SIEMENS

Since April 1, 1999, Siemens Semiconductor is Infineon Technologies. The next revision of this document will be updated accordingly.

Microcontrollers

ApNote AP0827

☑ : Additional file AP082701.EXE available

ADIS51 - Disassembler with One-Line Assembler for the 8051 and C500 8-Bit Microcontroller Family

ADIS51 allows the analysis of program code for the microcontrollers of the 8051/C500 family. Program code in different formats can be loaded and displayed in Hex-dump or Disassembler format. Single instructions can be input using the built-in Assembler. Program code flow and instructions can be analysed and written into a Log-file in a list or assembler source format. SFR and bit symbol operation is provided using a symbol definition file.

Richard Schmid, Microcontroller Product Definition, Siemens HL Munich

Contents **Page**

AP0827 ApNote - Revision History		
Actual Revision : 10.97 Rel. 01 Previous Revision : none (Original Version)		
Page of actual Rev.	Page of prev.Rel.	Subjects (changes since last release)

1 Starting of ADIS51

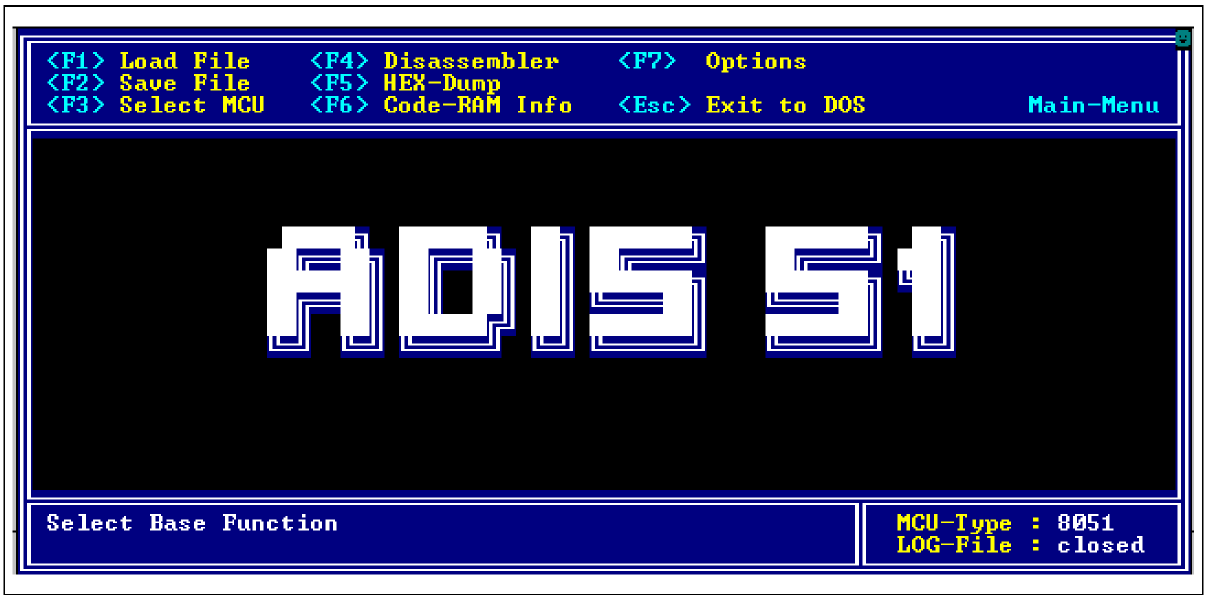
The program ADIS51.EXE is an MS-DOS program. It can be started under MS-DOS or in an MS-DOS window of a Windows operating system. If no parameter is added, ADIS51 comes up in a 25-line/80-columns text mode. It is also possible to initiate the program using a 43- or 50-line text mode. This mode is selected calling the program by ADIS51 /43 .

After ADIS51 has been started the title screen is displayed. Pressing any further key will turn on the main menu screen.

When ADIS51 is loaded, the program searches for a file called ADIS51.SYM, which contains the device, the special function register, and the bit symbol definitions. If this file is not present in the directory where ADIS51 is located and started, ADIS51 will operate without any microcontroller specific register or bit symbol. This capability of symbol definitions in an external file allows a very flexible adaption of ADIS51 to all types of the 8051/C500 family microcontrollers with their different SFR and bit symbols. The syntax of the symbol definition file is shown in appendix C of this ApNote.

2 Main Menu

Figure 1 shows the main menu in 25-line mode.



```
<F1> Load File     <F4> Disassembler     <F7>  Options
<F2> Save File     <F5> HEX-Dump
<F3> Select MCU    <F6> Code-RAM Info    <Esc> Exit to DOS               Main-Menu
```

```
Select Base Function                                      MCU-Type  : 8051
                                                          LOG-File  : closed
```

Figure 1 :
Main Menu Screen of ADIS51

The main menu screen of ADIS51 is divided into three sections :

- The upper area of the screen (3 lines), **the help area**, shows the actual available basic functions of the program and the related key which is used to select these basic functions.
- The middle area of the screen, **the display area**, is used to display the content of the 64K byte memory buffer in the Disassembler or Hex-dump menu. The number of text lines of this area depends on the text mode, in which ADIS51 is started (16, 34, or 41 lines).
- The lower area of the screen (2 lines), **the status area**, is used to display status information and error messages (see Appendix A). File names and addresses are also input in this screen area. The right part of the status area shows the actual selected type of microcontroller, as defined in the symbol definition file, and the state of the Log-File (closed or open).

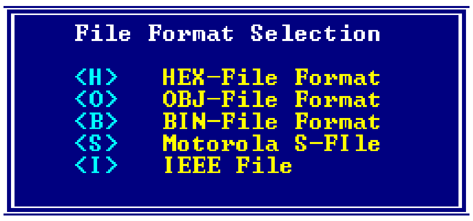
The seven basic functions of ADIS51 can be selected from the main menu. These basic functions are assigned to the function keys F1 to F7 as follows :

<F1>	Loading of a data file in different formats into the ADIS51 memory buffer
<F2>	Writing ADIS51 memory buffer contents into a file using HEX- or BIN- format
<F3>	Selection of a microcontroller type (and its symbols)
<F4>	Starting the ADIS51 Disassembler / One-Line Assembler function
<F5>	Starting the ADIS51 Hex-dump function
<F6>	Display of the ADIS51 memory buffer usage (Code RAM Info)
<F7>	Selection of different options

Pressing **<Esc>** in the main menu will terminate the ADIS51 program and return to DOS. Pressing any other key in the main menu results in an optical (error message) and acustical warning.The following chapters describe each of the seven main menu basic functions in detail.

