SIEMENS

Microcontrollers

ApNote AP1640

☑ : Additional file
AP164002.EXE available

ADIS16X - Disassembler with One-Line Assembler for the C16X 16-Bit Microcontroller Family

ADIS16X allows the analysis of program code for the 16-bit microcontrollers of the C16X family. Program code in different formats can be loaded and displayed in Disassembler or Hex-dump format. Single instructions can be input using the built-in One-Line Assembler. Program code can be 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 Munich

Contents **Page**

AP1640 ApNote - Revision History		
Actual Revision : 11.97 Rel. 02 Previous Revision : 10.97 Rel. 01 (Original Version)		
Page of actual Rev.	Page of prev.Rel.	Subjects (changes since last release)
-	-	AP164002.EXE : new version of ADIS16X.EXE (V3.03) included; operand decoding of JNB, JBC, JNB, and JNBS instructions corrected.

1 Starting of ADIS16X

The program ADIS16X.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 ADIS16X 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 ADIS16X /43 .

After ADIS16X has been started the title screen is displayed. Pressing any further key will turn on the main menu screen.

When ADIS16X is loaded, the program searches for a file called ADIS16X.SYM, which contains the device, the special function register, and the bit symbol definitions. If this file is not present in the directory where ADIS16X is located and started, ADIS16X 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 ADIS16X to all types of the C166 family microcontrollers with their different SFR and bit symbols. The syntax of the symbol definition file is shown in appendix C of this ApNote.

Depending on the available main memory resources of the PC, ADIS16X allocates a memory buffer for its operation with up to 256 KByte in portions of 64K Bytes. Therefore, the memory buffer for ADIS16X starts at 00000_H up to at maximum $3FFFF_H$ in quantities of 64 Kbyte pages.
Important : C166 data can be only handled within this absolute memory address area.

2 Main Menu

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

Figure 1 :
Main Menu of ADIS16X

The main menu screen of ADIS16X 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 keys which are used to select these basic functions.
- The middle area of the screen, **the display area**, is used to display the contents of the 256K 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 ADIS16X 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 ADIS16X can be selected from the main menu. These basic functions are assigned to function keys F1 to F7 as follows :

<F1> Loading of a data file in different formats into the ADIS16X memory buffer
<F2> Writing ADIS16X memory buffer contents into a file using HEX- or BIN- format
<F3> Selection of a microcontroller type (and its symbols)
<F4> Starting the ADIS16X Disassembler / One-Line Assembler function
<F5> Starting the ADIS16X Hex-dump function
<F6> Display of the ADIS16X memory buffer usage (Code RAM Info)
<F7> Selection of different options

Pressing **<Esc>** in the main menu will terminate the ADIS16X 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

ADIS16X provides a 256K 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 ADIS16X. HEX- and OBJ-file formats are the Intel type of format.

Figure 2 :
Data File Load Selection Window

Pressing **<F1>** in the main menu of ADIS16X opens the window above and requests an input for the selection of the data file format. After the selection of one data file format, ADIS16X 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 ADIS16X is located in either relative or absolute format. When a data file has been found and opened, ADIS16X analyses its content and transfers the relevant data into the memory buffer.

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 (data records) and 02_H (address records). Only code or data information which is placed in these records is processed and transferred into the memory buffer. From OBJ-files only records with record type $B9_H$ and 05_H are analysed. Symbol informations stored in HEX- and OBJ-files are not used by ADIS16X.

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-File

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 “Extended Address Record“ (record type=02_H) is preceeded and an "End-of-File" record “:00000001FF“ (record type=01_H) is appended to a data block which has been written into the HEX 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** three devices have been defined.

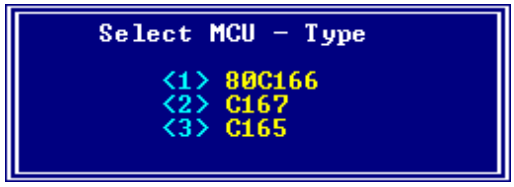

Figure 3 :
Microcontroller Type Selection Window

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> : Disassemblers / One-Line Assemblers 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 ADIS16X program. It allows to display the contents of the memory buffer as C166 instruction mnemonics and to alter or enter single instructions by using the one-line assembler capability.

When entering the disassembler menu, ADIS16X 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.

