CP1 + Befehlssatz

Befehle, die Flags beeinflussen

Befehl		Flag (PSW)				Bemerkung
		[z]ero	[c]arry	[l]ess	[g]reater	
Original CP1	CP1 +					
ADD	ADD		x			abweichend zum Original wird kein Fehler ausgegeben, sondern das Carry-Flag gesetzt
	ADI		x			
SUB	SUB		x			abweichend zum Original wird kein Fehler ausgegeben, sondern das Carry-Flag gesetzt
	SBI		x			
	MUL		x			
	INC		x			
	DEC		x			
VGL	CPE	x				
VGR	CPG	x				
VKL	CPL	x				
	CPI	x	x	x	x	
	CMP	x	x	x	x	
	SLC		x			
	SRC		x			

Tabelle 1

Adressierungbezeichnungen

a	Akku
r	Register (B, C, D, E)
mem	direkte Speicherzelle
@mem	indirekte Speicherzelle, das Datum in mem verweist auf die selektierte Speicherzelle. mem beinhaltet einen Zeiger auf die Speicherzelle
addr	Speicheradresse
@addr	indirekte Spungzieladresse, der Inhalt der Adresse beinhaltet das Sprungziel
const	8-Bit Konstante (0 - 255)
reg	Register B to E
bit	Bitpostion (1 bis 8) innerhalb einer Speicherstelle, des Akkus oder eines Ports. WICHTIG: Zählung der Bitposition beginnt bei 1 und nicht wie ansonsten üblich mit 0
sp	Stackpointer
psw:x	die Flags des Programmstatusworts zero ; carry ; less ; greater : psw.z ; psw.c ; psw.l ; psw.g

Tabelle 2

CP1+ Mnemonic und Opcode

	0	1	2	3	4	5	6	7	8	9
00	DB const	HLT	CDIS	CDEL const	MVI a, const	MOV a, mem	MOV mem, a	ADD a, mem	SUB a, mem	JMP addr
10	CPE a, mem	JZ addr	CPG a, mem	CPL a, mem	NOTB a	ANDB a, mem	IN(B) p1 *	OUT(B) p1 **	OUT(B) p2 ***	MOV a,@mem
20	MOV @mem, a	JMP @addr	reserved	reserved	reserved	DJNZ a, addr	INC a	DEC a	MVI b, const	MVI c, const
30	MVI d, const	MVI e, const	MOV a, b	MOV a, c	MOV a, d	MOV a, e	MOV b, a	MOV c, a	MOV d, a	MOV e, a
40	MOV b, c	MOV b, d	MOV b, e	MOV c, b	MOV c, d	MOV c, e	NOT a	XOR a, mem	XRI a, const	CPI a, const
50	CMP a, mem	JPL addr	JPG addr	JC addr	SLC a	SRC a	AND a, mem	ANI a, const	OR a, mem	ORI a, const
60	ADI a, const	SBI a, const	MUL a, b	INT const	CALL addr	RET				

Tabelle 3

CP1 originale Mnemonic und Opcode

Die Mnemonic von CP1+ und CP1 sind unterschiedlich, bei gleicher Funktionalität. D.h. dass es für dieselbe Funktionalität (Opcode) unterschiedliche Namen (Mnemonic) gibt. Der (virtuelle) Prozessor des CP1 und des CP1+ interpretiert nur die Opcodes zur Ausführung und nicht deren Namen. Aus diesem Grund bewirken die ersten 24 Opcodes auf beiden Systemen dasselbe.

	0	1	2	3	4	5	6	7	8	9
00	DB const	HLT	ANZ	VZG const	AKO const	LDA mem	ABS mem	ADD mem	SUB mem	SPU addr
10	VGL mem	SPB addr	VGR mem	VKL mem	NEG	UND mem	P1E *	P1A **	P2A ***	LIA @mem
20	AIS @mem	SIU @addr	reserved	reserved	reserved					

Tabelle 4

* dem Opcode 16 sind 2 Befehle zugeordnet:
IN p1 → 8 - Bit Leseoperation
INB p1,bit → 1 - Bit Leseoperation
** dem Opcode 17 sind 2 Befehle zugeordnet:
OUT p1 → 8 - Bit Schreiboperation
OUTB p1,bit → 1 - Bit Schreiboperation
*** dem Opcode 16 sind 2 Befehle zugeordnet:
OUT p2 → 8 - Bit Schreiboperation
OUTB p2 → 1 - Bit Schreiboperation

ADD a, <mem>

Funktion: add akku with memory value

Opcode: 07

Beschreibung: addiere zum Akku den Inhalt der in mem angegebenen Speicherstelle hinzu. Ist das Ergebnis größer als ein 8-Bit Wert (255) wird das Carry-Flag im Programmstatuswort (PSW) gesetzt (und signalisiert somit ein Überlaufen des Akkus)

Operation: ADD a, <mem>

(A) ← (A) + Speicherzelle [mem]

wenn (A) + <mem> > 255 dann PSW:C = 1; (A) ← (A) + <mem> - 256

Beispiel:

```
.org 100
db 10       ; Speicherstelle 100 mit Wert 10 beschreiben
.
.
mvi a,240   ; A = 240
add a,100   ; den Wert aus Zelle 100 hinzuaddieren

; der Akku hat den Wert 250, das Carry-Flag ist
; nicht gesetzt.

add a,100   ; den Wert aus Zelle 100 hinzuaddieren

; der Akku hat den Wert 250, das Carry-Flag ist gesetzt.
```

