

Development Tools for the HCS12

- Development tools are essential for those who want to learn the HCS12 or develop products based on the HCS12.
- The necessary software tools include *text editor*, *terminal programs*, *cross assembler*, *debugger*, and *an integrated development environment 'IDE'*.
- Any text editor can be used, however, *programmer's editor* will improve the program entering productivity.
- A *communication program* also called *terminal program* allows the PC to communicate with the development board.
- There are several general purpose *terminal programs* available such as *Hyper terminal*, *Kermit*, or *ProComm*.
- Cross assembler translates source code into executable machine instruction. There are several such free cross assembler posted on the web site www.microcontroller.com.

Development Tools *continued* ...

- The *source-level debugger* is a program that allows one to find problems in one's code at the assembly level language.
- A debugger can display the content of registers and memory and program code in separate windows.
- The BDM mode of the HCS12 offers an alternative for implementing the *source-level debugger*.
- Ideally, *integrated development environment* (IDE) software would provide an environment that combines the following software tools so that the user can perform all development activities without needing to exit any program:
 - A text editor
 - A project manager
 - A cross assembler and/or a compiler
 - A source level debugger
 - A communication program

Development Tools *continued* ...

- A full-blown IDE is certainly very useful but can cost dearly.
- Many of today's cross-assembler and compiler vendors add a text editor, project manager, and a terminal program into their cross software to make their product more competitive.
- There are also freeware IDEs in this category. Both **AsmIDE** (by Eric Engler) and the **MiniIDE** (from Mgtek) combine an assembler, text editor, and a terminal program.

HCS12 Demo & Evaluation Board:

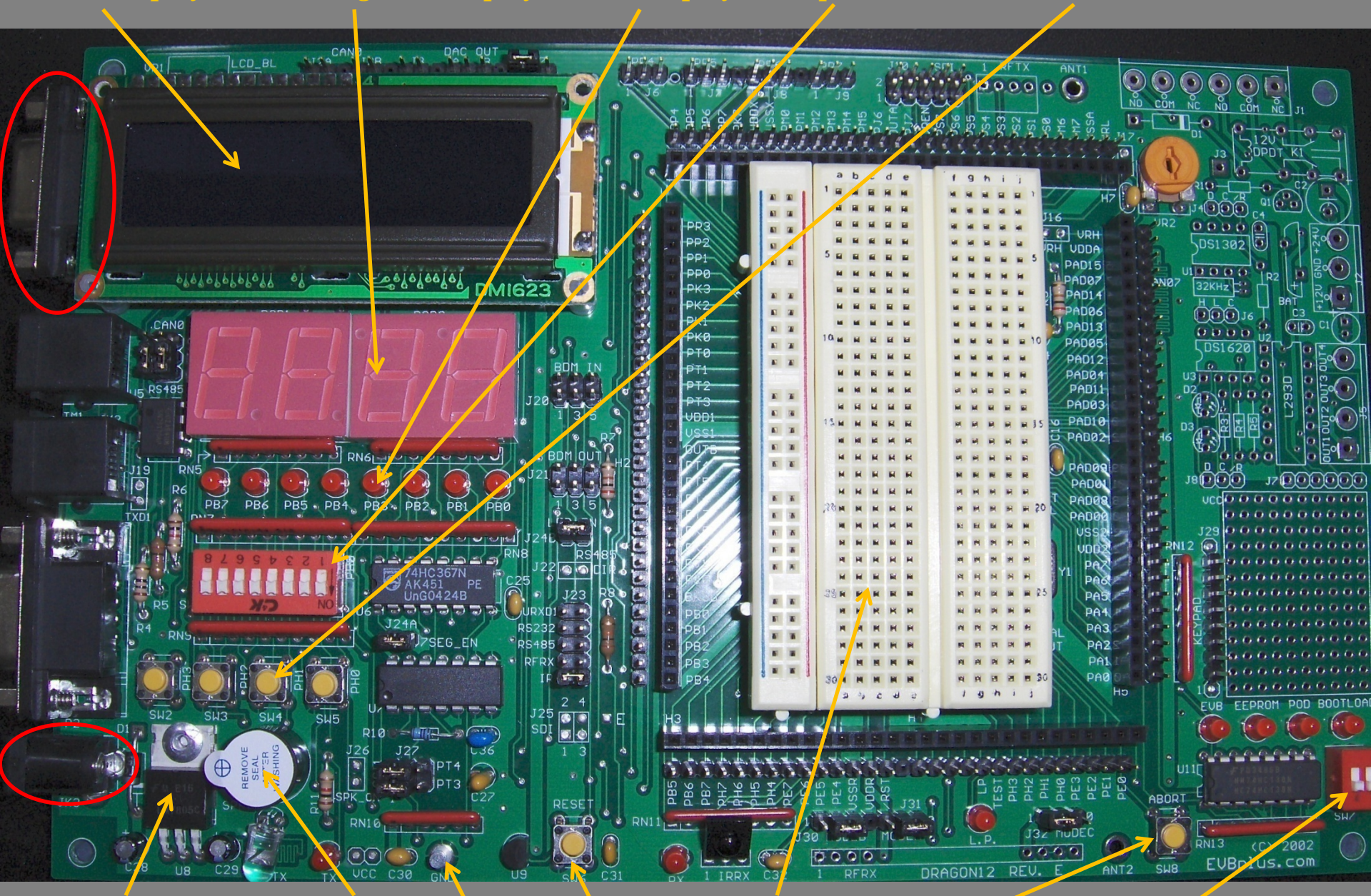
- There are several HCS12DP256-based demo boards that use the D-Bug12 as their debug monitor.
- One such board is the **Dragon12** from Wytec and the other is **SSE256** from Shuan Shizu.

