EZASS V2.1

EPROM Zur ASSembler-programmierung

mit dem NDR-Computer

Ausgabe 2

1/86

Graf Elektronik Systeme GmbH

ges

Inhaltsverzeichnis:

1. Einführung

Es stehen soviele Programmiersprachen zur Verfügung, daß jeder die Sprache verwenden kann, die seinen Bedürfnissen und seinem Geschmack entsprechen.

Allerdings sind die Sprachen, die von einem Interpreter aus bearbeitet werden, ziemlich langsam. Deshalb sind nicht alle Probleme mit den höheren Programmiersprachen zu lösen.

Schnelle Programme werden direkt im ASCII-Code (Hexadezimalziffern) programmiert. Doch dies erschwert dem Programmierer die Übersicht. Deshalb lohnt sich der Aufwand eines Übersetzungsprogrammes.

Das Übersetzungsprogramm macht dem Programmierer die unübersichtlichen Hexadezimalziffern zu einer gut verständlichen und leicht merkbaren Programmiersprache. Das verringert Programmierfehler und erleichtert die Programmkorrektur.

Zugleich erhalten wir einen weiteren Vorteil: Übersetzungsprogramme für Quellprogramme in Assembler führen Prüfungen auf Programmierfehler durch, indem sie die richtige Verwendung des Vokabulars der Assemblersprache überwachen, und hierdurch bleiben nur noch Fehler in der Logik des Programmablaufes, die alleine der Programmierer erkennen kann.

Dieses Übersetzungsprogramm wird von **ges** als **Z80 - Debugger Assembler** (Assembler und Disassembler) in einem 8kByte EPROM für die ROA64k angeboten.

2. Prinzipbeschreibung

2.1 Assembler

Die Assemblersprache jedes Mikoroprozessors besteht aus einem Befehlssatz, von dem jeder eine Zeile des Quellprogrammes belegt. Eine Zeile des Quellprogrammes besteht aus:

< Marke > < Kennfeld > < Operand > < Bemerkung >

z.B.:

```
                LD          A,0FFH    ; Lade FF nach A
LOOP:           DEC         A         ; Decrementiere A
                JP NZ ,     LOOP      ; Springe wenn A
                                      ; ungleich Null
                                      ; nach LOOP
                HALT
```

Das < Kennfeld > ist das wichtigste Feld und muß immer vorhanden sein. Es enthält eine Gruppe von Buchstaben, die den Befehl des Quellprogrammes in codierter Form darstellt.

Mit einer < Marke > wird ein Befehl durch einen Namen identifiziert, da während des Schreibens des Programmes meistens nicht bekannt ist, wo der Befehl im Speicher stehen wird. Man gibt dem Befehl einen Namen (Marke) um bei bedingten oder unbedingten Sprüngen eine Zielmarke angeben zu können.

Im < Operandenfeld > wird dem Kennfeld mitgeteilt, mit welchem Operanden in der entsprechenden Zeile gearbeitet wird. Auch werden hier die Sprungziele bestimmt.

Das letzte Feld < Bemerkung > wird beim Übersetzen überlesen. Es trägt aber wesentlich zur Übersicht bei, da die Programmzeile hier erklärt wird.

2.2 Disassembler

Oft besteht der Wunsch, ein Programm, das im Maschinencode vorhanden ist, wieder in die Mnemonik von Assembler zurück zu übersetzen. Dies kann z.B.: bei Fremdprogrammen erforderlich sein, um Systemanpassungen vorzunehmen.

Ein Disassebler hat eine komplizierte Aufgabe. Er muß versuchen, sich in einen Programmierer "hineinzudenken", um den Assembler-Code zu regenerieren. Daß dies nicht immer gelingt, ist selbstverständlich. Ein so erzeugtes Listing muß meist neu überarbeitet werden.

Die Problemem ergeben sich mit der Sprungmarke und den Datenbereichen, da Datenbereiche vom Objektcode her gesehen nicht vom Programm unterschieden werden können.

Eine Zeile des Disassemblers besteht aus

‹Adresse›	‹ASCII-Code›	‹Assembler-Code›
z.B.: 8800	3EFF	LD A,0FFH
8802	3D	LOOP:DEC A
8803	C20288	JP NZ,LOOP
8806	76	HALT

Das Feld ‹ Assembler-Code › ist das wichtigste,da hier übersichtlich vom Hexadezimal- in den Assemblercode übersetzt wird.

Im Feld des ‹ ASCII-Code › ist der Befehl in Hexadezimalziffern beschrieben.

Das ‹ Adressfeld › zeigt den Ort, an dem der Befehl abgelegt wird.

3. Aufbauanleitung

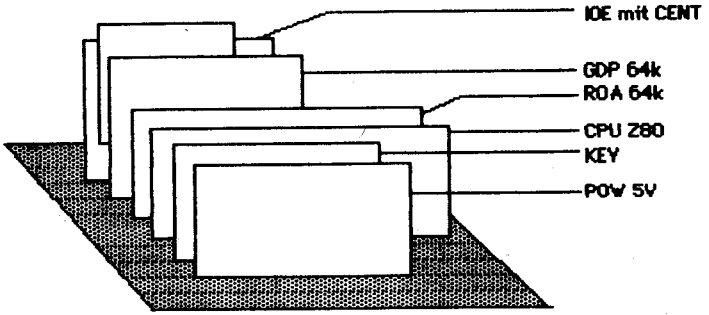
ACHTUNG CMOS !!!

