A3-Flachbettplotter

*Die folgenden Beiträge sind Auszüge aus einer umfangreichen Projektarbeit des Speziallehrgangs für Mikroelektronik aus dem Jahr 1986. Das Thema, ein **A3-Flachbettplotter**, wurde von den Schülern selbst gewählt; die Hardware haben sie selbst beigesteuert.*

1986 verfügten wir am TGM noch über keine Entwicklungssoftware für die 8051-Familie. Daher war auch ein Assembler/Disassembler zu schreiben, sozusagen ein Abfallprodukt der gesamten Entwicklung. Das hätte den Gesamtaufwand für das halbjährige Projekt zu stark ansteigen lassen, daher wurde nach einer Möglichkeit gesucht, diese Hilfsmittel einfacher zu bekommen. Gefunden wurde in einem Chip-Spezial der in TURBO-PASCAL geschriebene Assembler INLASS, dessen Mnemonics durch einen Pseudobefehl umschaltbar waren. Daher war die Erweiterung auf einen weiteren Prozessortyp, hier auf den 8052, kein unüberwindliches Problem mehr. Während der Assemblerteil durch Modifikation eines vorhandenen Programms entstand, ist der Disassembler eine Eigenentwicklung. Den Abschluß bildet die Beschreibung der Kommunikation zwischen dem 8052 und dem PC.

*Der 8052-BASIC hat den Vorteil eines eigenen BASIC-Interpreters, sodaß Programmentwicklung besonders wenig zusätzlichen Aufwandes bedarf. Siehe auch Beitrag **80C32-Minimodul**.*

Für Interessenten am Gesamtprojekt können wir weitere Unterlagen anbieten:

TGM-LIT-016:	*Auszug aus Chip-Spezial: INLASS-Assembler*
TGM-LIT-017:	*BASIC-Dokumentation für 8052, INTEL*
TGM-LIT-018:	*Datenblatt 8052-BASIC, INTEL*
TGM-LIT-019:	*Gesamtdokumentation Flachbettplotter, inklusive Schaltbild*

Der Mikrocontroller für den Plotter ist auf einer Europakarte 100x160 mm aufgebaut und enthält neben dem Controller 8052 selbst auch einen 32k EPROM-Speicher, 8k RAM und wegen der damit belegten Portleitungen auch einen zusätzlichen 8255 mit insgesamt 32 zusätzlichen IO-Leitungen. Die Platine ist im Prinzip nicht für den Plotter spezialisiert und kann auch für andere Steuerungsaufgaben verwendet werden, die mit dem 8052 gebaut werden sollen.

Auf Anfrage steht kostenlos ein fertiger Print im Europakartenformat 100x160 zur Verfügung. Sollten mehrere Mitglieder sich diesen Mikrocontroller aufbauen wollen, stehen auch die Filme für das Layout zur Verfügung.

8052 - INLINE - ASSEMBLER

Paul KOSTAL, Peter ULLRICH, TGM, NT/EL, SLME86 TGM-DSK-190, TGM-LIT-016,017,018,019
TGM-DSK-190: INLASS.PAS, PARAMSTR.BIB, SCANLINE.INC, ERROR.INC, SUCHBAUM.INC, BERECHNE.INC, HL1-Z80.INC, HL2-6502.INC, HL3-8080.INC, HL4A8086.INC, HL4B8086.INC, HL5-8052.INC, INL.INC, DOPASS.INC, INLOAD52.PAS

Von den Dateien auf TGM-DSK-190 wurden die Dateien DOPASS, INL, INLASS und INLOAD52 gegenüber dem Original in TGM-LIT-016 verändert. Alle anderen Dateien entsprechen dem Original.. Neu ist HL5-8052.INC. Für einen Vergleich finden Sie auch die jeweiligen Originale auf der Diskette: INLOAD.ORI, DOPASS.ORI, INL.ORI, INLASS.ORI und DOPASS.ORI. Die lauffähigen Versionen sind: INLASS.COM und INLOAD52.COM.

