit-Manipulationsbefehle

Mnemonic	Bedeutung	
SE<f>	Setzen eines Bedingungs-Flags	(*set*)
CL<f>	Löschen eines Bedingungs-Flags	(*clear*)
BSET	Setzen eines Bits	(*bit set*)
BCLR	Rücksetzen eines Bits	(*bit clear*)
BCHG	Invertieren eines Bits	(*bit change*)
TST	Prüfen eines bestimmten Flags oder Bits	(*test*)
BF...	Bitfeld-Befehle,	
	insbesondere:	
BFCLR	Zurücksetzen der Bits auf '0'	(*clear*)
BFSET	Setzen der Bits auf '1'	(*set*)
BFFFO	Finden der ersten '1' in einem Bitfeld	(*find first one*)
BFEXT	Lesen eines Bitfeldes	(*extract*)
BFINS	Einfügen eines Bitfeldes	(*insert*)

(<f> Abkürzung für ein Flag, z.B. C *carry flag*)

Schiebe- und Rotationsbefehle

Mnemonic	Bedeutung	
SHF	Verschieben eines Registerinhaltes	(*shift*)
	insbesondere:	
ASL	arithmetische Links-Verschiebung	(*arithm. shift left*)
ASR	arithmetische Rechts-Verschiebung	(*arithm. shift right*)
LSL	logische Links-Verschiebung	(*logical shift left*)
LSR	logische Rechts-Verschiebung	(*logical shift right*)
ROT	Rotation eines Registerinhaltes	(*rotate*)
	insbesondere:	
ROL	Rotation nach links	(*rotate left*)
RCL	Rotation nach links durchs Übertragsbit	(*rotate with carry left*)
ROR	Rotation nach rechts	(*rotate right*)
RCR	Rotation nach rechts durchs Übertragsbit	(*rotate with carry right*)
SWAP	Vertauschen der beiden Hälften eines Registers	

Befehle zur Programmsteuerung

Sprung und Verzweigungsbefehle

Mnemonic	Bedeutung	
JMP	unbedingter Sprung zu einer Adresse	(*jump*)
Bcc	Verzweigen, falls die Bedingung cc erfüllt ist	(*branch*)
BRA	Verzweigen ohne Abfrage einer Bedingung	(*branch always*)

Unterprogrammaufrufe und Rücksprünge, Software-Interrupts

Mnemonic	Bedeutung
JSR, CALL	unbedingter Sprung in ein Unterprogramm (*jump to subroutine*)
BSRcc	Verzweigung in ein Unterprogramm, falls die Bedingung cc gilt (*branch to subroutine*)
RTS	Rücksprung aus einem Unterprogramm (*return from subroutine*)
SWI,TRAP,INT	Unterbrechungsanforderung durch Software (*software interrupt*)
RTI, RTE	Rücksprung aus einer Unterbrechungsroutine (*return from interrupt/exception*)

Bedingungen für Sprünge

cc	Bedingung	Bezeichnung
CS	CF = 1	*branch on carry set*
CC	CF = 0	*branch on carry clear ,*
VS	OF = 1	*branch on overflow*
VC	OF = 0	*branch on not overflow*
EQ	ZF = 1	*branch on zero/equal*
NE	ZF = 0	*branch on not zero/equal*
MI	SF = 1	*branch on minus*
PL	SF = 0	*branch on plus*
PA	PF = 1	*branch on parity/parity even*
NP	PF = 0	*branch on not parity/parity odd*
nicht vorzeichenbehaftete Operanden		
LO	CF = 1 (vgl. CS)	*branch on lower than*
LS	CF v ZF = 1	*branch on lower or same*
HI	CF v ZF = 0	*branch on higher than*
HS	CF = 0 (vgl. CC)	*branch on higher or same*
vorzeichenbehaftete Operanden		
LT	SF≠OF = 1	*branch on less than*
LE	ZF v (SF≠OF) = 1	*branch on less or equal*
GT	ZF v (SF≠OF) = 0	*branch on greater than*
GE	SF≠OF = 0	*branch on greater or equal*

(Bezeichnungen: ≠ Antivalenz, v logisches ODER)

Sonstige Befehle

Systembefehle

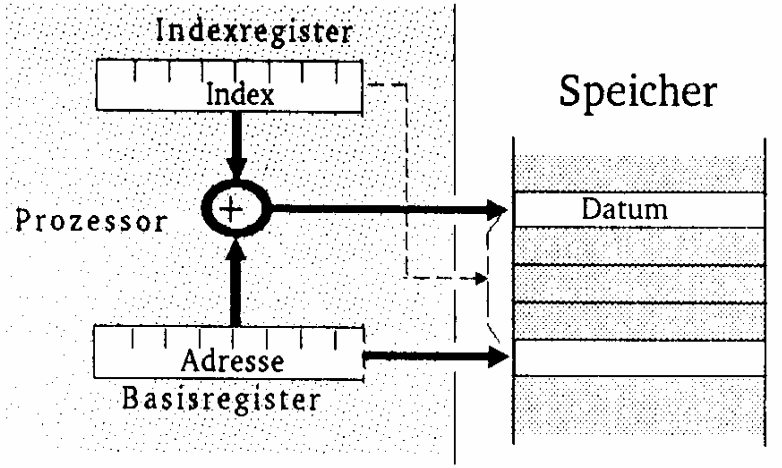
Mnemonic	Bedeutung
NOP	keine Operation, nächsten Befehl ansprechen (*no operation*)
WAIT	Warten, bis ein Signal an einem speziellen Eingang auftritt
SYNC	Warten auf einen Interrupt
HALT, STOP	Anhalten des Prozessors, Beenden jeder Programmausführung
RESET	Ausgabe eines Rücksetz-Signals für die Peripherie-Bausteine
SVC	(geschützter) Aufruf des Betriebssystem-Kerns (*supervisor call*)

Stringbefehle

Mnemonic	Bedeutung
MOVS	Transferieren eines Blocks (*move string*)
INS	Einlesen eines Blocks von der Peripherie (*input string*)
OUTS	Ausgabe eines Blocks an die Peripherie (*output string*)
CMPS	Vergleich zweier Blöcke (*compare string*)
COPS	Kopieren eines Blocks (*copy string*)
SCAS	Suchen eines Zeichens (Wortes) in einem Block (*scan string*)

Der Registersatz

- **Datenregister**
  - ⇨ **Integerregister**
  - ⇨ **Akkumulator**
- **Adressregister**
  - ⇨ **Basisregister**
  - ⇨ **Indexregister**
- **Spezialregister**
  - ⇨ **Statusregister**
  - ⇨ **Programmzähler**
  - ⇨ **Stackpointer**
  - ⇨ **Segmentregister**