3 Basic Function <F1> : Loading of a Data File

ADIS51 provides a 64K byte memory buffer which is assigned to be used as program memory buffer for instructions or data, which are generated by assembler and compiler programs. These instructions and data are normally stored with absolute addresses in data files using different formats. **Figure 2** shows the types of data formats which can be loaded by ADIS51. HEX- and OBJ-file formats are the Intel type of format.



```
File Format Selection

<H>    HEX-File Format
<O>    OBJ-File Format
<B>    BIN-File Format
<S>    Motorola S-FIle
<I>    IEEE File
```

Figure 2 :
Data File Load Selection Window

Pressing **<F1>** in the main menu of ADIS51 opens the window above and requests an input for the selection of the data file format. After the selection of one data file format, ADIS51 requests for the input of the file name of the data file. Additionally, for the BIN-file format a memory buffer start address is requested which defines the memory buffer address, where the binary data is placed (with incrementing memory buffer address). The other data file formats all provide address information whithin the data file which is used to locate the data file contents in the memory buffer.

The path of the data files names to be input must be referenced to the directory where ADIS51 is located in either relative or absolute format. When a data file has been found and opened, ADIS51 analyses its content and transfers the relevant data into the memory buffer. During this transfer each byte of the memory buffer, which is loaded by a byte of the data file, is marked. This marking is important for the code analysis capability. Further, the bytes which have been loaded, are displayed in yellow color in the disassembler and hex-dump function.

After the loading of a data file content, the Code RAM Info function (**<F6>** in the main menu) gives an overview of the memory buffer usage. Prior to loading of the memory buffer with the contents of a data file, the memory buffer is completely written with 00_H. This memory buffer initialization feature can be switched off by an option (**<F7>** in the main menu).

Remarks for HEX-, OBJ-files :
HEX-files are scanned for records with record type 00_H. Only code or data information which is placed in these records is transferred into the memory buffer. From OBJ-files only records with record type 05_H are transferred into the memory buffer. Symbol informations stored in HEX- and OBJ-files are not used by ADIS51.

Invalid data file formats and checksum errors in data files are detected and generate an error message in the status area.

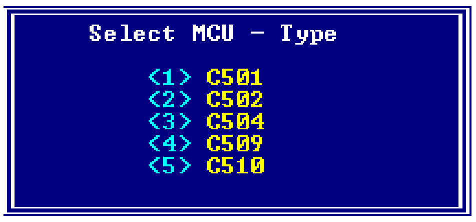
4 Basic Function <F2> : Writing Memory Buffer Contents into a HEX- or BIN-Files

This function allows to generate data files with the content of the memory buffer. It is invoked by pressing **<F2>** in the main menu. After the input of the data file name to be generated, start and end address must be input. If a data file already exists, a warning message occurs and it must be selected whether the data to be transferred should overwrite the old information in the existing data file or if it should be appended to an existing data file. This allows to store several memory buffer parts in one data file.

All address values must be input as hexadecimal numbers. For generated HEX-files, an "End-of-File" record “:00000001FF“ is automatically appended to a data block which has been written into the data file. Therefore, if multiple memory buffer blocks are written into one HEX-file, the "End-of-File" records should be deleted manually (except the last one) by using e.g. a text editor.

5 Basic Function <F3> : Selection of a Microcontroller Type

This function allows to select a microcontroller device with its related SFR- and bit symbols. This function is only available, if the device with its symbols has been defined in the symbol definition file ADIS51.SYM. In the example of **figure 3** five devices have been defined.



```
Select MCU - Type

    <1> C501
    <2> C502
    <3> C504
    <4> C509
    <5> C510
```

Figure 3 :
Microcontroller Type Selection Window

After the selection of the microcontroller type the name of the microcontroller is displayed in the right corner of the status area. Also all SFR and bit symbols, which are defined for the selected type of MCU in the symbol definition file ADIS51.SYM, are activated.

6 Basic Function <F4> : Disassembler / One-Line Assembler Menu

6.1 General Operation

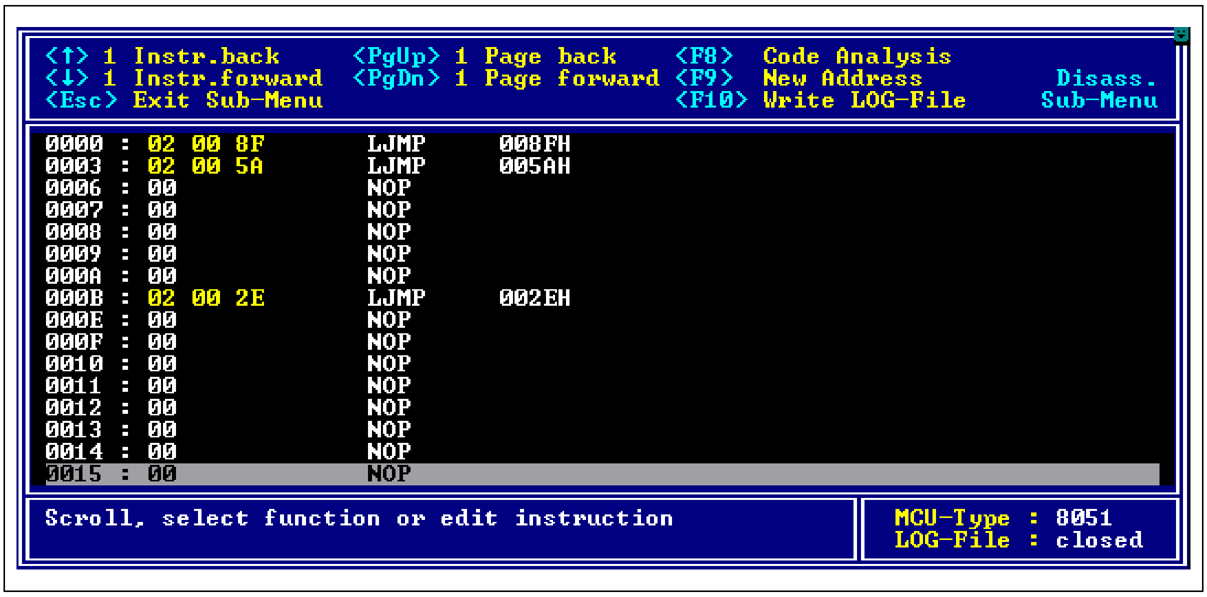
After pressing **<F4>** in the main menu and after the input of a start address, the disassembler window with its related menu is opened. This function is the main part of the ADIS51 program. It allows to display the contents of the memory buffer as 8051 instruction mnemonics and to alter or enter single instructions by using the one-line assembler capability. Additionally, a built-in code analysis capability makes it easy to check an unknown content of a data file, which has been loaded into the memory buffer.

