

## Inhalt

Kapitel 0:	Vorwort, Was leistet JASMIN? für wen?
Kapitel 1:	Freeware, bitte lesen!!
Kapitel 2:	Grundlegendes über den Assembler JASMIN
Kapitel 3:	Übersicht über die Maschinensprache-Befehle
Kapitel 4:	Assembler-Direktiven

Kapitel 0	Was leistet JASMIN? für wen?
-----------	------------------------------

Der Name "JASMIN" steht für Jan ASseMbler INput. Das Programm wandelt einen Quelltext in 8088/8086-Maschinensprache in ein ausführbares Programm um. JASMIN übersetzt einen Quelltext direkt in eine COM-Datei und unterstützt keine OBJ- und EXE-Dateien.

JASMIN ist nicht so mächtig, wie man das heute von einem Assembler erwartet. Das war auch nicht mein Ziel. JASMIN habe ich für den Einsatz auf dem Atari Portfolio entwickelt und daher das Programm einfach und speicherplatzsparend gehalten. Die Bildschirmausgaben sind deswegen auch nicht länger als 40 Zeichen/Zeile. Genaugenommen ist JASMIN ein Maschinenspracheübersetzer, der die Mnemoniks (= symbolische Operationsschlüssel, z.B. MOV AX) in eine dem Computer verständliche Folge von Zeichen umsetzt. JASMIN unterstützt nicht den gewohnten Befehlsumfang eines Assemblers!

JASMIN eignet sich für all diejenigen, die unter MS-DOS nur begrenzt Speicherplatz auf Disketten, Ramcards, etc. zur Verfügung haben. Aber auch für Einsteiger in Maschinensprache ist JASMIN hervorragend geeignet, da der Programmierer alle CPU Befehle verwenden kann ohne ein umständliches Gerüst von Assemblerbefehlen in sein Programm einbinden zu müssen (CODE SEGMENT ASSUME CS:CODE,DS:DATEN ORG 100h START: ... CODE ENDS ...)

Kapitel 1	Freeware, bitte lesen!!
-----------	-------------------------

Ich möchte dieses Programm allen Interessierten kostenlos zur Verwendung überlassen und dazu ermutigen es auch an andere weiterzugeben, solange ALLE Dateien OHNE Änderungen kopiert werden !!

Seit dem 7.10.2006 ist fuer dieses Programm auch der Source-Code beigefuegt. Dieser darf in nicht-kommerziellen Programmen verwendet werden. Jegliche komerzielle Verwendung des Sourcecodes im Ganzen oder in Teilen bedarf der schriftlichen Einwilligung des Autors!

### Kommandozeilenaufruf

Sie rufen den Assembler von DOS aus wie folgt auf:

```
JASMIN [ [Quelldatei[.JAS]] [Zieldatei[.COM]] ]
```

Das heißt im Klartext: Sie können das Programm mit keinem, einem oder mit 2 Kommandozeilenparametern aufrufen. Wenn Sie ...

- ... keinen Parameter angeben, werden Sie nach den Namen für Quell- und Zieldatei gefragt. Wenn Sie die Frage nach der Zieldatei nur mit <Return> beantworten, so wird der Name der Quelldatei mit der Extension ".COM" verwendet.
- ... einen Parameter angeben, so wird dieser Dateiname für die Quelldatei verwendet, der Name der Zieldatei wird wie im letzten Fall ermittelt (selber Name + ".COM")
- ... zwei Parameter eingeben, wird der Erste als Quelldateiname, der Zweite als Zieldateiname verwendet.

Hinweis: Wenn Sie einen Dateinamen ohne Punkt eingeben, so wird beim Quelldateinamen grundsätzlich die Extension ".JAS" bzw beim Zieldateinamen die Extension ".COM" angehängt. Wenn Sie einen Dateinamen ohne Extension eingeben möchten, so muß dieser mit einem Punkt abgeschlossen sein!