ADI a, <const>

Funktion: add immediately akku with const

Opcode: 60

Beschreibung: addiere zum Akku den Inhalt Konstante const hinzu. Ist das Ergebnis größer als ein 8-Bit Wert (255) wird das Carry-Flag im Programmstatuswort (PSW) gesetzt (und signalisiert somit ein Überlaufen des Akkus)

Operation: ADI a, <const>

(A) ← (A) + <const>

wenn (A) + <const> > 255 dann PSW:C = 1; (A) ← (A) + <const> - 256

Beispiel:

```
.org 100
db 10       ; Speicherstelle 100 mit Wert 10 beschreiben
.
.
mvi a,240   ; A = 240
add a,100   ; den Wert aus Zelle 100 hinzuaddieren

; der Akku hat den Wert 250, das Carry-Flag ist nicht
; gesetzt.

add a,100   ; den Wert aus Zelle 100 hinzuaddieren

; der Akku hat den Wert 250, das Carry-Flag ist gesetzt
```

AND a, <mem>

Funktion: logical AND accu with memory value

Opcode: 56

Beschreibung: führt eine logische UND Verknüpfung (AND) aller Bits des Akkuinhalts mit den Bits des Inhaltes der angegebenen Speicherstelle durch. Eine UND Verknüpfung bedeutet: Nur wenn beide beteiligten Bits eine logische 1 enthalten, wird das Ergebnis zu 1 wird, ist dem nicht so, ist das Ergebnis 0.

Eine logische UND - Verknüpfung wird u.a. benutzt, um gezielt einzelne Bits eines Datums löschen zu können

Beispiel:

```
          0100.1001 binär   (  73 dezimal )
   and    1011.1111 binär   ( 191 dezimal )
-----------------------------------------
Ergebnis: 0000.1001 binär   (   9 dezimal )
```

Operation: AND a, <mem>

(A) ← (A) & <mem>

Beispiel:

```
mvi a, 191                  ; 1011.1111
mov 100, a
mvi a, 73                   ; 0100.1001
and a, 100
; Akkuinhalt ist nun 9      ; 0000.1001
```

ANDB a, <mem>

Funktion: logical and the first bit in a

Opcode: 15

Beschreibung: führt eine logische UND Verknüpfung des ersten Bits im Akku mit dem ersten Bit der Speicherstelle mem durch

Operation: ANDB a, <mem>

(A:1) ← **(**A:1) & (mem:1)

Beispiel:

```
.org 100
db 1          ; Speicherstelle 100 = 1
db 0          ; Speicherstelle 101 = 0
.
.org 0
mvi a,1
andb a, 100   ; das Ergebnis in A = 1
andb a, 101   ; das Ergebnis in A = 0
```

ANI a, <const>

Funktion: immediately logical AND accu with const

Opcode: 57

Beschreibung: führt eine logische UND Verknüpfung (AND) aller Bits des Akkuinhalts mit den Bits der angegebenen Konstante durch. Eine UND Verknüpfung bedeutet: Nur wenn beide beteiligten Bits eine logische 1 enthalten, wird das Ergebnis zu 1 wird, ist dem nicht so, ist das Ergebnis 0.

Eine logische UND - Verknüpfung wird u.a. benutzt, um gezielt einzelne Bits eines Datums löschen zu können

Beispiel:

```
          0100.1001 binär   (  73 dezimal )
   and    1011.1111 binär   ( 191 dezimal )
-----------------------------------------
Ergebnis: 0000.1001 binär   (   9 dezimal )
```

Operation: ANI a, <const>

(A) ← (A) & <const>

Beispiel:

```
mvi a, 73                  ; 0100.1001
ani a, 191                 ; 1011.1111
; Akkuinhalt ist nun 9     ; 0000.1001
```

CALL <addr>

Funktion: call a subroutine

Opcode: 64

Beschreibung: ruft das Unterprogramm an Adresse addr auf

Operation: CALL <addr>

(SP [Inhalt]) ← (PC)
(SP) ← (SP+1)
(PC) ← <addr>

Beispiel:

```
; Blinkprogramm
.org 0
mvi a,1
out p2
mvi a,10
call 15
mvi a,0
out p2
mvi a, 10
call 15
jmp 0

; Verzoegerungsunterprogramm: warte A * 0,1 s
.org 15
mov e,a      ; A zwischenspeichern
cdel 100     ; Verzoegerung 0,1 s
djnz 16      ; Anzahl (A) mal wiederholen
mov a,e      ; A restaurieren
ret          ; zurueck zum aufrufenden Programm
```

CDEL <const>

Funktion: call delay

Opcode: 03

Beschreibung: verzögert die Programmausführung um const Anzahl Millisekunden

Operation: CDEL const

wait const * 1 ms

CDIS

Funktion: call display

Opcode: 02

Beschreibung: zeigt den Inhalt des Akkus dreistellig auf der Siebensegmentanzeige an (Zahlenwerte von 0 .. 255)

Operation: CDIS

display ← A

CMP a, <mem>

Funktion: compare accu with memory value

Opcode: 50

Beschreibung: vergleiche den Akkuinhalt mit dem Inhalt der angegebenen Speicherstelle. Das Ergebnis des Vergleichs wird im Programmstatuswort (PSW) gespeichert.