When entering the disassembler menu, ADIS51 begins a linear disassembling of 2000 instructions located in the memory buffer starting from the start address. The first instructions are then output in the display area. The instruction, which is located at the start address, appears in a (grey) highlighted scrollbar with the cursor placed at the first character of the instruction. The highlighted instruction can be modified or the scrollbar can be moved using the cursor keys :

<Cursor Up>	moving scrollbar one instruction back	(optional scroll screen)
<Cursor Down>	moving scrollbar one instruction forward	(optional scroll screen)
<Page Up>	moving scrollbar one screen page back	
<Page Down>	moving scrollbar one screen page forward	

It is not possible to move the scrollbar in the screen area directly to an address which is less than the start address. For this operation a new start address must be defined (using **<F9>**). Further, the scrollbar cannot be moved behind the last (of the 2000) instructions which have been disassembled.

Figure 4 shows an example of the disassembler menu. Code bytes in yellow colour have been loaded from a data file before. White coloured bytes are spare memory buffer bytes. After a execution of a code analysis, cyan coloured bytes indicate bytes, which have been already analysed .



```
0000 : 02 00 8F      LJMP    008FH
0003 : 02 00 5A      LJMP    005AH
0006 : 00            NOP
0007 : 00            NOP
0008 : 00            NOP
0009 : 00            NOP
000A : 00            NOP
000B : 02 00 2E      LJMP    002EH
000E : 00            NOP
000F : 00            NOP
0010 : 00            NOP
0011 : 00            NOP
0012 : 00            NOP
0013 : 00            NOP
0014 : 00            NOP
0015 : 00            NOP
```

Scroll, select function or edit instruction

MCU-Type : 8051
LOG-File : closed

Figure 4 :
Disassembler/One-Line Assembler Menu of ADIS51

The one-line assembler capability accepts opcodes and operands as input of instructions as they are defined in **Appendix B**. For the one-line assembler 8- and 16-bit values of an instruction can be input either in decimal or hexadecimal notation, but they are always output as hexadecimal values.

If SFRs and bit symbols are defined for the active microcontroller device (see chapter 5), the disassembler displays SFR addresses and bit addresses with its symbol. The one-line assembler also accepts symbol names as an input for an operand. SFR and bit symbol output/input can be switched off by an option (see chapter 9).

After an instruction has been modified or input by using the cursor left/right keys and the **`<Del>`** key, it is assembled pressing the **`<Return>`** key. If the instruction has a correct format, its code bytes are written into the memory buffer location where the scrollbar is located. After this memory buffer update, the instructions which follow the actually assembled instruction are re-disassembled. Therefore, it may occur that the instruction flow is changed for the instructions, which are located directly behind the actually assembled instruction (e.g. replacing a 3-byte instruction by a 1-byte instruction).

Three more function keys are available in the disassembler menu :

`<F8>`	Analysis of program code in the memory buffer
`<F9>`	Input and definition of a new start address for disassembling
`<F10>`	Open/append/close a Log-file
`<Esc>`	Back to the Main Menu

6.2 Memory Buffer Code Analysis

When a data file is loaded into the memory buffer, sometimes its content is unknown but wants to be analysed and or changed and reassembled. This capability is supported with ADIS51 too in the disassembler menu.

One of the main difficulties in the analysis of a 8051 program code is to seperate code and data which is placed together in the 64K program memory area. The code analysis capability provides a half-automatical tracing of instructions. The analysis is started at the instruction which is just indicated with the scrollbar. The program code is analysed sequentially and each instruction which has been analysed is marked. Also destination addresses of jump and call instruction are marked and sequentially analysed. **Figure 5** shows the code analyse option window which is opened when **<F8>** is pressed in the disassembler menu.

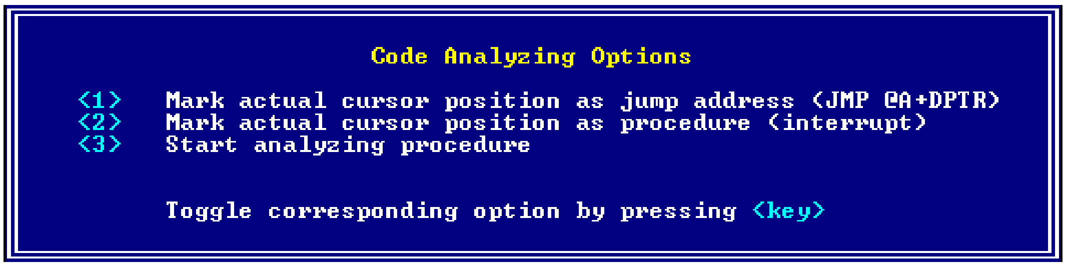

```
                    Code Analyzing Options

   <1>     Mark actual cursor position as jump address (JMP @A+DPTR)
   <2>     Mark actual cursor position as procedure (interrupt)
   <3>     Start analyzing procedure


           Toggle corresponding option by pressing <key>
```