CMOS-Bausteine sind hochempfindlich gegen elektrostatische Aufladung !

Bewahren oder transportieren Sie CMOS-Bausteine nur auf dem leitenden Schaumstoff auf !
Alle Pins müssen kurzgeschlossen sein !

Tip : Fassen Sie an ein geerdetes Teil (z.B.: Heizung, Wasserleitung oder an den Schutzkontakt der Steckdose bevor Sie einen Baustein berühren.

3.1 Konfiguration des NDR-Computers



(*** Bild 1 ***)

Für das Assembler-Programm wird das Grundprogramm EGRUND2 (auf Adresse 0000H), das Assemblerprogramm EZASS2 (auf Adresse 6000H) und ein RAM (8kByte) auf Adresse 8000H nötig.

3.2 Kurzbeschreibung der Baugruppen

3.2.1 CEN

Die CEN - Schnittstelle wird auf die IOE-Karte (auf Stiftleiste 2) gesteckt, um eine Anschlußstelle zum Druckerkabel zu erhalten. Ab 8/85 ist die CEN-Schnittstelle auf der IOE-Karte integriert.

3.2.2 CPU Z80

Diese Platine enthält den Prozessor Z 80 A als Zentraleinheit mit 4MHz - Taktversorgung. Sie benötigt die Speicherplatine ROA 64k.

3.2.3 GDP 64k

Der Grafik Display Prozessor GDP 9366 (Thomson) steuert die Ausgabe (Anzeige) auf dem Monitor. Neben der Ausgabe von Zahlen und Buchstaben kann auch eine hochauflösende Graphik erzeugt werden, wobei sich 256*512 Bildpunkte direkt ansteuern lassen. Diese Baugruppe enthält einen 64kByte dynamischen Speicher, der vier Bildschirmseiten speichern kann. Im Textmodus können 80 Zeichen pro Zeile sowie 24 Zeilen angezeigt werden.

3.2.4 IOE

Mit dieser Baugruppe wird die Verbindung zur Außenwelt (zum Drucker) hergestellt. Sie verfügt über 16 Ein- und Ausgabekanäle und wird auf Adresse 40H angesprochen.
Auf die IOE-Karte wird die CEN-Schnittstelle gesteckt. Daher darf nur eine 2*25polige Stiftleiste als Stiftleiste 2 eingelötet sein.

3.2.5 KEY

Die Platine KEY bietet die Möglichkeit, eine Standard-7-Bit-Tastatur anzuschließen. Die Anforderungen an die anzuschließende Tastatur sind:
7-Bit paralleler Ausgang, Codierung im ACSII-Code, Strobe-Signal zur Datenübernahme und 5V Betriebsspannung.

3.2.6 ROA 64k

Auf der Speicherplatine können 8*8kByte RAMs (bzw. das EPROM EGRUND 2 auf der Adresse 0000H, das EPROM EZASS 2 auf der Adresse 6000H mit einem 8kByte RAM auf der Adresse 8000H) gesteckt werden, die von 0000H bis FFFFH adressiert werden können.

3.3 Bestückungsanleitung

3.3.1 Lage der EPROMs auf der ROA 64k

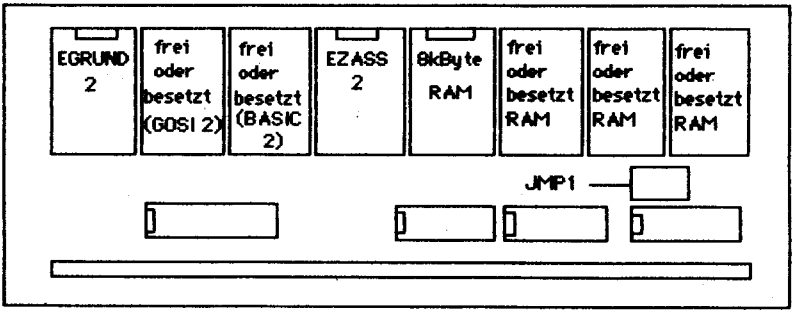


Bild 2: Lage der Speicherbausteine auf der ROA 64k

Auf der ROA 64k wird kein Jumper gesteckt. Dies entspricht der Adresse 0000h, mit der die Karte angesprochen wird.

3.3.2 Bestückungsanleitung der IOE

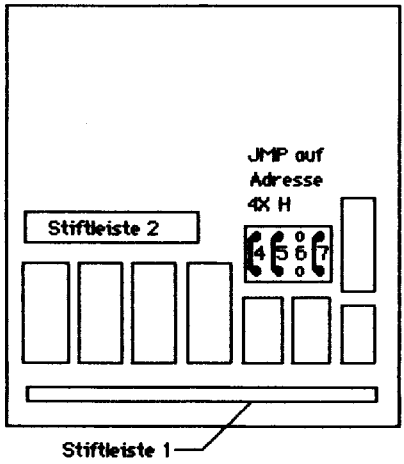


Bild 3 :
Bestückungsplan der IOE

Um die IOE auf die Adresse 4XH einzustellen, werden auf JMP 1 drei Brücken über 4,6 und 7 gelegt. Außerdem müssen e0 und e1 auf Masse gelegt werden. Damit wird die IOE mit allen Portadressen zwischen 40H und 4FH angesprochen. Für den Drucker sind die Portadressen 48H und 49H reserviert.