Ist der Akku gleich der Konstanten wird das Zero-Flag gesetzt, andernfalls gelöscht.
Ist der Akku kleiner als die Konstante wird das Less-Flag gesetzt, andernfalls gelöscht.
Ist der Akku größer als die Konstante wird das Greater-Flag gesetzt, andernfalls gelöscht.

Operation: CMP a, <mem>
wenn (A) = <mem> dann (PSW:Z) ← 1 andernfalls (PSW:Z) ← 0
wenn (A) < <mem> dann (PSW:L) ← 1 andernfalls (PSW:L) ← 0
wenn (A) > <mem> dann (PSW:G) ← 1 andernfalls (PSW:G) ← 0

Beispiel:

```
mvi a, 50
mov 100, a     ; Wert 50 in Speicherstelle 100 schreiben
mvi a, 10
cmp a,100
jz 20
jpg 30
jpl 40
.org 20
; hier fortfahren, wenn Akku = 50
.
.org 30
; hier fortfahren, wenn Akku > 50
.
.org 40
; hier fortfahren, wenn Akku < 50
```

Das Programm wird an Speicherstelle 40 fortgeführt

CPE a, <mem>

Funktion: compare equal

Opcode: 10

Beschreibung: vergleiche den Akkuinhalt mit dem Inhalt der in mem angegebenen Speicherstelle. Sind beide Inhalte gleich, wird das Zero-Flag im Programmstatuswort (PSW) gesetzt., andernfalls gelöscht

Operation: CPE a, <mem>

wenn (A) = mem dann (PSW:Z) ← 1 andernfalls (PSW:Z) ← 0

Beispiel:

```
.org 100
db 10       ; Speicherstelle 100 mit Wert 10 beschreiben
.
.
mvi a,10
cpe a,100
jz 20       ; mit Adresse 20 bei Gleichheit fortfahren

; hier fortfahren, wenn Akku ungleich 10
```

Das Programm wird an Speicherstelle 20 fortgeführt

CPG a, <mem>

Funktion: compare greater

Opcode: 12

Beschreibung: vergleiche den Akkuinhalt mit dem Inhalt der in mem angegebenen Speicherstelle. Ist der Akkuinhalt größer als der Speicherinhalt, wird das Zero-Flag im Programmstatuswort (PSW) gesetzt., andernfalls gelöscht

Operation: CPG a, <mem>

wenn (A) > mem dann (PSW:Z) ← 1 andernfalls (PSW:Z) ← 0

Beispiel:

```
.org 100
db 10       ; Speicherstelle 100 mit Wert 10 beschreiben
.
.
mvi a,10
cpg a,100
jz 20       ; mit Adresse 20 bei Gleichheit fortfahren

; hier fortfahren, wenn Akku ungleich 10
```

Das Programm wird NICHT an Speicherstelle 20 fortgeführt, da der Akkuinhalt nicht größer als der Speicherinhalt ist

CPI a, <const>

Funktion: compare immediately accu with const

Opcode: 49

Beschreibung: vergleiche den Akkuinhalt mit der angegebenen Konstante. Das Ergebnis des Vergleichs wird im Programmstatuswort (PSW) gespeichert.

Ist der Akku gleich der Konstanten wird das Zero-Flag gesetzt, andernfalls gelöscht. Ist der Akku kleiner als die Konst**DJNZ**ante wird das Less-Flag gesetzt, andernfalls gelöscht.

Ist der Akku größer als die Konstante wird das Greater-Flag gesetzt, andernfalls gelöscht.

Operation: CPI a, <const>
wenn (A) = <const> dann (PSW:Z) ← 1 andernfalls (PSW:Z) ← 0
wenn (A) < <const> dann (PSW:L) ← 1 andernfalls (PSW:L) ← 0
wenn (A) > <const> dann (PSW:G) ← 1 andernfalls (PSW:G) ← 0

Beispiel:

```
mvi a,10
cpi a,100
jz 20
jpg 30
jpl 40
.org 20
; hier fortfahren, wenn Akku = 100
.
.org 30
; hier fortfahren, wenn Akku > 100
.
.org 40
; hier fortfahren, wenn Akku < 100
```

Das Programm wird an Speicherstelle 40 fortgeführt

CPL a, <mem>

Funktion: compare lower

Opcode: 13

Beschreibung: vergleiche den Akkuinhalt mit dem Inhalt der in mem angegebenen Speicherstelle. Ist der Akkuinhalt kleiner als der Speicherinhalt, wird das Zero-Flag im Programmstatuswort (PSW) gesetzt., andernfalls gelöscht

Operation: CPL a, <mem>

wenn (A) < mem dann (PSW:Z) ← 1 andernfalls (PSW:Z) ← 0

Beispiel:

```
.org 100
db 10       ; Speicherstelle 100 mit Wert 10 beschreiben
.
.
mvi a,10
cpg a,100
jl 20       ; mit Adresse 20 bei Gleichheit fortfahren

; hier fortfahren, wenn Akku ungleich 10
```

