8052 - DISASSEMBLER

Paul KOSTAL, Peter ULLRICH, TGM,NT,SLME86 DISAS52.PAS, DIASSEM.INC, DISAS52.COM

Mit DISASS51 kann von jedem beliebigen COM-File ein Hexdump erstellt werden oder dieses disassembliert werden, wenn es im MCS-51 Code geschrieben ist.

Der Aufruf erfolgt folgendermaßen:

```
DISAS-52  filename
```

Der Filename wird ohne Extension angegeben, da immer ein COM - File gelesen wird. Wird kein Filename angegeben, so wird dieser nachträglich vom Programm angefordert. Es können aufgrund der im Pascal verwendeten INTEGERS nur Programme bis zu einer Länge von 32767 Bytes eingelesen werden.

Als Nächstes muß die Adresse angegeben werden, auf die das eingelesene Programm disassembliert werden soll.

Folgendes Menü wird nun angezeigt:

```
Ö--------------------------------------------------Ì
°                  Disassembler-Befehle            °
°                  --------------------            °
°                                                  °
°                                                  °
°  D            Disassembliere ab Adresstand       °
°  D XXXX       Disassembliere ab Adresse xxxx     °
°  M            Hexdump ab Adresstand              °
°  M XXXX       Hexdump ab Adresse xxxx            °
°  ^S           Anhalten/Weiter                    °
°  RETURN       Stopt Funktion                     °
°  PgUp         Seite vor                          °
°  PgDn         Seite rückwärts                    °
°  Home         Anfang                             °
°  End          Ende                               °
°  ? od. a      Zeigt dieses Menü                  °
°  QUIT         Ausstieg                           °
Û--------------------------------------------------ì
```

Ist man in dem Menüpunkt Disassembliere, so kann man die Steuertasten PgUp, Home, End nur begrenzt verwenden:

Home: Der Disassembler beginnt nicht am Programmanfang zu disassembieren, sondern ab der Adresse, die beim Aufruf dieses Menüpunkts gerade aktuell war (oder die durch xxxx angegebene).

End: Der Disassembler springt nicht an das Programmende, sondern an die bis zu diesem Zeitpunkt höchste, vorgekommene Adresse (Diese wird durch Drücken von PgDn erhöht). Ein Verlassen dieses Menüpunktes löscht aber wieder diesen Wert.

PgUp: Diese Funktion wird nur solange ausgeführt, bis die Adresse der Home - Taste erreicht ist.

Grund für die eingeschränkte Verwendung der Steuertasten: Normalerweise kann ein Disassembler nicht zu kleineren Adressen hin disassemblieren, da die Byteanzahlen sowie der OpCode selbst hier nicht eindeutig sind. So würde der Befehl:

```
74H 00H MOV A,#0
```

verkehrt gelesen ein NOP ergeben.

Der Disassembler speichert daher alle ersten Adressen pro disassemblierter Seite ab. Die erste abgespeicherte Adresse wird für die Home-Taste und die letzte abgespeicherte Adresse für die End-Taste verwendet. Die PgUp-Taste nimmt jeweils die nächstkleinere abgespeicherte Adresse, solange bis die erste erreicht ist.

1. Reservierte Symbole

Der Disassembler DISASS52 hat alle Special-Function-Register des 8052 implementiert. Tritt eine 8bit-Datenadresse auf, die auf ein SFR zutrifft, so wird dieses SFR-Symbol als Kommentar in Klammer an den jeweiligen Befehl angehängt. Handelt es sich bei dem Befehl um einen Bit-Befehl, so wird als Kommentar die zugehörige Byteadresse und die Bitposition (0 - 7) ausgegeben oder, falls es sich um ein fixes Bitsymbol handelt, dieses zugeordnet.

2. Änderungen gegenüber den 8052-Mnemonics

Da dieses Programm zusammen mit dem Inlass-8052 entwickelt wurde, wurden hier fast die gleichen Änderungen vorgenommen:

```
1. LCALL und LJMP  ->  CALL und JMP
2. SJMP            ->  JR
```

Im Gegensatz zum Inlass wurde hier als Negationszeichen wieder das "/" verwendet und die Befehle ACALL und AJMP wieder berücksichtigt, da diese hier keine Schwierigkeiten bereiten.

