Bei dem normalen 6502-Prozessor sind von 256 möglichen Codes nur 151 als Assembler-Befehle definiert. Die CPU interpretiert aber aus den restlichen Codes auch Befehle, die zwar zum Teil unsinnig sind, aber auch manche nützliche Operationen enthalten.
Mit dem vorhandenen Monitor-Disassembler kann man ein Programm mit solchen Befehlen nicht auslisten, da dann nur „???“ ausgedruckt wird. Ich habe nun einen „Extended Disassembler“ (**EDA**) programmiert, der die zusätzlichen Befehle richtig ausgibt.
Nach dem Eintippen des Programms sollte es mit „BSAVE EDA, A$94C8, L$0138“ auf Diskette gespeichert werden. Durch „BRUN EDA“ initialisiert sich die Routine selbst und kann dann vom Monitor aus wie der „L“-Befehl aufgerufen werden, wobei das „L“ durch Ctrl-Y zu ersetzen ist.
EDA ist keine gepatchte Language-Card-Version des alten Disassemblers, sondern wird in der Page 3 und direkt unter dem DOS-Puffer abgespeichert, so daß er verschwindend wenig vom freien Speicherplatz in Anspruch nimmt.
Somit läuft EDA auch auf Apple-II-Versionen mit nur 48K RAM oder unter ProDOS, das die Language-Card benutzt.

Im folgenden sind zwei Anwendungsbeispiele der zusätzlichen Op-Codes näher erläutert:

DEC/CMP adr.
INC/SBC adr.

Diese Befehle erniedrigen oder erhöhen den Inhalt einer Speicheradresse und vergleichen das Ergebnis mit dem Akkumulator.
Solche Operationen sind oft für die Ausführung von Schleifen nützlich, in denen z.B. ein Pointer von einer Anfangsadresse auf eine Endadresse heruntergezählt wird. Dies ist nun mit einem einzigen Befehl möglich.
Beispiel 1 zeigt eine solche Schleife, die einen Speicherbereich löscht.

AXM #Operand

Dieser Befehl verknüpft Akkumulator und X-Register durch die AND-Funktion, zieht davon den Operanden ab und speichert das Ergebnis in das X-Register. Das Carry-Flag muß nicht gesetzt oder gelöscht werden. Wird der Akkumulator vorher mit $FF geladen, so schafft dieser Befehl die Möglichkeit, direkt im X-Register zu addieren oder zu subtrahieren.
Beispiel 2 demonstriert diesen Befehl und den Zeitgewinn gegenüber der Benutzung der „echten“ Codes.

Eine Auflistung aller 93 zusätzlichen Codes kann der **Tabelle 1** entnommen werden*.
Bei vielen Befehlen handelt es sich nur um eine Aneinanderreihung zweier normaler 6502-Befehle, wobei die Mnemonics dann aus den zwei Befehlen und einem Schrägstrich bestehen.
Wegen der fehlenden Festlegung habe ich für die restlichen Codes die Mnemonics frei definiert. Bei ihnen druckt EDA dann noch ein „§“ davor, um sie von den anderen Befehlen abzusetzen.
Alle übrigen nicht aufgelisteten Codes „hängen den Prozessor auf“ (z.B. $02). Man könnte sie höchstens als HALT-Befehle benutzen, bei denen der Prozessor dann wartet, bis ein Reset oder Interrupt erfolgt.

* Eine analoge Matrixtabelle findet man in „incider“, 4/85, S. 121

Beispiel 1

```
                1   * X = High-Byte der Anfangsadresse-1
                2   * A = High-Byte der Endadresse
                3
A0 00           4           LDY  #0
84 06           5           STY  $6
85 07           6           STA  $7
98              7   LP1     TYA
91 06           8   LP2     STA  ($6),Y
C8              9           INY
D0 FB           10          BNE  LP2
8A              11          TXA
C7 07           12          DEC/CMP $7
90 F5           13          BCC  LP1
```

Beispiel 2

```
                1   * Subtraktion (z.B. X - 7 -> X)
                2   * 4 Mikrosekunden
                3
A9 FF           4           LDA  #$FF
CB 07           5           AXM  #7         ;C=0 bei Überlauf
                6
                7   * Subtraktion mit "echten" Opcodes
                8   * 8 Mikrosekunden
                9
8A              10          TXA
38              11          SEC
E9 07           12          SBC  #7
AA              13          TAX
                14
                15  * Addition (z.B. X + 13 -> X)
                16
A9 FF           17          LDA  #$FF
CB F3           18          AXM  #-13       ;C=1 bei Überlauf
                19  * Im Zweierkomplemet ist -13 = $F3
```

Tabelle 1: Die „geheimen“ 6502-Opcodes

Code:	Mnemonic:	Adress.:	Zeit:	Funktion:
03	ASL/ORA	(ind.,X)	8	ASL Adr. & ORA Adr.
07	-"-	zeropage	5	-"-
0F	-"-	absolute	6	-"-
13	-"-	(ind.),Y	8	-"-
17	-"-	z.page,X	6	-"-
1B	-"-	absol.,Y	7	-"-
1F	-"-	absol.,X	7	-"-
23	ROL/AND	(ind.,X)	8	ROL Adr. & AND Adr.
27	-"-	zeropage	5	-"-
2F	-"-	absolute	6	-"-
33	-"-	(ind.),Y	8	-"-
37	-"-	z.page,X	6	-"-
3B	-"-	absol.,Y	7	-"-
3F	-"-	absol.,X	7	-"-
43	LSR/EOR	(ind.,X)	8	LSR Adr. & EOR Adr.
47	-"-	zeropage	5	-"-
4F	-"-	absolute	6	-"-
53	-"-	(ind.),Y	8	-"-
57	-"-	z.page,X	6	-"-
5B	-"-	absol.,Y	7	-"-
5F	-"-	absol.,X	7	-"-
63	ROR/ADC	(ind.,X)	8	ROR Adr. & ADC Adr.
67	-"-	zeropage	5	-"-
6F	-"-	absolute	6	-"-
73	-"-	(ind.),Y	8	-"-
77	-"-	z.page,X	6	-"-
7B	-"-	absol.,Y	7	-"-
7F	-"-	absol.,X	7	-"-
83	STX/STA	(ind.,X)	6	Store (A AND X) Adr.
87	-"-	zeropage	3	-"-
8F	-"-	absolute	4	-"-
97	-"-	z.page,Y	4	-"-
A3	LDX/LDA	(ind.,X)	6	LDX Adr. & LDA Adr.
A7	-"-	zeropage	3	-"-
AB	-"-	immed.	2	-"-
AF	-"-	absolute	4	-"-
B3	-"-	(ind.),Y	5	-"-