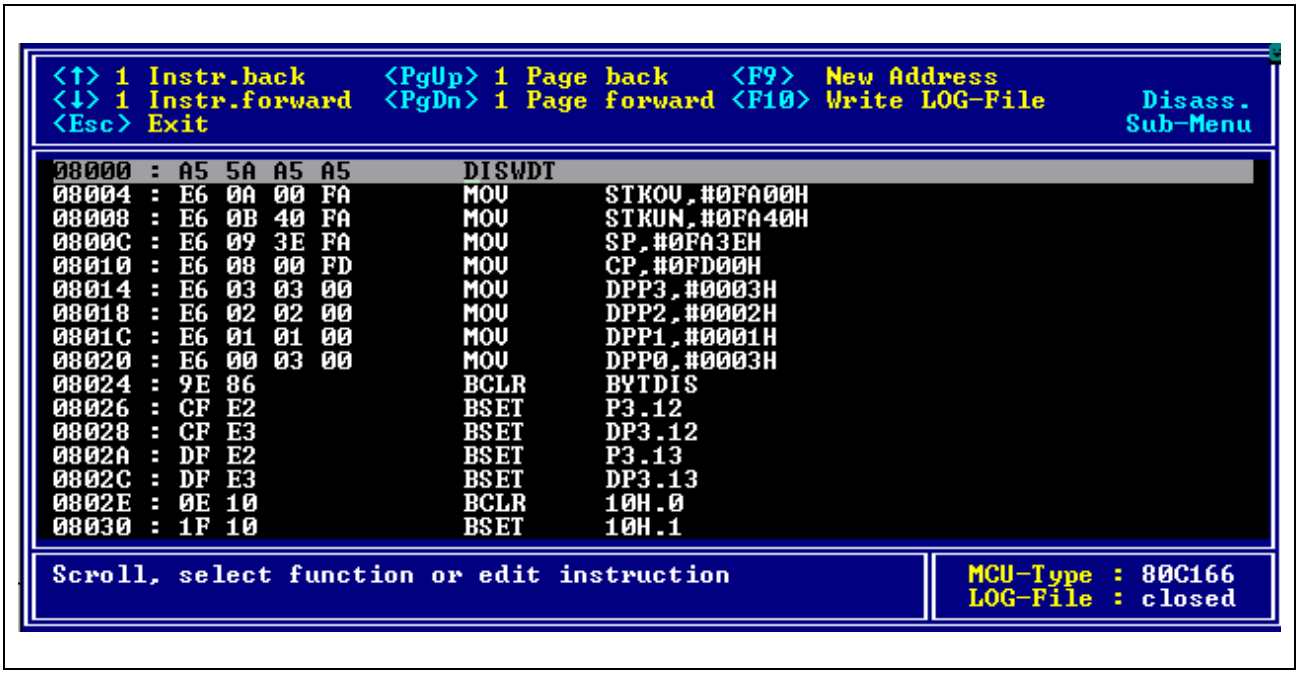

Figure 4 :
Disassembler/One-Line Assembler Menu of ADIS16X

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 byte and word values for 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 operands. SFR and bit symbol output/input can be switched off by an option (see chapter 8).

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 4-byte instruction by a 2-byte instruction).

The input of an odd start address for memory buffer disassembling will be corrected with a corresponding status message to the next even memory buffer address. Instructions, which follow ATOMIC and EXT* instructions, are assigned to the extended SFR area. Such instructions are marked with a “+“ in the rightmost column of the screen area.

Two more function keys are available in the disassembler menu :

<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 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. ADIS16X 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.

6.3 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. The disassembler data is written in the Log-file in a listing format as shown in the Log-file example 1 on the next page. The data in the first part of the Log-file example 1 has been generated with DPPi and symbols decoding enabled. For the second part DPPi and symbols decoding has been disabled.