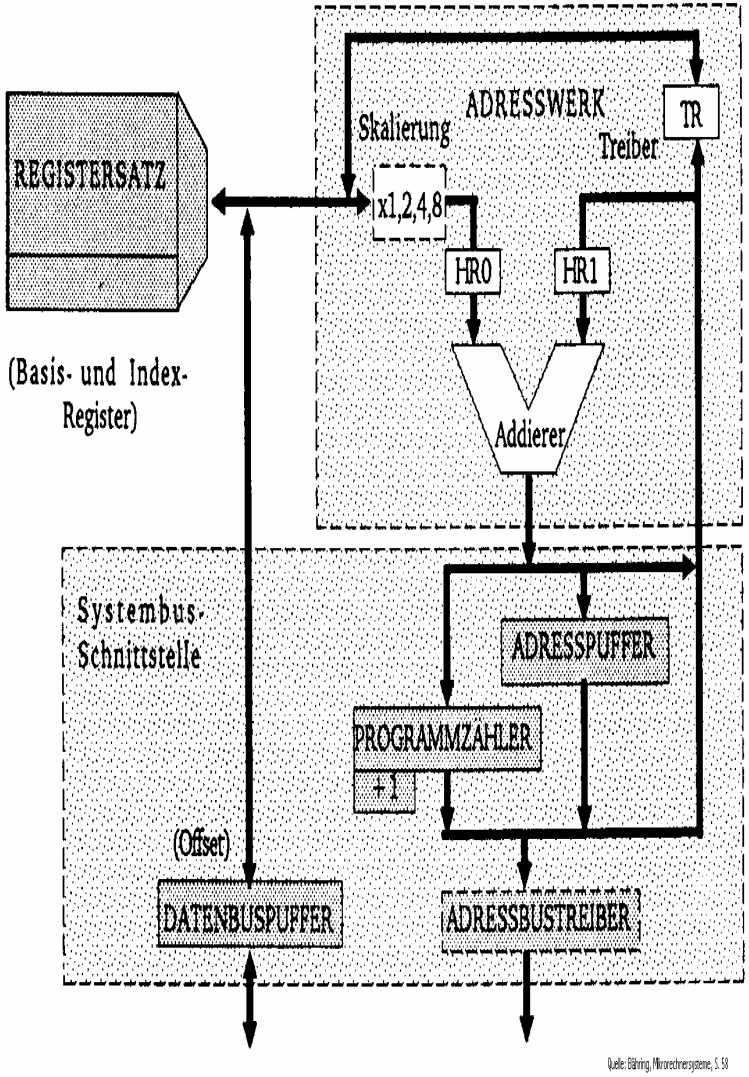


Die Register im Intel 80x86

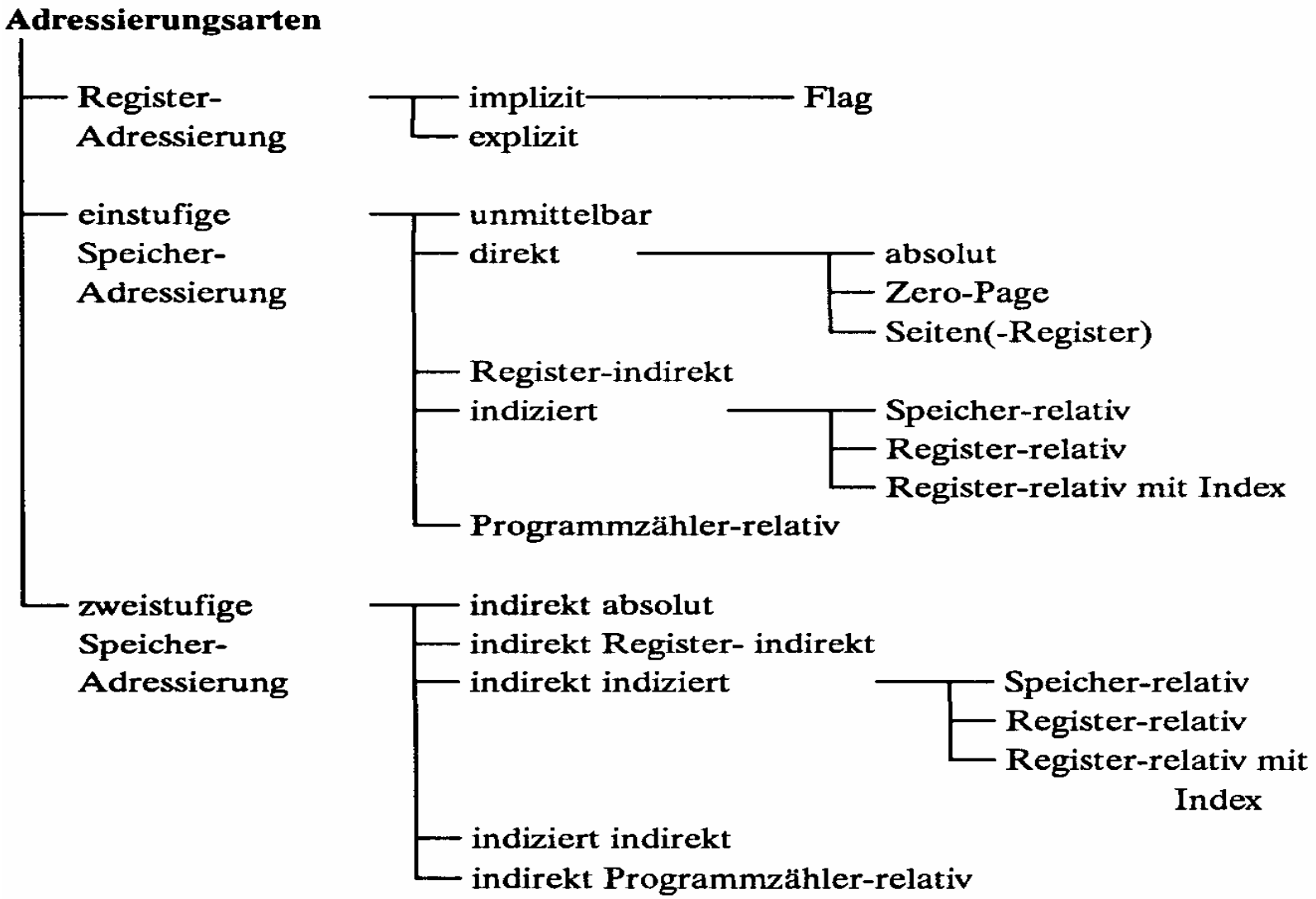
- AX (AH und AL)
  - accumlator
  - Akkumulator
- BX (BH und BL)
  - base register
  - Basisregister zur Adressierung der Anfangsadresse einer Datenstruktur
- CX (CH und CL)
  - count register
  - Schleifenzähler, wird bei Schleifen und Verschiebeoperationen benötigt
- DX
  - data register
  - Datenregister Register für den zweiten Operand
- SI und DI
  - source register und destination register
  - Indexregister für die Adressierung von Speicherbereichen
- SP
  - stack pointer
  - Verwaltung eines Stapelbereichs

Das Adresswerk

- Nach den Vorgaben des Steuerwerks werden Speicheradressen gebildet
  - ⇨ aus Registerinhalten
  - ⇨ aus Speicherzellen
- Adressaddierer
- TR-Register speichert den Inhalt des aktuellen Adresszählers bei Sprüngen
- Adressprüfung bei Byte-, Halbwort-, Doppelwort- und Quadwort-Zugriffen

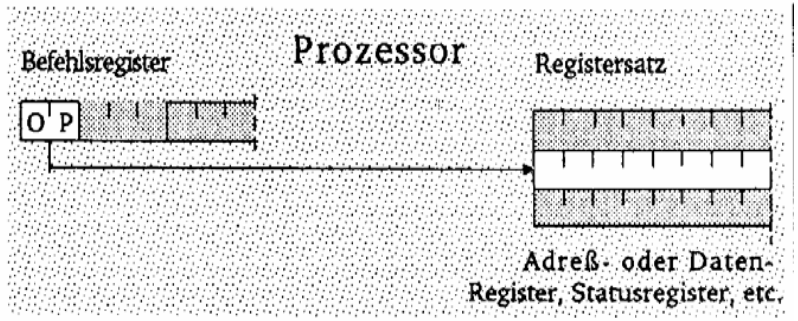


Adressierungsarten

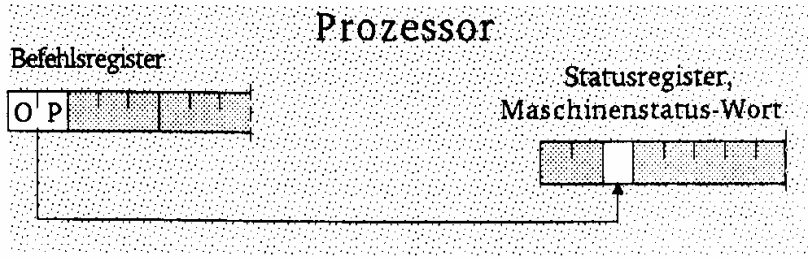


Register- Adressierung

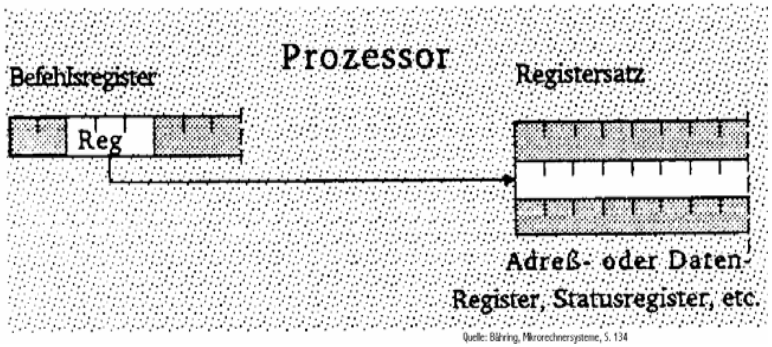
- **Implizite Adressierung**
  - ⇨ **Adresse des Operands ist im OP-Code enthalten**
  - ⇨ **Beispiel: LSRA**
    - **logical shift right accumulator**
- **Flag-Adressierung**
  - ⇨ **ein einzelnes Bit wird angesprochen**
  - ⇨ **Beispiel: SEC**
    - **set carry flag**
- **Explizite Adressierung**
  - ⇨ **Adresse des Operandenregisters wird im OP-Code angegeben**
  - ⇨ **Beispiel: DEC r0**
    - **decrement R0**



Quelle: Bähring, Mikrorechnersysteme, S. 130



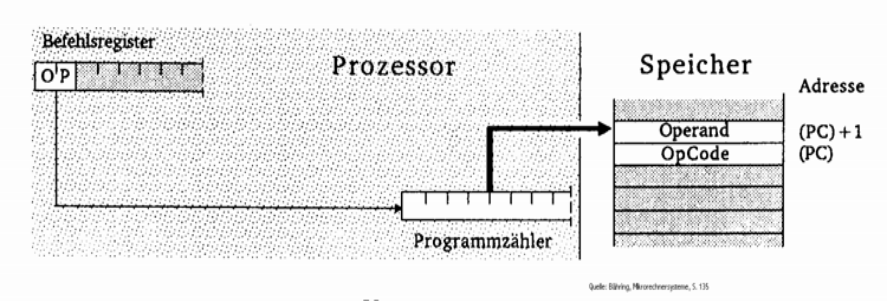
Quelle: Bähring, Mikrorechnersysteme, S. 133



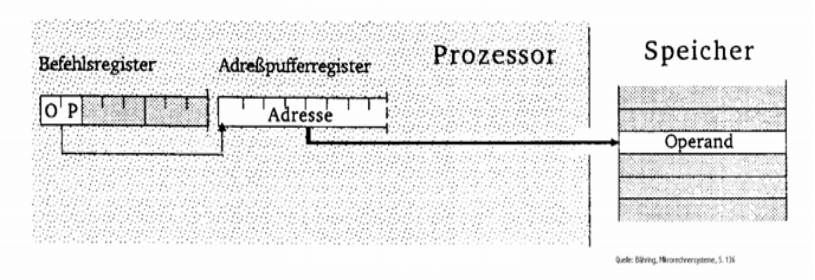
Quelle: Bähring, Mikrorechnersysteme, S. 134

Einstufige Adressierung

- **Unmittelbare Adressierung**
  - ⇨ **Der Befehl enthält den Operanden**
  - ⇨ **Beispiel: LDA #$A3**
    - **load accu 3_{16}**

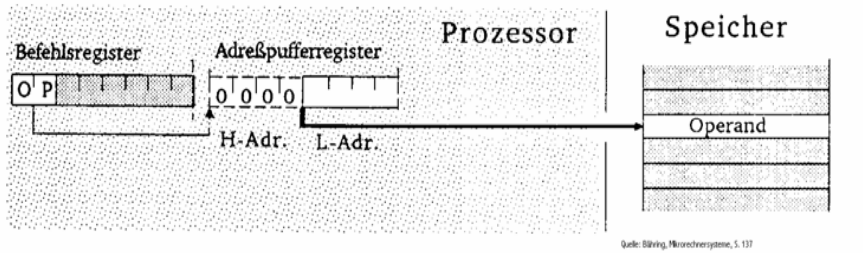


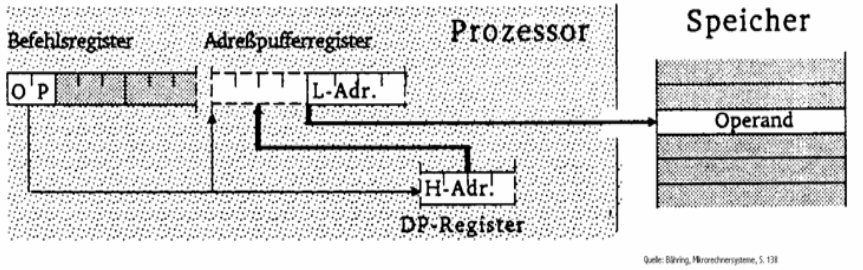
- **Absolute Adressierung**
  - ⇨ **Das Speicherwort das dem Befehl folgt enthält die Adresse**
  - ⇨ **Beispiel: JMP $07FE**



Seitenadressierung

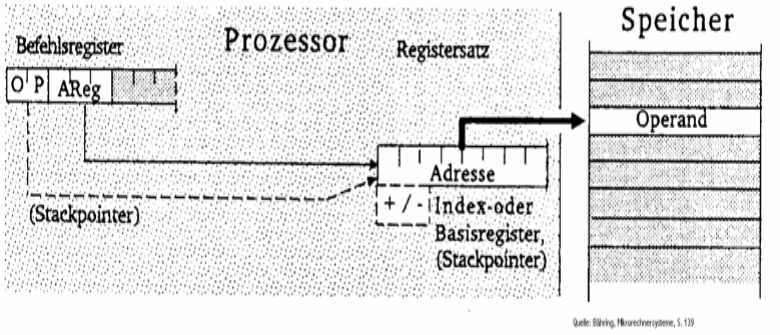
- Bei Prozessoren mit unterschiedlicher Daten- und Adressbusbreite
  - ⇨ man spart Speicherplatz und Zeit des Lesens der höherwertigen Bits
- Zero-Page Adressierung
  - ⇨ schneller Zugriff auf die Speicherseite 0
  - ⇨ Beispiel: INC $007F
    - erhöhe Speicherzelle $7 um 1
- Seiten-Register-Adressierung
  - ⇨ Höherwertige Adressteil wird von einem Register zur Verfügung gestellt
  - ⇨ Beispiel: LDA R0, <$7F