Development Tools *continued* ...

➤ The Dragon12 Demo Board from Wytec has the following features:

1. 24-MHz bus speed generated from a 4-MHz crystal
2. D-Bug12 monitor
3. 16×2 LCD kit (4-bit data transfer)
4. 8 LEDs
5. 8 switches for data input
6. Keypad connector
7. 4 buttons for input
8. 4 seven-segment displays
9. Buzzer for playing songs
10. Potentiometer
11. Infrared transceiver
12. CAN transceiver
13. A small breadboard
14. BDM in & BDM out connectors
15. USB Connector
16. 24LC16 serial EEPROM with I²C interface
17. LTC1661 DAC with SPI interface and 10-bit resolution

LCD Display Four 7-segment Display 8 LED Display 8 Dip Switches 4 Push Buttons



Voltage Regulator

Speaker

GND

Reset

Small Breadboard

Abort

Mode Select

Microprocessors

St. Mary's University

L2-5

The D-Bug12 Monitor:

➤ The D-Bug12 is a monitor designed for the HCS12 microcontrollers. Version 4 of the D-Bug12 supports the following devices:

- MC9S12Dx256
- MC9S12A256
- MC9S12Dx128
- MC9S12H256
- MC9S12A128

➤ This monitor is used in different HCS12 demo boards from several companies.

➤ The default baud rate out of reset is **9600**. However, the baud rate can be set to a higher value if a higher communication speed is desired.

➤ The D-Bug12 monitor has four operating modes. When the D-Bug12 monitor first starts (at power up or reset), it reads the logic levels of the **PAD0** and **PAD1** pins to enter into different operating modes.

The D-Bug12 Monitor Continued ...

- The four operating modes are as follows:

PAD1	PAD0	Operating mode
0	0	D-Bug12; EVB
0	1	Jump to internal EEPROM
1	0	D-Bug12; POD
1	1	Serial Bootloader

EVB Mode:

- The EVB mode is the most important mode for the beginners to learn the HCS12 microcontroller using a demo board.
- In this mode, the monitor operates as a **ROM** resident monitor/debugger executing from the on-chip flash memory.
- In this mode, user **loses** the *flash memory*, **1024** bytes of on-chip **SRAM**, and one of the **SCI** serial port.

EVB Mode Continued ...

- The portion of the SRAM that maybe used by the user begins at \$1000 and ends at \$3BFF.
- D-Bug12 uses the remainder of the SRAM that begins at \$3C00 and ends at \$3FFF.
- The user program can also be loaded onto the on-chip EEPROM for execution.
- When powering up the demo board or pushing RESET button, D-Bug12 displays the following text and issues the symbol ‘>’ on the next line that means you are in D-Bug12 monitor environment and it is ready to accept command(s).

D-Bug 12 4.0.0b24

Copyright 1996 - 2002 Motorola Semiconductor

For Commands type "Help"

>

- When running a program, D-Bug12 will place the terminal cursor on a blank line, where it will remain until the control is returned back to D-Bug12.