1. Allgemeines

Als Grundlage diente der Turbo-Assembler INLASS (TGM-LIT-016). Dieser ist in Turbo-Pascal geschrieben und daher leicht verständlich veränderbar. Der Assembler besteht aus einem Hauptprogramm und mehreren Include-Prozeduren für die Prozessoren Z80, 6502, 8080 und 8086. Diese sollen nun mit dem Prozessor 8052 erweitert werden. Die neue Prozedur erhielt den Filenamen HL5-8052.INC. Diese Prozedur behandelt nur die Umsetzung der Mnemonics in den entsprechenden OpCode. Die Behandlung der Pseudo-OpCodes sowie der Labels erfolgt ausschließich im Hauptprogramm.

Auf Grund des speziellen Adreßformates der Prozessorfamilie MCS-51 und, um die neue Prozedur in den bestehenden INLASS einzubinden, mußten auch die anderen Include-Files (Prozeduren) geringfügig verändert werden (siehe Punkt 8.6.). Das oben erwähnte Adressformat wird nicht wie üblich in der Reihenfolge "Low-Adress, High-Adress" sondern in der Reihenfolge "High-Adress, Low-Adress" im Speicher des MCS-51 abgelegt.

Da der INLASS ein relocatibles (=verschiebares) Programmfile erzeugt, kann dies mit dem Programm INLOAD52 in ein unverschiebbares COM-File umgewandelt werden. Dieses INLOAD ist gleich dem INLOAD für den Z80 (Datei INLOAD.ORI), nur daß hier wieder das andere Adreßformat des MCS-51 berücksichtigt wurde. (siehe 8.4.)

Die Bedienung und die Pseudo-OpCodes des INLASS sind der Beschreibung des INLASS zu entnehmen. Der Befehlsaufbau des MCS-51 und seine Handhabung sind dem Datenbuch (INTEL) des MCS-51 zu entnehmen.

2. Abweichungen von den 8052-Mnemonics

1. Die Befehle AJMP und ACALL wurden nicht berücksichtigt. Grund: Die Sprungadresse dieser Befehle ist teilweise mit dem Opcode verknüpft. Da aber im 1.Pass diese noch nicht bekannt ist und die Prozedur nur im 1.Pass durchlaufen wird, kann der OpCode für diese zwei Befehle nicht generiert werden.

2.Die Befehle LJMP und LCALL wurden auf JMP und CALL umgenannt. Grund: Wegen Punkt 1 ist nun eine Trennung zwischen A.. und L.. nicht nötig.

3. Der relative Sprungbefehl SJMP wurde in JR umbenannt.

4. Bei den Bit-Befehlen ANL und ORL wurde das Komplement-Zeichen "/" durch "!" ersetzt. Grund: Das Zeichen "/" ist vom Assembler schon als PseudoOpCode für die Division reserviert.

3. Reservierte Symbole

Folgende Symbole sind mit fixen Adressen belegt.

3.1. RAM-ADRESSEN

```
Symbol    Name                          Adresse
-------------------------------------------------
ACC       Accumulator                     0E0H
B         B-Register                      0F0H
PSW       Program Status Word             0D0H
SP        Stack Pointer                    81H
DPH       Data Pointer High-Byte           83H
DPL       Data Pointer Low-Byte            82H
P0        Port 0                           80H
P1        Port 1                           90H
P2        Port 2                          0A0H
P3        Port 3                          0B0H
IP        Interrupt Priority Control      0B8H
IE        Interrupt Enable Control        0A8H
TMOD      Timer/Counter Mode Control       89H
TCON      Timer/Counter Control            88H
T2CON     Timer/Counter 2 Control         0C8H
TH0       Timer/Counter 0 (High-Byte)      8CH
TL0       Timer/Counter 0 (Low-Byte)       8AH
TH1       Timer/Counter 1 (High-Byte)      8DH
TL1       Timer/Counter 1 (Low-Byte)       8BH
TH2       Timer/Counter 2 (High-Byte)     0CDH
TL2       Timer/Counter 2 (Low-Byte)      0CCH
RCAP2H    Timer/Counter 2 Capture
          Register (High-Byte)            0CBH
RCAP2L    Timer/Counter 2 Capture
          Register (Low-Byte)             0CAH
SCON      Serial Control                   98H
SBUF      Serial Data Buffer               99H
PCON      Power Control                    97H
```