Das Programm wird NICHT an Speicherstelle 20 fortgeführt, da der Akkuinhalt nicht kleiner als der Speicherinhalt ist

DB <const>

Funktion: define a byte

Opcode: 00

Beschreibung: speichert an der aktuellen Speicherstelle ein einzelnes Datum

Operation: DB <const>

<addr>) ← <const>

Beispiel:

```
.org 100
db 170
```

```
; an der Speicherstelle 100 wird der Wert 170 abgelegt
```

DEC a

Funktion: decrement a

Opcode: 27

Beschreibung: vermindere den Akkuinhalt um 1

Operation: DEC a

(A) ← (A) - 1

Beispiel:
```
mvi a, 10
dec a
; Akkuinhalt ist nun 9
```

DJNZ a

Funktion: decrement accu, jump if not zero

Opcode: 25

Beschreibung: vermindere Akku um 1 und springe an die angegebene Adresse, wenn der Akku nicht zu Null geworden ist

Operation: DJNZ a, <addr>

(A) ← (A) - 1
wenn (A) > 0 dann (PC) ← <addr> andernfalls (PC) ← (PC) + 1

Beispiel:
```
.org 0
mvi a, 10
.org 1
cdis
cdel 250    ; 1/4 Sekunde warten
djnz a, 1   ; zaehlt 10 bis 1 herunter
cdis        ; und zeigt hier 0 an
hlt
```

HLT

Funktion: halt

Opcode: 01

Beschreibung: Stopt das Programm und kehrt je nach Programmaufruf zur Tasteneingabe oder zum Terminal zurück

INC a

Funktion: increment a

Opcode: 26

Beschreibung: erhöhe den Akkuinhalt um 1

Operation: INC a

(A) ← (A) + 1

Beispiel:
```
mvi a, 10
inc a
; Akkuinhalt ist 11
```

IN p1

Funktion: input a byte from port 1 to accu

Opcode: 16.00x (gültige Werte für x: 1..8)

Beschreibung: liest ein 8-Bit Datum von Port 1 in den Akku ein

Operation: IN p1

(A) ← (P1)

Beispiel:
```
in p1
; Akku hat den eingelesenen Wert von Port 1
```

INB p1, <bit>

Funktion: input a bit from Port 1 to accu

Opcode: 16.000

Beschreibung: Setzt den Akku auf 1, wenn das angegebene Bit am Port 1= 1 ist andernfalls wird Akku zu 0

Operation: INB p1, <bit>

wenn (P1:bit) = dann (A) ← 1 andernfalls (A) ← **0**

Beispiel:
```
inb p1, 3
; Akku hat den Wert des von Port 1 eingelesenen dritten
; Bits
```

INT <const>

Funktion: interrupt program by calling a system function

Opcode: 63

Beschreibung: unterbricht den Programmauflauf und führt eine Systemfunktion durch (siehe Softwareinterrupts). Ein Softwareinterrupt besteht aus der Interruptfunktion, die in <const> angegeben ist sowie weiteren Unterfunktionen, die in einem Register angewählt wird.

Operation: INT <const>

Beispiel:
```
mvi a,25     ; A = 25
mvi b,3      ; Funktion 3
int 1        ; des Interrupts 1 aufrufen
             ; Funktion 3 wählt Zeitbasis 100 ms
             ; Programm wird um 25 * 100 ms = 2,5 Sek.
             ; verzoegert
```

JC <addr>

Funktion: jump on carry flag

Opcode: 53

Beschreibung: führe das Programm Adresse addr fort, wenn das Carry-Flag im Programmstatusword (PSW) gesetzt ist.

Instruktionen, die das Carry-Flag beeinflußen siehe Tabelle 1

Operation: JC <addr>

wenn (PSW:C) = 1 dann (PC) ← <addr> andernfalls (PC) ← (PC) + 1

Beispiel:

```
mvi a, 254
inc a         ; Akku = 255, Carry = 0
jc 60         ; kein Sprung erfolgt
inc a         ; Ueberlauf, Ergebnis = 256, passt nicht
              ; in den Akku: A= 0, Carry = 1
jc 60         ; Sprung an Adresse 60
.
.org 60
; Programm hier bei Operationsueberlauf fortfuehren
```

Das Programm wird nach dem zweiten JC - Befehl an Adresse 60 fortgeführt

JMP <addr>

Funktion: jump unconditional

Opcode: 09

Beschreibung: führe das Programm ohne eine Bedingung an Adresse addr fort

Operation: JMP <addr>

(PC) ← <addr>

Beispiel:

```
jmp 20      ; mit Adresse fortfahren
..
.org 20
; mit Programmausführung hier fortfahren
```

JMP <@memr>

Funktion: jump indirect unconditional

Opcode: 21

Beschreibung: führe das Programm ohne Bedingung an der Stelle fort, deren Adresse in der Speicherstelle mem steht.