3. Relative Sprünge

Alle relativen Sprungadressen und auch die ACALL- und AJMP-Adressen werden sofort berechnet und als absolute Adresse ausgegeben. Dies hat den Vorteil der besseren Übersicht, und man erspart sich das mühsame Berechnen der Adressen, um einen Programmverlauf zu verfolgen. Der relative Sprungwert ist außerdem auch aus den Hexcodes ersichtlich.

4. Beispielausdrucke

Das Beispielprogramm BEISP wurde folgendermaßen erstellt:

```
1. Texteditor    ->   BEISP.SOU
2. INLASS        ->   BEISP.PRN
                 ->   BEISP.INL
3. INLOAD52      ->   BEISP.COM
4. DISAS-52
```

BEISP.SOU

```
Ö-----------------------------------------------------------Ì
°          .8052                                            °
°STROUT    EQU     6                                        °
°          ORG     3000H                                    °
°;                                                          °
°          MOV     A,#STROUT       ;Ausgabe eines Strings   °
°          MOV     R3,#HI(MLDG)    ;HighAdresse des Strings °
°          MOV     R1,#LO(MLDG)    ;LowAdresse des Strings  °
°          CALL    30H             ;Befehl ausführen        °
°          RET                     ;Zurück zum Basic        °
°;                                                          °
°MLDG      DB  'Dies ist ein Beispiel'                      °
°          DB  00H                                          °
Û-----------------------------------------------------------ì
```

BEISP.PRN

```
PASS 1
0 Fehler aufgetreten.
PASS 2
*0000                       .8052
*0000 (0006)                    STROUT    EQU 6
*0000                                 ORG 3000H
*3000
*3000 /$74/$06                        MOV A,#STROUT     ;Ausgabe eines Strings
*3002 /$7B/$30                        MOV R3,#HI(MLDG)  ;HighAdresse des String
*3004 /$79/$0A                        MOV R1,#LO(MLDG)  ;LowAdresse des String
*3006 /$12/$00/$30                    CALL 30H          ;Befehl ausführen
*3009 /$22                            RET               ;Zurück zum Basic
*300A
*300A /$44/$69/$65/$73/$20/$69        MLDG     DB 'Dies ist ein Beispiel'
*3010 /$73/$74/$20/$65/$69/$6E
*3016 /$20/$42/$65/$69/$73/$70
*301C /$69/$65/$6C
*301F /$00                            DB 00H
0 Fehler aufgetreten.

MLDG     * 300A
STROUT   = 0006
```

Ausdrucke des DISAS-52

(Alle Zahlenwerte sind Hexadezimal!!)

1. Hexdump

```
Ö-------------------------------------------------------Ì
°3000  74 06 7B 30 79 0A 12 00 30 22 44 69 65 73 20 69°
°      t  .. {  0  y  .. .. .. 0  "  D  i  e  s     i °
°3010  73 74 20 65 69 6E 20 42 65 69 73 70 69 65 6C 00°
°      s  t     e  i  n     B  e  i  s  p  i  e  l  .. °
°3020  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00°
°      .. .. .. .. .. .. .. .. .. .. .. .. .. .. .. ..°
°3030  00 00 00 00 00 00                              °
°      .. .. .. .. .. ..  .....                      °
°                                                     °
Û-------------------------------------------------------ì
```

2. Disassembler

```
Ö-------------------------------------------------------Ì
°3000  74 06      MOV   A,#06                   t     °
°3002  7B 30      MOV   R3,#30                  {0    °
°3004  79 0A      MOV   R1,#0A                  y     °
°3006  12 00 30   CALL  0030                      0   °
°3009  22         RET                           "     °
°300A  44 69      ORL   A,#69                   Di    °
°300C  65 73      XRL   A,73                    es    °
°300E  20 69 73   JB    69,3084    ;(2D.1)       is   °
°3011  74 20      MOV   A,#20                   t     °
°3013  65 69      XRL   A,69                    ei    °
°3015  6E         XRL   A,R6                    n     °
°3016  20 42 65   JB    42,307E    ;(28.2)       Be   °
°3019  69         XRL   A,R1                    i     °
°301A  73         JMP   @A+DPTR                 s     °
°301B  70 69      JNZ   3086                    pi    °
°301D  65 6C      XRL   A,6C                    el    °
°301F  00         NOP                                 °
°3020  00         NOP                                 °
°3021  00         NOP                                 °
°3022  00         NOP                                 °
°3023  00         NOP                                 °
°                                                     °
°ASCII-Text:                                          °
°t {0y   0"Dies ist ein Beispiel                      °
Û-------------------------------------------------------ì
```

Da der Disassembler zwischen Daten- und Programmbereich nicht unterscheiden kann, werden OpCodes auch als ASCII und ASCII-Daten (oder andere Daten) auch als OpCodes dargestellt. (siehe die ASCII-Daten: "Dies ist ein Beispiel" bei denen die danebenstehenden Befehle sinnlos sind. Genauso die neben dem Programm stehenden Buchstaben.)