Eingelötete Brücke = Logisch 0
Nicht eingelötete Brücke = Logisch 1

Bit	7 6 5 4	3 2 1 0	
	0 1 0 0	x x x x	Binär
Adr.	4	X	Hexa-dezimal

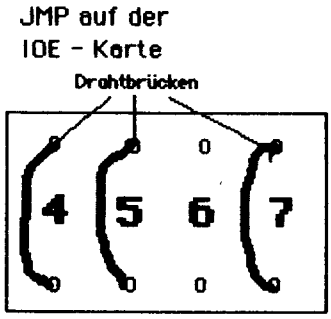


Bild 4: JMP auf der IOE

4. Kontrolle der Aufbauanleitung

Bevor Sie nun an das Einschalten denken, prüfen Sie zuerst, ob Sie alles richtig gesteckt haben. Ein paar Fragen zur ihrer Hilfe:

- Haben Sie die richtige Konfiguration gewählt ?
- Sitzt das EGRUND2, das EZASS2 und das RAM an der richtigen Stelle ?
- Sind alle Jumper auf der ROA64k gezogen ?
- Ist die IOE auf Adresse 4XH eingestellt?

Wenn Sie alle diese Fragen kontrolliert haben, kann nichts mehr schief gehen.

5. Grundbegriffe

5.1 Syntax des Debugger-Zeilenassembler

Der Zeilenassembler unterscheidet sich von einem "normalen" Assembler dadurch, daß hier nicht das gesammte Quellprogramm mit einem Editor vorher erstellt und anschließend assembliert wird, sondern es wird Zeile für Zeile eingegeben und jede Zeile sofort übersetzt.

5.1.1 Arithmetische Ausdrücke

+	Addition
-	Subtraktion
!	Oder-Verknüpfung
&	Und-Verknüpfung
	Nicht-Verknüpfung
-	Vorzeichen
$	Programmzähler
()	Klammerrechnung
symbol	definiertes Symbol
nnnnn	dezimale Zahl
nnnnh	sedezimale Zahl
0nnnnh	
"A"	Der ASCII-Code wird geladen

Beispiel :

```
LD HL,5+4-0FFH+3&(7-9)
```

Die mathematischen Konventionen wie Punkt vor Strich (Und vor Oder) werden berücksichtigt.
Groß- und Kleinschreibung wird identisch behandelt.

5.1.2 Hexadezimale Zahlen

Hexadezimale Zahlen werden mit einem "h" im Anschluß an die Zahl gekennzeichnet. Beginnt eine Hex-Zahl mit einem Buchstaben (A-F), so ist vor die Zahl eine Null ("0") zu schreiben.

Beispiel: 7AD4H
0FFFFH

5.1.3 Marken und Namen

Ein Name besteht aus einem Buchstaben, gefolgt von Buchstaben oder Ziffern.
Eine Marke wird durch einen nachfolgenden Doppelpunkt gekennzeichnet.

```
SCHLEIFE: NOP
          JP   SCHLEIFE
```

Groß- und Kleinbuchstaben sind als Eingabe möglich, werden aber nicht unterschieden.

5.1.4 Pseudobefehle

ORG ausdruck

Damit wird der Programmzähler gesetzt.

Beispiel:

```
ORG 9000h
```

OFF ausdruck

Der Wert von "ausdruck" wird auf den Programmzählerstand addiert und dort das Programm abgelegt. Den Befehl braucht man, wenn man Anwenderprogramme für die Adresse Null konstruieren will.

DEFB ausdruck,ausdruck ....
DEFB "texte","texte"

Ablage von Bytes oder Texten. Die Ausdrücke können auch gemischt verwendet werden. Auch einzelne Anführungszeichen sind möglich.

DEFW ausdruck, ausdruck .....

Ablage von Worten im Speicher. Dabei wird zunächst der niederwertige Teil, dann der höherwertige Teil.

name:= ausdruck

Zuweisung eines Wertes an einen Namen. Dies entspricht der konventionellen EQU - Anweisung, wurde aber im Hinblick auf die Konsistenz mit dem Grundprogramm so gewählt.

Beispiel:

```
ALPHA:=100
LD HL,ALPHA
LD BC,ALPHA
```

name:=%

Dadurch wird ein Name aus der Symboltabelle entfernt und kann danach neu als Marke definiert werden.

END Programmemde und Rückkehr ins Menue.

; Kommentarzeile, sie kann immer hinter dem Text eingegeben werden, zum Beispiel für die Protokollierung auf dem Drucker.

5.2 Pseudobefehle des Disassemblers

\+ Es wird ein Byte
"CR" weitergesprungen

\- Es wird ein Byte zurückgesprungen

* Der Bilschirm wird gelöscht und die Adresse wird an den oberen Bildschirmrand geschoben.

5.3 Hinweise zur Programmierung

5.3.1 Vorwärtssprünge

Dieses Menue wird mit einer END - Anweisung im Programm beendet.Da der Assembler im Ein-Pass-Verfahren arbeitet, kann der Assembler bei Vorwärtssprüngen das Sprungziel nicht definieren.

Deshalb wird im Sprungbefehl die Zieladresse durch ein "$" (Dollar) ersetzt. Der Befehl springt dann zunächst auf seine eigene Adresse. während des Programmierens wird die Zieladresse erreicht, die sich der Programmierer merkt.