Dragon12-Plus-USB Board

- Download [Dragon12-Plus-USB software and documentation](http://www.evbplus.com/dragon12p_usb_files/dragon12p_usb.html)
- http://www.evbplus.com/dragon12p_usb_files/dragon12p_usb.html
- After download is done, extract the zip file and then click on setup.bat.
- The c:\Dragon12P\asmchanges.txt is an important file to read and it covers the history of AsmIDE's upgrades.
- The user's manual can be found in c:\Dragon12P\document.

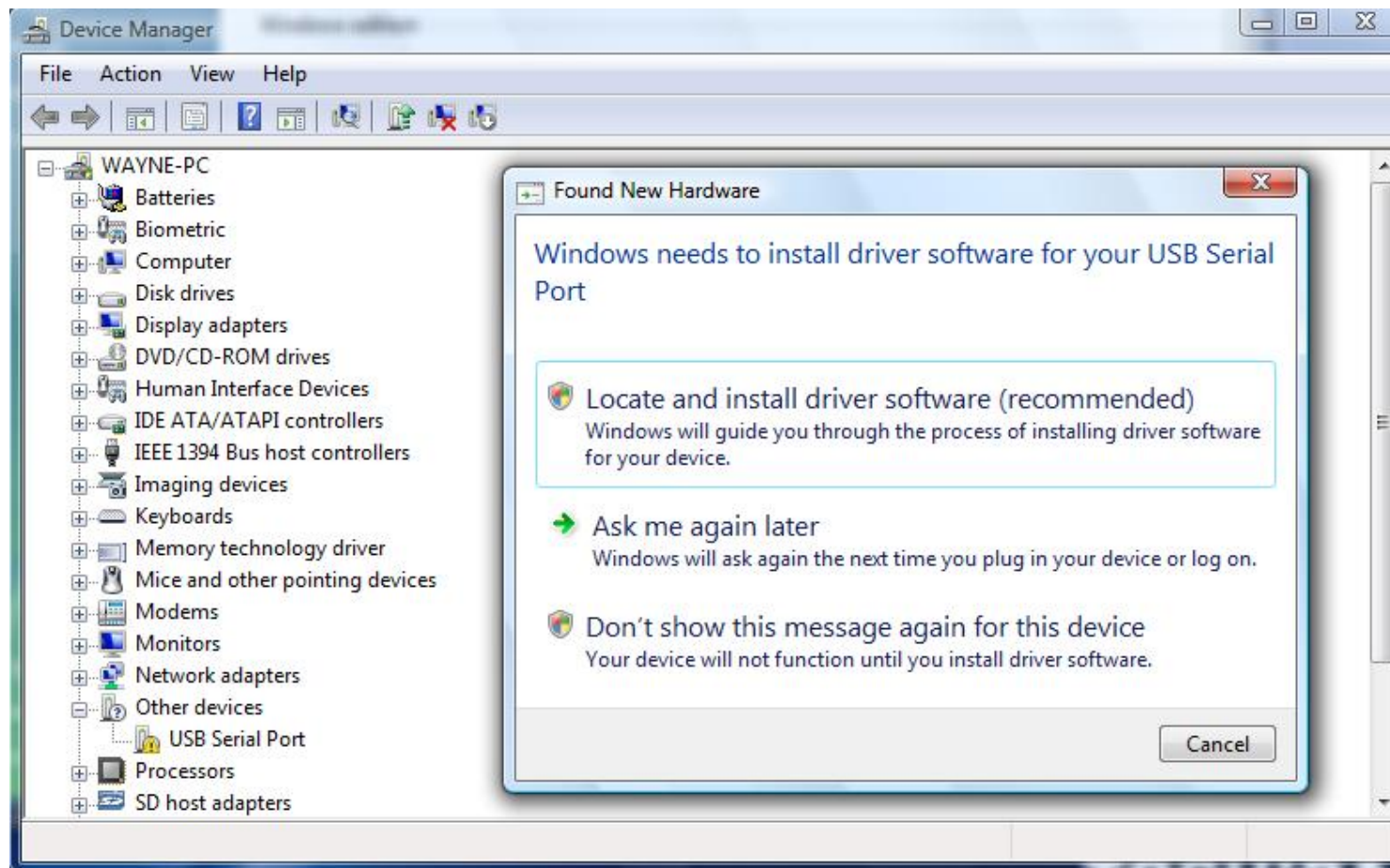
Verify monitor type on your Dragon12-Plus-USB board:

- Plug in the AC adapter that comes with your package and observe the LEDs.
- If your board is pre-installed with Freescale D-Bug12 monitor, the 8 port B LED indicators will light up from **left to right** and the speaker will chirp once when the board is powered up. If the chirp is too soft you can remove the sticker on the speaker to increase the volume.