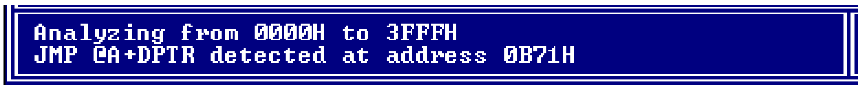
Figure 5 :
Code Analysis Options

Two functions are provided in the code analysis capability of ADIS51, which cannot be solved automatically by a sequential instruction tracing :

– Manually marking of instruction placed at interrupt start adresses as a subroutine
– Manually marking of instructions after a JMP @A+DPTR instruction as a jump destination

Interrupt procedures are normally not reached by a sequential tracing of the program code. Therefore, instructions at possible interrupt addresses in the memory buffer must be marked manually as the first address of a subroutine. After this interrupt address marking, the code must be analysed again (normally from address 0000_H).

As a second problem, sequential tracing stops when it detects a JMP @A+DPTR instruction and outputs a warning as shown in **figure 6**. All addresses of JMP @A+DPTR instructions should be noted. After the end of the analysing procedure, the instructions around the JMP @A+DPTR instruction must be checked manually. Thereafter, instructions following the JMP @A+DPTR instruction can be marked as a jump destination and the code must be re-analysed again.

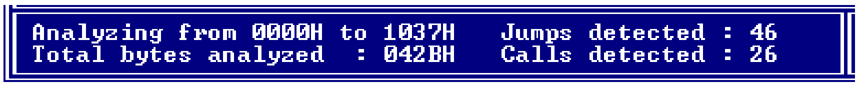


```
Analyzing from 0000H to 3FFFH
JMP @A+DPTR detected at address 0B71H
```

Figure 6 :
JMP @A+DPTR Instruction Warning Message

If a data file has been loaded into the memory buffer, the data bytes which have been loaded are displayed in yellow colour. When the analysing procedure is started, start and end address of the memory buffer area where data has been loaded into is always detected. Memory buffer bytes, which have been analysed already, are displayed in cyan colour. This colour marking allows easy checking of the analysing result.

After the termination of a code analysis run a status message with an analysing summary is displayed in the status area (see **figure 7**). This summary informes the total block size and the number of analysed bytes as well as the number of jump and call labels which have been detected.



```
Analyzing from 0000H to 1037H   Jumps detected : 46
Total bytes analyzed  : 042BH   Calls detected : 26
```

Figure 7 :
Code Analysis Status Result

The analysing result is also stored in a result file ADIS51.TXT. This result file includes additionally the addresses of the jump and call locations which have been detected during the analysis run. The jump and call locations are numbered with Jxxx and Cxxx.

Note: When pressing **`<Shift D>`** once in the main menu, a hidden debug option of ADIS51 is enabled (no message occurs). When this option is enabled, the result file also includes the memory buffer address sequence which has been executed during the last code analysis run.

Example of a Code Analysis Result File :

```
;=========================================================================
;                      ADIS51 V4.0 - Code Analysis Result
;=========================================================================
;
Bytes analysed from 0000H to 1037H
Total code bytes analysed  : 042BH
Jumps detected : 46
Calls detected : 26

List of all jump addresses :

J001:   008FH
J002:   00F5H
..
..
..
J046:   06DEH

List of all call addresses :

C001:   0066H
C002:   0073H
..
..
..
C026:   06D6H
```

6.3 Input of a New Address

Pressing the **<F9>** key in the disassembler menu allows to enter a new start address for the memory buffer disassembling procedure. ADIS51 again begins the linear disassembling of 2000 instructions located in the memory buffer starting from the new entered start address. The instruction, which is located at the start address, appears in the highlighted scrollbar.

This capability of defining a new start address is required when large areas of the memory buffer must be disassembled or if the destination address of a jump instruction, which should be disassembled, is e.g. outside the 2000 actually disassembled instruction area of the memory buffer.

6.4 Generation of a Log-File

The Log-file generation capability in the disassembler menu allows to write disassembler data from the memory buffer into an ASCII file. Two formats for disassembler data can be choosen :

– Listing format or
– Assembler format

The listing format includes instruction addresses as well as the data bytes followed by the instruction mnemonics. When the assembler format is used for Log-File generation, the Log-file itself can be used as an input source file for an 8051 assembler program.

If the memory buffer has been already analysed, jump and call addresses as well as instruction labels and subroutine labels are replaced by the symbolic names (Jxxx and Cxxx) which have been detected during the code analysis run. The absolute addresses are inserted as a comment in the lines where the Jxxx and Cxxx labels occur.

Sometimes there are some gaps in the data of the memory buffer (bytes displayed in white colour). These bytes have not been written into the memory buffer by loading of a data file before. After a code analysis run has been executed, these “white“ bytes are skipped and no more written into the Log-file.

After pressing **`<F10>`** in the disassembler menu, ADIS51 request the input of the Log-file name. As a default name ADIS51.LOG is used. This file name can be overwritten. When a Log-file with the referenced file name already exists, it must be selected whether data should overwrite or should be appended to an already existing Log-file. This capability allows to add multiple disassembler log data and Hex-dump data together in one Log-file.

When the Log-file name is accepted by ADIS51 it must be additionally selected whether symbols (**`<S>`**) or data (**`<D>`**) should be written. If **`<S>`** is selected, SFR and bit symbols are written into the Log-file. If **`<D>`** is selected, a start and end address of the memory buffer area to be written into the Log-file is requested.

The following two examples of a Log-file shows a memory buffer area generated in listing format and assembler format.