5. Struktogramm

```
Ö-Ú-----------------------------------------------------Ì
° ° Filename von Tastatur lesen                        °
° Û-----------------------------------------------------À
°   UNTIL  File exists                                 °
û-------------------------------------------------------À
°   File in Puffer einlesen                            °
ûÚ------------------------------------------------------À
°°  Menübild ausgeben                                  °
ûÙÚ-----------------------------------------------------À
° ° Tastatur einlesen                                  °
° û-----------------------------------------------------À
° °  D                                                 °
° ûÚ-------------Ì                    Taste =         °
° °°            °  H                                  °
° °°            ûÚ-------------Ì                      °
° °°            °°             °  ? od. ß             °
° °° Adresse holen°° Adresse holenûÚ-------------Ì     °
° ûé-------------éé-------------À°             û-------À
° °° Disassemb   °° Hexdump     °° Menübild    ° Fehler °
° ÛÙ-------------ÙÙ-------------ÙÙ-------------Ù-------À
°   UNTIL  Taste = 'quit'                              °
Û-------------------------------------------------------ì
```

HEXDUMP

```
Ö--Ú----------------------------------------------------Ì
°  ° Ausgabe der Adresse in Hex                        °
°  û----------------------------------------------------À
°  °    FOR Zähler = 1 bis 16                          °
°  °  Ö---------------------------------------------- À
°  °  ° Ausgabe Hexzahl aus Puffer                    °
°  û--Ù----------------------------------------------- À
°  °    neue Zeile                                     °
°  û----------------------------------------------------À
°  °    FOR Zähler = 1 bis 16                          °
°  °  Ö---------------------------------------------- À
°  °  ° Ausgabe ASCII-Charakter aus Puffer            °
°  û--Ù----------------------------------------------- À
°  °    neue Zeile                                     °
°  û----------------------------------------------------À
°  ° Tastatur einlesen                                 °
°  û----------------------------------------------------À
°  ° PgUp                                              °
°  û---------------Ì                          Taste =  °
°  °               ° PgDn                              °
°  °               û--------------Ì                    °
°  ° Pointer eine  ° Pointer eine  ° Home              °
°  ° Seite zurück  ° Seite vor     û-----------Ì       °
°  û---------------é--------------À           ° End    °
°  °   Pointer<0   °  Pointer>Pgr- °           û----------Ì°
°  ° N          Y ° N   ende    Y °           °          ûÀ
°  û---Ú----------é--Ú------------À           ° Pointer= °°
°  °   ° Pointer=0 °  °Pointer=ende° Pointer=0 ° Pgr.ende °°
°  Û---Ù----------Ù--Ù------------Ù-----------Ù----------ÙÀ
°   UNTIL  Taste = RETURN                              °
Û-------------------------------------------------------ì
```

DISASSEMB

6. Programmbeschreibung

(F)...Funktion, (P)...Prozedur

6.1. HEX (F)

```
Aufruf:      String:=HEX(Integer)
Stringlänge: 2
Integer:     0 - 255
```

Eine ganze Zahl kleiner 256 wird in einen Hexstring umgewandelt. Die Zahl wird in High- und Lownibble (Division durch 16) zerlegt. Man erhält jeweils eine Zahl zwischen 0 und 15. Entsprechend dieser wird dann aus dem Konstantenfeld `hd` das zugehörige Zeichen ausgelesen und in `HEX` zurückübergeben.

6.2 HEX2 (F)

```
Aufruf:      String:=HEX2(Integer)
Stringlänge: 4
Integer:     0 - 65535 (wird als positive Zahl gesehen)
```

Eine ganze Zahl wird in einen Hexstring umgewandelt. Da in Pascal Integers nur bis +32767 gehen, und ein Erhöhen dieser zu negativen Zahlen führt (-32768), muß zuerst einmal festgestellt werden, ob die Zahl positiv oder negativ ist. Die Zahl wird dann in High- und Lowbyte aufgeteilt (Division durch 256). War die Zahl positiv, so werden High- und Lowbyte mit der Funktion `HEX` jeweils in einen Hexstring umgesetzt und dann zusammengefügt zurückgegeben. War die Zahl aber negativ, so muß von der Zahl das 2er-Komplement gebildet werden. Durch den begrenzten Wertebereich des Pascal, muß dies aber extra für das Highbyte (1er-Komplement) und das Lowbyte (2er-Komplement) erfolgen, wobei der Übertrag zu beachten ist.