Note: The Dragon12-Plus-USB board comes with a built-in USB interface based on the **FT232RL**. The driver for the FT232RL must be installed properly before using the Dragon12-Plus-USB board.

Install the FT232RL driver:

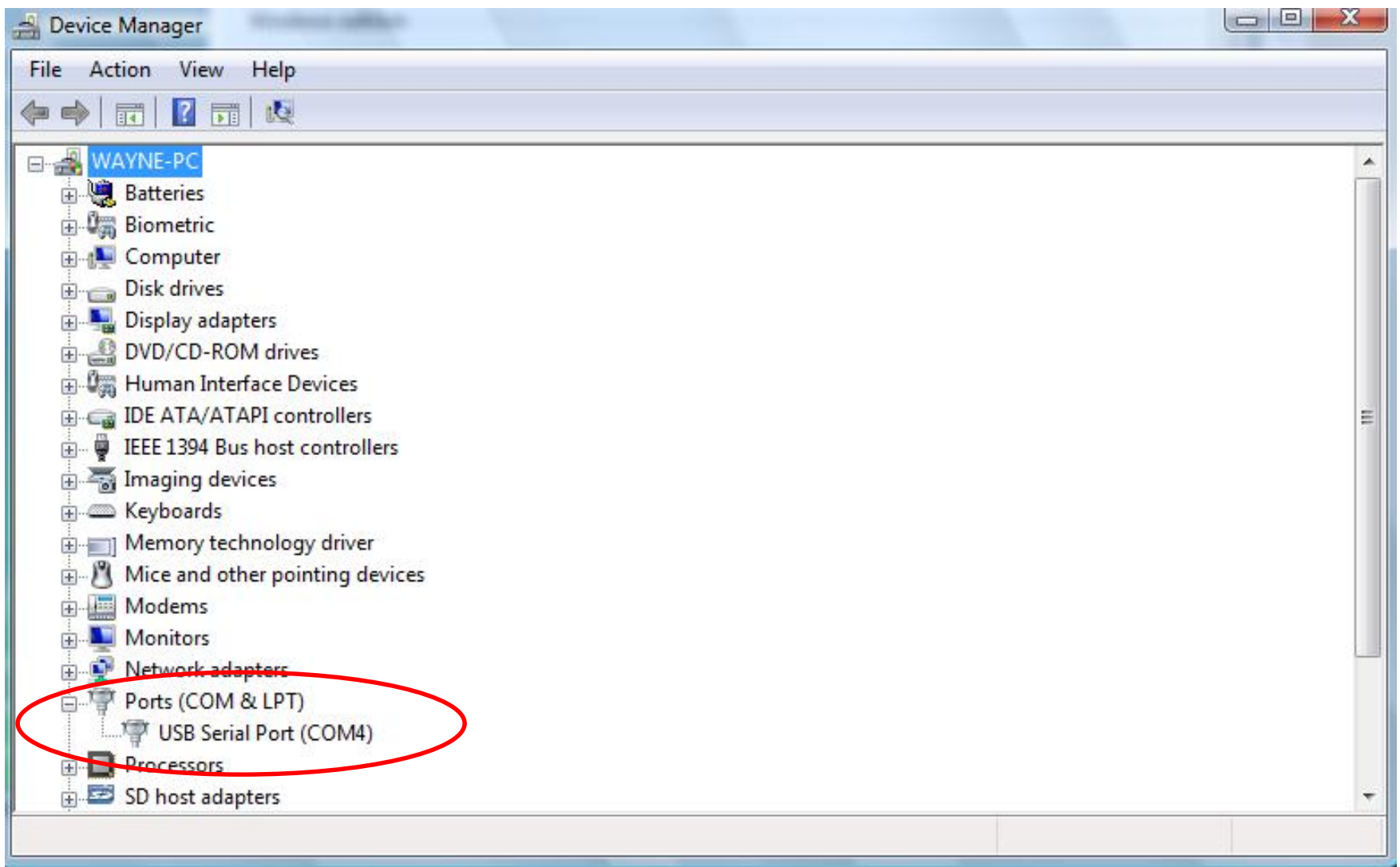
- Installing the FT232RL driver is very easy. The Windows XP, Vista and Windows 7 will search web and download the newest driver and install it on your PC automatically.
- As soon as you connect the FT232RL based adapter or board to a USB port on your PC, the following "Found New Hardware" screen will appear. If not, or you have Windows 98 then you may have to download the driver manually from: <http://www.ftdichip.com/Drivers/VCP.htm>



- Select the top choice "Locate and install driver software", then windows will install the driver for you.
- Once the driver for the FT232RL is installed you have to verify the installation See the procedures below.

