Fast Assembler

für SpartaDOS X

MMMG Soft 1995, Public Domain 2021
2021-12-01
holgerjanz@msn.com

Inhalt

Einführung

Fast Assembler ist ein Datei zu Datei Assembler für SpartaDOS X. Der Assembler kann Objektdateien und ausführbare Dateien für SpartaDOS X und Atari DOS erzeugen. Der Quelltext muss in ATASCII Dateien vorliegen.

Der Assembler hat folgende Eigenschaften und Funktionen:
- Unterstützung von SpartaDOS X und Atari DOS Objektdateien
- Unterstützung von verschiebbaren SpartaDOS X Blöcken (relocatable Blocks)
- Unterstützung von SpartaDOS X Symbolen
- Maximale Dateilänge von Quelltext und Objektdate hängt nur vom Dateisystem ab, da er ein echter Datei zu Datei Assembler ist

Fast Assembler ist vollständig kompatibel mit Quick Assembler, ist aber nicht kompatibel mit MAC/65 oder Atari Assembler Editor. Die Syntax von Fast Assembler wird vom 6502 Cross-Assembler MADS unterstützt. MADS ist für macOS, Linux und Windows verfügbar.

Das Fast Assembler Paket enthält folgende Dateien und Verzeichnisse:

`FA.COM`
Dies ist ein ausführbare Datei im SpartaDOS X Format. Dies ist die einzige Datei, die zum Starten des Assemblers benötigt wird. Diese Datei kann man in ein Verzeichnis kopieren, welches in $PATH enthalten ist, um den Assembler einfach von allen Verzeichnissen starten zu können.

`FA.MAN`
Dies ist die Manual-Page für Fast Assembler (siehe SpartaDOS X Anweisung MAN). Hier sind zum Nachschlagen die wichtigsten Informationen zusammen gefasst. Diese Datei sollte in ein Verzeichnis kopiert werden, welches in $MAN enthalten ist.

`EXAMPLES>`
Dieses Verzeichnis enthält gut kommentierte Beispiele.

`DOCS>`
Dieses Verzeichnis enthält Dokumentation im Atari Format.

`SOURCES>`
Dieses Verzeichnis enthält den Quelltext für Fast Assembler. Fast Assembler kann sich selbst assemblieren und kann so unter SpartaDOS X weiterentwickelt werden.

`TEST>`
Dieses Verzeichnis enthält eine Testumgebung. Es enthält die Batch-Datei TEST.BAT zu starten der Tests. Es werden für alle Tests auch die Verzeichnisse EXAMPLES und SOURCES benötigt.

Aufrufsyntax

```
FA src[.ext] [dst[.ext]] [/L|S|E]
```

Es wird die Source Datei `src[.ext]` assembliert und das Ergebnis als Object Datei `dst[.ext]` gespeichert. Die Standarderweiterungen sind `.ASM` und `.OBJ`. Es können aber andere verwendet werden. Laufwerk und Pfad werden vom Ort des Aufrufs des Assemblers übernommen.

Mit der Option /L werden alle Programmzeilen bei der Assemblierung ausgegeben. Die Zeilen werden ausgegeben, wie der Assembler sie liest, d.h. alle Includes sind aufgelöst und die Zeilennummern werden fortlaufend vergeben.

Mit der Option /S wird eine Zusammenfassung am Ende der Assemblierung ausgegeben, z.B. Gesamtanzahl von Labels, Blöcken und Speicher. Der Speicher wird im folgenden Format ausgegeben:

```
Memory: $code-$stat_mem-§stat_end $dyn_mem-$dyn_end
```

Dabei ist $code die Adresse, an die der Assembler geladen wurde (MEMLO zum Zeitpunkt des Starts des Assemblers). Der Begin der statischen Puffer, z.B. für die Zeilenverarbeitung und den Dateizugriff, wird mit $stat_mem angegeben und das Ende mit $stat_end. Die dynamischen Puffer, z.B. für die Symboltabelle, beginnen bei $dyn_mem (MEMLO zum Zeitpunkt des Starten der Assemblierung) und enden bei $dyn_end. Hiermit kann man prüfen wieviel Speicher bei der Assemblierung verbraucht wurde. Wenn $dyn_end den Wert von MEMTOP erreicht (siehe SpartaDOS X Kommando MEM), kommt es zum Fehler: Out of Memory.