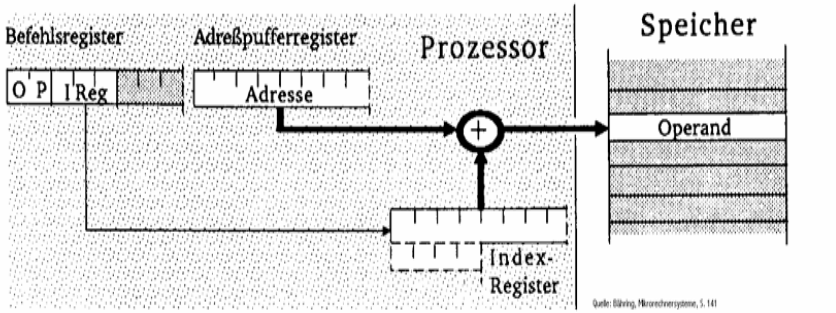
Register-Indirekte Adressierung

- **Auch Zeigeradressierung**
  - ⇨ **Der Inhalt eines Registers wird als Adresse des Operanden verwendet**
- **postincrement:** `LD R1,(R0)+`
  - ⇨ **Lade R1 mit dem Inhalt der Speicherzelle, auf die R0 zeigt, und incrementiere anschließend R0**
- **preincrement:** `INC +(R0)`
  - ⇨ **Erhöhe zunächst das Register R0 um 1 und danach die Speicherzelle, auf die das neue R0 zeigt**
- **postdecrement:** `LD R1,(R0)-`
  - ⇨ **Lade R1 mit dem Inhalt der Speicherzelle, auf die R0 zeigt, und decrementiere anschließend R0**
- **predecrement:** `CLR -(R0)`
  - ⇨ **Dekrementiere zunächst R0 und lösche die Speicherzelle, auf die das neue R0 zeigt**

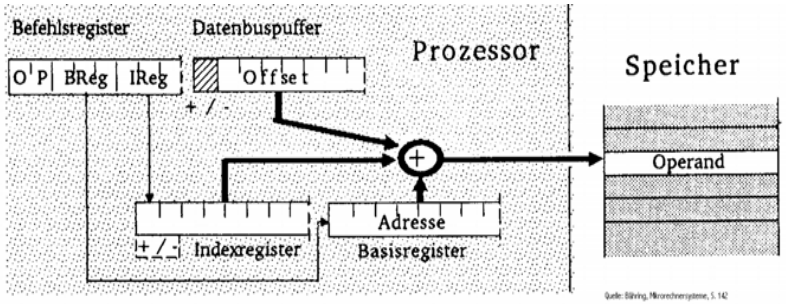


Indizierte Adressierung

- **Speicher-relative Adressierung**
  - ⇨ **Der Basiswert, der zum Indexregister addiert wird, ist im Befehlswort enthalten**
  - ⇨ **Beispiel `ST R1, $A704(R0)`**
    - **Speichere R1 an die Adresse, die sich aus der Summe des Inhalts des Registers R0 und $A704 ergibt**

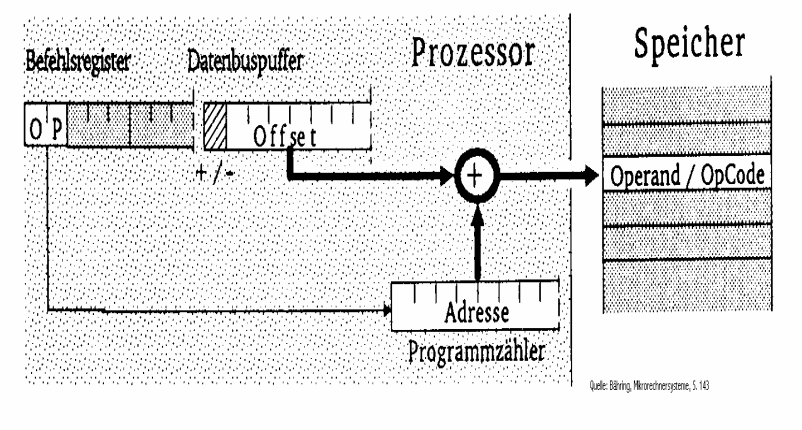


- **Register-relative Adressierung mit Offset**
  - ⇨ **Der Basiswert befindet sich in einem speziellen Basisregister**
  - ⇨ **Der Inhalt des Indexregister und ein Offset wird zum Basiswert addiert**
  - ⇨ **autoincrement und autodecrement**
  - ⇨ **Beispiel: `ST R1, $A7(B0)(I0)+`**
    - **Speichere R1 an die Adresse die sich durch Addition von B0, I0 und dem Offset ergibt und incrementiere I0 anschließend**



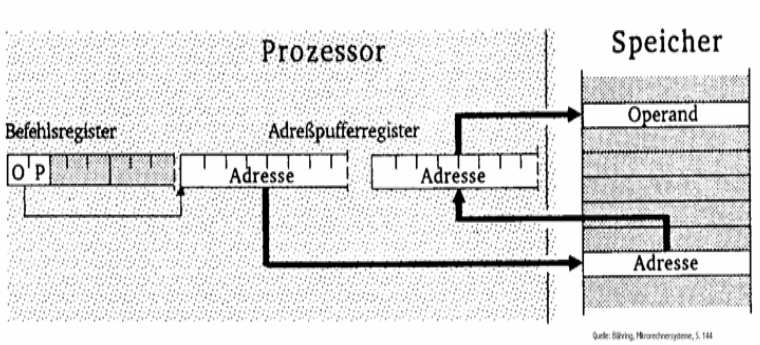
Programmzähler-relative Adressierung

- Der im Befehlscode angegebene Offset wird zum aktuellen Befehlszähler hinzuaddiert
- Beispiel: `BCS $47(PC)`
  - ⇨ Verzweige an die Adresse PC+$47 sofern das Carry-Flag gesetzt ist

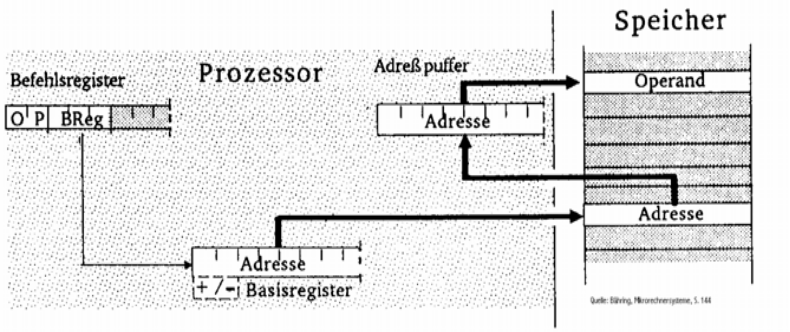


Zweistufige Speicheradressierung

- **Indirekte absolute Adressierung**
  - ⇨ **Der Befehl enthält eine absolute Adresse, die auf ein Speicherwort zeigt. Dieses Speicherwort enthält die gesuchte Adresse**
  - ⇨ **Beispiel: `LDA ($A345)`**
    - **Lade den Accu mit dem Inhalt des Speicherworts, dessen Adresse in $A345 steht**

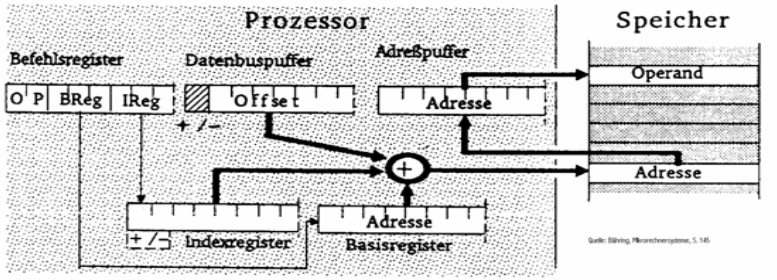


- **Indirekte Register-indirekte Adressierung**
  - ⇨ **Der Befehl bezeichnet ein Register, dessen Inhalt die Speicherzelle ist, deren Inhalt als Adresse für das Speicherwort verwendet wird**
  - ⇨ **Beispiel: LD R1, ((R0))**
    - **Lade R1 mit dem Inhalt der Adresse, die in in der Speicherzelle steht, auf die R0 zeigt**

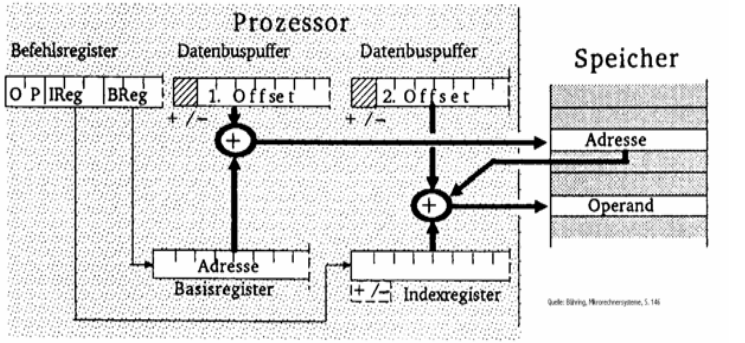


Zweistufige Speicheradressierung

- **Indirekte indizierte Adressierung**
  - ⇨ **Die Adresse des Speicherworts wird aus der Summe von Offset, Basisregister und Indexregister gebildet. Dieses Speicherwort enthält die Adresse des Ziels**
  - ⇨ **Beispiel: `INC ($A7(B0)(I0))`**
    - **Erhöhe die Speicherzelle mit der Adresse $A7+B0+I0 um 1**