Operation: JMP <@mem>

PC ← Speicherzelle [Speicherzelle [mem]]

Beispiel:

```
.org 100          ; Speicherzelle 100 waehlen
db 40             ; Sprungadresse in Zelle 100
.
.org 10
.
jmp @100          ; Springe an die Adresse in Zelle 100
.
.org 40
; Hier wird das Programm fortgefuehrt
```

JPG <addr>

Funktion: jump on greater flag

Opcode: 52

Beschreibung: führe das Programm Adresse addr fort, wenn das Greater-Flag im Programmstatusword (PSW) gesetzt ist.

Das Greater-Flag wird gesetzt, wenn das Ergebnis eines Vergleichs der Instruktionen CPI oder CMP festgestellt hat, dass der Akkuinhalt größer als der Vergleichswert ist.

Operation: JPG <addr>

wenn (PSW:G) = 1 dann (PC) ← <addr> andernfalls (PC) ← (PC) + 1

Beispiel:

```
mvi a, 34
cpi a, 19
jpg 50      ; mit Adresse 50 fortfahren wenn A > 19 ist
; Programm hier fortfuehren, wenn A nicht groesser 19 ist
..
.org 50
; Programm hier fortfuehren, wenn A > 19 ist
```

Das Programm wird an Speicherstelle 50 fortgeführt

JPL <addr>

Funktion: jump on less flag

Opcode: 51

Beschreibung: führe das Programm Adresse addr fort, wenn das Less-Flag im Programmstatusword (PSW) gesetzt ist.

Das Less-Flag wird gesetzt, wenn das Ergebnis eines Vergleichs der Instruktionen CPI oder CMP festgestellt hat, dass der Akkuinhalt kleiner als der Vergleichswert ist.

Operation: JPL <addr>

wenn (PSW:L) = 1 dann (PC) ← <addr> andernfalls (PC) ← (PC) + 1

Beispiel:

```
mvi a, 15
cpi a, 19
jpl 50      ; mit Adresse 50 fortfahren wenn A < 19 ist
; Programm hier fortfuehren, wenn A nicht kleiner 19 ist
..
.org 50
; Programm hier fortfuehren, wenn A kleiner 19 ist
```

Das Programm wird an Speicherstelle 50 fortgeführt

JZ <addr>

Funktion: jump on zero flag

Opcode: 11

Beschreibung: führe das Programm Adresse addr fort, wenn das Zero-Flag im Programmstatusword (PSW) gesetzt ist.

Das Zero Flag wird gesetzt, wenn das Ergebnis eines Vergleichs der Instruktionen CPE, CPG, CPL "wahr" ist. Das Zero Flag wird zudem gesetzt, wenn ein Vergleich mittels CPI oder CMP die Gleichheit festgestellt hat.

Operation: JZ <addr>

wenn (PSW:Z) dann (PC) ← <addr> andernfalls (PC) ← (PC) + 1

Beispiel:

```
mvi a, 19
cpi a, 19
jz 50      ; mit Adresse 50 bei Gleichheit fortfahren
; Programm bei Ungleichheit hier fortfuehren
..
.org 50
; mit Programmausführung bei Gleichheit hier fortfahren
```

MOV a, <mem>

Funktion: move a value to akku from memory

Opcode: 05

Beschreibung: beschreibe den Akku mit dem Speicherinhalt von mem

Operation: MOV a,<mem>

(A) ← Speicherzelle [mem]

Beispiel:

```
.org 110    ; Speicherstelle 110 auswaehlen
db 84       ; Zahlenwert in den Speicher schreiben
.
.
mov a,110

;der Akku hat den Wert 84
```

MOV a, @<mem>

Funktion: move a value to akku indirect form memory

Opcode: 19

Beschreibung: beschreibe den Akku indirekt mit dem Speicherinhalt, der in mem angegeben ist. mem ist somit ein Zeiger auf einen anderen Speicherinhalt.

Operation: MOV a, @<mem>

(A) ← Speicherzelle [Speicherzelle [mem]]

Beispiel:

```
.org 50
db 84      ; Speicherzelle 50 mit Wert 84 beschreiben
.org 100
db 50      ; Speicherstelle 100 mit Wert 50 beschreiben
.
.
mov a,@100
```

der Akku hat den Inhalt 84, weil in der Speicherstelle 100 der Wert 50 steht und dieses die Adresse ist unter dem der gewünschte Wert gespeichert ist.

MOV @<mem>, a

Funktion: move indirect a value to memory from akku

Opcode: 20

Beschreibung: beschreibe indirekt den Speicherinhalt, dessen Speicherort in mem angegeben ist mit dem Inhalt des Akkus

Operation: MOV @<mem>, a

(A) → Speicherzelle [Speicherzelle [mem]]

Beispiel:

```
.org 100
db 50       ; Speicherstelle 100 mit Wert 50 beschreiben
.
.
mvi a,12    ; A = 12
mov a,@100  ; und indirekt in den Speicher schreiben
```

Inhalt der Speicherstelle 50 hat den Wert 12 (weil in der Speicherstelle 100 der Wert 50 steht und dieses die Adresse ist unter dem der Akkuinhalt abgelegt wurde)

MOV <mem>, a

Funktion: move a value to memory from akku

Opcode: 06

Beschreibung: beschreibe den Speicherinhalt von mem mit dem Akkuinhalt

Operation: MOV <mem>, a

(A) → Speicherzelle [mem]