Log-File Example 1 : Disassembler Output

```
================================================================================
                          ADIS16X V3.0 - LOG-File
================================================================================
;
;       Log-File output with DPPi decoding and symbols decoding enabled (C167)
;
08000 : A5 5A A5 A5       DISWDT
08004 : E6 0A 00 FA       MOV      STKOV,#0FA00H
08008 : E6 0B 40 FA       MOV      STKUN,#0FA40H
0800C : E6 09 3E FA       MOV      SP,#0FA3EH
08010 : E6 08 00 FD       MOV      CP,#0FD00H
08014 : E6 03 03 00       MOV      DPP3,#0003H
08018 : E6 02 02 00       MOV      DPP2,#0002H
0801C : E6 01 01 00       MOV      DPP1,#0001H
08020 : E6 00 03 00       MOV      DPP0,#0003H
08024 : 9E 86             BCLR     ALECTL0
08026 : CF E2             BSET     P3.12
08028 : CF E3             BSET     DP3.12
0802A : DF E2             BSET     P3.13
0802C : DF E3             BSET     DP3.13
0802E : 0E 10             BCLR     10H.0
08030 : 1F 10             BSET     10H.1
08032 : FA 00 36 80       JMPS     0,8036H
08036 : B5 4A B5 B5       EINIT
0803A : F6 F0 48 FD       MOV      DPP3:3D48H,R0
0803E : F2 5E 48 FD       MOV      S1BG,DPP3:3D48H
;
;       Log-File output with DPPi decoding and symbols decoding disabled (C167)
;
08000 : A5 5A A5 A5       DISWDT
08004 : E6 0A 00 FA       MOV      0AH,#0FA00H
08008 : E6 0B 40 FA       MOV      0BH,#0FA40H
0800C : E6 09 3E FA       MOV      09H,#0FA3EH
08010 : E6 08 00 FD       MOV      08H,#0FD00H
08014 : E6 03 03 00       MOV      03H,#0003H
08018 : E6 02 02 00       MOV      02H,#0002H
0801C : E6 01 01 00       MOV      01H,#0001H
08020 : E6 00 03 00       MOV      00H,#0003H
08024 : 9E 86             BCLR     86H.9
08026 : CF E2             BSET     0E2H.12
08028 : CF E3             BSET     0E3H.12
0802A : DF E2             BSET     0E2H.13
0802C : DF E3             BSET     0E3H.13
0802E : 0E 10             BCLR     10H.0
08030 : 1F 10             BSET     10H.1
08032 : FA 00 36 80       JMPS     0,8036H
08036 : B5 4A B5 B5       EINIT
0803A : F6 F0 48 FD       MOV      0FD48H,R0
0803E : F2 5E 48 FD       MOV      S1BG,0FD48H
```

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 proposed. 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.

7 Basic Function <F5> : HEX-Dump Menu

7.1 General Operation

Pressing **<F5>** in the main menu activates the Hex-dump function of ADIS16X 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 5 shows an example of a Hex-dump screen