Im zweiten Durchlauf wird auf den unteren Bildschirmrand geachtet. Dort wird die Adresse und der Inhalt des zu bearbeitenden Befehles angezeigt. Nun kann der tatsächliche Wert der Zieladresse eingetragen werden. Mit einem "CR" wird der Inhalt übernommen und der nächste Befehl angezeigt. Wenn allerdings der (Sprung-) Befehl neu geschrieben wird, so wird der alte Befehl gelöscht und der neu geschriebene in das Programm übernommen.

Beispiel:

```
.....
.....
JP  Z , LOOP
.....
.....
.....
.....
.....
LOOP: LD B,A
.....
.....
```

1. Schritt: Statt "JP Z,LOOP" wird statt dem Sprungziel LOOP ein "$" geschrieben. Der Befehl heißt dann "JP Z,$".

   Der Sprungbefehl springt (nach einem "CR") auf seine eigene Adresse.

2. Schritt: Die Adresse der Zeile des Sprungzieles "LOOP: LD B,A" schreibt sich der Programmierer auf. Z.B. sol der befehl auf Adresse 88C0H liegen.

3. Schritt: Das Programm ist zu Ende geschrieben. Der Programmierer ruft von neuem das Assemblerprogramm aus dem Assembler-Grundprogramm (mit "1") auf.

4. Schritt: Die Befehle werden mit einem "CR" übernommen
Angelangt beim Sprungbefehl "JP Z, $" wird
anstelle des "$" die Adresse des Sprungzieles
eingegeben, z.B. "JP Z, 88C0H"

5.3.2 Fehlerbehandlung

Fehler werden in der zu bearbeitenden Zeile an der vermeindlichen Stelle markiert. Man gibt die Zeile erneut ein. Der Programmzähler verändert sich nicht, daher ist das eingegebene Programm im Speicher dann korrekt abgelegt. Wenn man erneut ins Assemblermenue geht, nachdem man ein Progreamm beendet hat, so ist der Programmzähler auf dem letzten Wert und man kann dort fortfahren mit der Programmeingabe. Will man das nicht, so kann man mit dem ORG - Befehl eine Adresse festlegen.

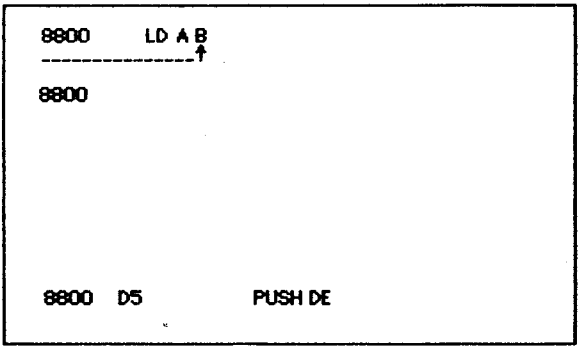


Bild 5 : Fehlerbehandlung

5.3.3 Speicherbelegung des Assembler

Der Assembler verwendet den Speicherbereich 8F00H bis 8FFFH, der nicht als Ziel für Programme verwendet werden darf. Die Symboltabelle des Grundprogrammes wird verwendet, daher kann man die definierten Symbole im Menue "Symbole" im Grundprogramm ausgeben; sie werden auch mit auf Kassette gerettet.

5.4 Zusammenstellung des Befehlssatzes

In diesem Kapitel beziehen wir uns auf das Buch von Osborne: Programmierung eines Mikroprozessors.
Im Folgenden finden sie die Zusammenstellung des Befehlssatzes.

In der nachfolgenden Tabelle werden die Symbole mit folgender Bedeutung verwendet:

A0	Akkumulator A0
A1	Akkumulator A1
AY	Entweder A0 oder A1
ADDR	Eine 16-Bit-Speicheradresse
C	Statusanzeige Übertrag (carry)
DATA	Eine 8-Bit-Dateneinheit
DZ0	Datenzähler DZ0
DZ1	Datenzähler DZ1
DZ2	Datenzähler DZ2
DZX	Beliebiger Datenzähler
DISP	8-Bit-Adreßverschiebung
DST	BEliebiges Zielregister
I	Beliebige Statusanzeige
O	Statusanzeige Überlauf (overflow)
P	Nummer eines E/A-Tors
PZ	Programmzähler
R	Beliebiges Register
S	Statusanzeige Vorzeichen (sign)
SZ	Stapelzeiger
SRC	Beliebiges Quellregister
SW	Statusanzeigen
Z	Statusanzeige Nullergebnis (zero)
[ ]	Inhalt. Ist die Bezeichnung eines Registers, E/A-Tors oder eine Speicheradresse in eckigen Klammern angegeben, dann ist jeweils der Inhalt gemeint.
[[ ]]	Implizite Speicheradressierung; Inhalt der Speicherstelle, deren Adresse Inhalt eines Registers ist.
∧	Logisches UND
∨	Logisches ODER
∀	Logisches EXCLUSIV-ODER
←	Datenübertragung in Pfeilrichtung
← →	Datenaustausch zwischen den angegebenen Stellen

(*** Bild 6A ***)