Log-File Example Part 1 : Listing Format

```
;=========================================================================
;                           ADIS51 V4.0 - LOG-File
;=========================================================================
;
 0000 : 02 00 8F              LJMP    J007                     ;008FH
 0003 : 02 00 5A      C001:   LJMP    J004                     ;005AH

 000B : 02 00 2E      C002:   LJMP    J001                     ;002EH

 001B : 02 00 3E      C003:   LJMP    J002                     ;003EH

 0023 : 02 00 5D      C004:   LJMP    J005                     ;005DH

 002B : 02 00 4D      C005:   LJMP    J003                     ;004DH
 002E : C2 8C         J001:   CLR     TR0
 0030 : 75 8A 00              MOV     TL0,#00H
 0033 : 75 8C D8              MOV     TH0,#0D8H
 0036 : B2 02                 CPL     02H
 0038 : D2 8C                 SETB    TR0
 003A : 12 02 63              LCALL   C012                     ;0263H
 003D : 32                    RETI
 003E : C2 8E         J002:   CLR     TR1
 0040 : 75 8D FA              MOV     TH1,#0FAH
 ....
 ....
 0080 : 53 C8 C0      C008:   ANL     0C8H,#0C0H
 0083 : 75 CA 00              MOV     0CAH,#00H
 0086 : 75 CB 00              MOV     0CBH,#00H
 0089 : 22                    RET
 008A : C2
 008B : 9F
 008C : C2
 008D : 9E
 008E : 22
 008F : 75 81 20      J007:   MOV     SP,#20H
 0092 : C2 B5                 CLR     P3.5
 ....
 ....
```

Comment :
Up to address $002B_H$ there are some gaps in the data of the memory buffer. After address 0040_H the Log-file is truncated up to address 0080_H. The bytes at addresses $8A_H$ to $8E_H$ have not been detected as valid instructions during the analysis and therefore they are displayed as memory bytes only.

Log-File Example Part 2 : Assembler Format

```
;=========================================================================
;                            ADIS51 V4.0 - LOG-File
;=========================================================================
;
      CSEG     AT       00000H
      LJMP    J007                   ;008FH
C001: LJMP    J004                   ;005AH

      CSEG     AT       0000BH
C002: LJMP    J001                   ;002EH

      CSEG     AT       0001BH
C003: LJMP    J002                   ;003EH

      CSEG     AT       00023H
C004: LJMP    J005                   ;005DH

      CSEG     AT       0002BH
C005: LJMP    J003                   ;004DH
J001: CLR     TR0
      MOV     TL0,#00H
      MOV     TH0,#0D8H
      CPL     02H
      SETB    TR0
      LCALL   C012                   ;0263H
      RETI
J002: CLR     TR1
      MOV     TH1,#0FAH
....
....
C008: ANL     0C8H,#0C0H
      MOV     0CAH,#00H
      MOV     0CBH,#00H
      RET
      DB      0C2H
      DB      09FH
      DB      0C2H
      DB      09EH
      DB      022H
J007: MOV     SP,#20H
      CLR     P3.5
```

Comment :

When using the assembler format for a Log-file generation absolute code segment definitions (CSEG) are inserted into the Log-file. The bytes $8A_H$ to $8E_H$ (4. line below label C008:)), which have not been detected as valid instructions during the analysis, are included using the “DB“ (“define byte“) directive.

The Log-file example 3 below shows the Log-file content when SFR and bit symbols are written into a Log-file.

Log-File Example Part 3 : SFR and Bit Symbols Format

```
ADIS51 V4.0 - List of SFR Symbols for MCU 8051 (in Listing Format)

 80: P0
 81: SP
 ..
 ..
 F0: B

ADIS51 V4.0 - List of SFB Symbols for MCU 8051

 80: P0.0
 81: P0.1
 ..
 ..
 F7: B.7


;    ADIS51 V4.0 - List of SFR Symbols for MCU 8051 (in Assembler Format)

     P0       DATA   80H
     SP       DATA   81H
     ..
     ..
     B        DATA   0F0H

;    ADIS51 V4.0 - List of SFB Symbols for MCU 8051

     IT0      BIT    88H
     IE0      BIT    89H
     ..
     ..
     CY       BIT    0D7H
```

7 Basic Function <F5> : Hex-Dump Menu

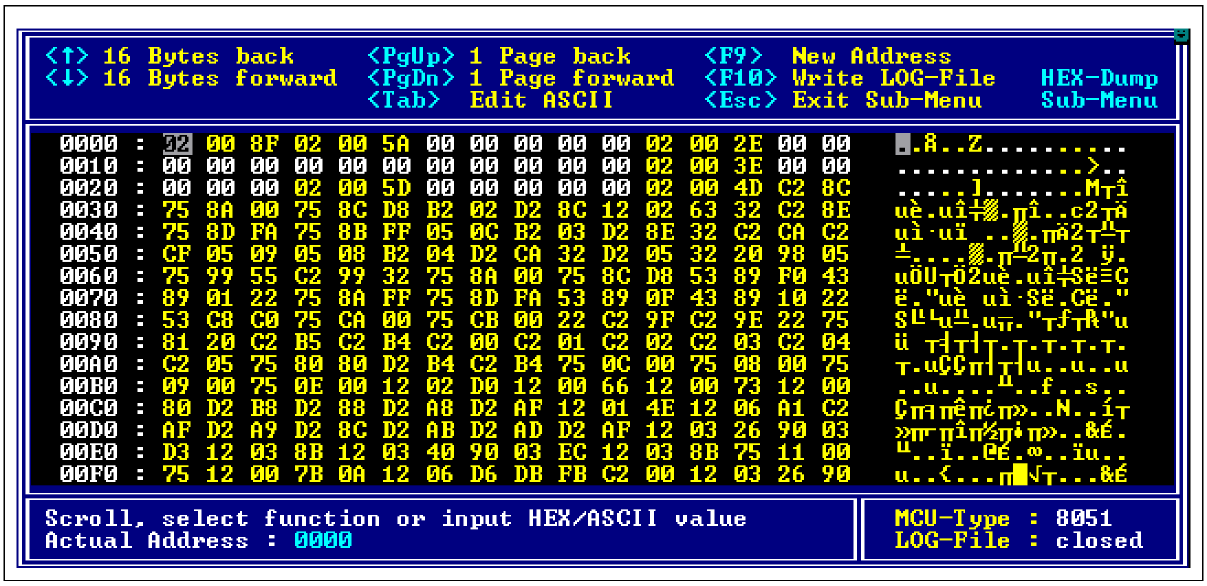
7.1 General Operation

Pressing **`<F5>`** in the main menu activates the Hex-dump function of ADIS51 and requests to input a memory buffer start address (hexadecimal address value followed by a **`<Return>)`**. Beginning at this address, the contents of the memory buffer are displayed in hexadecimal and ASCII notation (16 bytes in each row). Depending on the cursor position, the highlighted byte can be modified by entering a new hexadecimal value or an ASCII character. The **`<Tab>`** key is used to switch the cursor of the highlighted byte from hexadecimal display to ASCII display and vice versa. The highlighted byte can be moved using the cursor keys. The address of the highlighted byte is displayed additionally in the status area of the screen.