3.2. BIT-ADRESSEN

```
Symbole: P0, P1, P2, P3, IP, IE, TCON, T2CON, SCON, PSW, ACC,
B (Name und Adresse siehe vorher)

Aufbau der Bit-Adresse:    Symbol.x          (x = 0 - 7)
```

3.3. DIREKTE BIT-ADRESSEN

```
ByteSymbol  Symbol    Name                              Adresse
-----------------------------------------------------------------
TCON.0      IT0       Interrupt 0 Typ-Controllbit         88H
TCON.1      IE0       Interrupt 0 Edge-Flag               89H
TCON.2      IT1       Interrupt 1 Typ-Controllbit         8AH
TCON.3      IE1       Interrupt 1 Edge-Flag               8BH
TCON.4      TR0       Timer 0 Run-Controllbit             8CH
TCON.5      TF0       Timer 0 Overflow-Flag               8DH
TCON.6      TR1       Timer 1 Run-Controllbit             8EH
TCON.7      TF1       Timer 1 Overflow-Flag               8FH

SCON.0      RI        Rceive Interrupt-Flag               98H
SCON.1      TI        Transmit Interrupt-Flag             99H
SCON.2      RB8       Received Bit 8                      9AH
SCON.3      TB8       Transmitted Bit 8                   9BH
SCON.4      REN       Serial Reception Enable             9CH
SCON.5      SM2       Serial Mode                         9DH
SCON.6      SM1       Serial Mode                         9EH
SCON.7      SM0       Serial Mode                         9FH
IE.0        EX0       External Interrupt 0 Enable        0A8H
IE.1        ET0       Timer 0 Overflow-Interr. Enable    0A9H
IE.2        EX1       External Interrupt 1 Enable        0AAH
IE.3        ET1       Timer 1 Overflow-Interr. Enable    0ABH
IE.4        ES        Serial Port Interrupt Enable       0ACH
IE.5        ET2       Timer 2 Overflow-Interr. Enable    0ADH
IE.7        EA        All Interrupts Enable              0AFH
IP.0        PX0       External Interrupt 0 Priority      0B8H
IP.1        PT0       Timer 0 Interrupt Priority         0B9H
IP.2        PX1       External Interrupt 1 Priority      0BAH
IP.3        PT1       Timer 1 Interrupt Priority         0BBH
IP.4        PS        Serial Port Interrupt Priority     0BCH
IP.5        PT2       Timer 2 Interrupt Priority         0BDH

T2CON.0     CP/RL2    Capture/Reload Flag                0C8H
T2CON.1     C/T2      Timer/Counter Select               0C9H
T2CON.2     TR2       Timer 2 Run-Controllbit            0CAH
T2CON.3     EXEN2     External Timer 2 Enable-Flag       0CBH
T2CON.4     TCLK      Transmittclock-Flag                0CCH
T2CON.5     RCLK      Receiveclock-Flag                  0CDH
T2CON.6     EXF2      Timer 2 External-Flag              0CEH
T2CON.7     TF2       Timer 2 Overflow-Flag              0CFH

PSW.0       P         Parity Flag                        0D0H
PSW.1       ---       - reserved -                       ----
PSW.2       OV        Overflow Flag                      0D2H
PSW.3       RS0       Register-Bank Select               0D3H
PSW.4       RS1       Register-Bank Select               0D4H
PSW.5       F0        Flag 0                             0D5H
PSW.6       AC        Auxiliary Carry Flag               0D6H
PSW.7       CY        Carry Flag                         0D7H
```

Die reservierten Symbole können an jeder Stelle als Argument angegeben werden. Der Assembler setzt automatisch die richtige Adresse für das Symbol ein.