Beispiel:

```
mvi a,20
mov 110,a

; der Inhalt von Speicherzelle 110 ist nun 20
```

MOV a, reg

Funktion: move a value from register to accu

Opcode: 32 : mov a, b 33 : mov a, c 34 : mov a, d 35 : mov a, e

Beschreibung: kopiere den Registerinhalt in den Akku

Operation: MOV a, reg

(A) ← (reg)

Beispiel:

```
mvi e, 100
mov a, e
; Akku hat nun den Inhalt 100
```

MOV reg, a

Funktion: move a value from accu to register

Opcode: 36 : mov b, a 37 : mov c, a 38 : mov d, a 39 : mov e, a

Beschreibung: kopiere den Akkuinhalt in ein Register

Operation: MOV reg, a

(reg) ← (A)

Beispiel:

```
mvi a, 102
mov e, a
; Register E hat nun den Inhalt 102
```

MOV reg, reg

Funktion: move the value in a register to another register

Opcode: 40 : mov b, c 41 : mov b, d 42 : mov b, e

43 : mov c, b 44 : mov c, d 45 : mov c, e

Beschreibung: kopiere den Inhalt eines Registers in ein anderes Register

Hinweis: Das Kopieren eines Registers in eines der Register D oder E ist NICHT möglich. Somit fungieren die Register D und E hautpsächlich als "Datenrettungsregister" für den Akku

Operation: MOV reg, reg

(reg x) ← (reg x)

Beispiel:

```
mvi e, 103
mov b, e
; Register B hat nun den Inhalt 103
```

MUL a, b

Funktion: multilicate accu with register B

Opcode: 62

Beschreibung: multiplziere den Akku mit dem Inhalt von Register B. das Ergebnis größer als 255 wird das Carry-Flag im Programmstatuswort (PSW) gesetzt (und signalisiert somit ein Überlaufen - overflow - des Akkus)

Operation: MUL a, b

(A) ← A * <reg B>

wenn (A) * <reg B> > 255 dann PSW:C = 1; (A) ← ((A) * <reg B>) - 256

Beispiel:

```
mvi a,90     ; A = 90
mvi b,2      ; B = 2
mul a,b      ; A = 180, Carry = 0
mul a,b      ; A = 104, Carry = 1

; der Akku hat den Wert 104, das Carry-Flag ist gesetzt
```

MVI a, <const>

Funktion: move a value immediately to akku from const

Opcode: 04

Beschreibung: beschreibe den Akku mit dem Wert in const

Operation: MVI a, <const>

(A) ← const

Beispiel:

```
mvi a,23

; der Akku hat den Wert 23
```

MVI reg, <const>

Funktion: move a value immediately to register from const

Opcode: 28 : mvi b, const 29 : mvi c, const 30 : mvi d, const 31 : mvi e, const

Beschreibung: beschreibe ein Register mit dem Wert in const

Operation: MVI reg, const

(reg) ← <const>

Beispiel:

```
mvi b, 100
; Register B hat nun den Inhalt 100
```

NOT a

Funktion: invert accu

Opcode: 46

Beschreibung: invertiere alle Bits des Akkus. D.h. aus jeder 1 wird eine 0, aus jeder 0 wird eine 1. Da der CP1(+) im Vergleich mit anderen Computersystemen Zahlen dezimal darstellt ist die Invertierung nicht sofort ersichtlich. Der max. mit 8 Bit größte darstellbare Zahl ist $2^8 = 255$ entspricht binär = 1111.1111. Werden bei einer dezimal dargestellten Zahl alle Bits invertiert, so ist das Ergebnis die Differenz der ursprünglichen Zahl bis 255.

Beispiel:

```
 73 dezimal = 0100.1001 binär

    wird nach Invertierung

182 dezimal = 1011.0110 binär
```

73 + 182 = 255

Operation: NOT a

(A) ← ¬ (A)

Beispiel:

```
mvi a, 73                   ; 0100.1001
not a
; Akkuinhalt ist nun 182    ; 1011.0110
```

NOTB a

Funktion: not the first bit in a

Opcode: 14

Beschreibung: invertiert das erste Bit im Akku (nur dieses, Zählung beginnt bei 0 und nicht bei 1)

Operation: NOTB a

(A:1) ← ¬ (A:1)

Beispiel:

```
mvi a,1
notb a    ; Bit 1 wird invertiert und ist jetzt 0
notb a    ; Bit 1 wird invertiert und ist jetzt wieder 1
```

OR a, <mem>

Funktion: logical OR accu with memory value

Opcode: 58

Beschreibung: führt eine logische ODER Verknüpfung (OR) aller Bits des Akkuinhalts mit den Bits des Inhaltes der angegebenen Speicherstelle durch. Eine ODER Verknüpfung bedeutet: Wenn eines der beiden beteiligten Bits eine logische 1 enthält, wird das Ergebnis zu 1 wird, ist dem nicht so, ist das Ergebnis 0.