Mit der Option /E werden Assemblerfehler zu SpartaDOS X Fehlern beginnend mit 224 ($E0) konvertiert, wenn bei der Assemblierung Fehler auftreten. Dies kann in der Batch-Verarbeitung mit `IF ERROR` genutzt werden (siehe Abschnitt Fehlermeldungen am Ende des Dokuments).

Beispiele

Atari DOS Programm - RAINBOW.ASM

Dieses Beispiel zeigt, wie Atari DOS Programme erstellt werden können. Es ist ein kleines Demo mit klassischen Farblaufen.

Assemblieren mit:

```
FA RAINBOW.ASM RAINBOW.COM
```

Ausführen mit:

```
RAINBOW
```

Mit START wird das Programm beendet.

Das Programm besteht aus zwei Blöcken. Ein Block enthält das eigentliche Programm, der Zweite die Startadresse zum automatischen Starten. Ein klassisches Atari DOS Programm. Blöcke werden mit dem Pseudoanweisung **blk** (siehe Abschnitt Pseudoanweisungen) eingeleitet. Der zweite Parameter enthält den Typ. DOS steht für Atari DOS Block. Als dritter Parameter folgt die Anfangsadresse des Blocks. Die Anweisung blk DOS $3000 entspricht der Anweisung ORG $3000 oder *=$3000 in anderen Assemblern.

```
* Example for FA
* Demo Atari Rainbow Color
* Atari DOS executable
* press START to end

RTCLK   equ $14
RUNAD   equ $02e0
COLPF2  equ $d018
CONSOL  equ $d01f
WSYNC   equ $d40a
VCOUNT  equ $d40b

* main program Atari DOS block
        blk DOS $3000
* reset console keys
start   lda #$08
        sta CONSOL
loop    clc
* calc next color
        lda VCOUNT
        adc RTCLK
* wait for sync
        sta WSYNC
* set color
        sta COLPF2
* check START
        lda CONSOL
        and #$01
        bne loop
        rts

* run address Atari DOS block
        blk DOS RUNAD
        dta a(start)

        end
```

SpartaDOS X Programm - HELLO.ASM

Dies ist ein Beispiel für ein SpartaDOS X Programm. Es kann nicht unter Atari DOS ausgeführt werden. In diesem Beispiel wird verschiebbare Blöcke und Symbole verwendet.

Assemblieren mit:

```
FA HELLO.ASM HELLO.COM
```

Ausführen mit:

```
HELLO
```

Nach der Eingabe eines Namens z.B. Holger, grüsst es:

```
Hello Holger !
```

Es werden die SpartaDOS Symbols PRINTF und GETS benutzt. Diese Symbole werden erst zur Laufzeit des Programs eingebunden. Sie werden mit JSR gerufen und erwarten ihre Parameter direkt im Anschluss.

Wenn man einen Fehler beim Namen des Symbols macht, kommt es zu einem Laufzeitfehler.
Als Beispiel sollte die Zeile:

```
PRINTF  smb 'PRINTF'
```

wie folgt geändert werden:

```
PRINTF  smb 'PRINTFX'
```

Das Programm lässt sich assemblieren. Beim Start wird dann aber von SpartaDOS X folgende Fehlermeldung ausgegeben:

```
Symbol PRINTFX
154 Symbol not defined
```

Weitere Informationen über die Symbole können im „SpartaDOS X Programming Guide“ gefunden werden.