6.3 GetLoHi (P)

```
Aufruf:      GetLoHi(String,Integer1,Integer2)
Stringlänge: >4         eingang
Integer:     0 - 255    ausgang
```

Der String wird in eine positive Zahl umgewandelt. Auf Grund des begrenzten Wertebereiches für Integers werden Highbyte und Lowbyte getrennt umgewandelt und zurückgegeben.

6.4 GetAdr (P)

```
Aufruf:  GetAdr
Var. Befehl:   String:  1 Steuerzeichen    (Adresse)
```

Diese Routine prüft, ob die Variable `Befehl` eine Adresse beinhaltet. Enthält sie keine, oder lautet der Befehl 'quit' so wird diese Routine beendet. Ansonsten wird mit `GetLoHi` die Adresse geholt und verglichen, ob sie kleiner als die Startadresse des eingelesenen Programms ist. Wenn ja, wird die Adresse gleich 0 gesetzt, da intern der Zähler von 0 an läuft. War die Adresse größer, so wird die Startadresse des eingelesenen Programms abgezogen. (Grund wie vorher -> Zähler von 0 weg).

6.5 GetBefehl (P)

```
Aufruf: GetBefehl
```

Diese Routine erzeugt einen blinkenden Pfeil und liest anschließend von der Tastatur in den String `Befehl` ein. Von diesem werden dann alle Spaces gelöscht und das erste Zeichen (Steuerzeichen) in die Variable `Taste` gespeichert und in einen Großbuchstaben umgewandelt.

6.6 Hexdump (P)

```
Aufruf: Hexdump
```

Diese Routine erzeugt einen Hexdump (24 Zeilen). Wurde als Hexdumpadresse eine größere angegeben, als das Programm lang ist, so wird die Adresse auf die letzte Adresse gesetzt, die noch einen vollen Bildschirm `Hexdump` ermöglicht. Diese Adresse ergibt sich aus der letzten (höchsten) Adresse - 12 Zeilen * 16Bytes. Es folgt die Ausgabe der Adresse und der 16 Hexbytes und dann die Ausgabe der 16 zugehörigen ASCII-Zeichen, die wieder auf druckbare Zeichen geprüft werden. Ist die 24.Zeile erreicht, so wird die Tastatur abgefragt. Erfolgt eine unzulässige Eingabe (z.B. Drücken von `PgUp` wenn man schon auf der erstmöglichen Adresse ist), so wird die Ausgabe gesperrt (`ausg:=false`), und es erfolgt eine neue Tastaturabfrage. Dies geschieht solange, bis mit `RETURN` die Routine abgebrochen wird.

6.7 Disassem (P)

```
Aufruf: Disassem
```

6.7.1 Zeile (F)

```
Aufruf:          String1:=Zeile(Integer,String2)
Stringlänge1,2:  3
Integer:         0 - 65535 (pos.)
```

Es wird ein String zurückgegeben, der das Hexbyte der Adresse (Integer) enthält. In `String2` wird der ASCII-Charakter dieses Hexbytes zu den Bestehenden in `String2` dazugefügt und zuerst auf druckbare Zeichen geprüft. Weiters wird der Adreßzähler um eins erhöht.

6.7.2 SFR8 (F)

```
Aufruf:      String:=SFR8(Integer)
Stringlänge: 6
Integer:     1,2
```

Diese Routine vergleicht den Hexwert mit den Special-Function-Registern SFR und ordnet dem String entweder ein SFR oder einen Leerstring zu. Einem Wert kleiner 128 wird sofort ein Leerstring zugewiesen, da die SFR erst bei 128 beginnen. Der Wert `Integer` wird von dem derzeitigen Stand des Adreßzählers abgezogen (Da der Adreßzähler schon auf den nächsten Befehl zeigt). Das Ergebnis weist dann auf das gewünschte Hexbyte. `Integer` gibt daher an, ob das letzte oder vorletzte Byte verglichen werden soll.