### Bildschirmausgaben von JASMIN

```
JASMIN    --- Freeware ---   Version 1.6
8088/86 Jan laitenberger ASseMbler INput
max. size of COM file:      65535 bytes
```

Mit dieser Ausgabe werden Sie bei jeder Übersetzung begrüßt. Sie werden über die Größe des Speicherbereiches, den JASMIN für die zu übersetzende COM-Datei reservieren konnte, informiert. Wenn Sie keine Parameter beim Aufruf übergeben haben, folgen nun Fragen nach Quell- und Zieldateinamen. Danach wird die Assemblerdatei Zeile für Zeile übersetzt. Die aktuelle Zeile wird in eckigen Klammern angezeigt, während eine INCLUDE-Datei assembliert wird erscheint hinter den eckigen Klammern ein "I".

Tritt beim Übersetzungsvorgang ein Fehler auf, wird eine Fehlermeldung im Klartext ausgegeben, wenn sich die Datei "JASMIN.ERR" im Verzeichnis befindet. Ist diese Datei nicht vorhanden, so wird nur eine Fehlernummer ausgegeben. Nach der Fehlermeldung wird (zum leichteren Auffinden im Quelltext) die fehlerhafte Zeile angezeigt.

## Meine Philosophie

- Groß- und Kleinschreibung wird nicht unterschieden (auch nicht bei Adressmarken).
- Es ist unerheblich, wieviele Tabulatoren/Leerzeichen (Trennzeichen) in einer Quelltext-Zeile stehen, der Maschinenbefehl darf jedoch nicht durch Trennzeichen unterbrochen werden, auf einen Maschinenbefehl muß mindestens ein Trennzeichen folgen, wenn auf den Befehl Operanden folgen.
- Speicheroperanden müssen immer in eckigen Klammern eingefasst sein, sonst werden sie als Adressmarken oder Imm-Zahlen interpretiert.
- Adressmarken werden mit "@" eingeleitet, die Definition einer Marke wird mit einem Doppelpunkt abgeschlossen und darf keine Trennzeichen enthalten.

In diesem Kapitel werden Sie nicht lernen, wie man in Maschinensprache programmiert, dafür sollten Sie sich lieber ein gutes Buch kaufen. Ich möchte an dieser Stelle nur ansprechen, welche Befehle es gibt, was sie machen und mit welchen Operanden man sie aufruft. Der Befehlssatz der Intel 8088/86-CPU ist in Liste 1 zusammengestellt. Diese Befehle können in JASMIN-Quelltexten verwendet werden.

AAA	AAD	AAM	AAS	ADC	ADD	AND	CALL
CBW	CLC	CLD	CLI	CMC	CMP	CMPSB	CMPSW
CWD	DAA	DAS	DEC	DIV	HLT	IDIV	IMUL
IN	INC	INT	INTO	IRET	JA	JAE	JB
JBE	JCXZ	JE	JG	JGE	JL	JLE	JMP
JNA	JNAE	JNB	JNBE	JNE	JNG	JNGE	JNL
JNLE	JNO	JNP	JNS	JNZ	JO	JP	JPE
JPO	JS	JZ	LAHF	LDS	LEA	LES	LOCK
LODSB	LODSW	LOOP	LOOPE	LOOPNE	LOOPNZ	LOOPZ	MOV
MOVS	MOVSW	MUL	NEG	NOP	NOT	OR	OUT
POP	POPF	PUSH	PUSHF	RCL	RCR	REP	REPE
REPNE	REPZ	REPZ	RET	RETF	ROL	ROR	SAHF
SAL	SAR	SBB	SCASB	SCASW	SHL	SHR	STC
STD	STI	STOSB	STOSW	SUB	TEST	WAIT	XCHG
XLAT	XOR						

Liste 1

Jeder Befehl wird entweder mit keinem, einem oder mit zwei Operanden aufgerufen. Dabei kann ein Operand ein Register, eine Adressmarke, eine Speicherstelle oder eine Konstante sein.

Register (reg, reg8, reg16):

AX, BX, CX, DX, SI, DI, BP, SP, CS, DS, ES, SS und das Flag-Register sind 16-Bit-Register. Jedes Register hat eine bestimmte Bedeutung. In Ihren Programmen können Sie AX ... BP Werte zuweisen ohne größeren Schaden anzurichten. CS ... SS werden beim Start eines von JASMIN übersetzten Programms geladen und enthalten alle den gleichen Wert. (JASMIN erzeugt ".COM"-Dateien!)