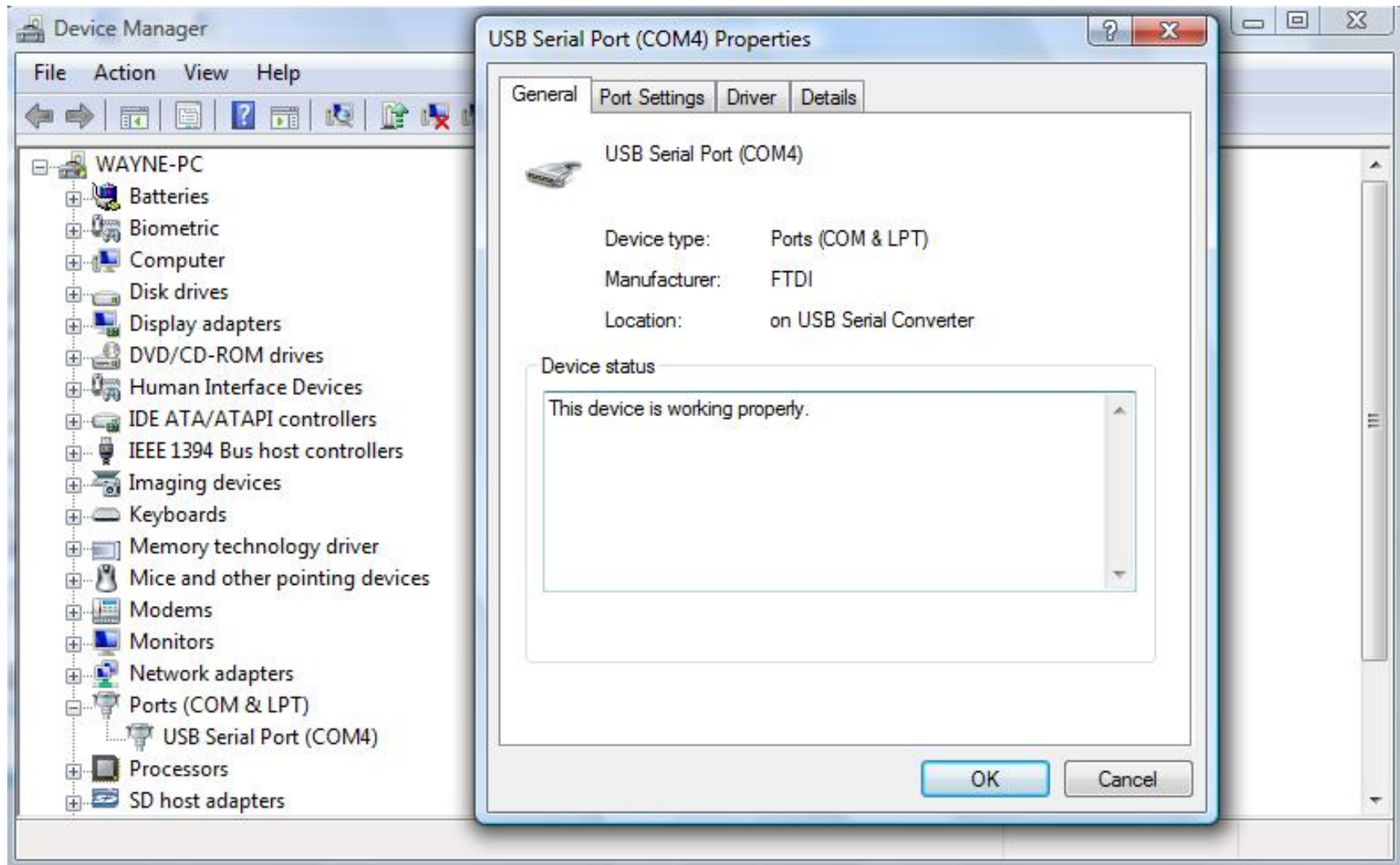
Verify the FT232RL driver:

- To verify if the FT232RL driver is properly installed and determine the COM port assignment of the FT232RL by Device Manager:
 - ☐ For Windows XP, Windows 2000, Windows ME, or Windows 98: Click **Start -> Control Panel -> System -> Hardware -> Device Manager**. Double-click on Ports (COM & LPT1).
 - ☐ For Windows Vista: Click **Start -> Computer -> System Properties -> Device Manager**. Double-click on Ports (COM & LPT1).
- In the following picture you should see the USB Serial Com Port (COM4) under the Ports. This also means that the Device Manager has assigned the adapter to COM4 port.