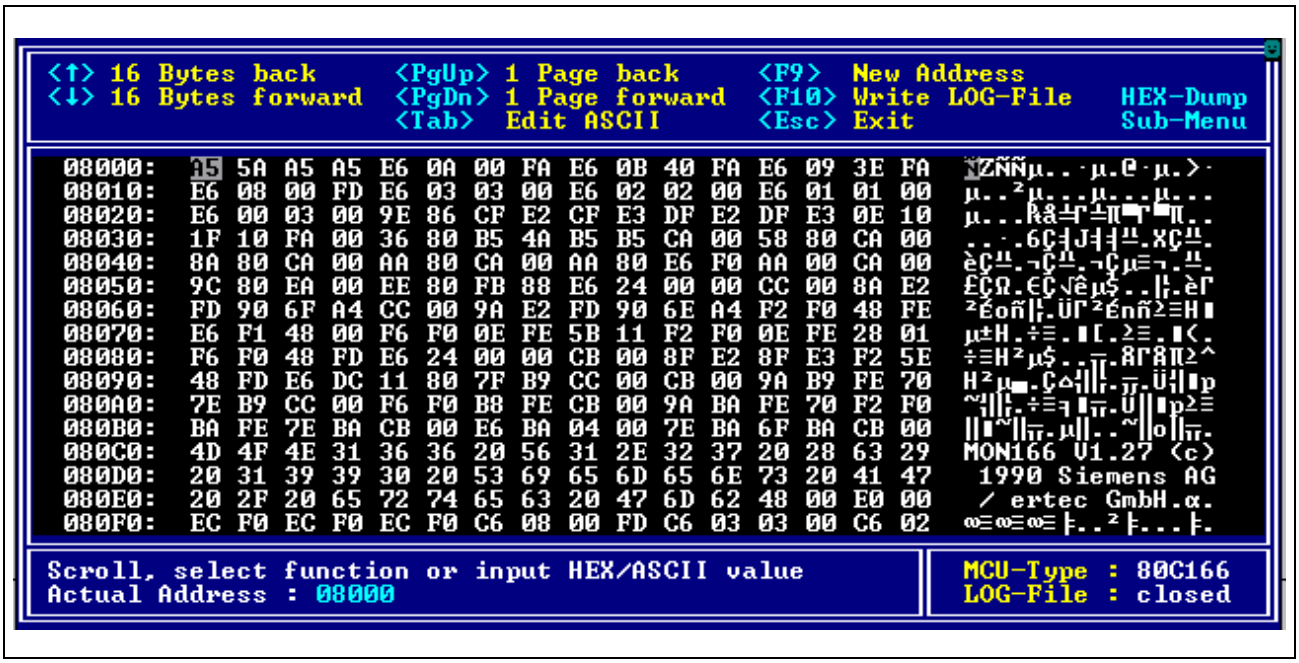

Figure 5 :
Hex-Dump Function of ADIS16X

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. 256 Kbyte. For using this feature the file should be loaded as BIN file into the memory buffer at address 00000_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, ADIS16X 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. The format of the hex data includes memory buffer addresses and hex data with 16 bytes in one row, as hexadecimal and ASCII characters.

The Log-file example 2 below shows the Hex-dump log file output of the data as shown in **figure 5**.

Log-File Example 2 : Hex-Dump Output

```
================================================================================
                          ADIS16X V3.0 - LOG-File
================================================================================

08000:  A5 5A A5 A5 E6 0A 00 FA E6 0B 40 FA E6 09 3E FA  ¥Z¥¥æ..úæ.@úæ.>ú
08010:  E6 08 00 FD E6 03 03 00 E6 02 02 00 E6 01 01 00  æ..ýæ...æ...æ...
08020:  E6 00 03 00 9E 86 CF E2 CF E3 DF E2 DF E3 0E 10  æ...•†ÏâÏãßâßã..
08030:  1F 10 FA 00 36 80 B5 4A B5 B5 CA 00 58 80 CA 00  ..ú.6_µJµµÊ.X_Ê.
08040:  8A 80 CA 00 AA 80 CA 00 AA 80 E6 F0 AA 00 CA 00  Š_Ê.ª_Ê.ª_æðª.Ê.
08050:  9C 80 EA 00 EE 80 FB 88 E6 24 00 00 CC 00 8A E2  œ_ê.î_û^æ$..Ì.Šâ
08060:  FD 90 6F A4 CC 00 9A E2 FD 90 6E A4 F2 F0 48 FE  ý_o¤Ì.šâý_n¤òðHþ
08070:  E6 F1 48 00 F6 F0 0E FE 5B 11 F2 F0 0E FE 28 01  æñH.öð.þ[.òð.þ(.
08080:  F6 F0 48 FD E6 24 00 00 CB 00 8F E2 8F E3 F2 5E  öðHýæ$..Ë._â_ãò^
08090:  48 FD E6 DC 11 80 7F B9 CC 00 CB 00 9A B9 FE 70  HýæÜ._-¹Ì.Ë.š¹þp
080A0:  7E B9 CC 00 F6 F0 B8 FE CB 00 9A BA FE 70 F2 F0  ~¹Ì.öð¸þË.šºþpòð
080B0:  BA FE 7E BA CB 00 E6 BA 04 00 7E BA 6F BA CB 00  ºþ~ºË.æº..~ºoºË.
080C0:  4D 4F 4E 31 36 36 20 56 31 2E 32 37 20 28 63 29  MON166 V1.27 (c)
080D0:  20 31 39 39 30 20 53 69 65 6D 65 6E 73 20 41 47   1990 Siemens AG
080E0:  20 2F 20 65 72 74 65 63 20 47 6D 62 48 00 E0 00   / ertec GmbH.à.
080F0:  EC F0 EC F0 EC F0 C6 08 00 FD C6 03 03 00 C6 02  ìðìðìðÆ..ýÆ...Æ.
```

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 locations of a data file are unknown. Each block character in the memory buffer usage window represents an address area of 1K byte (see **figure 6**). 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.