AX, BX, CX, DX können auch als AH,AL, BH,BL, CH,CL und DH,DL (8-Bit-Register) angesprochen werden.

Adressmarken (adr, shortadr):

JASMIN gibt Ihnen die Möglichkeit für Sprünge oder für Variablen symbolische Namen zu definieren. Diese Marken sind im Prinzip Konstanten, deren Wert bei der Übersetzung des Quelltextes ermittelt wird. Eine Marke muß mit einem Klammeraffen ("@" ) eingeleitet werden. Die Definition einer Adressmarke endet mit einem Doppelpunkt.

Beispiel:

```
@Loop:          ; Definition einer Adressmarke
...             ; weitere gültige Anweisungen
JMP @Loop       ; und Sprung zurück zur Marke
```

**Speicherstellen (mem, mem8, mem16):**

In eckigen Klammern eingeschlossene Werte geben den Offset einer Speicherstelle an. Sie können damit auf beliebige Stellen im Speicher zugreifen. JEDER Zugriff auf eine Speicherstelle muß in eckigen Klammern "[]" eingeschlossen sein! Das ist nicht bei allen Assemblern so! Nicht eingeklammerte Zahlen bzw. Angaben mit OFFSET werden von JASMIN als imm (s.u.) betrachtet!

Wenn Sie auf Speicherstellen in anderen Segmenten zugreifen möchten, so können Sie dies mit einem Segment-Präfix ("ES:", "SS:", "CS:", "DS:") anzeigen. Dieser Befehl muß eine Zeile vor dem eigentlichen Speicherzugriff stehen.

Beispiele:

```
MOV AH, [1234]    ; AH wird mit dem Wert an Speicherstelle 1234
                  ; geladen
```

```
MOV BYTE PTR [10], 5 ; weisen Sie einer Speicherstelle eine Kon-
                    ; stante zu, müssen Sie über BYTE / WORD PTR
                    ; festlegen, ob es sich um eine 8- oder 16-
                    ; Bit Operation handeln soll.
```

```
@var: DW 0FF00h
```

```
...
MOV AX, [OFFSET @var] ; bzw. kürzer: MOV AX, [?@var]
                    ; so können Sie auf einen Wert an einer
                    ; Speicherstelle, die durch eine symbolische
                    ; Adressmarke gekennzeichnet ist, zugreifen
```

```
MOV BYTEPTR["$${}"], "?" ; ist gleich: MOV BYTEPTR[2424h], 3Fh
```

```
MOV ES, 0          ; direkte Zuweisung für die Segmentregister
                    ; CS, DS, ES, SS ist nicht zulässig!
                    ; ALSO: MOV AX, 0 und dann MOV ES, AX
```

```
MOV AX, B800h      ; Textbildschirmspeichersegment CGA/EGA/VGA
MOV ES, AX          ; in das Extra-Segment laden
ES:                ; nächster Zugriff in diesem Segment!
MOV DL, [0]         ; lese das Zeichen, das in der linken oberen
                    ; Ecke auf dem Bildschirm steht -> DL
```

```
; meine Philosophie, aber schlechter Stil -
; die folgenden Anweisungen werden akzeptiert:
```

```
MoV B Y T E ptr[9999h], 5
    @labell: MOV A X, word PT R [10 99 h]
add A X , A X
    ReT
```

```
call @LABEL 1
```

; Dieser Code ist fast kaum zu lesen. Vermeiden Sie es also bitte,  
; die Freiheiten, die Ihnen JASMIN zugesteht voll auszunützen!