Eine logische ODER - Verknüpfung wird u.a. benutzt, um gezielt einzelne Bits eines Datums löschen zu können

Beispiel:

```
          0000.1001 binär   (  9 dezimal )
   and    0100.0000 binär   ( 64 dezimal )
-----------------------------------------
Ergebnis: 0100.1001 binär   ( 73 dezimal )
```

Operation: OR a, <mem>

(A) ← (A) | <mem>

Beispiel:
```
mvi a, 64                   ; 0100.0000
mov 100, a
mvi a, 9                    ; 0000.1001
and a, 100
; Akkuinhalt ist nun 73     ; 0100.1001
```

ORI a, <const>

Funktion: immediately logical OR accu with memory value

Opcode: 59

Beschreibung: führt eine logische ODER Verknüpfung (OR) aller Bits des Akkuinhalts mit den Bits der angegebenen Konstante durch. Eine ODER Verknüpfung bedeutet: Wenn eines der beiden beteiligten Bits eine logische 1 enthält, wird das Ergebnis zu 1 wird, ist dem nicht so, ist das Ergebnis 0.

Eine logische ODER - Verknüpfung wird u.a. benutzt, um gezielt einzelne Bits eines Datums löschen zu können

Beispiel:
```
          0000.1001 binär   (  9 dezimal )
   and    0100.0000 binär   ( 64 dezimal )
-----------------------------------------
Ergebnis: 0100.1001 binär   ( 73 dezimal )
```

Operation: ORI a, <const>

(A) ← (A) | <const>

Beispiel:
```
mvi a, 9                    ; 0000.1001
ori a, 64                   ; 0100.0000
; Akkuinhalt ist nun 73     ; 0100.1001
```

OUT p1

Funktion: output from accu to port 1

Opcode: 17.000

Beschreibung: setzt Port 1 mit dem Inhalt des Akkus

Operation: OUT p1

(P1) ← (A)

Beispiel:
```
mvi a,170
out p1        ; der Wert 170 wurde auf Port 1 ausgegeben
```

OUTB p1, <bit>

Funktion: output the first bit in accu to port 1

Opcode: 17.00x (gültige Werte für x: 1..8)

Beschreibung: setzt / löscht ein Bit in Port 1 in Abhängigkeit des ersten Akkubits

Operation: OUTB p1,<bit>

(P1:bit) ← (A:1)

Beispiel:

```
mvi a,1
outb p1,6    ; setzt das 6. Bit in Port 1
cdel 250     ; warten
mvi a,0
outb p1,6    ; löscht das 6. Bit in Port 1
```

OUT p2

Funktion: output from accu to port 2

Opcode: 18.000

Beschreibung: setzt Port 2 mit dem Inhalt des Akkus

Operation: OUT p2

(P2) ← (A)

Beispiel:

```
mvi a,85
out p2       ; der Wert 85 wurde auf Port 2 ausgegeben
```

OUTB p2, <bit>

Funktion: output the first bit in accu to port 2

Opcode: 18.00x (gültige Werte für x: 1..8)

Beschreibung: setzt / löscht ein Bit in Port 2 in Abhängigkeit des ersten Akkubits

Operation: OUTB p2,<bit>

(P2:bit) ← (A:1)

Beispiel:

```
mvi a,1
outb p2,6    ; setzt das 6. Bit in Port 2
cdel 250     ; warten
mvi a,0
outb p2,6    ; löscht das 6. Bit in Port 2
```

RET

Funktion: return from subroutine

Opcode: 65

Beschreibung: Rückkehr vom Unterprogramm zum aufrufenden Programm

Operation: ret

(SP) ← (SP) - 1
(PC) ← (SP [inhalt])

Beispiel:

```
; Blinkprogramm
.org 0
mvi a,1
out p2
mvi a,10
call 15
mvi a,0
out p2
mvi a, 10
call 15
jmp 0

; Verzoegerungsunterprogramm: warte A * 0,1 s
```

```
.org 15
mov e,a     ; A zwischenspeichern
cdel 100    ; Verzoegerung 0,1 s
djnz 16     ; Anzahl (A) mal wiederholen
mov a,e     ; A restaurieren
ret         ; zurueck zum aufrufenden Programm
```

SBI a, <mem>

Funktion: subtract immediately const from akku

Opcode: 61

Beschreibung: subtrahiere vom Akku die Konstante const. Ist das Ergebnis kleiner als 0 wird das Carry-Flag im Programmstatuswort (PSW) gesetzt (und signalisiert somit ein Unterlaufen - underflow - des Akkus)

Operation: SUB a, <mem>

(A) ← A - <const>

wenn (A) - <const> < 0 dann PSW:C = 1; (A) ← (256 + (A)) - <const>

Beispiel:

```
mvi a,12    ; A = 12
sbi a,10    ; den Wert 10 subtrahieren

; der Akku hat den Wert 2, das Carry-Flag ist nicht
; gesetzt.

sbi a,10    ; den Wert 10 subtrahieren

; der Akku hat den Wert 248, das Carry-Flag ist gesetzt
```

SLC a

Funktion: shift accu left through carry

Opcode: 54

Beschreibung: verschiebe jedes einzelne Bit des Akkus um eine Position nach links. Bit-Nummer 8 wird in das Carry-Flag transferiert. Dezimal entspricht eine Schiebeoperation nach links einer Multiplikation * 2.