Art	Mnemonik	Operand(en)	Bytes	Status C	O	S	Z	Operationen bei Befehlsausführung
E/A-Befehle	INB	P	1,2					(A0) ← (P) Eingabe nach A0 von E/A-Tor P (P=0, 1, 2 für INB)
	OUTB	P	1,2					(P) ← (A0) Ausgabe von A0 nach E/A-Tor P (P=0, 1, 2 für OUTB)
Speicherzugriffsbefehle	LDB	AY,ADDR	3					(A0) ← (ADDR), (A1) ← (ADDR) Lade A0 oder A1 unter direkter Adressierung
	LDB	AY,@DZX	1					(A0) ← ((DCX)), (A1) ← ((DCX)) Lade A0 oder A1 unter impliziter Adressierung
	LDB	AY,@DZX+	1					(A0) ← ((DCX)), (A1) ← ((DCX)), and (DCX) ← (DCX) + 1 Lade A0 od. A1 unter impliziter Adressierung u. inkrementieren von DZX
	LDB	AY,@DZX−	1					(A0) ← ((DCX)), (A1) ← ((DCX)), and (DCX) ← (DCX) − 1 Lade A0 od. A1 unter impliziter Adressierung u. dekrementieren von DZX
	STB	AY,ADDR	3					(ADDR) ← (A0), (ADDR) ← (A1) Speichere A0 oder A1 unter direkter Adressierung ab
	STB	AY,@DZX	1					((DCX)) ← (A0), ((DCX)) ← (A1) Speichere A0 oder A1 unter impliziter Adressierung ab
	STB	AY,@DZX+	1					((DCX)) ← (A0), ((DCX)) ← (A1), and (DCX) ← (DCX) + 1 Speichere A0 od. A1 unter impliz. Adressierung ab u. inkrementiere DZX
	STB	AY,@DZX−	1					((DCX)) ← (A0), ((DCX)) ← (A1) and (DCX) ← (DCX) − 1 Speichere A0 od. A1 unter impliz. Adressierung ab u. dekrementiere DZX
	LDBNZ	AY,@DZX,ADDR	2					(A0) ← ((DCX)), (A1) ← ((DCX)) and (DCX) ← (DCX) + 1 plus branch Lade A0 od. A1 unter impliz. Adressierung, inkrementiere DZX u. verzweige
	STBNZ	AY,@DZX,ADDR	2					((DCX)) ← (A0), ((DCX)) ← (A1) and (DCX) ← (DCX) + 1 plus branch Speichere A0 od. A1 unter impliz. Adressierung, inkrement. DZX u. verzweige
Befehle für Operationen mit Speicherinhalten	ADDCB	AY,ADDR	3	X	X	X	X	(A0) ← (A0) + (ADDR) + (C), (A1) ← (A1) + (ADDR) + (C) Addiere mit Übertrag direkt adressierten Speicherinhalt zu A0 oder A1
	ADDCB	AY,@DZX	1	X	X	X	X	(A0) ← (A0) + ((DCX)) + (C), (A1) ← (A1) + ((DCX)) + (C) Addiere mit Übertrag implizit adressierten Speicherinhalt zu A0 oder A1
	ADDCDB	AT,ADDR	3	X	X	X	X	(A0) ← (A0) + (ADDR) + (C), (A1) ← (A1) + (ADDR) + (C) Addiere dezimal mit Übertrag direkt adressierten Speicherinh. zu A0 od. A1
	ADDCDB	AY,@DZX	1	X	X	X	X	(A0) ← (A0) + ((DCX)) + (C), (A1) ← (A1) + ((DCX)) + (C) Addiere dezimal mit Übertrag implizit adressierten Speicherinh. zu A0 od. A1
	SUBCDB	AY,ADDR	3	X	X	X	X	(A0) ← (A0) − (ADDR) − (C), (A1) ← (A1) − (ADDR) − (C) Subtrahiere dezimal mit Borgen direkt adressierten Speicherinh. von A0 od. A1
	SUBCDB	AY,@DZX	1	X	X	X	X	(A0) ← (A0) − ((DCX)) − (C), (A1) ← (A1) − ((DCX)) − (C) Subtrahiere dezimal m. Borgen implizit addressierten Speicherinh. von A0 od. A1

(*** Bild 6B ***)