Konstanten (imm, imm8, imm16):

Zahlen im Binär-, Dezimal- oder Hexadezimalsystem  
oder als ASCII-String (imm8: 1 Zeichen, imm16: 2 Zeichen)

Beispiele:

```
                                ; dezimal:
0101b                          ; 5
0FFFFh                         ; 65535
WORDPTR -10h                   ; 65520 (=0FFF0h)
BYTEPTR -10h                   ; 240   (=0F0h)
10d                             ; 10
255
"A"                             ; 65
"AB"                           ; 24889 (6139h)  "A" ist das niederwertige
                                ;                "B" das höherwertige Byte
```

Befehl Opcode	Bedeutung	Operanden
AAA	ASCII Adjust After Addition	keine
AAD	ASCII Adjust Before Division	keine
AAM	ASCII Adjust After Multiplication	keine
AAS	ASCII Adjust After Subtraction	keine
ADC	Add with Carry addiert den zweiten Operanden zum ersten, zusätzlich wird das Carryflag mitaddiert	reg,reg reg,mem mem,reg reg,imm mem,imm
ADD	Integer Addition	s. ADC
AND	Boolean AND	s. ADC
CALL	Call Procedure	adr reg16 mem16
CBW	Convert Byte to Word	keine
CLC	Clear Carry Flag	keine
CLD	Clear Direction Flag	keine
CLI	Clear Interrupt Flag	keine
CMC	Complement Carry Flag	keine
CMP	Compare Integers	s. ADC
CMPSB	Compare String Bytes	keine
CMPSW	Compare String Words	keine
CWD	Convert Word to Doubleword	keine
DAA	Decimal Adjust AL After Addition	keine
DAS	Decimal Adjust AL After Subtraction	keine
DEC	Decrement	reg mem
DIV	Unsigned Division	reg mem
HLT	Halt	keine

IDIV	Integer Signed Division	reg mem
IMUL	Integer Signed Multiplikation	reg mem
IN	Input from I/O Port	Akku,imm8 Akku,DX
INC	Increment	reg mem
INT	Software Interrupt	imm8
INTO	Interrupt on Overflow	keine
IRET	Interrupt Return	keine
JA/JNBE	Jump Above/Jump Not Below or Equal	Disp
JAE/JNB	Jump Above or Equal/Jump Not Below	Disp
JB/JNAE	Jump Below/Jump Not Above or Equal	Disp
JBE/JNA	Jump Below or Equal/Jump Not Above	Disp
JCXZ	Jump CX=Zero	Disp
JE/JZ	Jump Equal/Jump Zero	Disp
JG/JNLE	Jump Greater/Jump Not Less or Equal	Disp
JGE/JNL	Jump Greater or Equal/Jump Not Less	Disp
JL/JNGE	Jump Less/Jump Not Greater or Equal	Disp
JLE/JNG	Jump Less or Equal/Jump Not Greater	Disp
JMP	Jump	adr reg16 mem16
JNE/JNZ	Jump Not Equal/Jump Not Zero	Disp
JNO	Jump No Overflow	Disp
JNP/JPO	Jump No Parity/Jump Parity Odd	Disp
JNS	Jump No Sign	Disp
JO	Jump if Overflow	Disp
JP/JPE	Jump if Parity/Jump if Parity Even	Disp
JS	Jump if Sign	Disp
LAHF	Load AH with Flags	keine
LDS	Load DS with Pointer	reg16,mem32
LEA	Load Effective Adress	reg16,mem16
LES	Load ES with Pointer	reg16,mem32
LOCK	Assert Hardware LOCK	keine
LODSB	Load String Byte	keine
LODSW	Load String Word	keine
LOOP	Decrement CX and Branch	Disp
LOOPE	Decrement CX and check Zeroflag	Disp
LOOPNE	Decrement CX and check Zeroflag	Disp
LOOPNZ	Decrement CX and check Zeroflag	Disp
LOOPZ	Decrement CX and check Zeroflag	Disp
MOV	Move Data	reg,reg reg,mem mem,reg reg,imm mem,imm sreg,reg16 sreg,mem16 reg16,sreg mem16,sreg
MOVSB	Move String Byte	keine
MOVSW	Move String Word	keine
MUL	Unsigned Multiplication	reg mem
NEG	Negate Integer	reg