Beispiel:

```
      Carry       Akku
        0      0100.1001 binär   (  73 dezimal )
 SLC a
        0      1001.0010 binär   ( 146 dezimal )
 SLC a
        1      0010.0100 binär   (  36 dezimal )
```

Operation: SLC a

(A:n + 1) ← (A:n) = 1 .. 7
(carry) ← (A:8)

Beispiel:

```
.org 0
mvi a,1
out p2              ; Bitmuster auf P2 anzeigen
cdel 250            ; warten
slc a               ; alle Bits verschieben
jc 0                ; bei Überlauf von vorne anfangen
jmp 1               ; ansonsten neues schieben
```

Programm stellt ein Lauflicht nach links dar

SRC a

Funktion: shift accu right through carry

Opcode: 55

Beschreibung: verschiebe jedes einzelne Bit des Akkus um eine Position nach rechts. Bit-Nummer 1 wird in das Carry-Flag transferiert. Dezimal entspricht eine Schiebeoperation nach rechts einer Division / 2.

Beispiel:

```
      Carry       Akku
        0      1001.0010 binär   ( 146 dezimal )
 SRC a
        0      0100.1001 binär   (  73 dezimal )
 SRC a
        1      0010.0100 binär   (  36 dezimal )
```

Operation: SRC a

(A:n) ← (A:n+1) = 1 .. 7
(carry) ← (A:1)

Beispiel:

```
.org 0
mvi a,128           ; = 0x80 = 1000.000 binaer
out p2              ; Bitmuster auf P2 anzeigen
cdel 250            ; warten
src a               ; alle Bits verschieben
jc 0                ; bei Überlauf von vorne anfangen
jmp 1               ; ansonsten neues schieben
```

Programm stellt ein Lauflicht nach rechts dar

SUB a, <mem>

Funktion: subtract memory value from accu

Opcode: 08

Beschreibung: subtrahiere vom Akku den Inhalt der in mem angegebenen Speicherstelle. Ist das Ergebnis kleiner als 0 wird das Carry-Flag im Programmstatuswort (PSW) gesetzt (und signalisiert somit ein Unterlaufen - underflow - des Akkus)

Operation: SUB a, <mem>

(A) ← A - Speicherzelle [mem]]

wenn (A) - <mem> < 0 dann PSW:C = 1; (A) ← (256 + (A)) - <mem>

Beispiel:

```
.org 100
db 10       ; Speicherstelle 100 mit Wert 10 beschreiben
.
.
mvi a,12    ; A = 12
sub a,100   ; den Wert aus Zelle 100 subtrahieren

; der Akku hat nun den Wert 2, das Carry-Flag ist nicht
; gesetzt.
sub a,100   ; den Wert aus Zelle 100 subtrahieren
```

```
; der Akku hat nun den Wert 248 und das Carry-Flag ist
; gesetzt.
```

XOR a, <mem>

Funktion: logical exclusive or accu with memory value

Opcode: 47

Beschreibung: führt eine logische Exklusiv-Oder Verknüpfung (XOR) aller Bits des Akkuinhalts mit den Bits des Inhaltes der angegebenen Speicherstelle durch. Eine Exklusiv-Oder Verknüpfung bedeutet: Nur eine der beiden beteiligten Bits darf eine logische 1 enthalten, damit das Ergebnis zu 1 wird, ist dem nicht so, ist das Ergebnis 0.

Beispiel:

```
          0100.1001 binär   ( 73 dezimal )
   xor    0000.1111 binär   ( 15 dezimal )
-----------------------------------------
Ergebnis: 0100.0110 binär   ( 70 dezimal )
```

Operation: XOR a, <mem>

(A) ← (A) ^ <mem>

Beispiel:

```
mvi a, 73                   ; 0100.1001
mov 100, a
mvi a, 15                   ; 0000.1111
xor a, 100
; Akkuinhalt ist nun 70     ; 0100.0110
```

XRI a, <const>

Funktion: immediately logical exclusive or accu with const

Opcode: 48

Beschreibung: führt eine logische Exklusiv-Oder Verknüpfung (XOR) aller Bits des Akkuinhalts mit den Bits einer Konstanten durch. Eine Exklusiv-Oder Verknüpfung bedeutet: Nur eine der beiden beteiligten Bits darf eine logische 1 enthalten, damit das Ergebnis zu 1 wird, ist dem nicht so, ist das Ergebnis 0.

Beispiel:

```
          0100.1001 binär   ( 73 dezimal )
   xor    0000.1111 binär   ( 15 dezimal )
-----------------------------------------
Ergebnis: 0100.0110 binär   ( 70 dezimal )
```

Operation: XRI a, <const>

(A) ← (A) ^ <const>

Beispiel:

```
mvi a, 15                   ; 0000.1111
xri a, 73                   ; 0100.1001
; Akkuinhalt ist nun 70     ; 0100.0110
```

.ORG <const>

Funktion: set origin memory adress

Opcode: keiner

Beschreibung: Pseudo - Mnemonic (ohne Opcode). Setzt im Terminalprogramm die

Ursprungsadresse an der ein Befehl abgelegt wird.

Operation: .ORG <const>

<addr> ← <const>

Beispiel:

```
.org 0
; nachfolgende Befehle werden ab der Adresse 0 und
; folgende gespeichert
mvi a,1
```