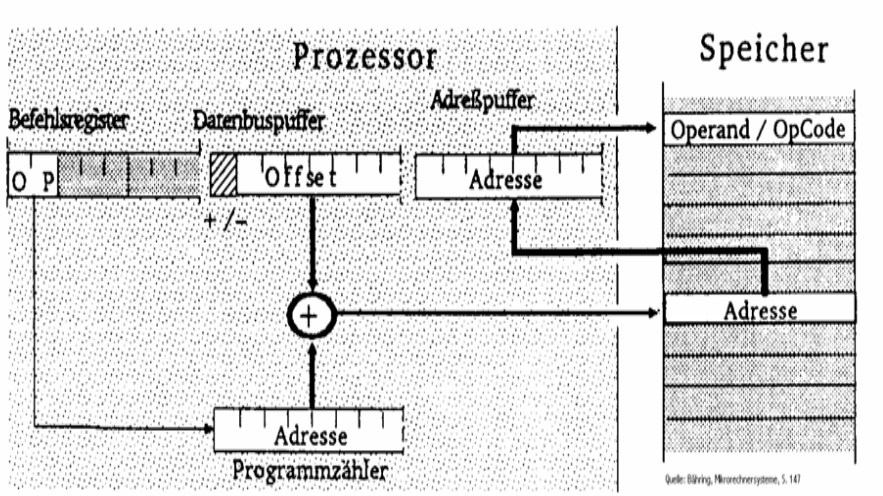


- **Indizierte indirekte Adressierung**
  - ⇨ **Die Adresse des Speicherworts wird aus dem 1. Offset und dem Basisregister gebildet. Der Inhalt dieses Speicherworts wird zum Indexregister und dem 2. Offset hinzuaddiert und bildet die Adresse des gesuchten Speicherworts**
  - ⇨ **Beispiel: `INC $A7($10(B0))(I2)`**
    - **Addiere den Offset $10 zum Inhalt des Basisregisters. Der Inhalt dieser Speicherzelle plus Indexregister und zweiter Offset $A7 ergibt den Wert der gesuchten Adresse**



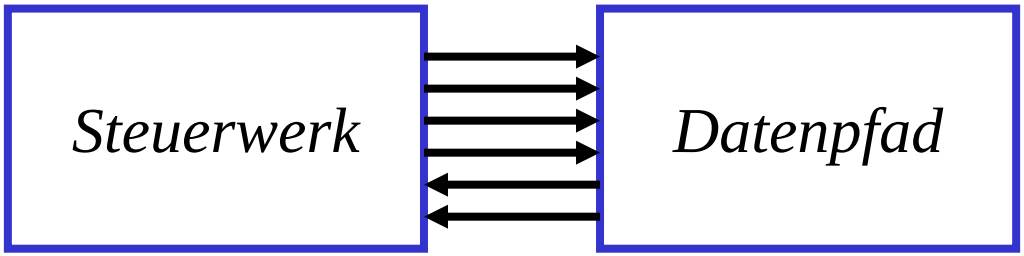
Zweistufige Speicheradressierung

- **Indirekte Programmzähler-relative Adressierung**
  - ⇨ **Die Summe aus Programmzähler und Offset ergeben die Adresse, die auf das Ziel zeigt**
  - ⇨ **Beispiel: JMP ($A7(PC))**
    - **Springe an die Stelle die im Speicherwort mit der Adresse PC plus $A7 steht.**



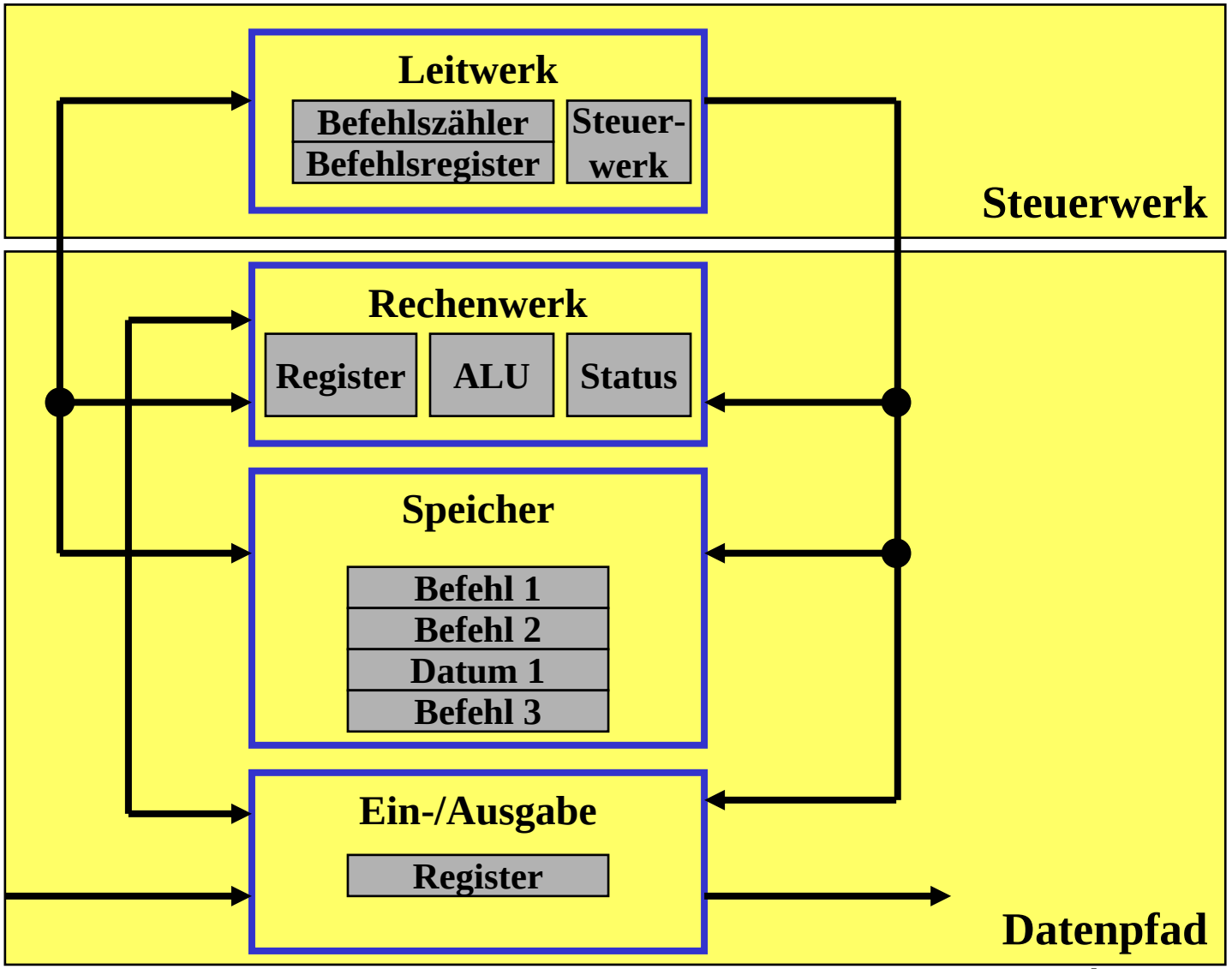
7 Ein einfacher Rechner

- **VonNeumann Architektur**
  - **Ein-/Ausgabe**
  - **Speicher**
  - **Rechenwerk**
  - **Leitwerk (Steuerwerk)**



- **Erweiterung von Steuerwerken**
  - **Ein Steuerwerk bestimmt den Ablauf der Berechnung**
  - **Der Datenpfad bestimmt den Fluss der Operanden und Ergebnisse**
  - **Daten und Programm stehen in einem gemeinsamen Speicher**

Von Neumann Architektur



Befehlsablauf im von Neumann-Rechner

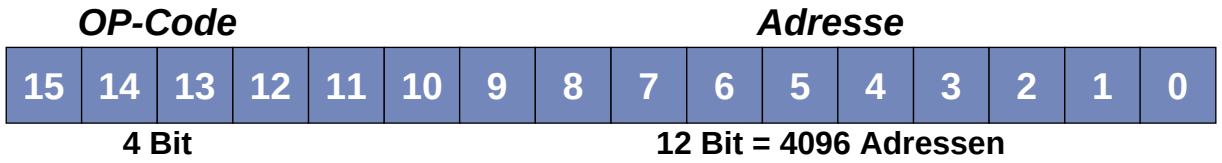
- **Lesen**
  - ⇨ **Einen neuen Programmzähler-Wert (PC) bestimmen**
  - ⇨ **Bestimmung der Speicheradresse des Quelloperanden**
  - ⇨ **Lesezugriff auf den Speicher**
  - ⇨ **Speichern des gelesenen Wertes im Zielregister**
- **Schreiben**
  - ⇨ **Einen neuen Programmzähler-Wert (PC) bestimmen**
  - ⇨ **Bestimmung der Speicheradresse des Zieloperanden**
  - ⇨ **Lesezugriff auf das Quellregister**
  - ⇨ **Schreibzugriff auf den Speicher**

Befehlsablauf im von Neumann-Rechner

- **Verknüpfung von Operanden**
  - ⇨ **Einen neuen Programmzähler-Wert (PC) bestimmen**
  - ⇨ **Auslesen der Operanden aus dem Registerblock**
  - ⇨ **Verknüpfung der Operanden in der ALU**
  - ⇨ **Schreiben des Ergebnisses in den Registerblock**
- **Verzweigungen und Sprünge**
  - ⇨ **Einen neuen Programmzähler-Wert (PC) bestimmen**
  - ⇨ **Berechnung der Adresse des Sprungziels**
  - ⇨ **Prüfung der Sprungbedingung (bei Verzweigungen)**
  - ⇨ **Überschreiben des Befehlszählers, wenn der Sprung ausgeführt werden soll**

Der Toy-Rechner: Allgemeine Informationen

- **Quelle: Phil Koopman, *Microcoded versus hard-wired control*, BYTE, Januar 1987, S. 235**
- **Spezifikation**
  - **einfacher aber vollständiger Mikrorechner**
  - **einfacher Aufbau mit Standardbausteinen**
  - **RISC-Rechner**
    - sehr einfacher Befehlssatz (12 Befehle)
    - alle Befehle in einem Takt (2 Phasen-Takt)
  - **Befehlsformat**