NOP	No Operation	mem
NOT	Boolean Complement	keine
		reg
		mem
OR	Boolean OR	reg, reg
		reg, mem
		mem, reg
		reg, imm
		mem, imm
OUT	Output to Port	imm8, Akku
		DX, Akku
POP	Pop Value off Stack	reg
		segreg
		(nicht CS)
POPF	Pop Stack into Flagregister	keine
PUSH	Push Value onto Stack	reg
		segreg
		mem16
PUSHF	Push Flagregister onto Stack	keine
RCL	Rotate Through Carry Left	reg, 1
		reg, CL
		mem, 1
		mem, CL
RCR	Rotate Through Carry Right	s. RCL
REP	Repeat String Prefix	MOVS
		STOS
REPE/REPZ/REPNE/REPNZ	Repeat String Prefix and chk ZF	CMPS
		SCAS
RET	Return from Subroutine	[imm8 imm16]
RETF	Return from Subroutine	[imm16]
ROL	Rotate Left	s. RCL
ROR	Rotate Right	s. RCL
SAHF	Store AH in Flagregister	keine
SAL	Shift Arithmetic Left	s. RCL
SAR	Shift Arithmetic Right	s. RCL
SBB	Subtraction with Borrow	s. OR
SCASB	Scan String Byte	keine
SCASW	Scan String Word	keine
SHL	Shift Left (identisch mit SAL)	s. SAL
SHR	Shift Right Logical	s. RCL
STC	Set Carry Flag	keine
STD	Set Direction Flag	keine
STI	Set Interrupt Flag	keine
STOSB	Store String Byte	keine
STOSW	Store String Word	keine
SUB	Subtraction with Integers	s. OR
TEST	Test Bits	s. OR
WAIT	Wait Until Not Busy	keine
XCHG	Exchange	reg, reg
		reg, mem
XLAT	Translate Byte	keine
XOR	Boolean Exclusive OR	s. OR

Tabelle 1

In Tabelle 1 sind die gültigen Kombinationen von Opcodes mit Operanden dargestellt.



Die Abkürzungen bedeuten:

adr	Adress(mark)e
Akku	AX (16-Bit) oder AL (8-Bit)
Disp	Adressmarke, die nicht weiter als +- 127 Byte entfernt ist
imm	beliebige Zahl (8086 => höchstens 16 Bit!)
imm8	Zahl mit 8-Bit
imm16	Zahl mit 16-Bit
mem	BYTE PTR [????]
mem16	WORD PTR [????]
reg	AX, BX, CX, DX, AH, AL, BH, BL, CH, CL, DH, DL, SI, DI, BP, SP
reg8	AH, AL, BH, BL, CH, CL, DH, DL
reg16	AX, BX, CX, DX, SI, DI, BP, SP
segreg	CS, DS, ES, SS

Wenn es nicht explizit anders angegeben ist, so müssen die Operanden in ihrer Größe übereinstimmen! Beispiel: reg,mem kann sein reg8,mem8 oder reg16,mem16 - nichts anderes.

\$ Das Dollarzeichen kann bei bedingten Sprüngen verwendet werden um Adressmarkierungen (in übersichtlichen Fällen) einzusparen. Es wird dadurch das ausführbare Programm nicht kleiner! Geben Sie dem Sprungbefehl als Operand \$+x. Dabei geben Sie mit x die Entfernung des Sprungziels an. Für x=0 wird zum nächsten Befehl im Programm gesprungen. VORSICHT: Sprünge dieser Art sind unübersichtlich! Sie können durch eine falsche Angabe auch in die Mitte eines Befehls springen und damit beliebig viel Unsinn anstellen!

Beispiele:

```
...
JCXZ $ + 11      ; CX=0 => Schleife überspringen
CALL @ReadKey    ; warte auf Tastendruck, Taste -> AX
CMP  AX, 27      ; ESC gedrückt ?
JE   $+2         ; JA => Schleife verlassen
LOOP $-11        ; wieder zum CALL zurückspringen
...
```

BYTEPTR Eine der beiden Anweisungen werden Sie benötigen, wenn Sie bei einer Instruktion die Operanden mem und imm verwenden möchten. In diesem Fall kann der Assembler nicht wissen, ob der immediate-Wert ein Byte oder ein Wort darstellt - Sie müssen dies durch BYTE/WORD PTR angeben!  
 Hinweis: Es ist egal ob BYTE/WORDPTR beim Speicher- oder Immediate-Operand angegeben wird. Die Angabe ist nicht nötig, wenn die Größe eines Operanden bestimmt werden kann (wenn es sich z.B. bei einem Operand um ein Register und beim Zweiten um eine Zahl oder Speicherstelle handelt.)

