Technische Universität Chemnitz
Fakultät für Informatik
Professur Technische Informatik
Dr.-Ing. B. Naumann

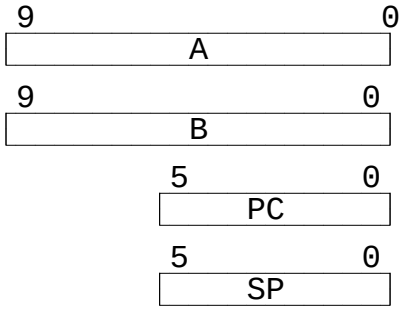
01.07.2002

LC1 - Befehlsbeschreibung

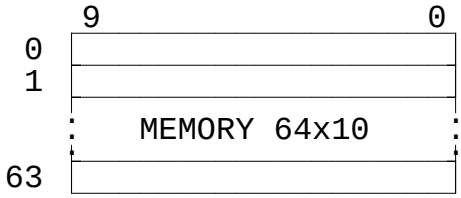
Inhaltsverzeichnis

1. Programmiermodell

Register



Speicher



Flags

SF	OF

2. Befehle

Neben den Komponenten des Programmiermodells werden die folgenden Bezeichnungen verwendet:

adr ($0 \leq$ **adr** ≤ 63) für Speicheradresse
n ($0 \leq$ **n** ≤ 63) für Rotationsweite

Die **Flags** werden von den Befehlen unterschiedlich beeinflußt:

- Das Flag bleibt unverändert
A Das Flag wird beeinflußt (affected)

Das **Sign Flag SF** ist das höchstwertige Bit des Registers A (SF = A_9). Das **Overflow Flag OF** wird nur durch die Befehle ADD und SUB nach den dort geltenden Regeln beeinflußt.

X bedeutet, daß das entsprechende Bit beliebig belegt sein kann.

LDA adr — Load Register A

Kodierung:	Operation:	Flags:
0000 | adr	PC := PC + 1 A := MEMORY(adr)	SF | A OF | -

LDB adr — Load Register B

Kodierung:	Operation:	Flags:
0001 | adr	PC := PC + 1 B := MEMORY(adr)	SF | - OF | -

MOV adr — Move Register A

Kodierung:	Operation:	Flags:
0010 | adr	PC := PC + 1 MEMORY(adr) := A	SF | - OF | -

MAB — Move Register A to Register B

Kodierung:	Operation:	Flags:
0011 | XXXXXX	PC := PC + 1 B := A	SF | - OF | -

ADD — Add Register B to Register A

Kodierung:	Operation:	Flags:
0100 | XXXXXX	PC := PC + 1 A := A + B	SF | A OF | A

SUB — Subtract Register B from Register A

Kodierung:	Operation:	Flags:
0101 | XXXXXX	PC := PC + 1 A := A - B	SF | A OF | A

AND — Logical AND

Kodierung:	Operation:	Flags:
0110 | XXXXXX	PC := PC + 1 A := A $\wedge$ B; bitweise	SF | A OF | -

NOT — Logical NOT

Kodierung:	Operation:	Flags:
0111 | XXXXXX	PC := PC + 1 A := /A; bitweise	SF | A OF | -

JMP adr — Jump

Kodierung:	Operation:	Flags:
1000 | adr	PC := PC + 1 PC := adr	SF | - OF | -

JPS adr — Jump if Sign

Kodierung:	Operation:	Flags:
1001 | adr	PC := PC + 1 if SF = 1 then PC := adr	SF | - OF | -

JPO adr — Jump if Overflow

Kodierung:	Operation:	Flags:
1010 | adr	PC := PC + 1 if OF = 1 then PC := adr	SF | - OF | -

CAL adr — Call Procedure

Kodierung:	Operation:	Flags:
1011 | adr	PC := PC + 1 MEMORY(SP) := PC SP := SP - 1 PC := PC + 1 PC := adr	SF | - OF | -

RET — Return from Procedure

Kodierung:	Operation:	Flags:
1100 | XXXXXX	PC := PC + 1 SP := SP + 1 PC := MEMORY(SP)	SF | - OF | -

RRA n	Rotate Register A Right by n Bits	
Kodierung:	Operation:	Flags:
1101 | n	PC := PC + 1 A := A0,A9...A1; n-mal A> 9 8 7 6 5 4 3 2 1 0	SF | A OF | -

RLA n	Rotate Register A Left by n Bits	
Kodierung:	Operation:	Flags:
1110 | n	PC := PC + 1 A := A8...A0,A9; n-mal 9 8 7 6 5 4 3 2 1 0 <A	SF | A OF | -

HLT		Halt
Kodierung:	Operation:	Flags:
1111 | XXXXXX	stoppt den Prozessor	SF | - OF | -

3. Assemblerdirektive

Es werden die folgenden Bezeichnungen verwendet:

label (max. 8 Zeichen, mit einem Buchstaben beginnend)
const (-512 ≤ const ≤ 511) für Konstante

[label:] DEF const	Define
Operation: MEMORY(label) := const	

4. Beispielprogramm

Anm.: In den Programmen sollten anstelle der numerischen Speicheradressen (**adr**) stets symbolische Speicheradressen (**label**) verwendet werden.

Multiplikation zweier ganzer Zahlen

```
     WEITER:   LDA FAKTOR_1   ; A := <FAKTOR_1>
               LDB ZAEHLER    ; B := <ZAEHLER>
               SUB            ; A := A - B
               JPS FERTIG     ; IF A < 0 THEN GOTO FERTIG
               LDA PRODUKT    ; A := <PRODUKT>
               LDB FAKTOR_2   ; B := <FAKTOR_2>
               ADD            ; A := A + B
               MOV PRODUKT    ; <PRODUKT> := A
               LDA ZAEHLER    ; A := <ZAEHLER>
               LDB EINS       ; B := 1
               ADD            ; A := A + B
               MOV ZAEHLER    ; <ZAEHLER> := A
               JMP WEITER     ; GOTO WEITER
     FERTIG:   LDA PRODUKT    ; A := <PRODUKT>
               HLT            ; STOP
     FAKTOR_1: DEF 5          ; FAKTOR 1
     FAKTOR_2: DEF -2         ; FAKTOR 2
     PRODUKT:  DEF 0          ; PRODUKT
     ZAEHLER:  DEF 1          ; ZAEHLER
     EINS:     DEF 1          ; KONSTANTE 1
```