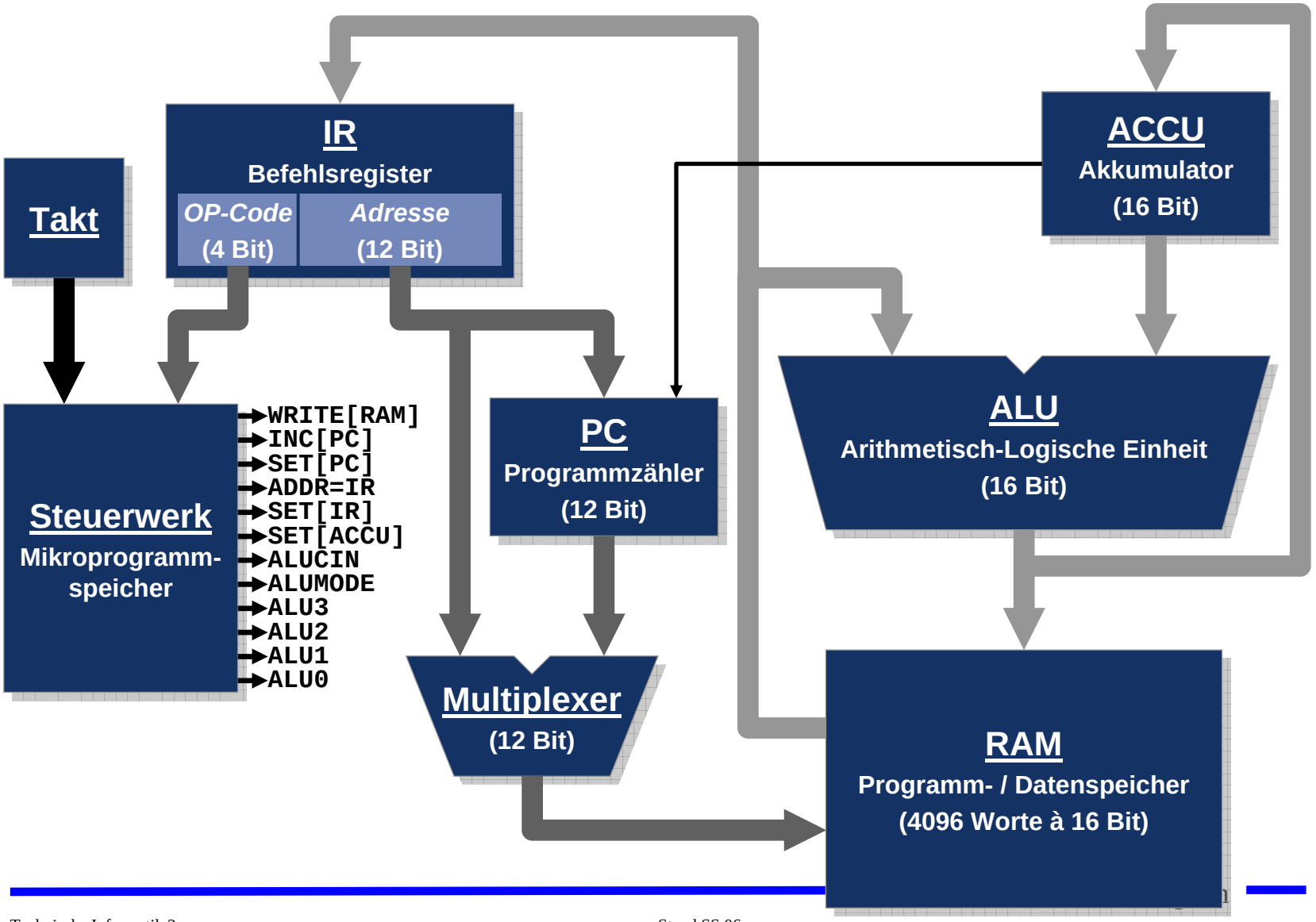


  - **1-Adress-Maschine**
    - Akkumulator liefert implizit immer einen der Operanden
    - Befehl spezifiziert zweiten Operanden (falls notwendig)
    - Ergebnis einer Operation wird immer in den Akkumulator geschrieben

Toy-Befehlssatz

	Mnemonic	Bedeutung
0	`STO <Adresse>`	speichere den Inhalt des ACCUs ins RAM
1	`LDA <Adresse>`	lade den ACCU mit dem Inhalt der Adresse
2	`BRZ <Adresse>`	springe nach Adresse, wenn der ACCU Null ist
3	`ADD <Adresse>`	addiere den Inhalt der Adresse zum ACCU
4	`SUB <Adresse>`	subtrahiere den Inhalt der Adresse vom ACCU
5	`OR  <Adresse>`	logisches ODER des ACCUs mit dem Inhalt der Adresse
6	`AND <Adresse>`	logisches UND des ACCUs mit dem Inhalt der Adresse
7	`XOR <Adresse>`	logisches ExODER des ACCUs mit dem Inhalt der Adresse
8	`NOT`	logisches NICHT der Bits im ACCU
9	`INC`	inkrementiere den ACCU
10	`DEC`	dekrementiere den ACCU
11	`ZRO`	setze den ACCU auf Null
12	`NOP`	keine Operation
13	`NOP`	keine Operation
14	`NOP`	keine Operation
15	`NOP`	keine Operation

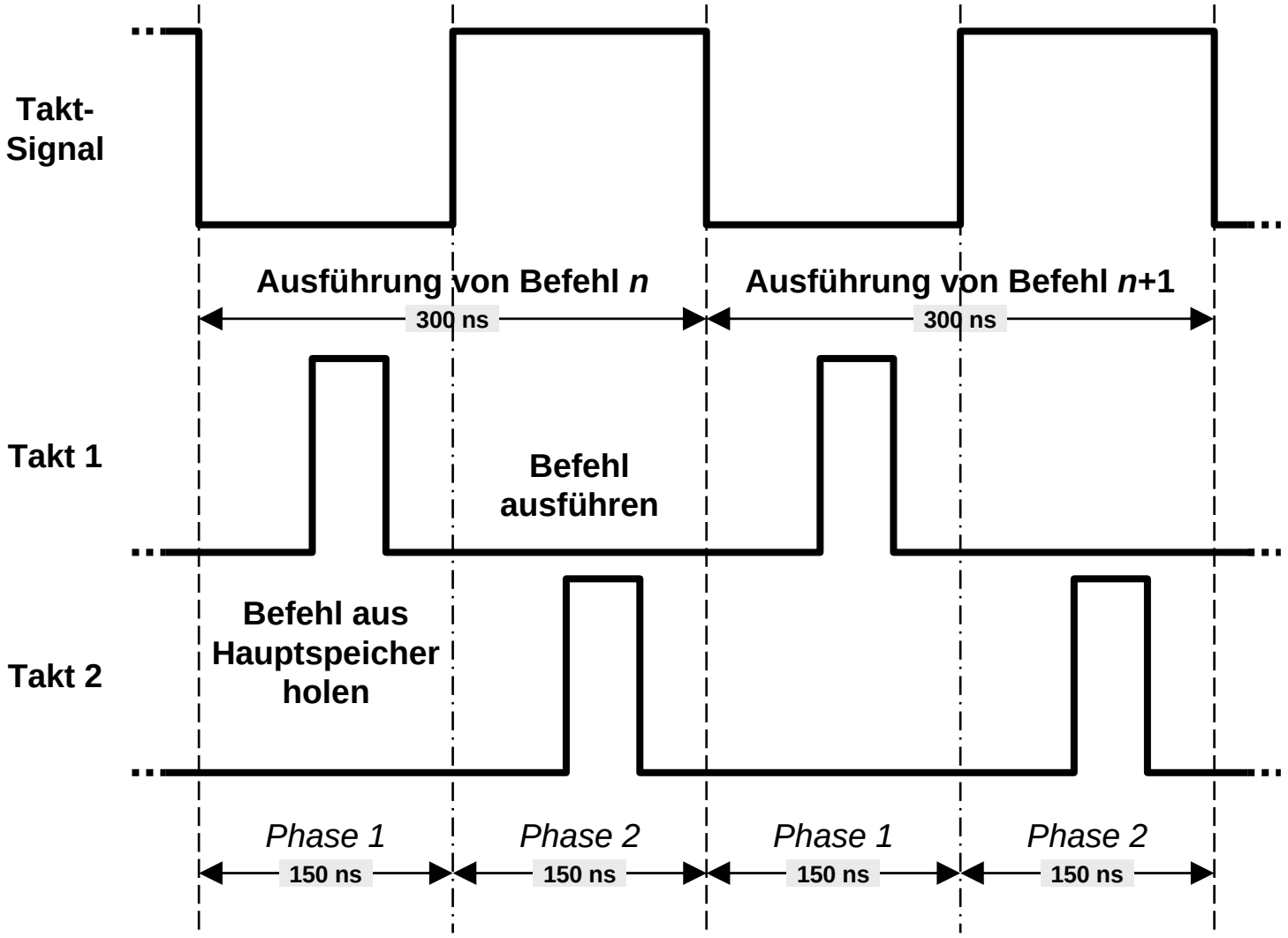
Blockschaltbild



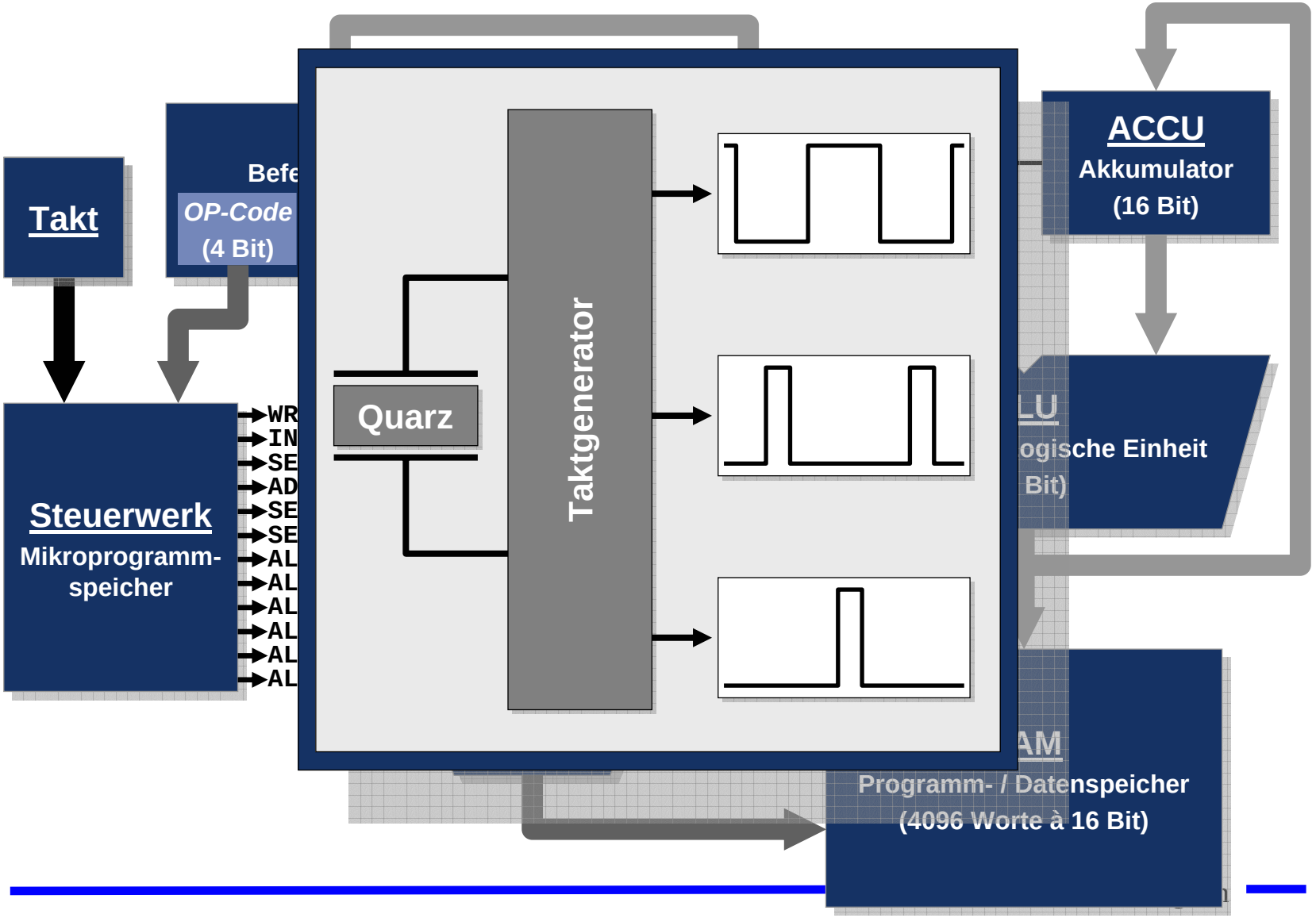
Steuerung im 2-Phasen-Takt

OP-Code	Operation	Phase	Steuerung
0	STO	1	ADDR=IR; ALU=ACCU; WRITE[RAM]
		2	ADDR=PC; SET[IR]; INC[PC]
1	LDA	1	ADDR=IR; ALU=RAM; SET[ACCU]
		2	ADDR=PC; SET[IR]; INC[PC]
2	BRZ	1	SET[PC]
		2	ADDR=PC; SET[IR]; INC[PC]
3	ADD	1	ADDR=IR; ALU=ACCU+RAM; SET[ACCU]
		2	ADDR=PC; SET[IR]; INC[PC]
4	SUB	1	ADDR=IR; ALU=ACCU–RAM; SET[ACCU]
		2	ADDR=PC; SET[IR]; INC[PC]
...			
8	NOT	1	ALU=~ACCU; SET[ACCU]
		2	ADDR=PC; SET[IR]; INC[PC]
9	INC	1	ALU=ACCU+1; SET[ACCU]
		2	ADDR=PC; SET[IR]; INC[PC]
...			
12 – 15	NOP	1	ADDR=PC; SET[IR]; INC[PC]
		2	–