Einschränkung!!: Die reservierten Symbole können nicht in EQU-Anweisungen verwendet werden. Dies würde zu der Fehlerausgabe "Unbekanntes Label oder Symbol" führen.

```
        Motor     EQU  P0
        MOV       A,Motor     ->   ist nicht möglich

        Motor     EQU  080H
        MOV       A,Motor     ->   ist möglich

        MOV       A,P0        ->   ist möglich
```

Grund: Im ersten Pass sind die Labels (auch EQU-Definitionen) noch nicht bekannt. Daher kann die Prozedur HL5-8052 noch nicht die zugehörigen Adreßwerte einsetzen, da sie z.B. das Symbol Motor nicht kennt. Erst im zweiten Pass sind die Labels bekannt. Jetzt wird aber die Prozedur nicht mehr aufgerufen, da alle Befehle schon übersetzt sind. Für das Hauptprogramm ist dann aber bei der Zuweisung von P0 zu Motor, P0 nicht bekannt.

4. INLOAD52

Dieses Programm ist ähnlich aufgebaut, wie das in der INLASS-Beschreibung beschriebene INLOAD für den Z80. INLOAD52 macht aus einem mit INLASS assemblierten 8052 Programm ein .COM-File mit Hexcodes ohne Trennzeichen. Dieses ist dann unverschiebbar, wenn absolute Sprünge verwendet wurden. Es wird daher nach Aufruf dieses Programms die Startadresse verlangt. Diese wird in hexadezimaler Form eingegeben ($xxxx). In dezimaler Form werden nur Zahlen bis 32767 akzeptiert. Wird nur Return eingegeben, so wird - wenn vorhanden - die mit ORG definierte Start-Adresse eingesetzt. Ansonsten wird $0000 als Startadresse verwendet.

5. Änderung der bestehenden Include-Files

Folgende Änderungen mußten vorgenommen werden, um die Prozedur HL5-8052 einzubinden. Die Zeilennummern sind dem Listing entnommen.

```
 1. File:   INLASS.PAS
    Zeile:  28
     " Prozessor:
(MPC_Z80,MPC_6502,MPC_8080,MPC_8086,MPC_8052);"
     Zeile:  40
      " {$I HL5-8052.INC} "


 2.  File:   DOPASS.INC
     Zeile:  zw. 25 u. 26 löschen
     Zeile:  105
      " ELSE IF o='.8052' THEN PseudoOp:=p8052 "
     Zeile:  204
      " p8052: Prozessor := MPC_8052; "
     Zeile:  231
      " MPC_8052 : HexLine5(OPCODE,Arg1,Arg2,L)
```

Folgende Änderung mußte auf Grund des schon in Punkt 1. erwähnten speziellen Adreßformats des MCS-51 vorgenommen werden:

```
File:   INL.INC
Zeile:  79
  " IF fc=0 THEN IF Prozessor=MPC_8052 THEN "
  "   z:=z+'$'+HexByte(Hi(e))+'/$'+HexByte(Lo(e)) ELSE "
```

6. Die Schnittstelle zwischen Hauptprogramm und Prozedur

Die Prozedur (HL5-8052) erhält vom Hauptprogramm folgende Parameter:

```
Object
Argument1
Argument2
```

Die Prozedur gibt an das Hauptprogramm einen String mit dem übersetzten Hexcode zurück der folgende Form haben muß:

```
'$'+Hexcode+'/$'+Hexcode+.....
```

Soll ein Argument zurückgesendet werden, so muß nicht getestet werden, ob dies eine Zahl oder ein Symbol ist, sondern es muß lediglich angegeben werden, ob es, je nachdem wie es das Objekt verlangt, ein 8-bit Argument oder ein 16-bit Argument sein soll. Das Hauptprogramm prüft dann, ob das Argument dieser Forderung entspricht. Die Argumentgröße wird folgendermaßen gekennzeichnet:

```
/<   8-bit Argument
/    16-bit Argument
```

Ein 3-Byte Befehl `Obj Arg1,Arg2` muß dann in folgender Form an das Hauptprogramm zurückgegeben werden:

```
'$'+Objcode+'/<'+Arg1+'/<'+Arg2
```

wobei die Reihenfolge von `Arg1` und `Arg2` vom Prozessor und dem jeweiligen Op-Code abhängt. Wird aber schon in der Prozedur sichergestellt, daß sich ein Argument um eine erlaubte Zahl handelt, so kann auch das Argument in der Form '$'+Hexcode zurückgesendet werden.

