

# Zusammenfassung der Assemblerbefehle des 8051

Befehl	Bezeichnung	Syntax	Wirkung / Beispiel
<b>Befehle zum Datentransfer</b>			
MOV	Move	<b>MOV [Ziel],[Quelle]</b>	MOV P1,P3 Kopiert den Inhalt von P3 nach P1
SWAP	Swap	<b>SWAP A</b>	SWAP A vertauscht die Bits 0 bis 3 mit den Bit 4-7 im A-Register
PUSH	Push	<b>PUSH [Byte in RAM]</b>	PUSH legt ein Byte im Stack ab ----- PUSH Acc PUSH 0 PUSH 1 ----- > A, R0 und R1 sichern
POP	Pop	<b>POP [Byte im RAM]</b>	Pop liest ein Byte vom Stack zurück ----- POP 1 POP 0 POP Acc ----- > A, R0 und R1 sichern
<b>Unbedingte Sprünge</b>			
AJMP	Absolute Jump	<b>AJMP [Sprungziel]</b>	Springt zu einem bestimmten Label ( 11 Bit-Adresse / Reichweite )
LJMP	Long Jump	<b>LJMP [Sprungziel]</b>	Springt zu einem bestimmten Label ( 16 Bit-Adresse / Reichweite )
<b>Bedingte Sprünge</b>			
JB	Jump if Bit is set	<b>JB [Bit],[Sprungziel]</b>	JB P1.0, Label1 Springt nach Label1, wenn P1.0 auf 1 steht
JNB	Jump if Bit is not set	<b>JNB [Bit],[Sprungziel]</b>	JB P1.0, Label1 Springt nach Label1, wenn P1.0 auf 0 steht
JBC	Jump if Bit is set and clear bit	<b>JBC [Bit],[Sprungziel]</b>	JB P1.0, Label1 Springt nach Label1, wenn P1.0 auf 1 steht und Löscht P1.0 (-> Ist schneller, als Löschbefehl)
JC	Jump if Carry is set	<b>JMP [Sprungziel]</b>	Springt zum Label, wenn das Carry Flag den Wert 1 hat

JNC	Jump if Carry is not set	<b>JNC [Sprungziel]</b>	Springt zum Label, wenn das Carry Flag den Wert 0 hat
JZ	Jump if A is zero	<b>JZ [Sprungziel]</b>	Springt zum Label, wenn das gesamte A-Register (alle Bits) 0 sind
JNZ	Jump if A is not zero	<b>JNZ [Sprungziel]</b>	Springt zum Label, wenn das gesamte A-Register (alle Bits) 1 sind
CJNE	Compare and Jump if not equal	<b>CJNE [Op1],[Op2],[Sprungziel]</b>	Vergleicht Operator 1 mit Operator 2 und verzweigt nach Label, falls die beiden Operatoren nicht übereinstimmen
DJNZ	Decrement and Jump if not zero	<b>DJNZ [Op],[Sprungziel]</b>	Zuerst wird der Operator verringert, ist der Operator danach nicht 0, so wird zum Label gesprungen.
JMP	Jump	<b>JMP [Sprungziel]</b>	Springt zum Sprungziel (Berechnet die Entfernung selbst – anders als AJMP/LJMP) Kann auch über Speicher genutzt werden (Sprungtabelle)

#### **Unterprogrammaufrufe**

ACALL	Absolute Call	<b>ACALL [Sprungziel]</b>	Ruft ein Unterprogramm auf und springt wieder zurück zur Aufrufstelle – der Stack wird automatisch gesetzt, es muss aber genug Platz frei sein.
LCALL	Long Call	<b>LCALL [Sprungziel]</b>	Ruft ein Unterprogramm auf und springt wieder zurück zur Aufrufstelle – der Stack wird automatisch gesetzt, es muss aber genug Platz frei sein.
CALL	Call a Sub Routine	<b>CALL [Sprungziel]</b>	Springt zum Unterprogramm – ermittelt automatisch die Entfernung