Das Programm besteht aus zwei Blöcken, einen verschiebbaren Block (Reloc-Block) und einem Speicher reservierenden Block (Empty-Block). Beiden ist gemein, dass die Ladeadresse zum Zeitpunk der Assemblierung nicht bekannt ist. Solche Blöcke werden von SpartaDOS immer automatisch ab MEMLO geladen und MEMLO wird danach weiter gesetzt. Weiterhin werden dann die Adressen im Programm angepasst. Für den Empty-Block wird von SpartaDOS nur MEMLO weitergesetzt, d.h. der Speicher wird nicht initialisiert. Nach dem Ende des Programms, wird MEMLO von SpartaDOS X wieder zurückgesetzt.

```
* Example for FA
* Hello from Atari
* SpartaDOS X executable with
* relocatable code

* used SpartaDOS X symbols
* see SDX programming guide
PRINTF  smb 'PRINTF'
GETS    smb 'GETS'

* name buffer size
nam_siz equ $40

* relocatable code block
        blk reloc main
* print question
        jsr PRINTF
        dta c'What''s your name?'
        dta b($9B,$00)
* get name, due to code
* relocatability you cannot use
* lda <name and ldx >name because at
* compile time we do not know where
* the program will be located!
        lda namev
        ldX namev+1
        ldy #nam_siz
        jsr GETS
* print greetings
        jsr PRINTF
        dta c'Hello %s !'
        dta b($9b,$00)
        dta v(name)
        rts
* vector is needed for get name
namev   dta v(name)

* relocatable data block
* program counter is incremented!
* by BLK EMPTY, symbols must be
* definded beforehand
name    equ *
        blk empty nam_siz main

* blocks for address and symbol update
        blk update addresses
        blk update symbols

        end
```

Speicherresidentes Programm - TSR/TSR_CALL.ASM

In SpartaDOS X kann man sehr einfach speicherresidentes Programm schreiben. Dies sind Programme, die nicht aus dem Speicher entfernt werden, wenn sie beendet sind.

Dies kann man mit dem Symbol INSTALL erreichen. Es ist bei Start eines Programmes mit 0 initialisiert. Wenn beim beenden des Programs INTALL den Wert $FF hat, wird es nicht aus dem Speicher entfernt, d.h. MEMLO wird nicht zurückgesetzt. So kann man sehr einfach zusätzliche Treiber schreiben.

Man kann seinen Unterprogrammen auch einen Namen geben und so neue SpartaDOS X Symbole definieren. Dies geschieht mit folgendem Blockbefehl:

```
blk update new <address> '<name>'
```

Wenn der Name mit @ anfängt, so wirdes in COMMAND auch als Kommando erkannt.

Diese Beispiel besteht aus zwei Programmen:
1) TSR.ASM definiert ein neues Symbol @GREET, welches auch als Kommando bekannt gemacht wird (der Name beginnt mit @)
2) TSR_CALL.ASM benutzt dieses Symbol. Aus diesem Grund funktioniert es nur, wenn TSR vorher gestartet wurde und damit auch das Symbol @GREET bekannt ist. Wenn nicht kommt es zu folgenden Fehler:

```
Symbol @GREET
154 Symbol not defined
```

Wenn TSR ausgeführt wurde, kann man GREET direkt aus SpartaDOS X aufrufen, oder man ruft das Programm TSR_CALL, welches das Symbol aufruft.

Kommunikation zwischen Programmen - CALLER/CALLEE.ASM

In SpartaDOS X kann man aus einem Programm (CALLER) direkt ein anderes Programm (CALLEE). Dazu wird die Funktion U_LOAD. Mit dem Symbol FLAG kann man dem gerufenen Program einen Wert übergeben. Es gibt noch weitere Möglichkeiten, Daten an das gerufenen Programm zu übergeben. Um das Beispiel einfach zu halten, werden diese hier nicht benutzt. Das gerufenen Program kann mit dem Symbol STATUS einen Rückgabewert setzen.

Das Programm CALLER ruft ein anderes Program und setzt vorher FLAG. Der Wert für FLAG und der Programmname werde über die Kommandozeile übergeben, z.B.:

```
CALLER 2 CALLEE.COM
```

Das Program CALLEE liest FLAG, multipliziert den Wert mit 8 und setzt mit dem Ergebnis STATUS. Hierzu wird die Funktion MUL_32 benutzt. Diese Funktion erwartet die Parameter an bestimmten Adressen relativ zu COMTAB und schreibt so auch das Ergebnis.