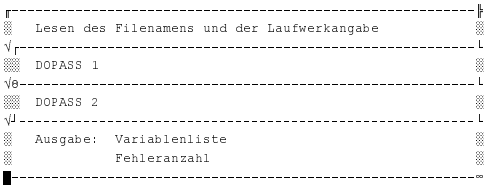
Relative Adressen (Labels) müssen mit

```
'/~'+Adresse bzw. Label
```

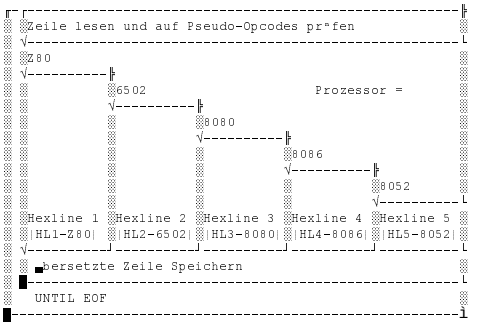
zurückgesendet werden. Absolute Adressen werden wie obige Argumente behandelt.

7. Struktogramme

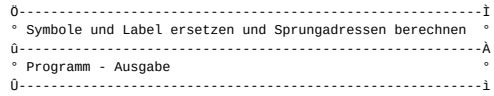
7.1 INLASS



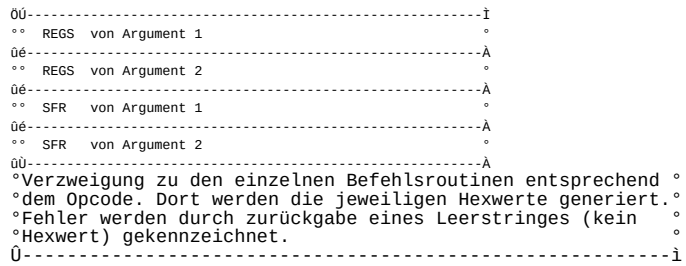
7.2 DOPASS1



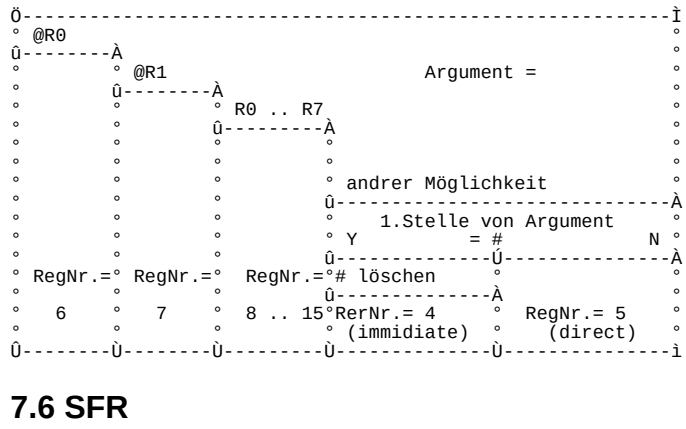
7.3 DOPASS2



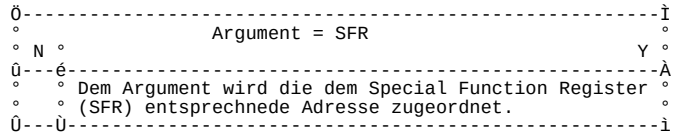
7.4 HEXLINE5



7.5 REGS



7.6 SFR



8. Programmbeschreibung HL5-8052

```
Eingangsvariablen:  o          ->  Mnemonics
                    a1,a2,(a3) ->  Argumente
Ausgangsvariable:   H          ->  Hexcodezeile
```

(F) bzw. (P) bei den Programmnamen bedeuten 'Funktion' und 'Prozedur'.