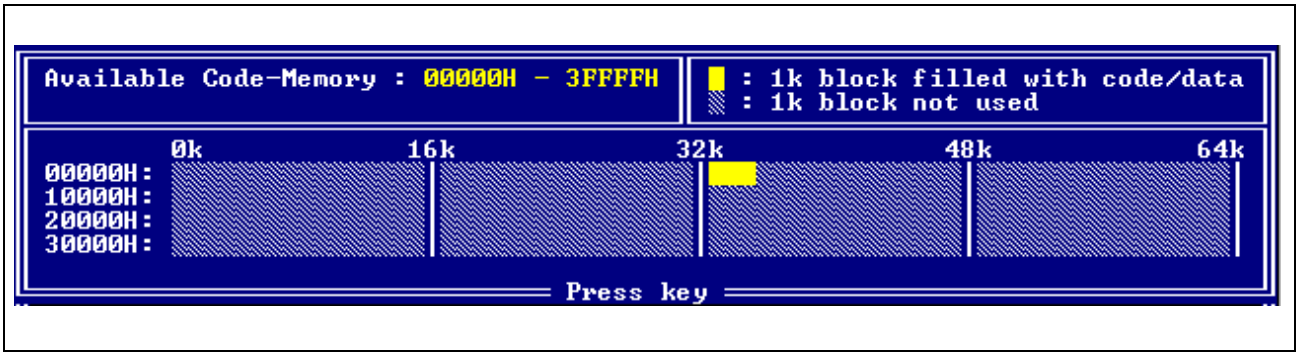

Figure 6 :
Memory Buffer Usage Window

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 7**). Pressing another key closes the options window again.

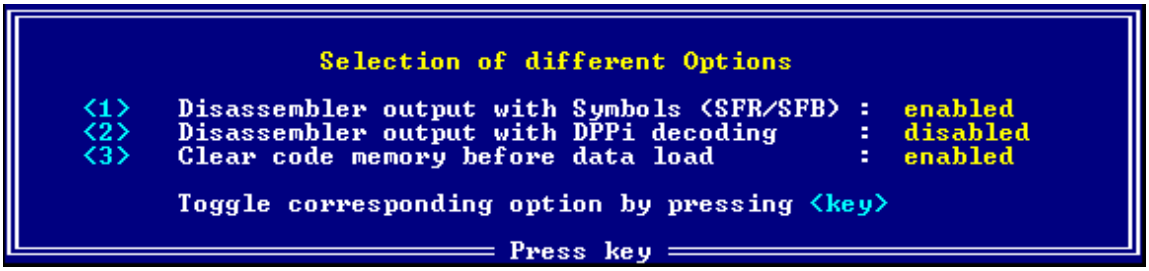

Figure 7 :
Options Window

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 (ADIS16X.SYM).

The second option allows to switch on/off the DPPi decoding of instructions when they are displayed in the disassembler menu or written in a Log-file. An instruction “MOV DPP1,#0001H“ is disassembled as “MOV 01H,#0001H“ when option 2 is disabled.

If the third option is enabled, every load of a data file is preceeded by a memory buffer clear operation (256K memory buffer is loaded with 00_H). This memory buffer clear operation can be disabled.

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 OBJ-file format or Invalid IEEE-file format	During loading of a OBJ or IEEE data file a wrong file format has been detected. The load operation has been aborted.
Code memory is not available at this address	During loading of a data file a non existing memory buffer address has been requested. 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.
Odd address - corrected to next even ddress	The start address for the disassemble function is an odd address and corrected to the next (higher) even address.
Symbol definition file: not available or invalid format	During the loading of ADIS16X 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 file 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 Formats

Opcodes	Operands
ADD, ADDC; SUB, SUBC, AND, OR, XOR, CMP	Rw,Rw Rw,[Rwi] Rw,[Rwi+] Rw,#data3 reg,#data16 reg,mem mem,reg (not for CMP)
ADDB, ADDCB, SUBB, SUBCB, ANDB, ORB, 'XORB, CMPB	Rb,Rb Rb,[Rwi] Rb,[Rwi+] Rb,#data3 reg,#data8 reg,mem mem,reg (not for CMPB)
MUL, MULU PRIOR	Rw,Rw
CPLB, NEGB	Rb
BCLR, BSET	bitaddr
BCLR, BSET BMOV, BMOVN, BAND, BCMP, BOR, BXOR	bitaddr,bitaddr
BFLDL, BFLDH	bitoff,#mask8, #data8
CMPD1, CMPD2, CMPI1, CMPI2	Rw,#data4 Rw,#data16 Rw,mem
SHL,SHR,ROL, ROR, ASHR	Rw,Rw Rw,#data4
MOV	Rw,Rw Rw,#data4 Rw,[Rw] Rw,[Rw+] Rw,[Rw+#data16] [Rw],Rw [Rw],[Rw] [Rw],[Rw+] [Rw+],[Rw] [-Rw],Rw [Rw+#data16],Rw [Rw],mem mem,[Rw] reg,#data16 reg,mem mem,reg