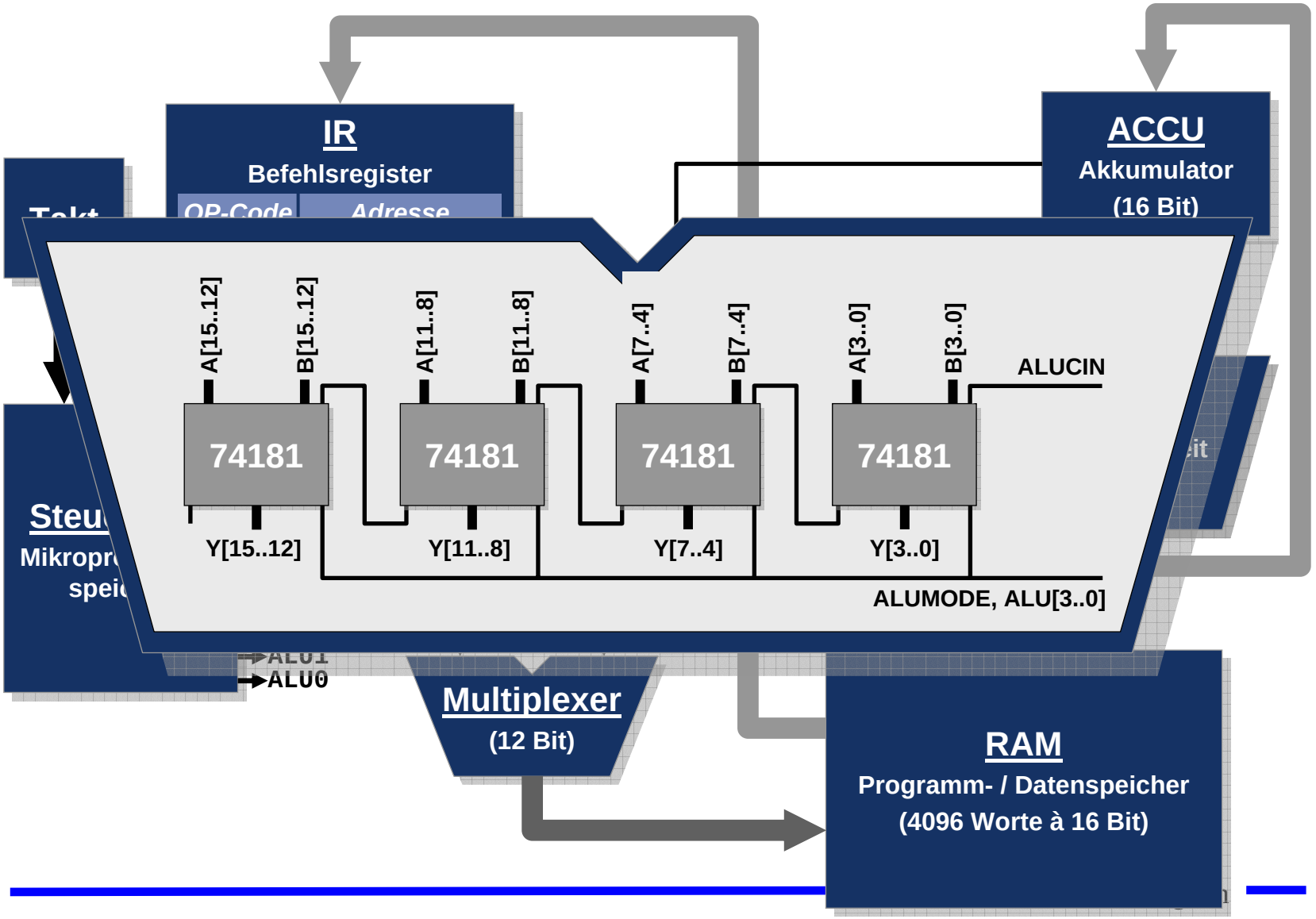
Ablaufsteuerung – 2-Phasen-Takt



Der Taktgenerator



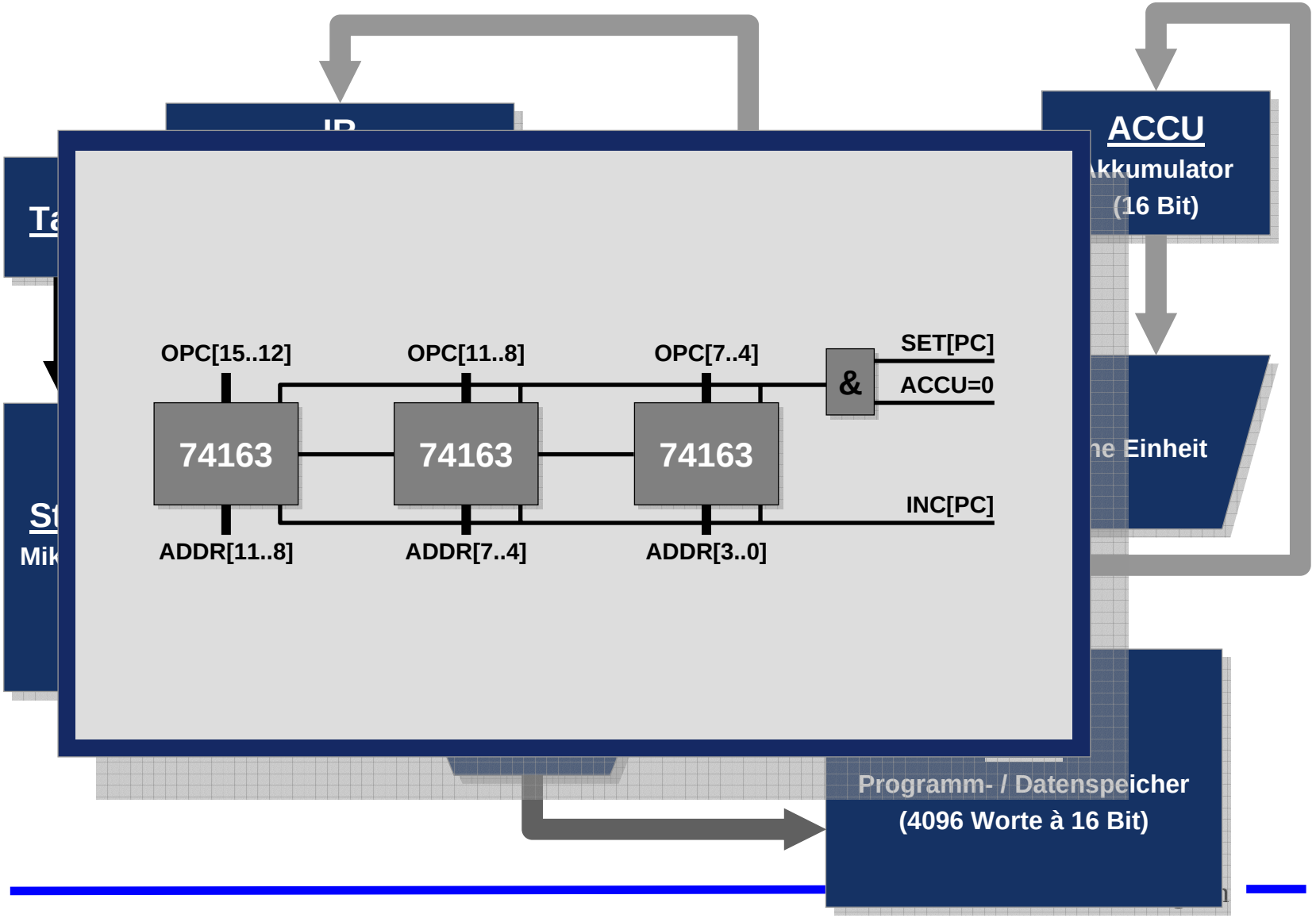
Die ALU

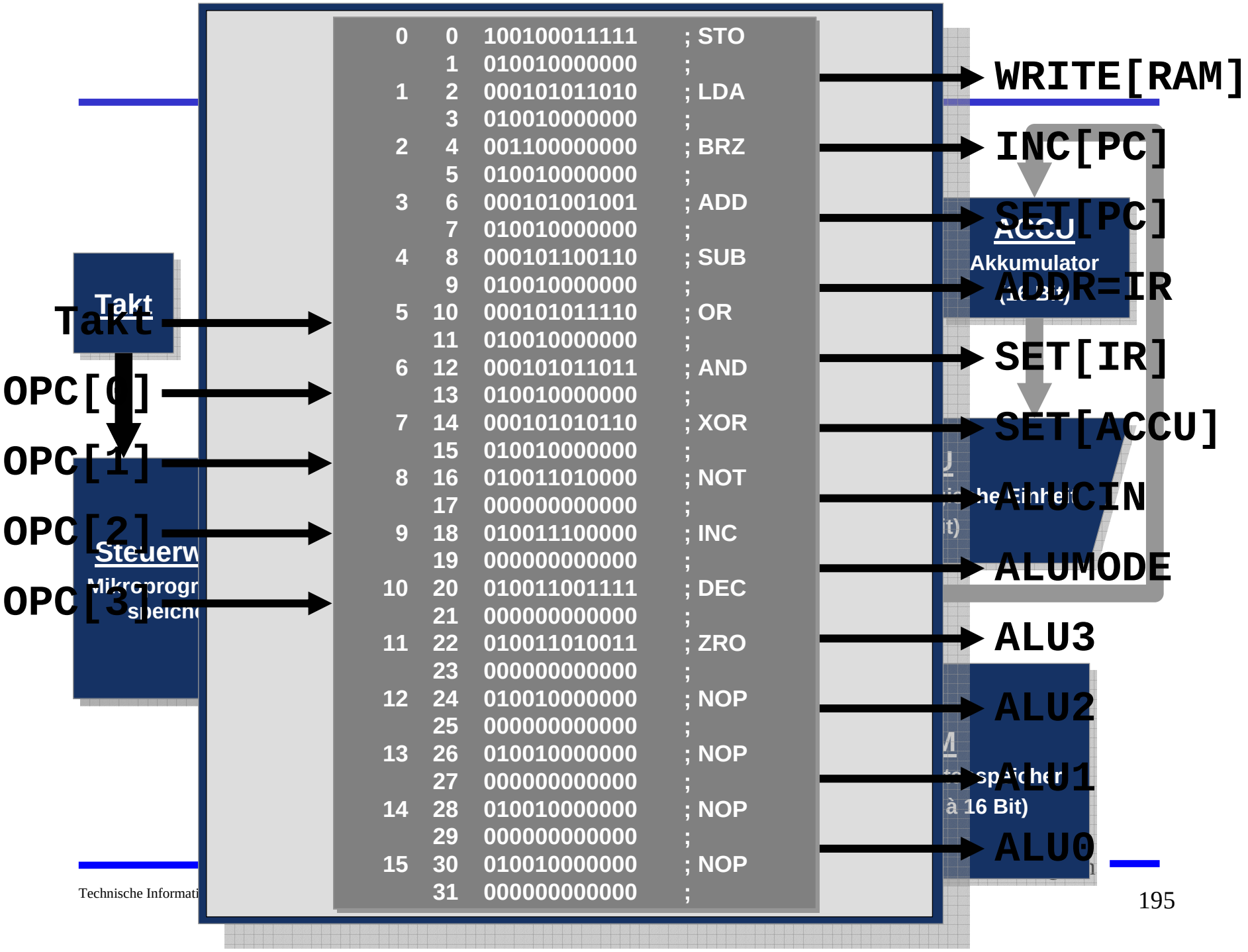

IR
Befehlsregister
OP-Code
Adresse
ACCU
Akkumulator
(16 Bit)
A[15..12]
B[15..12]
A[11..8]
B[11..8]
A[7..4]
B[7..4]
A[3..0]
B[3..0]
ALUCIN
74181
74181
74181
74181
Y[15..12]
Y[11..8]
Y[7..4]
Y[3..0]
ALUMODE, ALU[3..0]
Multiplexer
(12 Bit)
RAM
Programm- / Datenspeicher
(4096 Worte à 16 Bit)

Ansteuerung einer ALU vom Typ TTL-Series 74181

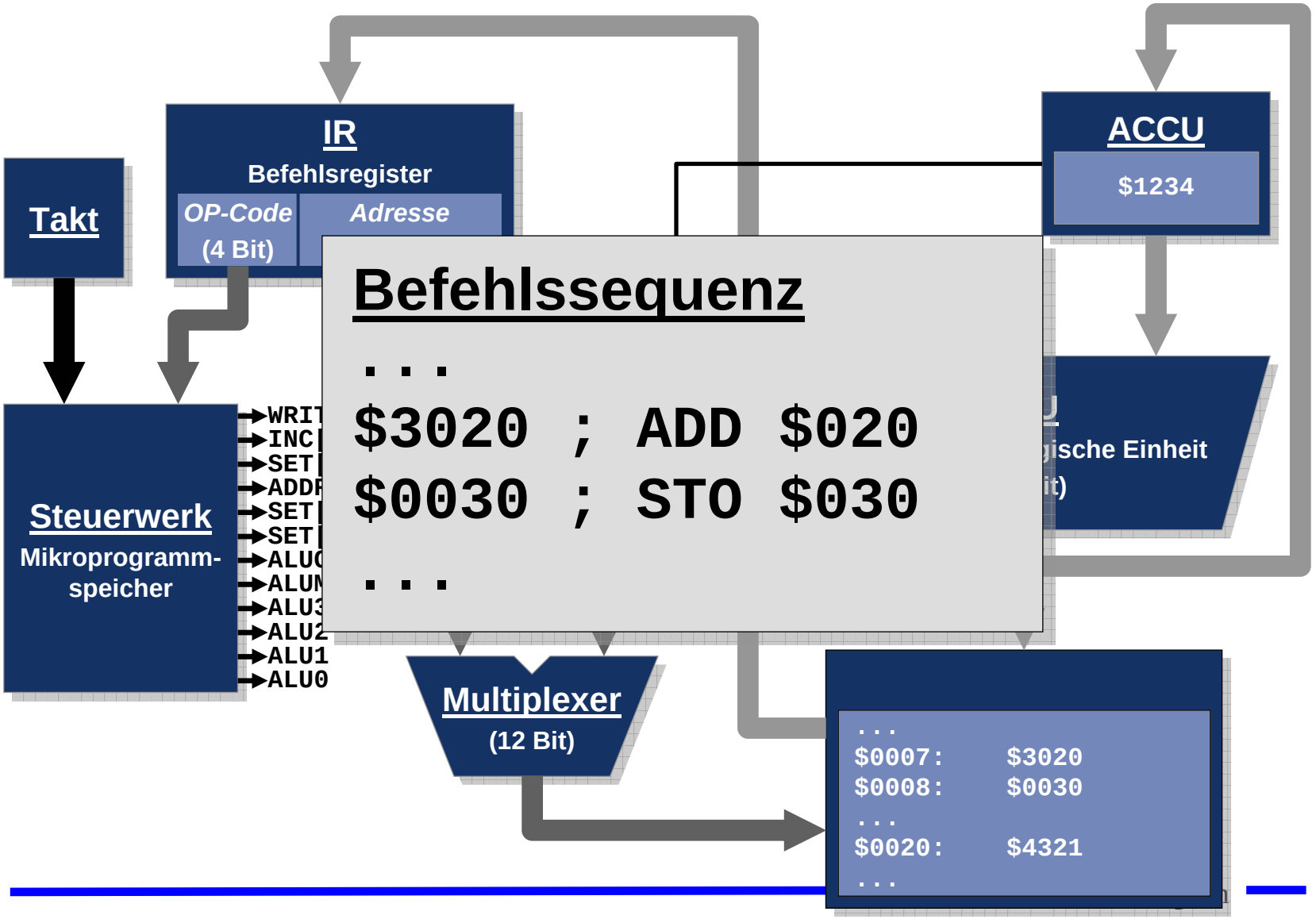