After activation of the Hex-dump function, the Hex-dump menu is entered. The help area of the screen displays the functions which are available now :

`<Cursor Down>`	Decrement address of highlighted byte by 16	(optional scroll screen)
`<Cursor Up>`	Increment address of highlighted byte by 16	(optional scroll screen)
`<Cursor Right>`	Increment address of highlighted byte by one	(optional scroll screen)
`<Cursor Left>`	Decrement address of highlighted byte by one	(optional scroll screen)
`<Page Up>`	Scroll Hex-dump screen by one screen page back	
`<Page Down>`	Scroll Hex-dump screen by one screen page forward	
`<F9>`	Input of a new address for the Hex-dump function	
`<F10>`	Open/append/close a Log-file	
`<Tab>`	Switch cursor of highlighted byte from hexadecimal to ASCII input	
`<Esc>`	Back to the Main Menu	

Figure 8 shows an example of a Hex-dump screen. Bytes in yellow colour have been loaded from a data file before. White coloured bytes are spare memory bytes. After a execution of a code analysis, cyan coloured bytes indicate bytes, which have been already analysed.



```
<↑> 16 Bytes back      <PgUp> 1 Page back      <F9>  New Address
<↓> 16 Bytes forward   <PgDn> 1 Page forward   <F10> Write LOG-File    HEX-Dump
                       <Tab>  Edit ASCII       <Esc> Exit Sub-Menu     Sub-Menu

 0000 : 02 00 8F 02 00 5A 00 00 00 00 00 02 00 2E 00 00    ..Å..Z..........
 0010 : 00 00 00 00 00 00 00 00 00 00 00 02 00 3E 00 00    .............>..
 0020 : 00 00 00 02 00 5D 00 00 00 00 00 02 00 4D C2 8C    .....].......M┬î
 0030 : 75 8A 00 75 8C D8 B2 02 D2 8C 12 02 63 32 C2 8E    uè.uî╪▓.╥î..c2┬Ä
 0040 : 75 8D FA 75 8B FF 05 0C B2 03 D2 8E 32 C2 CA C2    uì·uï ..▓.╥Ä2┬╩┬
 0050 : CF 05 09 05 08 B2 04 D2 CA 32 D2 05 32 20 98 05    ╧....▓.╥╩2╥.2 ÿ.
 0060 : 75 99 55 C2 99 32 75 8A 00 75 8C D8 53 89 F0 43    uÖU┬Ö2uè.uî╪Së≡C
 0070 : 89 01 22 75 8A FF 75 8D FA 53 89 0F 43 89 10 22    ë."uè uì·Së.Cë."
 0080 : 53 C8 C0 75 CA 00 75 CB 00 22 C2 9F C2 9E 22 75    S╚└u╩.u╦."┬ƒ┬₧"u
 0090 : 81 20 C2 B5 C2 B4 C2 00 C2 01 C2 02 C2 03 C2 04    ü ┬╡┬┤┬.┬.┬.┬.┬.
 00A0 : C2 05 75 80 80 D2 B4 C2 B4 75 0C 00 75 08 00 75    ┬.uÇÇ╥┤┬┤u..u..u
 00B0 : 09 00 75 0E 00 12 02 D0 12 00 66 12 00 73 12 00    ..u....╨..f..s..
 00C0 : 80 D2 B8 D2 88 D2 A8 D2 AF 12 01 4E 12 06 A1 C2    Ç╥╕╥ê╥¿╥»..N..í┬
 00D0 : AF D2 A9 D2 8C D2 AB D2 AD D2 AF 12 03 26 90 03    »╥⌐╥î╥½╥¡╥»..&É.
 00E0 : D3 12 03 8B 12 03 40 90 03 EC 12 03 8B 75 11 00    ╙..ï..@É.∞..ïu..
 00F0 : 75 12 00 7B 0A 12 06 D6 DB FB C2 00 12 03 26 90    u..{...╓█√┬...&É

Scroll, select function or input HEX/ASCII value    MCU-Type  : 8051
Actual Address : 0000                               LOG-File  : closed
```

Figure 8 :
Hex-Dump Function of ADIS51

The Hex-dump function is left by pressing **`<Esc>`**. The generation of a Log-file is described in chapter 7.3.

Note : Generally, the Hex-dump function can be also used for binary analysis/modifcation of a file with a size of max. 64 Kbyte. For using this feature the file should be loaded as BIN file into the memory buffer at address 0000_H. After the load operation start and end address can be detected manually and bytes can be modified. Finally the data can be written again from the detected start to end address into a BIN file.

7.2 Input of a New Address

Pressing the **`<F9>`** key in the Hex-dump menu allows to enter a new start address for the Hex-dump output. Thereafter, ADIS51 displays the data bytes located in the memory buffer starting with the data byte of the new entered start address in the left upper corner of the screen area.

7.3 Generation of a Log-File

The Log-file generation capability in the Hex-dump menu allows to write data from the memory buffer in hex notation into an ASCII file. Two formats for hex data can be choosen :

– Listing format or
– Assembler format

The listing format for hex data includes memory buffer addresses and hex data with 16 bytes in one row, as hexadecimal and ASCII characters.

When using the assembler format for a Log-file generation an absolute code segment definition (CSEG) is inserted into the Log-file. The data bytes are written into the Log-file using the “DB“ (“define byte“) directive. The example below shows both types of Hex-dump data written into one Log-file.