8.1 OPCodeNummer (F)

```
Aufruf: on:=OpCodeNummer(o)
```

Diese Funktion vergleicht die Eingangsvariable `o` mit den erlaubten Mnemonics des 8052. Bei Gleichheit wird der Variablen `on` je nach Mnemonics ein Wert zwischen 1 und 41 zugeordnet. Trifft kein Vergleich zu, dann wird `on` gleich Null (-> unbekanntes Mnemonics). Die Vergleichs-Mnemonics sind im Konstanten-Feld `OpCodeName` in alphabetischer Reihenfolge gespeichert. Dadurch kann eine sehr schnelle Vergleichsroutine verwendet werden, die mit größer/kleiner Vergleichen arbeitet. Es wird immer verglichen, in welcher Hälfte des Konstantenfeldes das gesuchte Mnemonics liegt. Diese Hälfte wird dann wieder halbiert, und wie vorher verfahren, bis das gesuchte Mnemonics gefunden ist.

8.2 SFR (F)

```
Aufruf: a:=SFR(a)
```

Diese Funktion vergleicht, ob das Argument ein reserviertes Symbol, d.h. ein Special Function Register ist. Das Argument a wird nacheinander mit den SFR-Namen verglichen. Bei Gleichheit wird die zugehörige Adresse aus dem Feld SFRAdr statt dem SFRNamen zurückgegeben.

8.3 Regs (P)

```
Aufruf: Regs(a,R)
```

Diese Prozedur prüft, ob das Argument ein Register, ein indirekter Registerbefehl, ein direktes oder ein immediate Argument ist.

```
R0 - R7        Register
@R0, @R1       indirekt
#xxxx          immediate
xxxx           direkt
```

Je nach den oben angeführten Fällen wird R ein bestimmter Wert zugeordnet. Die Werte ergaben sich aus dem Bitaufbau der OpCodes des 8052:

```
Argument       OpCode          Wert
-------------------------------------
R0 - R7        yyyy 1rrr   ->  8 - 15
@R0, @R1       yyyy 011r   ->  6, 7
#xxxx          yyyy 0100   ->  4
xxxx           yyyy 0101   ->  5
```

yyyy richtet sich nach dem OpCode. Dieser Bit-Aufbau ist nur bei manchen OpCodes anders, die dann gesondert behandelt werden müssen. Der Vorteil dieser Zuordnung ist, daß dann unabhängig vom Argument der High-Teil des OpCodes nur mit R verort werden muß. -> z.B. $20 or R. Dies wird z.B. in der folgenden Prozedur AOP verwendet. xxxx kann eine beliebige 8bit oder 16bit-Zahl sein oder ein Label oder Symbol.

8.4 AOP (A-Operation) (P)

```
Aufruf: AOP(Hex,H)
```

Diese Prozedur prüft zuerst, ob das 1.Argument der Akkumulator ist. Wenn ja, bildet sie nach vorher erwähnter Methode zusammen mit der übergebenen Zahl Hex den OpCode und legt diesen in H ab. Ist R2 (Index2 für 2.Argument) aus vorheriger Prozedur kleiner 6 (-> #xxxx oder xxxx), dann muß noch das 2.Argument xxxx an H angehängt werden. Da die "A,"-Befehle nur mit 8bit-Argumenten arbeiten muß vor xxxx noch /< eingefügt werden (xxxx kann eine Zahl oder ein Symbol sein). Die verwendete Prozedur HexByte(x) erzeugt aus einer Integer einen String.

8.5 HLogik (P)

```
Aufruf: HLogik(Hex,H)
```

Diese Prozedur wird von den Logik-Befehlen ANL, ORL und XRL verwendet. Zuerst wird die Prozedur AOP aufgerufen, da diese auch für alle Logikbefehle mit A gilt. Lag kein A-Befehl vor, so sendet die Prozedur AOP einen Leerstring zurück H="". Ist dies der Fall, so muß Argument 1 ein direct-Argument R1=5 sein, ansonsten liegt eine unerlaubte Anweisung vor. Ist nun R1=5, so gibt es nur mehr 2 Möglichkeiten. Entweder Argument 2 ist A oder ein "immediate"-Argument. In beiden Fällen wird dann der Opcode wie in der Prozedur AOP gebildet und in H abgelegt.