Beide Programme geben die Werte von FLAG und STATUS aus die sie gesetzt oder übergeben bekommen haben.

Program zum Vergleich von Dateien - EX5COMP.ASM

In diesem Beispiel wird gezeigt, wie ein neuer externer Befehl für SpartaDOS X entwickelt werden kann. Er ist ähnlich dem Befehl COMP und vergleicht zwei Dateien. Ein Unterschied ist, dass das Ergebnis nicht nur mit PRINTF ausgegeben wird, sondern auch STATUS gesetzt wird. Wenn die Dateien nicht gleich sind, wird der STATUS auf $FF(255) gesetzt. Da dies ein Wert über $80(128) ist, wird er von SpartaDOS X als Fehler interpretiert. Da die Fehlernummer $FF(255) keinen Fehlertext besitzt, erfolgt folgende Ausgaben:

```
255 System ERROR
```

Das Setzen einer Fehlernummer hat den Vorteil, dass man in einer Batch-Datei mit dem SpartaDOS X Befehl IF ERROR 255 darauf reagieren kann. Dies kann man z.B. für automatische Tests nutz, wenn eine zu testendes Programm eine Datei erzeugt und das erwartete Ergebnis in einer anderen Datei schon vorliegt.

Aufbau des Quelltexts

Der Quelltext ist zeilenorientiert. Jede Zeile kann einen Anweisung enthalten. Es darf auch Leerzeilen geben, diese dürfen keine weiteren Leerzeichen enthalten. Wenn z.B. die Zeile 42 ein Leerzeichen enthält, wird folgende Fehlermeldung erzeugt:

```
42:
ERROR: Unexpected eol
```

Kommentarzeilen beginnen mit *. Alles nach * bis zum Zeilenende wird vom Assembler ignoriert. Beispiel:

```
* This is a comment!
```

Kommentare können auch nach einem Anweisung folgen. Dort müssen sie nicht mit * eingeleitet werden. Der Assembler ignoriert alle Zeichen nach einem Anweisung bis zum Zeilenende. Eine Anweisungszeile ist wie folgt aufgebaut:

```
[label] opcode [operand] [comment]
```

Beispiel mit Label und Kommentar:

```
start   ldx #$08 set X register
```

Beispiel ohne Label und ohne Kommentar:

```
        ldy #$10
```

Um das niederwertige oder höherwertige Byte eines Werts zu nutzen, müssen die Operatoren < und > benutzt werden. Beispiel:

```
        lda <my_address
        ldy >my_address
```

Der Assembler unterstützt alle offiziellen 6502 Anweisungen (Opcodes) mit ihren Adressierungsarten.

Ein Besonderheit gegenüber anderen 6502 Assemblern sind die Operationen, die sowohl mit dem Akkumulator als auch mit anderen Adressierungsarten benutzt werden können. Hier muss für den Akkumulator das Zeichen @ benutzt werden.

Beispiel Anweisung ASL (Arithmetic Shift Left) auf Adresse $3000:

```
        asl $3000
```

Beispiel Anweisung ASL auf Akkumulator:

```
        asl @
```

Assemblierung

Der Assembler verarbeitet die Quelltext Zeile für Zeile und im letzten Durchlauf wird die Objektdatei oder ausführbare Datei entsprechend geschrieben. Keine der Dateien wird dabei im Speicher gehalten, d.h. die maximale Größe der Dateien die verarbeitet oder erzeugt werden können, werden nur durch das Dateisystem beschränkt. Der Assembler hält nur die interne Symboltabelle im Speicher.