Log-File Example Part 4 : Hex-Dump Format

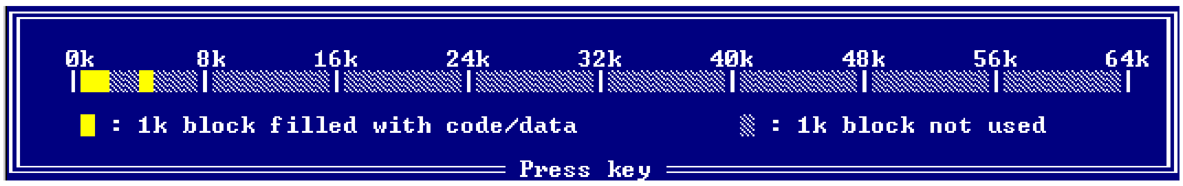
```
;==========================================================================
;                           ADIS51 V4.0 - LOG-File
;==========================================================================
;
 0000 : 02 00 8F 02 00 5A 00 00 00 00 00 02 00 2E 00 00   .._..Z..........
 0010 : 00 00 00 00 00 00 00 00 00 00 00 02 00 3E 00 00   .............>..
 0020 : 00 00 00 02 00 5D 00 00 00 00 00 02 00 4D C2 8C   .....].......MÂŒ
 0030 : 75 8A 00 75 8C D8 B2 02 D2 8C 12 02 63 32 C2 8E   uŠ.uŒØ².ÒŒ..c2Â•

CSEG  AT    00000H
DB  002H,000H,08FH,002H,000H,05AH,000H,000H,000H,000H,000H,002H,000H,02EH,000H,000H
DB  000H,000H,000H,000H,000H,000H,000H,000H,000H,000H,000H,002H,000H,03EH,000H,000H
DB  000H,000H,000H,002H,000H,05DH,000H,000H,000H,000H,000H,002H,000H,04DH,0C2H,08CH
DB  075H,08AH,000H,075H,08CH,0D8H,0B2H,002H,0D2H,08CH,012H,002H,063H,032H,0C2H,08EH
DB  075H,08DH,0FAH,075H,08BH,0FFH,005H,00CH,0B2H,003H,0D2H,08EH,032H,0C2H,0CAH,0C2H
```

If data shall be written into a Log-file by pressing **`<F10>`** in the Hex-dump menu and a Log-file is not open (status message "LOG-File : closed"), a file name is requested to be input in the status area. As default, ADIS16X.LOG is porposed. If a Log-file is already open (status message "LOG-File : open"), it must be selected whether the actual Log-file shall be closed or whether the data should be appended to the end of the actual Log-file.

8 Basic Function <F6> : Overview on the Memory Buffer Usage

This function can be used to get an information about the usage of the memory buffer after a data file has been loaded into the memory buffer (basic function **<F1>** “Loading of a Data File“). This function is useful if e.g. the code size of a data file is unknown. Each block character in the memory buffer usage window represents an address area of 1K byte (see **figure 9**). A yellow (or bold) block character indicates that the 1k block of the memory buffer has been loaded with data. A gray block character indicates that the address area of 1K byte has not been loaded with data.

The memory buffer usage window is again closed when any key is pressed.

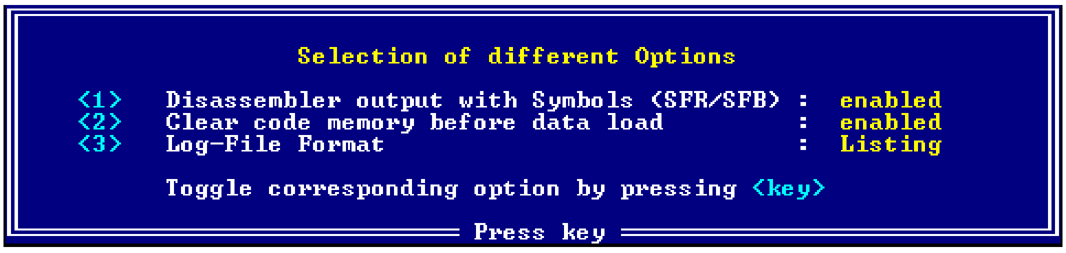


```
0k      8k      16k     24k     32k     40k     48k     56k     64k

█ : 1k block filled with code/data          ▒ : 1k block not used
                         Press key
```

Figure 9 :
Memory Buffer Usage Window

9 Basic Function <F7> : Selection of Options

With this function three options can be selected. Pressing the keys **<1>** to **<3>** always toggles the related option. The displayed value is always active (see **figure 10**). Pressing another key closes the options window again.



```
                Selection of different Options

   <1>    Disassembler output with Symbols (SFR/SFB) :  enabled
   <2>    Clear code memory before data load         :  enabled
   <3>    Log-File Format                            :  Listing

          Toggle corresponding option by pressing <key>

                            Press key
```

Figure 10 :
Options Window

The first option can be used to enable/disable SFR- and bit-symbols in disassembler outputs and one-line ssembler inputs. The symbols must be defined in an external symbol definition file (ADIS51.SYM).

If the second option is enabled, every load of a data file is preceeded by a memory buffer clear operation (64K memory buffer is loaded with 00_H). This memory buffer clear operation can be disabled.

The third option allows to select the Log-file format. The Log-file can be generated in a type of listing format or in an assember input format, which can be directly used by the standard 8051/C500 assembler programs as an input source file. More details about the format of a Log-file is given in chapter 6 and 7.