6.7.3 SFR1 (F)

```
Aufruf:      String:=SFR1(Integer)
Stringlänge: 6
Integer:     1,2
```

Diese Routine vergleicht den Hexwert mit den Bit-Symbolen und ordnet dem `String` entweder ein Bit-Symbol oder einen Leerstring zu. Bei einem Wert kleiner 128 wird dem String die Byteadresse zugeordnet, die aus dem Hexwert (dieser ist in diesem Fall eine Bitadresse) errechnet wird (Bitwert/8 + 20h). Der Wert `Integer` wird von dem derzeitigen Stand des Adreßzählers abgezogen, da der Adreßzähler schon auf den nächsten Befehl zeigt. Das Ergebnis weist dann auf das gewünschte Hexbyte. `Integer` gibt daher an, ob das letzte oder vorletzte Byte verglichen werden soll.

6.7.4 Register (F)

```
Aufruf:      String:=Register(Integer)
Stringlänge: 4
Integer:     0 - 15
```

Diese Routine liefert abhängig vom Integerwert entweder ein Register (8-15), Indirekt Register (6,7), immediate Data (4) oder Data (5) zurück.

6.7.5 SprAdr (F)

```
Aufruf:      String:=SprAdr(Integer)
Stringlänge: 4
Integer:     0 - 224  step 32  (-> nur Bits 5-7)
```

Diese Routine berechnet die absolute Sprungadresse für ACALL- und AJMP-Befehle. Die Adresse ergibt sich folgendermaßen:

```
(Bits 11-15 des Adresszählers)+(Bits 5-7 des OpCode)+Argument
Adr.: 15-11                    10-8              7-0
```

Der OpCode wird mit `Integer` übergeben und das Argument aus dem `Pufer+1` geholt. Durch Division des Opcodes durch 32 werden die Bits 5-7 auf 0-3 transferiert und durch Multiplikation mit 256 auf die Bits 10-8. Die Adresse wird mit 11111000 00000000B verundet, um nur die Adreßbits 11-15 zu haben.

6.7.6 RelAdr (F)

```
Aufruf: RelAdr
```

Diese Routine berechnet die absolute Sprungadresse für relative Sprünge. Je nach dem, ob der Hexwert (rel. Adr) >127 ist oder nicht, wird der Wert zum Adreßstand addiert oder sein 2er Komplement subtrahiert.

6.7.7 OneByte - TwoByte - ThreByte (P)

```
Aufruf: Onebyte(String), Twobyte(String), Threbyte(String)
Stringlänge: 10
```

Es wird die Funktion Zeile so oft, wie die Byteanzahl aufgerufen und der Rest mit Spaces aufgefüllt, sodaß alle 3 Prozeduren einen gleich langen String liefern, der die Hexbytes des Befehls enthält. Zusätzlich wird durch die Funktion `Zeile` ein String ASCII gebildet, der die ASCII-Zeichen dieser Bytes beinhaltet.

Hauptprogramm von Disassemb

Es wird zuerst der Opcode entsprechend der derzeitigen Adresse eingelesen. Der Programmteil wird aufgeteilt in ACALL- und AJMP-Befehle, in Registerbefehle und in die restlichen Befehle.

Die ACALL- und die AJMP-Befehle werden dadurch erkannt, daß die Bits 0-4 gleich 00001b=1 (->AJMP) oder gleich 10001b=17 (->ACALL) sind.

Die Registerbefehle werden dadurch erkannt, daß das Lownibble des OpBytes = 6-15 ist. Ist dies der Fall, so wird nur für jede Registergruppe mit Hilfe des Highnibble des OpBytes der Befehl decodiert.

Alle anderen OpBytes werden direkt verglichen und einzeln decodiert.

Ist die 21.Zeile erreicht, so wird der ASCII-Text aller Hexcodes am Bildschirm ausgegeben. Danach wird die Tastatur abgefragt. Wurde PgDn gedrückt, so wird ab der letzten Adresse weiter disassembliert und diese Adresse in einem Puffer gespeichert, um für die Steuertaste PgUp zur Verfügung zu stehen, welche eine 'Seite' zurückblättert. Erfolgt eine unzulässige Eingabe (z.B. Drücken von PgUp, wenn man schon auf der erstmöglichen Adresse ist), so wird die Ausgabe gesperrt (`ausg:=false`), und es erfolgt eine neue Tastaturabfrage. Dies geschieht solange, bis mit RETURN die Routine abgebrochen wird.