Der Quelltext wird mindestens zweimal durchlaufen. Nach jedem Durchlauf wird die Anzahl der Bezeichner aufgeführt, die noch nicht aufgelöst werden konnten. Manchmal müssen mehr als zwei Durchläuft ausgeführt werden, z.B. nach folgender Definition:

```
a   equ b
b   equ c
c   equ 100
```

Im ersten Durchgang wird c bestimmt, im Zweiten von b und im Dritten von a. Es sollte darauf geachtet werden, so etwas zu vermeiden, da sich dadurch die Zeit für die Assemblierung unnötig verlängert (durch die Anzahl der Durchläufe). Folgende Definition kann in zwei Durchläufen verarbeitet werden:

```
c   equ 100
b   equ c
a   equ b
```

Der Assembler unterstützt maximal 8 Durchläufe.

Der Assembler unterstützt Includes. Die maximale Tiefe der Verschachtlung von Includes hängt davon ab, wieviele Dateien gleichzeitig geöffnet werden können. Es werden mindestens zwei Dateien geöffnet, für die Source und die Objekt Datei, für jede weitere Verschachtlung von Includes, wird eine weitere Source Datei geöffnet. Die Anzahl der gleichzeitig geöffneten Dateien kann mit dem SPARTA Treiber in CONFIG.SYS konfiguriert werden. Der Standardwert von SpartaDOS X ist 5, d.h. es ist eine maximale Tiefe der Verschachtlung von 3 möglich.

Im folgenden wird beschrieben, welche Ausdrücke benutzt werden können und welche Pseudoanweisunge (Anweisunge die keine 6502 Anweisunge darstellen, sondern zur Steuerung der Objekterzeugung oder der Verarbeitung von Ausdrücken und Werte dienen) der Assembler beherrscht.

Ausdrücke

Textausdrücke werden durch Anführungszeichen definiert, ' oder “. Um Anführungszeichen in einem Textausdruck darzustellen, müssen zwei aufeinander folgende benutzt werden, z.B.:

```
'Windows ''95'
```

Zahlen in Ausdrücken:

- Binäre Zahlen mit % z.B.:
  ```
  %101
  ```
- Hexadezimale Zahlen mit $ z.B.:
  ```
  $FDAACDE
  ```
- Dezimalzahlen z.B.:
  ```
  23454
  ```

Zahlen dürfen maximal 4 Bytes lang werden.

Zeichenwort in ATASCII z.B.:
'A' oder ''' für '

Zeichenwort in interner Darstellung z.B.:
"I" oder """ fuer "

Textausdrücke sind immer ein Byte lang und haben den Wert des ersten Zeichens, z.B.:
'A' und 'ABC' haben den gleichen Wert A.

Zusätzlich können folgende Operatoren verwendet werden:

`&` – Operator AND
`|` – Operator ORA
`^` – Operator EOR
`*` – Multiplikation
`/` – Division
`+` – Addition
`–` – Subtraktion

Sequenzen von Operatoren werden nach der oben genannten Prioritäten ausgewertet (AND höchste, Subtraktion niedrigste). Es wird immer mit 4 Bytes gerechnet. Maximal 32 Operatoren können pro Ausdruck verwendet werden. Die Priorität kann innerhalb eines Ausdrucks mit den eckigen Klammern geändert werden z.B.:

```
[[4+5]*[256/21]]&$FF
```

Es gibt zwei weitere Sonderzeichen:
* und ! - liefern den aktuellen Programmzähler.

* kann nur am Anfang eines Ausdrucks verwendet werden, während ! allgemein verwendet werden kann z.B.:

```
*+4
!–22
$2300+!
```

Pseudoanweisungen

Im Folgenden werden alle unterstützten Pseudoanweisungen beschrieben. Am Ende des Abschnitts werden die Pseudoanweisungen beschrieben, die nur zur Kompatibilität zu Quick Assembler unterstütz werden. Beim Erstellen neuer Programme werden diese nicht benötigt.

blk

Diese Anweisung deklariert einen neuen Programmblock. Die Gesamtzahl von Blöcke in einem Program ist auf 256 begrenzt. Es werden folgende Blockarten unterstützt:

blk n[one] a

Es wird ein Block ohne Header deklariert. Der Programmzähler wird auf die Adresse a gesetzt.

blk d[os] a

Deklariert einen DOS-Block mit $FFFF-Header oder ohne Header, wenn der Programmzähler schon auf a steht.

