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

11.05.1998

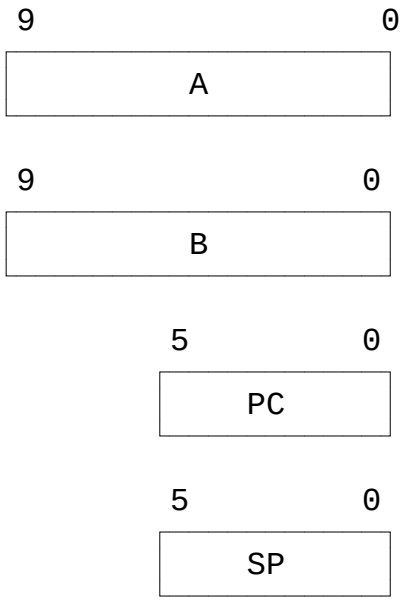
LC1 - Befehlsbeschreibung

Inhaltsverzeichnis

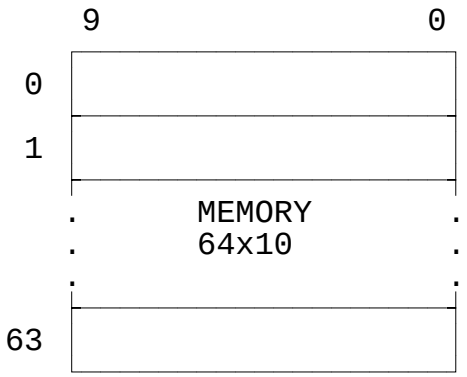
1. Programmiermodell 1
2. Befehle 1
3. Assemblerdirektive 4
4. Beispielprogramm 4

1. Programmiermodell

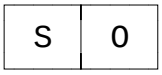
Register



Speicher



Flags



2. Befehle

Neben den Komponenten des Programmiermodells werden die folgenden Bezeichnungen verwendet:

Speicheradresse **adr** (0 ≤ **adr** ≤ 63)
Rotationsweite **n** (0 ≤ **n** ≤ 63)

Die Flags werden von den Befehlen unterschiedlich beeinflußt:

- Das Flag wird nicht beeinflußt
X Das Flag wird beeinflußt

In der Kodierung bedeutet **X**, daß das entsprechende Bit beliebig belegt sein kann.

LDA adr — Load Register A

Operation:
```
PC := PC + 1
A  := MEMORY(adr)
```

Flags:

S	O
X	-

Kodierung:

0000	adr

LDB adr — Load Register B

Operation:
```
PC := PC + 1
B  := MEMORY(adr)
```

Flags:

S	O
-	-

Kodierung:

0001	adr

MOV adr — Move Register A

Operation:
```
PC := PC + 1
MEMORY(adr) := A
```

Flags:

S	O
-	-

Kodierung:

0010	adr

MAB — Move Register A to Register B

Operation:
```
PC := PC + 1
B  := A
```

Flags:

S	O
-	-

Kodierung:

0011	XXXXXX

ADD — Add Register B to Register A

Operation:
```
PC := PC + 1
A  := A + B
```

Flags:

S	O
X	X

Kodierung:

0100	XXXXXX

SUB — Subtract Register B from Register A

Operation:
```
PC := PC + 1
A  := A - B
```

Flags:

S	O
X	X

Kodierung:

0101	XXXXXX

AND — Logical AND

Operation:
```
PC := PC + 1
A  := A & B
```

Flags:

S	O
X	-

Kodierung:

0110	XXXXXX

NOT — Logical NOT

Operation:
```
PC := PC + 1
A  := /A
```

Flags:

S	O
X	-

Kodierung:

0111	XXXXXX

JMP adr — Jump

Operation:
```
PC := adr
```

Flags:

S	O
-	-

Kodierung:

1000	adr

JPS adr — Jump if Sign

Operation:
```
PC := PC + 1
if S = 1 then PC := adr
```

Flags:

S	O
-	-

Kodierung:

1001	adr

JPO adr — Jump if Overflow

Operation:
```
PC := PC + 1
if O = 1 then PC := adr
```

Flags:

S	O
-	-

Kodierung:

1010	adr

CAL adr — Call Procedure

Operation:
```
PC := PC + 1
MEMORY(SP) := PC
SP := SP - 1
PC := adr
```

Flags:

S	O
-	-

Kodierung:

1011	adr

RET — Return from Procedure

Operation:
```
SP := SP + 1
PC := MEMORY(SP)
```

Flags:

S	O
-	-

Kodierung:

1100	XXXXXX

RRA n — Rotate Register A Right by n bits

Operation:
```
PC := PC + 1
(n mod 10)-mal:
```

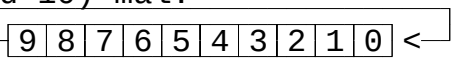
A ⟲ > | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Flags:

S	O
X	-

Kodierung:

1101	n

RLA n	Rotate Register A Left by n bits
Operation:	PC := PC + 1 (n mod 10)-mal: A 
Flags:	S O X -
Kodierung:	1110 n

HLT	Halt
Operation:	stoppt den Prozessor
Flags:	S O - -
Kodierung:	1111 XXXXXX

3. Assemblerdirektive

Es werden die folgenden Bezeichnungen verwendet:

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

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

4. Beispielprogramm

Anm.: Anstelle der numerischen Speicheradressen (**adr**) sollten 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
```