Appendix

A Error and Status Messages

Error-/Status Message	Cause
File access error	Input of an invalid file name or file does not exist.
Checksum error	During loading of a data file a checksum error has been detected. The data file load operation is aborted.
No absolute code/data has been loaded	During the reading of a data file no absolute located code/data information has been detected and loaded into the memory buffer. Probably the data file has a wrong format.
Invalid file format or Invalid OBJ-file format	During loading of a data file a wrong file format has been detected. The load operation has been aborted.
Invalid input	The character which has been input is invald and was not accepted.
Invalid address	Invalid address (only hexadecimal values allowed).
Invalid instruction	An invalid instruction has been input; the one-line assembler cannot disassemble the instruction (instruction format see Appendix B)
Display buffer exceeded	For further disassembling of the memory buffer a new start address must be input (using **`<F9>`** in the disassembler menu. Beginning with the start address, at maximum 2000 instructions can be displayed in one run.
Code memory limit reached	During the disassemble or hex dump function the upper limit of the memory buffer has been reached.
End of code memory - input lower address	In disassembler or hex dump function a new lower start address must be input. With the start address, which has been input, the display area cannot be filled completely with data.
Disassembler start address reached	The disassembler has reached the start address from which 2000 instruction have been disassembled (with incrementing address).
Start address is greater than end address	For log file generation the start address is greater than the end address.
`xxx`-File **`name`** is loaded	Data file **`name`** of type **`xxx`** is opened, analysed, and absolutely located code/data is loaded into the memory buffer.
File **`name`** is created	The content of the memory buffer is transferred into a data file (Hex- or BIN-format) with the filename **`name.`**
Log-File is created	Data is currently transferred into a Log-file.
Symbol definition file: not available or invalid format	During the loading of ADIS51 no symbol definition file has been found or the format of an existing symbol definition file is wrong.
MCU selection is not available - no MCU-type defined	No MCU types defined. A symbol definition has not been loaded.
File exists ! Overwrite/Append/Cancel ?	A file (data or log file) to be written already exyists or is open. The file can be closed, overwritten, appended or the action can be cancelled.

B One-Line Assembler Conventions

Instruction Format :

Opcodes	Operands
MOV	A,Rn A,direct A,@Ri A,#data Rn.A Rn,direct Rn,#data direct,A direct,Rn direct,direct direct,@Ri direct,#data @Ri,A @Ri,direct @Ri,#data DPTR,#data16
MOVC	A,@A+DPTR A,@A+PC
MOVX	A,@Ri A,@DPTR @Ri,A @DPTR,A
XCH	A,Rn A,direct A,@Ri
XCHD	A,@Ri
PUSH POP	direct
ADD ADDC SUBB	A,Rn A,direct A,@Ri A,#data

Opcodes	Operands
ANL ORL XRL	A,Rn A,direct A,@Ri A,#data direct,A direct,#data
INC DEC	A Rn direct @Ri
INC	DPTR
MUL DIV	AB
CLR CPLRL RLC RR RRC DA SWAP	A
PUSH POP	direct
XCH	A,Rn A,direct A,@Ri
XCHD	A,@Ri
ACALL AJMP	addr11
LCALL LJMP	addr16

Opcodes	Operands
JMP	@A+DPTR
SJMP JZ JNZ JC JNC	rel
JB JNB JBC	bit,rel
CJNE	A,direct,rel A,#data,rel Rn,#data,rel @Ri,#data,rel
DJNZ	Rn,rel direct,rel
CLR CPL SETB C	bit
ANL ORL MOV	C,bit
ANL ORL	C,/bit
MOV	bit,C
RET RETI NOP	-

The operands shown in table are abbreviations for the following inputs :

Rn	R0-R7
Rii	R0, R1
direct	0-255 or 00H-0FFH or SFR symbol
data	0-255 or 00H-0FFH
data16, addr16, rel	0-65535 or 0000H-0FFFFH
addrr11	0-2047 or 000H-07FH
bit	0-255 or 00H-0FFH or bit symbol or SFR.x (x=0-7)

Generally, numbers can be input as decimal or hexadecimal values. For hexadecimal values the standard convention is valid : a “0“ must preceed the value if the hexadecimal value begins with a letter (“A“ - “F“); the hexadecimal value ends with a “H“.

Inputs for the assembler are not case sensitive.

C Definition of the Symbol Definition File

```
;==============================================================================
;     Symbol Definition File for ADIS51 V4.0 - Rules
;==============================================================================
;
;     The symbol definition file for the ADIS51 V4.0 Disassembler allows to
;     define symbols for SFR's and bits of the SFR's for up to 15 different
;     8051 compatible microcontrollers. Such a symbol definition file is build
;     up according the following rules :
;
;     1.  Comment lines have a ";" in the first row of a line. All following
;         characters in this line are ignored.
;
;     2.  Empty lines (with 0DH, 0AH) can be inserted everywhere.
;
;     3.  The definition file has 3 sections. Each section is validated by
;         a keyword. The keywords of these sections are :
;         Keyword "[MCU]" --> MCU-Section: defines the names of the MCUs
;         Keyword "[SFR]" --> SFR-Section: defines the names of the Special
;                             Function Registers
;         Keyword "[SFB]" --> SFB-Section: defines the names of the bits of the
;                             Special Function Registers.
;         The keywords must be placed in the first row of a line.
;
;     4.  Definitions in the MCU-Section :
;         "xxxxxxx hhhh" : "xxxxxxx" starts in the first row of a column and
;                          is the (short-)name of the microcontroller;
;                          (max. 7 ASCII characters)
;                          "hhhh" is a 4-digit hexword, coded with a "1" in one
;                          of the 15 bit positions, starting with bit position
;                          0; this hexword is separated from the name with ex-
;                          actly one blank character;
;         The lines in the MCU-Section shall be ordered by ascending bit
;         positions of "hhhh".
;
;     5.  Definitions in the SFR-/SFB-Sections :
;         Lines of the SFR-/SFB-Sections have exactly 3 parts, separated each
;         by one blank character.
;         "aa xxxxxxx hhhh" : "aa" is a 2-digit hexbyte value, which defines
;                             the 8-bit address of a SFR or SFB and starts in
;                             the first row of a column;
;                             "xxxxxxx" is the name of the SFR/SFB with the
;                             address "aa"; (max. 7 ASCII characters);
;                             "hhhh" is a 4-digit hexword; a "1" in a dedicated
;                             bit position defines, that the symbol 'xxxxxxx'
;                             is valid for the MCU, which has also a "1" de-
;                             fined at the same bit position in the MCU-Section
;
;==============================================================================
;
```

Note : The file AP082701.EXE includes an example for a symbol definition file.