Art	Mnemonik	Operand(en)	Bytes	Status C	O	S	Z	Operationen bei Befehlsausführung
Befehle für Operationen mit Speicherinhalten (Forts.)	ANDB	AY,ADDR	3			X		(A0) ← (A0) ∧ (ADDR), (A1) ← (A1) ∧ (ADDR) UND-Verknüpfung mit A0 oder A1 unter direkter Adressierung
	ANDB	AY,@DZX	1			X		(A0) ← (A0) ∧ ((DCX)), (A1) ← (A1) ∧ ((DCX)) UND-Verknüpfung mit A0 oder A1 unter impliziter Adressierung
	ORB	AY,ADDR	3			X		(A0) ← (A0) ∨ (ADDR), (A1) ← (A1) ∨ (ADDR) ODER-Verknüpfung mit A0 oder A1 unter direkter Adressierung
	ORB	AY,@DZX	1			X		(A0) ← (A0) ∨ ((DCX)), (A1) ← (A1) ∨ ((DCX)) ODER-Verknüpfung mit A0 oder A1 unter impliziter Adressierung
	XORB	AY,ADDR	3			X		(A0) ← (A0) ⊻ (ADDR), (A1) ← (A1) ⊻ (ADDR) EXCLUSIV-ODER Verknüpfung mit A0 od. A1 unter direkter Adressierung
	XORB	AY,@DZX	1			X		(A0) ← (A0) ⊻ ((DCX)), (A1) ← (A1) ⊻ ((DCX)) EXCLUSIV-ODER Verknüpfung m. A0 od. A1 unter impliziter Adressierung
	CMPB	AY,@DZX	1	X	X	X	X	Vergleich des implizit adressierten Speicherinhalts mit A0 oder A1
Direktbefehle zum Laden	LDB	AY,#DATA	2					(DST) ← DATA Lade im Befehl enthaltene Daten in A0 oder A1
	LD	R,#DATA	3					(DST) ← DATA Lade im Befehl enthaltene Daten in DZ0, DZ1, DZ2 oder SZ
Direktbefehle für Operationen	ADDB	AY,#DATA	2	X	X	X	X	(A0) ← (A0) + DATA + (C), (A1) ← (A1) + DATA + (C) Addiere binär im Befehl enthaltene Daten zu A0 oder A1
	ANDB	AY,#DATA	2			X		(A0) ← (A0) ∧ DATA, (A1) ← (A1) ∧ DATA UND-Verknüpfung von im Befehl enth. Daten mit A0 oder A1
	ORB	AY,#DATA	2			X		(A0) ← (A0) ∨ DATA, (A1) ← (A1) ∨ DATA ODER Verknüpfung von im Befehl enth. Daten mit A0 oder A1
	CMPB	AY,#DATA	2	X	X	X	X	Vergleiche im Befehl enthaltene Daten mit A0 oder A1
Sprungbefehle	BR	ADDR	3					(PC) ← ADDR Sprung zur Adresse mit der Marke ADDR
	CALL	ADDR	3					((SP)) ← (PC), (PC) ← ADDR, (SP) ← (SP) + 1 Sprung in ein Unterprogramm, dessen Anfangsadresse die Marke ADDR trägt

(*** Bild 6C ***)

Art	Mnemonik	Operand(en)	Bytes	Status C	O	Z	S	Operationen bei Befehlsaufführung
Bedingte Verzweigungsbefehle	BE,BZ	DISP	2					Wenn [Z] = 1, [PZ] ← [PZ] + DISP Verzweige, wenn Z = 1
	BNZ	DISP	2					Wenn [Z] = 0, [PZ] ← [PZ] + DISP Verzweige, wenn Z = 0
	BGE,BC	DISP	2					Wenn [C] = 1, [PZ] ← [PZ] + DISP Verzweige, wenn C = 1
	BNC,BLT	DISP	2					Wenn [C] = 0, [PZ] ← [PZ] + DISP Verzweige, wenn C = 0
	BV	DISP	2					Wenn [O] = 1, [PZ] ← [PZ] + DISP Verzweige, wenn O = 1
	BNV	DISP	2					Wenn [O] = 0, [PZ] ← [PZ] + DISP Verzweige, wenn O = 0
	BP	DISP	2					Wenn [S] = 0, [PZ] ← [PZ] + DISP Verzweige, wenn S = 0
	BN	DISP	2					Wenn [S] = 1, [PZ] ← [PZ] + DISP Verzweige, wenn S = 0
Register-Register: Verschiebebefehle	MOV MOVB	SRC,DST SRC,DST	1 1					(DST) — (SRC) Verschiebe Inhalt von SRC nach DST SRC = A0, A oder SZ; DST = A1, DZ0, DZ1, DZ2 oder PZ
	XCH XCHB	SRC,DST SRC,DST	1 1					(DST ← — (SRC) Vertausche Inhalte von SRC und DST
Register-Register-Operationsbefehle	ADDB	A0,A1	1	X	X	X	X	(A0) — (A0) + (A1) + (C) Addiere A1 binar zu A0
	ADDD	A0,A1	1	X	X	X	X	(A0) — (A0) + (A1) + (C) Addiere A1 dezimal zu A0
	SUBDB	A0,A1	1	X	X	X	X	(A0) — (A0) - (A1) - (C) Subtrahiere A1 dezimal von A0
	ANDB	A0,A1	1			X		(A0) — (A0) ∧ (A1) UND-Verknüpfung von A1 und A0
	ORB	A0,A1	1			X		(A0) — (A0) ∨ (A1) ODER-Verknüpfung von A1 und A0
	XORB	A0,A1	1			X		(A0) — (A0) ⊻ (A1) EXCLUSIV-ODER-Verknüpfung von A1 und A0
	CMPB	A0,A1	1			X		Vergleiche Inhalte von A1 und A0
	ADD ADDB	SRC,DST SRC,DST	1	X	X	X	X	(DST) ← (DST) + (SRC) + (C) Addiere SRC binar zu DST SRC = A0 od. A, DST = DZ0, DZ1, DZ2 od. SZ

(*** Bild 6D ***)