Opcodes	Operands
MOVB	Rb,Rb Rb,#data4 Rb,[Rw] Rb,[Rw+] Rb,[Rw+#data16] [Rw],Rb [Rw],[Rw] [Rw],[Rw+] [Rw+],[Rw] [-Rw],Rb [Rw+#data16],Rb [Rw],mem mem,[Rw] reg,#data8 reg,mem mem,reg
MOVBS, MOVBZ	Rb,Rb reg,mem mem,reg
JMPA, CALLA	cc,caddr
JMPI, CALLI	cc,[Rw]
JMPS, CALLS	seg,caddr
JMPR	cc,rel
CALLR	rel
TRAP	#trap7
PCALL	reg,caddr
JB, JBC, JNB, JNBS	bitaddr,rel
POP, PUSH, RETP	reg
SCXT	reg,#data16 reg,mem
RET, RETS, RETI, SRST, IDLE, PWRDN, SRVWDT,EINIT, DISWDT, NOP	no operands
ATOMIC 1) EXTR 1)	#data2
EXTP 1) EXTPR 1)	Rw,#data2 pag,#data2
EXTS 1) EXTSR 1)	Rw,#data2 segm,#data2

1) These instructions are not available for the C166

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

Rw	R0-R15
Rwi	R0-R3
Rb	RL0-RH7
data2	1-4 or 1H-4H
data3	0-7 or 0H-7H
data4	0-15 or 00H-0FH
trap7	0-127 or 00H-7FH
data8,mask8	0-255 or 00H-0FFH
data12	0-4095 or 000H-7FFH
data16	0-65535 or 0000H-0FFFFH
SFR	symbol of a special function registers (SFR)
SFRb	symbol of a bitadressable SFR
SFB	symbol of a bitadressable Bits
SFRb.x	bit x of a bitadressable SFR (x=0-15)
data8.x	bit x of a bitadressable SFR witrh the address data8 (x=0-15)
Rw.x	bit x of the registers R0-R15 (x,x=0-15)
cc	symbol of a condition codes (z.B. CC_EQ)
caddr	absolute jump address : 0-65535 or 0000H-0FFFFH
rel	relative jump address : 0-65535 or 0000H-0FFFFH
pag	10-bit page number : 0-1023 or 00H-3FFH
seg	absolute segment address : 0-3 or 0H-3H
segm	absolute segment address : 0-255 or 00H-0FFH
DPPi	data pointer DPP0-DPP3 (i=0-3)

The following operands are abbreviations for several inputs shown above :

reg	=	Rw (for word instructions)	or
		Rb (for byte instructions)	or
		SFR	or
		data8	
mem	=	SFR	or
		data16	or
		DPPi:data16	
bitoff	=	Rw	or
		SFRb	or
		data8	
bitaddr	=	data12	or
		SFB	or
		SFRb.x	or
		data8.x	or
		Rw.x	

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 ADIS16X V3.0 - Rules
;=============================================================================
;
;     The symbol definition file for the ADIS16X V3.0 Disassembler allows to
;     define symbols for SFR's and bits of the SFR's for up to 15 different
;     80C16X 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 4 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 "[ESFR]"--> ESFR-Section: defines the names of the Extended
;                             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". A "1" in the highest bit position indicates that
;         the MCU is using ESFR register.
;
;     5.  Definitions in the SFR-/ESFR-/SFB-Sections :
;         Lines of the SFR-/ESFR-Sections have exactly 3 parts, separated each
;         by one blank character.
;         "aa xxxxxxxx hhhh" :"aa" is a 2-digit hexbyte value, which defines
;                             the 8-bit address of a SFR or ESFR and starts in
;                             the first row of a column;
;                             "xxxxxxxx" is the name of the SFR/ESFR with the
;                             address "aa"; (max. 8 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
;
;     6.  Definitions in SFB-Sections :
;         Lines of this section are build up in a similar way as in the SFR-/
;         ESFR-Section. The difference is that the bitaddress is decoded in a
;         16-bit value.
;         "aabb xxxxxxx hhhh" : "aabb" is a 4-digit hexbyte value, which con-
;                               tains the bit position (aa) and the 8-bit
;                               address bb of the bitaddressable SFR-/ESFR.
;                               "aabb" starts in the first row of a column.
;                               "xxxxxxx" is the name of the bit symbol
;                               (8 ASCII characters);
;                               "hhhh" is a 4-digit hexword; a "1" in a dedi-
;                               cated bit position defines, that the symbol
;                               'xxxxxxx' is valid for the MCU, which has also
;                               a "1" defined at the same bit position in the
;                               MCU-Section
;=============================================================================
;
```

Note : The file AP164001.EXE includes an example for a symbol definition file.