blk s[parta] a

Deklariert einen SpartaDOS X Block mit festen Ladeadressen mit Header $FFFA, Programmzähler wird auf a gesetzt.

blk r[eloc] m[ain]|e[xtended]

Deklariert einen verschiebbarer SpartaDOS X Block mit Header $FFFE im Haupt- (m[ain]) oder Extended-Speicher (e[xtended]).

blk e[mpty] a m[ain]|e[xtended]

Deklariert einen verschiebbarer SpartaDOS X Block mit Header $FFFE, um Bytes im Haupt- (m[ain]) oder Extended-Speicher (e[xtended]) zu reservieren. Der Programmzähler wird sofort um a erhöht.

blk u[pdate] s[ymbols]

Erzeugt einen SpartaDOS X Block mit Header $FFFB, der die Symboladressen in Sparta- oder Reloc-Blöcken aktualisiert.

blk u[pdate] a[ddresses]

Erzeugt einen SpartaDOS X Block mit Header $FFFD, zum Aktualisieren von Adressen in Reloc-Blöcken.

blk u[pdate] n[ew] a text

Erzeugt einen SpartaDOS X Block mit Header $FFFC, der ein neues Symbol in einem Reloc-Block mit der Adresse a deklariert. Wenn dem Symbolnamen ein @ vorangestellt ist und die Adresse aus dem Hauptspeicher stammt, kann ein solches Symbol über COMMAND.COM aufgerufen werden.

end

Ende der Quell- oder Include-Datei.

label equ a

Weist einem Namen einen Wert zu. Der Name darf maximal 240 lang sein.

Beispiel:

```
RUNAD equ $02e0
```

[label] dta x(expr)

Generiert Daten, wobei x wie folgt definiert werden kann:

- b(expr) – ein Byte
- a(expr) – zwei Bytes, Adresse
- v(expr) – zwei Bytes, Adresse, bedeutet für Reloc-Blöcke, dass sie aktualisiert werden muss, für andere Blöcke entspricht es a(expr)
- e(expr) – drei Bytes
- f(expr) – vier Bytes
- g(expr) – vier Bytes in umgekehrter Darstellung

- `l(expr)` – niederwertiges Byte
- `h(expr)` – höherwertiges Byte
- `c'ATASCII'` – Text in ATASCII
- `d'INTERN'` – Text in interner Darstellung.

Zusätzlich können numerische Daten als durch Komma getrennte Ausdrücke angegeben werden, z.B.:

```
myDat dta e(0,15000,$FFAACC)
      dta b($FF),a($C000)
```

icl source[.ext]

Fügt den Quelltext einer Datei ein. Laufwerk und Pfad werden standardmässig vom aktuellen Verzeichnis beim Start des Assembler übernommen, z.B.;

```
      icl 'FAMAIN.ASM'
```

label smb text

Deklaration der Verwendung eines SpartaDOS X Symbols. Nach der Verwendung von blk update symbols generiert der Assembler einen Block, der automatisch die Adresse der verwendeten Symbole im Programm aktualisiert z.B.:

```
pf    smb 'PRINTF'
      jsr pf
```

Der Assembler fügt nach jsr die richtige Symboladresse ein.

Diese Deklaration ist nicht transitiv, d.h. folgendes Beispiel führt zu einem Fehler bei der Assemblierung:

```
cm    smb 'COMTAB'
wp    equ cm-1.    (Error !)
      sta wp
```

Stattdessen muss man wie folgt vorgehen:

```
cm    smb 'COMTAB'
      sta cm-1     (ok !)
```

Alle Symboldeklarationen müssen zu erst im Programm definiert werden, also vor allen anderen Deklarationen sowie vor dem eigentlichen Programm.

Die folgenden Anweisungen sind nur für die Kompatibilität zu Quick Assembler notwendig:

opt b

Parameter für die Assemblierung, nur für die Kompatibilität zu Quick Assembler, wobei die Bits von b folgende Bedeutung haben:

Bits 0-1 Ausgabe des Quelltextes:

- `00` – keinen
- `01` – nur bei Fehler
- `10` – gesamter Quelltext

Bits 6-7 Objekttyp:

- `11` – ohne Header,

`01,10` – mit Atari DOS Header.