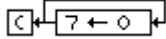

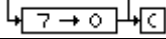
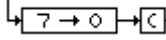
RET	Return from Sub Routine		Rücksprung zum Hauptprogramm, Rücksprungadresse wird vom Stack geholt und in de Programmzähler geschrieben
RETI	Return from Interrupt		Siehe RET – Unterschied: Rücksprung aus einer Interrupt-Routine, dabei werden die Interrupts wieder freigegeben

### Mathematik-Befehle

INC	Increment	<b>INC [Op]</b>	INC A erhöht das A-Register um 1
DEC	Decrement	<b>DEC [Op]</b>	DEC A vermindert das A-Register um 1
MUL	Multiplication	<b>MUL AB</b>	Multipliziert A mit B – die unteren 8 Bit (0-7) werden im A-Register gespeichert, die oberen 8 Bit im B-Register
DIV	Division	<b>DIV AB</b>	Dividiert A durch B – Ganzzahliges Ergebnis wird in A, der Rest in B gespeichert
ADD	Add	<b>ADD A,[Op]</b>	Addiert 2 Bytes, dabei ist der eine Summand auf das A-Register festgelegt. Der ursprünglich in A gespeicherte Wert geht verloren. Ist das Ergebnis größer als 256, so wird das Carry-Flag gesetzt und das A-Register hat den Wert, z.B. $260-256 = 4$
ADDC	Add with Carry	<b>ADDC A,[Op]</b>	Als zusätzlicher Input wird hier das Carry-Flag benutzt. $\{A+C+Op\}$
SUBB	Subtraction	<b>SUBB A,[Op]</b>	Ein Operator wird unter Berücksichtigung des Carry-Flag subtrahiert und das Ergebnis in A abgelegt $\{A-C-Op\}$

### Logische Operatoren

CLR	Clear	<b>CLR [Op]</b> ([Op]:C-Flag, A, Bit in RAM)	Setzt ein Bit bzw. das A-Register auf 0
SETB	Set Bit	<b>SETB [Op]</b> ([Op]:C-Flag, A, Bit in RAM)	Setzt ein Bit bzw. das A-Register auf 1
CPL	Complement Accumulator	<b>CPL [Op]</b> ([Op]:C-Flag, A, Bit in RAM)	Toggelt ein Bit, bzw. das A-Register

ANL	Logical And	<b>ANL [Ziel],[Quelle]</b>	Bildet das logische UND zweier Bits von [Quelle] und [Ziel]
ORL	Logical Or	<b>OR [Ziel],[Quelle]</b>	Bildet das logische ODER zweier Bits von [Quelle] und [Ziel]
XRL	Exclusive Or	<b>XRL [Ziel],[Quelle]</b>	Bildet das Exklusive Oder zweier Bits, d.h. das Erg. der Operation ist nur 1, wenn einer der beiden Input-Bits 1 ist.
RL	Rotate Left	<b>RL A</b>	Alle Bits werden um 1 Stelle nach Links verschoben, fällt das höherwertigste 7 Bit heraus, wird es an Stelle 0 wieder eingesetzt 
RLC	Rotate Left trough carry	<b>RLC A</b>	Alle Bits werden um 1 Stelle nach links verschoben, fällt das höherwertigste 7 Bit dadurch heraus, wird an Stelle 0 der Inhalt des C-Flags gesetzt. Das Bit das herausfällt, wird ebenfalls ins Carry-Flag geschrieben. 
RR	Rotate Right	<b>RR A</b>	Alle Bits werden um 1 Stelle nach rechts verschoben, fällt das niederwertigste 0 Bit heraus, wird es an Stelle 7 wieder eingesetzt 
RRC	Rotate Right trough carry	<b>RRC A</b>	Alle Bits werden um 1 Stelle nach rechts verschoben, fällt das niederwertigste 0 Bit dadurch heraus, wird an Stelle 7 der Inhalt des C-Flags gesetzt. Das Bit das herausfällt, wird ebenfalls ins Carry-Flag geschrieben. 
<b>Weitere Befehle:</b>			
NOP	No Operation	<b>NOP</b>	Dieser Befehl verbraucht genau einen Maschinenzklus.