ALU3	ALU2	ALU1	ALU0	ALUMODE = 0 (arithmetischer Modus) ALUCIN = 0	ALUMODE = 0 (arithmetischer Modus) ALUCIN = 1	ALUMODE = 1 (logischer Modus)
0	0	0	0	A	A + 1	¬A
0	0	0	1	A | B	(A | B) + 1	¬(A & B)
0	0	1	0	A | ¬B	(A | ¬B) + 1	¬A & B
0	0	1	1	–1	0	0
0	1	0	0	A + (A & ¬B)	A + (A & ¬B) + 1	¬(A & B)
0	1	0	1	(A | B) + (A & ¬B)	(A | B) + (A & ¬B) + 1	¬B
0	1	1	0	A – B – 1	A – B	A ⊕ B
0	1	1	1	(A & B) – 1	A & B	A & ¬B
1	0	0	0	A + (A & B)	A + (A & B) + 1	¬A | B
1	0	0	1	A + B	A + B + 1	¬(A ⊕ B)
1	0	1	0	(A | ¬B) + (A & B)	(A | ¬B) + (A & B) + 1	B
1	0	1	1	(A & B) – 1	A & B	A & B
1	1	0	0	A + A	A + A + 1	1
1	1	0	1	(A | B) + A	(A | B) + A + 1	A | ¬B
1	1	1	0	(A | ¬B) + A	(A | ¬B) + A + 1	A | B
1	1	1	1	A – 1	A	A