Art	Mnemonik	Operand(en)	Bytes	Status C	O	S	Z	Operationen bei Befehlsaufführung
Operationsbefehle	SHR	AY	1					Einfache Rechtsverschiebung des Inhalts von A0 oder A1
	SHL	AY	1					Einfache Linksverschiebung des Inhalts von A0 nach A1
	ROR	AY	1					Zyklische Rechtsverschiebung des Inhalts von A0 nach A1
	ROL	AY	1					Zyklische Linksverschiebung des Inhalts von A0 nach A1
	RORC	AY	1	X				Rechtsverschiebung des Inhalts von A0 oder A1 über Status C
	ROLC	AY	1	X				Linksverschiebung des Inhalts von A0 oder A1 über Status C
	SHRC	AY	1	X				Rechtsverschiebung d. Inhalts von A0 od. A1 mit abgezweigtem Überlauf
	SHLC	AY	1	X				Linksverschiebung d. Inhalts von A0 od. A1 mit abgezweigtem Überlauf
	SHRA	AY	1	X	X			Arithmetische Rechtsverschiebung des Inhalts von A0 oder A1
	SHLA	AY	1	X	X			Arithmetische Linksverschiebung des Inhalts von A0 oder A1
	SHR4	AY	1					Rechtsverschiebung des Inhalts von A0 oder A1 um 4 Bits
	SHL4	AY	1					Linksverschiebung des Inhalts von A0 oder A1 um 4 Bits
	SHR4	A	1					Rechtsverschiebung des Inhalts von A0 und A1 um 4 Bits
	SHL4	A	1					Linksverschiebung des Inhalts von A0 und A1 um 4 Bits
	INC	R	1				X	(R) ← (R) + 1 Register R inkrementieren (R = A0, A1, A, DZ0, DZ1 oder DZ2)
Stapelspeicherbefehle	PUSH	R	1					((SP)) ← (R); (SP) ← (SP) + 1 Inhalt von Register R und Zeiger SZ auf Stapelspeicher bringen R = A0, A1, DZ0, DZ1, DZ2, SZ, PZ
	POP	R	1					(R) ← ((SP)); (SP) ← (SP) - 1 Inhalt des Stapelspeichers unter Zeiger SZ in Register R bringen
	RET		1					(PC) ← ((SP)); (SP) ← (SP) - 1 POP-Befehl mit Programmzähler PZ als Zielregister
Param. Übergabe	LDB	AY (SP)+ n	1					(R) ← ((SP)+n); ((SP)) ← ((SP)) + 1 Parameter an Register R übergeben R = A0, A1, DZ0, DZ1 oder DZ2
	LD	DCX (SP)+ n						
Unterbrechungsbefehle	DI		1					Freigabe der Unterbrechungen
	EI		1					Sperren der Unterbrechungen
	RETI		1					(SW) ← ((SP)); (PC) ← ((SP)); Rücksprung aus Unterprogramm
Statusbefehle	SET	I	1	X	X	X	X	(I) ← 1 Statusanzeige I = 1 setzen (I = C, S, O, Z)
	CLR	I	1	X	X	X	X	(I) ← 0 Statusanzeige I = 0 setzen
	HALT		1					Halt-Befehl (zum Abbrechen des Programmablaufs)

(*** Bild 6E ***)

6. Beschreibung

6.1 Start des Assemblers

Nach dem Einschalten des NDR-Computers meldet sich das Grundmenue.

Nach Eingabe einer "2" gefolgt von einem "CR" fragt der Computer nach der Startadresse des Assemblers. Das Assembler-EPROM ist auf der Adresse 6000H gesteckt, Deshalb wird nun "6000" eingegeben, und es erscheint das Debugger - Grundmenue auf dem Bildschirm.

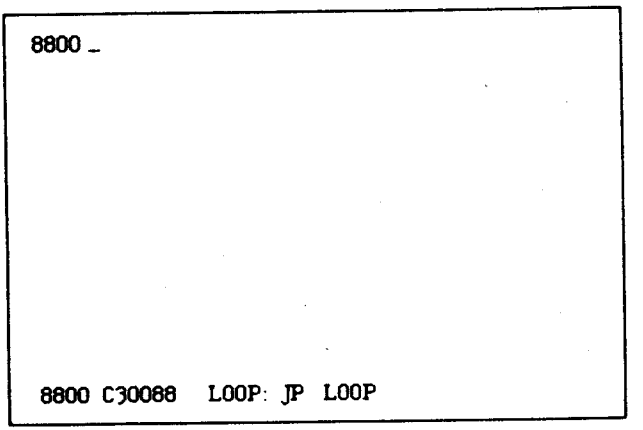
```
Debugger V2.1 (C) 1984 Rolf-Dieter-Klein

1=Assembler
2=Disassembler
3=starten
4=Einzelschritt
5=Protokoll an/aus
6=Grundprogramm
```

(***Bild 7***)

6.2 1=Assembler

Nach dem Drücken einer "1" kann man ein Assembler-programm, beginnend bei der Startadresse 8800H, editieren.



(*** Bild 8***)

Am oberen Bildschirmrand steht die Adrese 8800h, mit der beim Assemblieren begonnen wird.

Am unteren Bildschirmrand wird der Inhalt der Adresse angezeigt. Mit einem "+" bzw. "CR" wird der Befehl übernommen.

Wird ein neuer Befehl nun geschrieben, so wird dieser statt dem "alten" übernommen.

6.3 2=Disassembler

Mit "2" wird der Disassembler gestartet, der nach der Adresse des zu disassemblierenden Programmes frägt.

Die Befehle werden untereinander aufgelistet. Das Programm kann mit jeder Taste vorwärts gescrollt werden. Mit einem "." (Punkt) wird die Adresse an den oberen Bildschirmrand geschoben..

Der Ausstieg erfolgt mit der Taste "M".

6.4 3=starten

Mit "3" wird das geschriebene Asseblerprogramm gestartet. Nach dem Ablauf des Programmes meldet sich das Grundmenue.

6.5 4=Einzelschritt

Nach der Eingabe einer "4" wird nach der Startadresse des Programmes gefragt, das bearbeitet werden soll.
Es wird nun die Startadresse "8800" eingegeben, da der Debugger-Asembler immer auf Adressse 8800 den Programmbeginn festlegt. Es können aber auch Programmteile mit anderen Startadressen abgearbeitet werden.