Beispiel für Atari DOS mit Ausgabe des gesamten Quelltexts:

```
opt %01000010
```

org a

Definiert einen neuen Block und erzeugt einen Header wie mit **opt** definiert und setzt den Programmzähler auf a. Diese Anweisung ist nur für die Kompatibilität zu Quick Assembler nötig, für Blöcke siehe Pseudoanweisung **blk**.

lst all|bad|not

Option für die Ausgabe des Quelltextes bei der Assemblierung :

`not` – keine Ausgabe
`all` – kompletten Quelltext ausgeben
`bad` – nur Zeilen mit Fehlern ausgeben

Blöcke

Die wichtigste Neuheit in SpartaDOS X ist die Möglichkeit, einfach verschiebbare Programme zu schreiben. Da der 6502 Prozessor keine relative Adressierung aufweist (mit Ausnahme von kurzen bedingten Sprüngen), verwendeten die ICD Programmierer spezielle Prozess zum Laden von Programmblöcken. Der gesamte Prozess lädt einen verschiebbaren Block immer ab MEMLO. Dann wird MEMLO um die Länge des Blocks erhöht und dann wird ein speziellen Block zum Aktualisieren von Adressen geladen. Alle Adressen im Programmblock sind nullbasiert. Es reicht also aus, den MEMLO Wert zu addieren, um die richtige Adresse zu erhalten. Welche Adressen sollen aktualisiert werden und welche nicht? Dafür gibt es einen speziellen Block, der kodierte Zeiger auf diese Adressen enthält. Nach dem Reloc-Block oder den Reloc-Blöcken muss ein **blk update addresses** ausgeführt werden, damit das Programm ausgeführt werden kann. Auch nach Sparta-Blöcken, bei denen sich Anweisungen (oder Vektoren) auf Reloc- oder Empty-Blöcke beziehen, ist es notwendig.

Eine weitere Neuerung ist die Einführung von Symbolen. Einige SpartaDOS X Serviceroutinen werden durch Namen definiert. Diese Namen sind immer 8 Buchstaben lang (ebenso wie Dateinamen). Anstelle von Vektor- oder Sprungarrays (wie im Atari Betriebssystem) werden mit dem Pseudoanweisung **smb** definierte Symbole verwendet. Nach dem Lesen eines Blocks oder von Programmblöcken, lädt SpartaDOS X den Block zur Aktualisierung der Symbole und tauscht auf ähnliche Weise wie bei Reloc-Blöcke Adressen im Programm aus. Symbole können in Reloc- und Sparta-Blöcken verwendet werden.

Der Programmierer kann seine eigenen Symbole definieren, um Sparta DOS X Symbole zu ersetzen, oder völlig neue Symbole, die von anderen Programmen verwendet werden können. Dies erfolgt über den Block update new. Neue Symbole müssen in einem Reloc-Block implementiert werden.

Die Anzahl von verschiebbaren Blöcke (Reloc- oder Empty-Blöcke) wird von SpartaDOS X auf 7 begrenzt.

Blöcke können zu Ketten kombiniert werden, z.B.:

```
blk sparta $600
...

blk reloc main
...

blk empty $100 main
...

blk reloc extended
...

blk empty $200 extended
```

Dies bedeutet, dass sich Anweisungen in diesen Blöcken auf alle Blöcke in der Kette beziehen können. Diese Kette wird nicht durch Adress- oder Symbolaktualisierungen zerstört, sondern nur durch neue Symboldefinitionen oder einer anderen Blockart (z.B. DOS).

Eine solche Kette ist nur dann sinnvoll, wenn alle Blöcke in der gleichen Speicherart (main oder extended) geladen werden oder wenn das Programm den Speicher entsprechend wechselt.