Der Befehlszähler

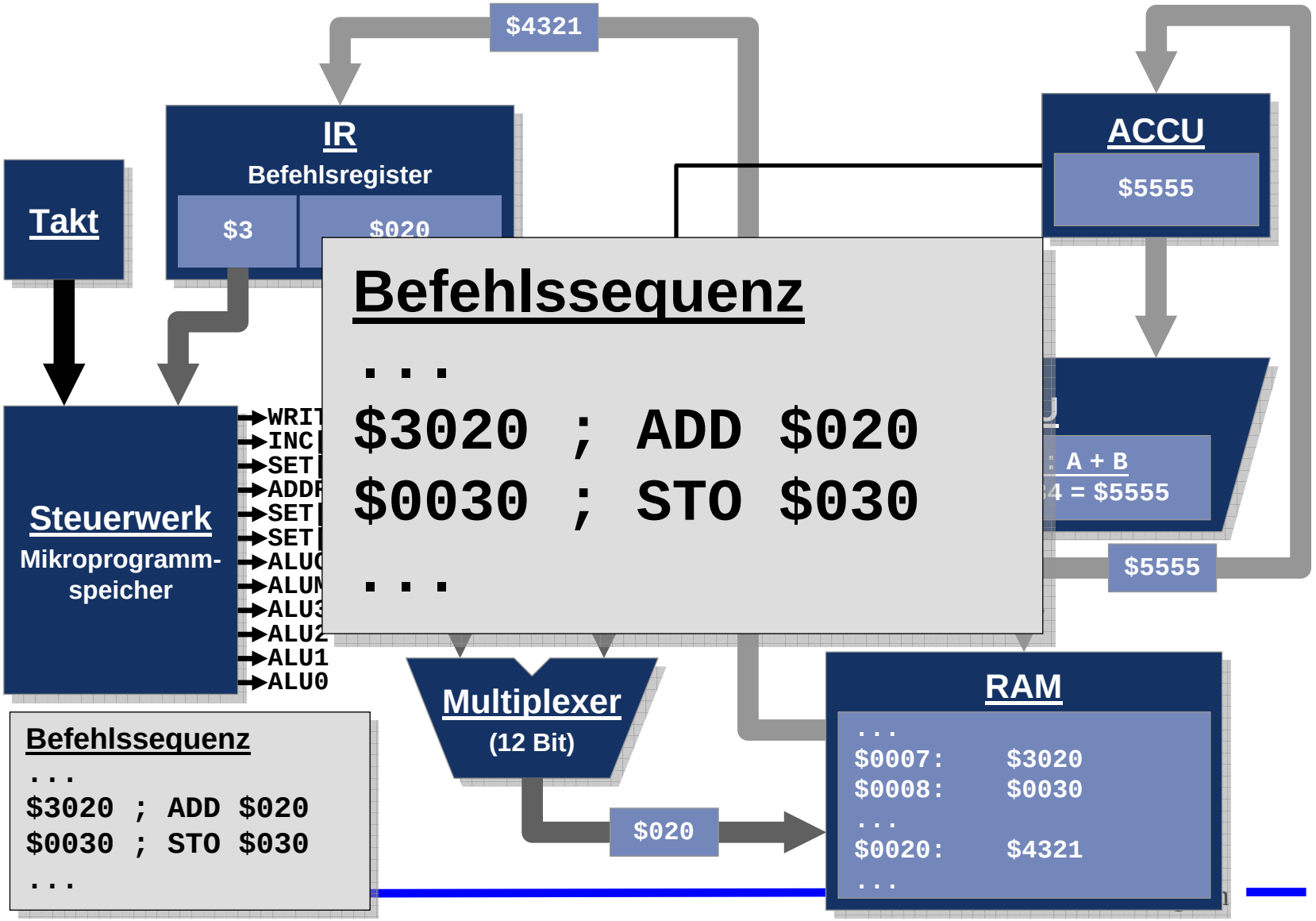



Takt
OPC[0]
OPC[1]
OPC[2]
OPC[3]
Steuerw
Mikroprogr
speich
0 0 100100011111 ; STO
1 010010000000 ;
1 2 000101011010 ; LDA
3 010010000000 ;
2 4 001100000000 ; BRZ
5 010010000000 ;
3 6 000101001001 ; ADD
7 010010000000 ;
4 8 000101100110 ; SUB
9 010010000000 ;
5 10 000101011110 ; OR
11 010010000000 ;
6 12 000101011011 ; AND
13 010010000000 ;
7 14 000101010110 ; XOR
15 010010000000 ;
8 16 010011010000 ; NOT
17 000000000000 ;
9 18 010011100000 ; INC
19 000000000000 ;
10 20 010011001111 ; DEC
21 000000000000 ;
11 22 010011010011 ; ZRO
23 000000000000 ;
12 24 010010000000 ; NOP
25 000000000000 ;
13 26 010010000000 ; NOP
27 000000000000 ;
14 28 010010000000 ; NOP
29 000000000000 ;
15 30 010010000000 ; NOP
31 000000000000 ;
WRITE[RAM]
INC[PC]
SET[PC]
ACCU
Akkumulator
ADDR=IR
SET[IR]
SET[ACCU]
ALUCIN
ALUMODE
ALU3
ALU2
ALU1
ALU0
à 16 Bit)

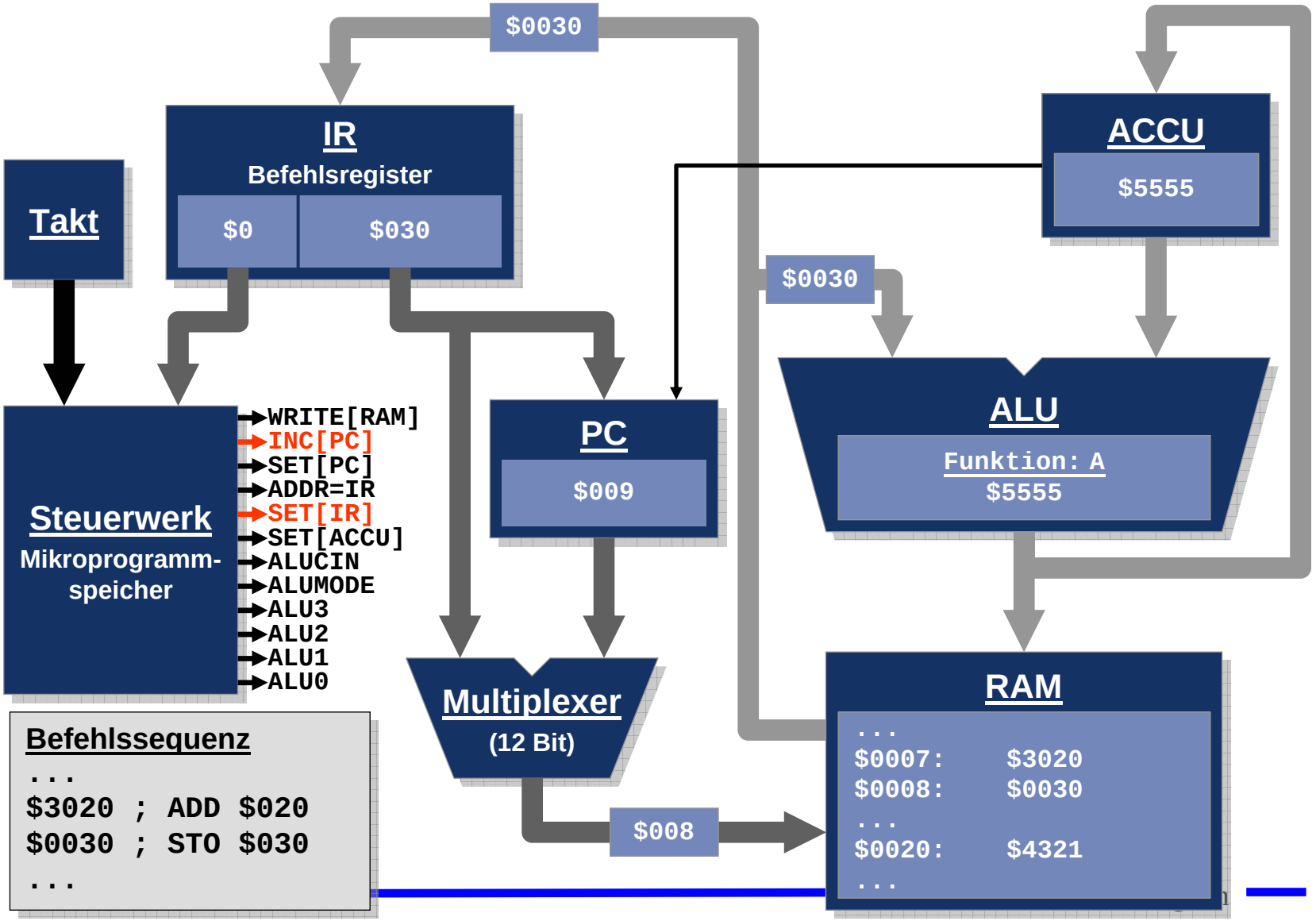
Ablauf eines Maschinenbefehls
Ausgangssituation



Ablauf eines Maschinenbefehls: Phase 1



Ablauf eines Maschinenbefehls: Phase 2



Toy-Programm

```
; Variablen:
;   Loopcount=$20, Result=$21 (enthaelt zunaechst 0)
; Labels:
;   loop=$2, end=$b
;
$0020 ; STO Loopcount       ; Auswerten des initialen ACCU-Inhalts
$200b ; BRZ end             ; Schon fertig?
#-------------------------
#loop:
$1021 ; LDA Result          ; Schleifenzaehler zu Result addieren
$3020 ; ADD Loopcount
$0021 ; STO Result
$1020 ; LDA Loopcount       ; Schleifenzaehler aktualisieren
$a000 ; DEC
$0020 ; STO Loopcount
$200b ; BRZ end             ; Fertig?
$b000 ; ZRO                 ; Nein,
$2002 ; BRZ loop            ; dann wieder von vorn
#-------------------------
#end:
$b000 ; ZRO
$200b ; BRZ end             ; Endlosschleife
```

Der Toy-Emulator

```
   Accu    :  $    1                Address :   Contents
   PC      :  $    7               ---------------------------------------
   InsnReg :  STO $20
                                     $   0  :   $  20  $200b  $1021  $3020
---------------------------------    $   4  :   $  21  $1020  $a000  $  20
                                     $   8  :   $200b  $b000  $2002  $b000
   Steps   :          79            $   c  :   $200c  $   0  $   0  $   0
                                     $  10  :   $   0  $   0  $   0  $   0
---------------------------------    $  14  :   $   0  $   0  $   0  $   0
                                     $  18  :   $   0  $   0  $   0  $   0
        T O Y - EMULATOR             $  1c  :   $   0  $   0  $   0  $   0
        (c) 1992-95, gjh             $  20  :   $   2  $  36  $   0  $   0
                                     $  24  :   $   0  $   0  $   0  $   0
   <Space>   Single Step             $  28  :   $   0  $   0  $   0  $   0
   <S>       Enter # of Steps        $  2c  :   $   0  $   0  $   0  $   0
   <A>       Set Accumulator         $  30  :   $   0  $   0  $   0  $   0
   <P>       Set Program Counter     $  34  :   $   0  $   0  $   0  $   0
   <M>       Set RAM Cell            $  3c  :   $   0  $   0  $   0  $   0
   <+>, <->  Scroll RAM-Window       $  40  :   $   0  $   0  $   0  $   0
                                     $  44  :   $   0  $   0  $   0  $   0
   <Q>, <^C> Game Over
```

Der Toy-Emulator

```
   Accu    :  $    1              Address :   Contents
   PC      :  $    8             ---------------------------------------
   InsnReg :  BRZ $b
                                   $   0  :   $  20  $200b  $1021  $3020
---------------------------------  $   4  :   $  21  $1020  $a000  $  20
                                   $   8  :   $200b  $b000  $2002  $b000
   Steps   :          80           $   c  :   $200c  $   0  $   0  $   0
                                   $  10  :   $   0  $   0  $   0  $   0
---------------------------------  $  14  :   $   0  $   0  $   0  $   0
                                   $  18  :   $   0  $   0  $   0  $   0
        T O Y - EMULATOR           $  1c  :   $   0  $   0  $   0  $   0
        (c) 1992-95, gjh           $  20  :   $   1  $  36  $   0  $   0
                                   $  24  :   $   0  $   0  $   0  $   0
   <Space>   Single Step           $  28  :   $   0  $   0  $   0  $   0
   <S>       Enter # of Steps      $  2c  :   $   0  $   0  $   0  $   0
   <A>       Set Accumulator       $  30  :   $   0  $   0  $   0  $   0
   <P>       Set Program Counter   $  34  :   $   0  $   0  $   0  $   0
   <M>       Set RAM Cell          $  3c  :   $   0  $   0  $   0  $   0
   <+>, <->  Scroll RAM-Window     $  40  :   $   0  $   0  $   0  $   0
                                   $  44  :   $   0  $   0  $   0  $   0
   <Q>, <^C> Game Over
```

Unterschiede zu realen Rechnern

	Toy Rechner	reale Prozessoren
Wortlänge	16 BitDaten 12 Bit Adresssen	 bis 100 Bit
Mikroinstruktionen	1 Routine pro Maschinenbefehl	mehrere Routinen pro Maschinenbefehl
Umfang des Mikroprogramms	384 Bit	300 000 Bit
Verzweigungsbefehle	1 Verzweigungsbefehl	10-33 Verzweigungsbefehle
Adressierungsmodi	1 Addressierungsmodus	1-21 Addressierungsmodi
Befehlssatz	12 Befehle	100 - 300 Befehle
Registersatz	1 Register (Akku)	32 - 512 Register