Jeder Einzelschritt wird nach einem "CR" abgearbeitet, der Inhalt der Speicher wird am Bildschirm angezeigt. Nach dem Programmende springt der Computer wieder zurück ins Grundmenue.

6.6 5=Protokoll an/aus

Mit diesem Punkt kann ein angeschlossener Drucker zum Protokollieren eingeschalten werden. Bitte achten Sie darauf, daß der Drucker auf "bereit" steht, da sonst das Programm hier stehen bleibt.

6.7 6=Grundprogramm

Mit "6" wird ins Grundprogramm zurückgesprungen. Es meldet sich das RDK-Grundprogramm.

7. Anwendungsbeispiele

```
Debugger ab V2.0 (C) 1984 Rolf-Dieter Klein

1=Assembler
2=Disassembler
3=Starten
4=Einzelschritt
5=Protokoll an/aus
6=Grundprogramm

 Eingabe >>5

 Eingabe >>1

8800            ; beispiel textausgabe
8800            startetext:
8800            text:=8c00h
8800 21008C     ld hl,text
8803 CD1E00     call write
8806 C9         ret
8807            org text
8C00 6400/800   defw 100,120 ; x y
8C04 3300       defb 33h,0
8C06 48656C6C   defb "Hallo",0
     6F00
8C0C            end

Debugger ab V2.0 (C) 1984 Rolf-Dieter Klein

1=Assembler
2=Disassembler
3=Starten
4=Einzelschritt
5=Protokoll an/aus
6=Grundprogramm

Eingabe >> 2

 Adresse:8800h
 8800 21008C    STARTETEXT: LD HL,TEXT
 8803 CD1E00     CALL WRITE
 8806 C9         RET

 8807 FF         RST 7
 8808 04         INC B
 8809 00         NOP
 880A 00         NOP
 880B 80         ADD A,B
 880C 04         INC B
 880D 2008       JR NZ,8817H
 880F 00         NOP
 8810 BF         CP A
 8811 FF         RST 7
 8812 EF         RST 5
 8813 FF         RST 7
 8814 7F         LD A,A
 8815 F7         RST 6
 8816 CF         RST 1
 8817 7F         LD A,A
 8818 00         NOP
 8819 A0         AND B
```

8. Technischer Standard

Ab Version 2.0 des Debugger-Assembler ist nun nicht nur der Assembler (wie inVersion 0.9), sondern auch der Disassembler vorhanden.

Wer die Version 0.9 besitzt und über den neuen Disassembler verfügen will braucht nur sein EPROM an

Graf Elektronik Systeme
Magnusstraße 13
8960 KEMPTEN

zu schicken und erhält - gegen eine Kopierkostenpauschale von DM 10,- das neue Debugger-EPROM der Version 2.1.

Literaturhinweis:

Es gibt eine Reihe von Büchern, in denen die Assemblersprache sehr übersichtlich erklärt wird. Wir möchten auf zwei Bücher hinweisen, die uns besonders geeignet erscheinen und die sie auch bei uns beziehen können.

Adam Osborne, Einführung in die Mikrocomputertechnik, te-wi Verlag München, DM 48,-

Rodnay Zaks, Programmierung des Z80, Sybex-Verlag Düsseldorf, DM 59,-

Außerdem sollten Sie jederzeit über die fortlaufenden Änderungen informiert werden. Dazu ist ein LOOP-Abonement unbedingt von Nöten, denn dort erfahren Sie am schnellsten unsere neuesten Änderungen und sind daher immer auf dem neuesten Stand.

```
8800             ; kleines Beïspiel
8800 210400      ld hl,4
8803 CD0F20      call schleife
8806 216400       ld hl,100
8809 CD0320       call schreite
880C 215A00       ld hl,90
880F CD0620       call drehe
8812 CD1220      call endschleife
8815 C9          ret
8816             end _
```

```
8816             ; Beispiel 2
8816             start:
8816 0605         ld b,5
8818             schleife:
8818 C5           push bc
8819 216400       ld hl,100
881C CD0320       call schreite
881F 214800       ld hl,72
8822 CD0620       call drehe
8825 C1           pop bc
8826 10F0         djnz schleife
8828 C9           ret
8829             end_
```

Graf Elektronik Systeme GmbH
Magnusstraße 13 · Postfach 1610
8960 Kempten (Allgäu)
Telefon: (08 31) 62 11
Teletex: 831804 = GRAF
Telex: 17 831804 = GRAF
Datentelefon: (08 31) 6 93 30

Filiale Hamburg
Ehrenbergstraße 56
2000 Hamburg 50
Telefon: (0 40) 38 81 51

Filiale München:
Georgenstraße 61
8000 München 40
Telefon: (0 89) 2 71 58 58

Öffnungszeiten der Filialen:
Montag – Freitag
10.00 – 12.00 Uhr, 13.00 – 18.00 Uhr
Samstag 10.00 – 14.00 Uhr

Verkauf:
Computervilla
Ludwigstraße 18 b
(bei Möbel-Krügel)
8960 Kempten-Sankt Mang

Öffnungszeiten:
Montag – Freitag
10.00 – 12.00 Uhr, 13.00 – 18.00 Uhr
langer Samstag 10.00 – 14.00 Uhr