8.6 HBit (P)

```
Aufruf: HBit(Hex,a,H)
```

Diese Prozedur prüft, ob das übergebene Argument a ein erlautes reserviertes Bitsymbol ist. In Hex wird wieder der Hexcode des Mnemonics übergeben, um dann damit den gesamten Opcode zu bilden und in H abzulegen. Der Befehl pos('.',a) sucht in dem String a nach dem Zeichen "." und übergibt die Zeichenstelle, an der dieses Zeichen gefunden wurde (nicht gefunden bedeutet 0). Die Stelle wird in s gespeichert.

1) Wurde ein "." gefunden (S>0) liegt eine reservierte Bit-Adresse vor:

```
Symbol.x            x ... 0 - 7
```

Der Befehl copy(a,1,S) kopiert vom String a S Zeichen ab dem 1.Zeichen in b. Dadurch ist nun in b nur das Symbol vorhanden ohne .x. Mit dem Befehl delete(a,1,S) wird ab dem 1.Zeichen S Zeichen von a weggeschnitten. Dies bedeutet, daß nur mehr x in a bleibt. Da a ein String ist, muß mit dem Befehl val(a,bit,error) daraus eine Integer-Zahl gemacht werden, um x als Zahl zu haben. Dabei wird in bit die Zahl übergeben und in Error der Fehler wie folgt übergeben:

```
String    bit  Error                 B ... Buchstaben
---------------------------           Z ... Ziffern
Z         Z    0
B         0    1
BZ        0    1
ZB        Z    1.Stelle v. B
Z1BZ2     Z1   1.Stelle v. B
```

Nur wenn Error gleich Null ist und bit 0 - 7 ist, war die Syntax richtig. Es wird nun das Symbol b mit den Elementen der Konstante BSFRName verglichen. Trifft ein Vergleich zu, so wird zur Startadresse der BSFR ($80) die Stelle an der das Symbol gefunden wurde s mal 8 addiert, da jedes BSFR 8 Bit beinhaltet. Die Bitstelle wird durch bit x angegeben und wieder addiert. Das Ergebnis ist die Bitadresse für das Eingegebene Symbol x .

2) Wurde kein "." gefunden, so liegt eine direkte reservierte Bit-Adresse oder ein mit EQU definiertes Symbol vor. Im ersten Fall wird String a mit den Elementen der Konstanten BitSFRName verglichen. Bei Gleichheit wird das zugehörige Element der Konstanten BitSFRAdr zur OpCode-Bildung herangezogen. War kein Vergleich zutreffend, so liegt direkt eine Adresse vor, oder ein Symbol, das aber mit EQU definiert werden muß. Es wird daher in diesem Fall das Argument a, so wie es war, an das Hauptprogramm zurückgesendet, zusammen mit dem "<" damit eine 8-Bit Adresse verlangt wird.

8.7 Cbit (P)

```
Aufruf: Cbit(Hex,H)
```

Diese Prozedur wird von den Bit-Befehlen mit C (ANL, ORL) verwendet. Ist das 1.Argument nicht "C", dann liegt ein Logik-Befehl vor, und es wird die Prozedur HLogik aufgerufen, an die der Hexcode unverändert weitergegeben wird. Ist a1 aber C dann wird Argument2 untersucht, ob es ein ! enthält. Dieses wird dann abgetrennt und zusammen mit dem dazugehörigen Hexcode die Prozedur HBit aufgerufen.

BEFEHLSBESONDERHEIT: Der Befehl CJNE benötigt 3 Argumente, da er wie folgt aufgebaut ist:

```
CJNE      a1,a2,a3
```

Das Hauptprogramm trennt zwar das erste Komma weg und zerteilt in a1 und a2, aber prüft nicht, ob noch ein zweites Komma folgt. (Prüft nur auf ein folgendes Semikolon zur Kommentartrennung). Es wird daher a2, a3 in dem Argument2 vom Hauptprogramm übergeben. Es muß daher nun a2 erst in a2 und a3 aufgeteilt werden. Dies geschieht dadurch, daß mit pos die Stelle des Kommas in a2 gesucht wird und dann der Teil hinter dem Komma mit copy in a3 kopiert wird und der hintere Teil mit dem Komma dann aus a2 gelöscht wird.