Anweisungen und Vektoren in Reloc- und Empty-Blöcken sollten sich nicht auf Sparta-Bloecke beziehen. Dies kann zu einem Fehler führen, wenn das Programm mit der SpartaDOS X Anweisung LOAD geladen wird und erst später verwendet wird. Während Reloc- und Empty-Blöcke sicher sind, ist nicht sicher, was sich im Speicher befindet, in dem sich der letzte Sparta-Block befand.

Es ist ebenso gefährlich, in Reloc- und Empty-Blöcken in Sparta-Blöcken zu referenzieren (Grund wie in Abschnitt oben). Bei der Installation von Overlays (.sys Dateien) mit INSTALL ist dies jedoch manchmal erforderlich und daher zulässig. Ein Sparta-Block kann auch initialisiert werden (über INITAD $2E2).

Adresskollisionen können zwischen Sparta-, Reloc- und Empty-Blöcken auftreten. Fast Assembler erkennt Verweise auf andere Blöcke anhand der Adressen, wobei für Reloc- und Empty-Blöcke Adressen ab $1000 angenommen werden. Beim Mischen dieser Blöcke, sollten sichergestellt werden, dass Sparta-Blöcke unter $1000 (z. B. $600) oder über dem letzten verschiebbaren Block liegen. In der Regel reicht $4000 aus. Dieser Fehler wird vom Assembler nicht erkannt.

Fehlermeldungen

`Undeclared label 224($E0)`
Undefinierter Name oder Symbol oder rekursive Definition.

`Label declared twice 225($E1)`
Name oder Symbol wurde zweimal definiert.

`Unexpected eol 226($E2)`
Unerwarteter Zeilenumbruch, die Zeile muss mindestens ein Zeichen oder einen Parameter enthalten.

`Too many passes 227($E3)`
Zu viele Durchläufe beim Assemblieren, die Anzahl von 8 Durchläufen wurde überschritten.

`Too big number 228($E4)`
Zahl zu gross, die Zahl hat den zulässigen 4-Byte-Bereich überschritten.

`String error 229($E5)`
Fehler im Textausdruck, keine geschlossenen Anführungszeichen oder leerer Ausdruck.

`Illegal symbol 230($E6)`
Unzulässiges Zeichen in der Zeile.

`Branch to far 231($E7)`
Der relative Sprung liegt ausserhalb des Bereichs +/-128 Bytes.

`Improper type 232($E8)`
Anweisung oder Adressierungstyp für den angegebenen Anweisung ist nicht zulässig.

`Label missing 233($E9)`
Fehlender Name, einer EQU oder SMB Definition muss immer ein Name vorangestellt werden.

`Expression expected 234($EA)`
Ausdruck erwartet, Operanden von mathematischen Operationen müssen numerische Ausrücke sein.

`Too many blocks 235($EB)`
Zu viele Blöcke, die Gesamtzahl der Blöcke darf 256 nicht überschreiten, und die Anzahl der RELOC- und EMPTY-Blöcke darf 7 nicht überschreiten.

`Undefined or too long 236($EC)`
Name undefiniert oder zu lang, bei der Definition von Blöcken muss ein Name vergeben werden und die Adresse darf nicht größer als 2 Byte sein.

`Improper block type 237($ED)`
Ungültiger Blocktyp, Z.B. darf eine UPDATE NEW Block nicht einem DOS Block folgen.

`Too long or invalid symbol 238($EE)`
Der Ausdruck ist zu gross oder enthält ein ungültiges Symbol.

`Parenthesis not balanced 239($EF)`
Keine geschlossenen Klammern oder keine passende geöffnete Klammer.

`Too many operations 240($F0)`
Zu viele Operationen im Ausdruck, die Anzahl der Operationen ist auf 32 begrenzt.

`Unexpected symbol 241($E1)`
Unerwartetes Symbol, Symbole können nicht in anderen Blöcken als SPARTA oder RELOC und in Blockdefinitionen verwendet werden.

`Internal error 242($E2)`
Interner Fehler, bitte schick mir eine Email, mit einer Anleitung zum Reproduzieren (holgerjanz@msn.com)