Beispiele:

```
MOV [1000h], WORD PTR 12h
CMP BYTEPTR [10h], 35
MOV BYTE PTR AL, 10h      ; Ok, aber unnötig
```

falsch ist:

```
MOV WORD PTR AH, [0]      ; WORDPTR wird ignoriert!
MOV BYTE PTR [1], WORD PTR 255 ; Operandengrößen
MOV AH, WORD PTR 20h      ; unterschiedlich
MOV BYTEPTR [0], 256      ; 256 > MAX(BYTE) !
```

DB Mit diesen Anweisungen kann ein Byte bzw. Wort direkt angegeben werden. Die nach DB folgenden Zahlen und Zeichen einer Zeichenkette werden an der auftretenden Stelle in die ausführbare Datei kopiert. Es können nach DB Zahlen, die mit einem Byte dargestellt werden können, oder Zeichenketten in Anführungsstrichen folgen. Mit DW können Wörter (16-bit Zahlen) definiert werden. Es sind hierbei - wie bei den Konstanten - Zeichenketten der Länge 2 zugelassen. Wenn Sie

nach DB/DW keine weiteren Angaben machen, wird 1 Byte/Wort freigehalten und mit 0 vorbelegt. Vorsicht: Wenn Sie keinen Sprung um einen Datenbereich machen, werden die mit DB/DW eingegebenen Zahlen als Code angesehen und ausgeführt!

Beispiele:

```
JMP SHORT @weiter
DW                                ; gleich wie DB 0,0
DB 0
DB "Hello world", 10h, 20h, 01101101b, "?"
DW 0, "??", 0
@weiter:
DB B8h,38h,29h                ; MOV AX,2938h
DW 20CDh                      ; INT 20h
```

INCLUDE Ein anderes Assembler-Listing kann an einer beliebigen Stelle im Programm eingebunden werden. Der Assembler behandelt den eingebundenen Text genauso wie das Programm, d.h. alle Label-Definitionen sind global und beim Assemblieren wird die Zeilenzahl in den Include-Dateien weitergezählt als würde der Inhalt direkt im Hauptprogramm stehen. Geschachtelte Include-Dateien sind nicht erlaubt! Dieser Befehl ist sinnvoll, wenn man Routinen geschrieben hat und diese in mehreren Programmen nutzen möchte. Dabei spielt es keine Rolle, ob die Include-Dateien am Anfang, am Ende oder irgendwo im Programm eingefügt werden. Bitte beachten Sie aber auch hier, daß Sie - falls die Routine nicht an der Stelle im Programm ausgeführt werden soll, an der sie eingebunden wurde - einen Sprung darüber hinweg machen müssen.

Beispiele:

```
JMP @Start                ; Include überspringen
INCLUDE C:\Params.INC     ; Prozeduren einbinden
@Start:
INCLUDE Init              ; Init einbinden und an
...                      ; dieser Stelle ausführen
CALL @GetPar              ; Funktion aus Params.INC
...                      ; ausführen
INT 20h                   ; Programm beenden
INCLUDE WrNum.INC         ; Prozeduren einbinden
```

OFFSET Mit diesen Schlüsselworten ist es möglich, die Adresse einer Marke zu bestimmen. OFFSET und ? sind identisch. Diesen Befehl können Sie dazu verwenden, um die Adresse in ein Register zu laden, oder um direkt auf die Speicherstelle mit der ermittelten Adresse zuzugreifen.

Beispiele:

```
                JMP SHORT @load
@x:             DW
@Name: DB "C:\JASMIN.DOC", 0
@load: MOV DX, OFFSET @Name ; Offsetadresse -> DX
        MOV AX, 3D00h       ; Datei öffnen
        INT 21h
        ...
        MOV DX, [?@x]
        INC BYTE PTR [OFFSET @x]
        ...
```