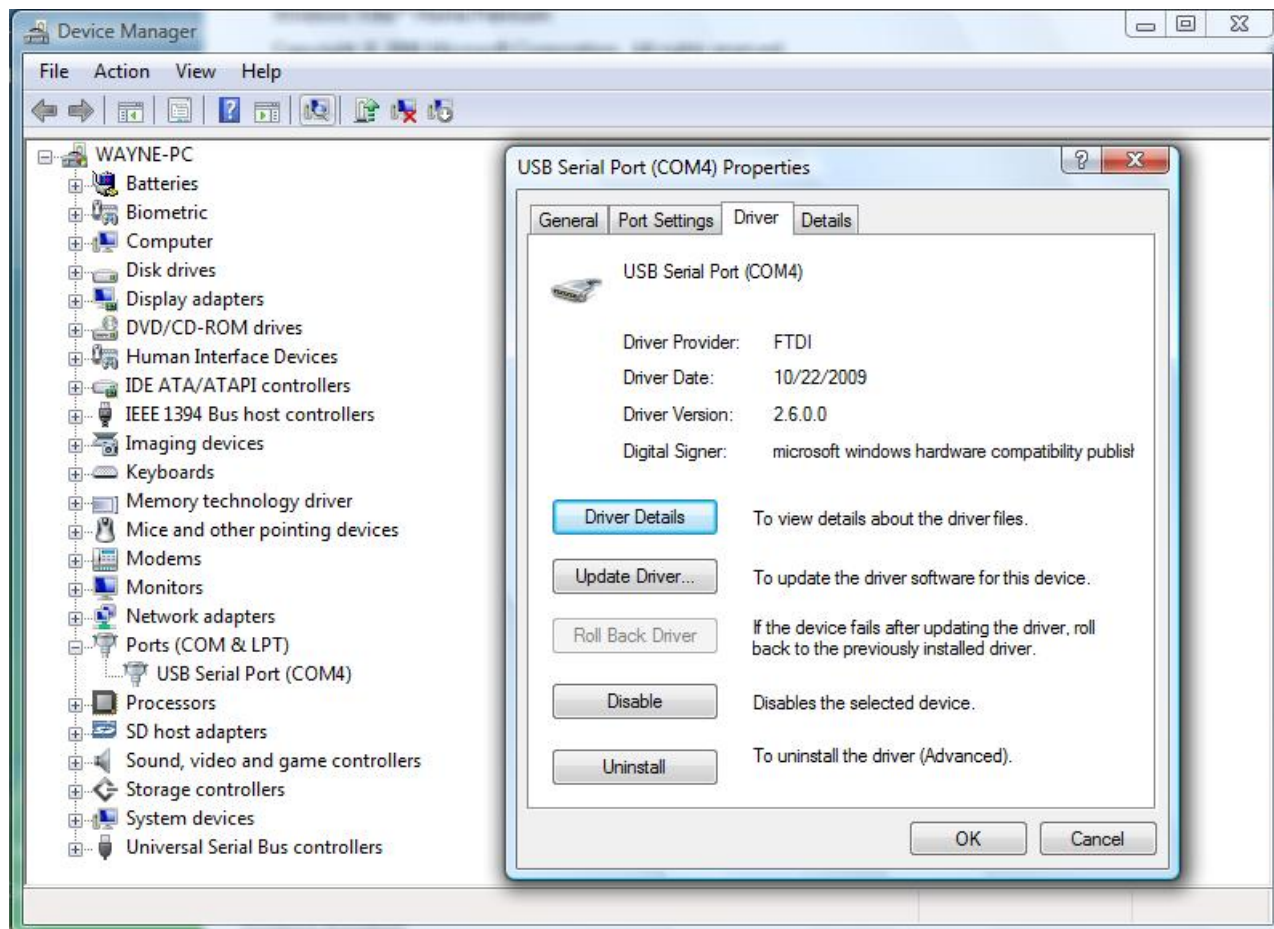


Note: The COM port assignment of the USB to Serial adapter varies on your PC hardware configuration. Device Manager will randomly assign an unused COM port number to the FT232RL. The COM4 is just an example here and it could be a different COM port on your PC. The same adapter can be assigned to different COM port numbers by different PCs

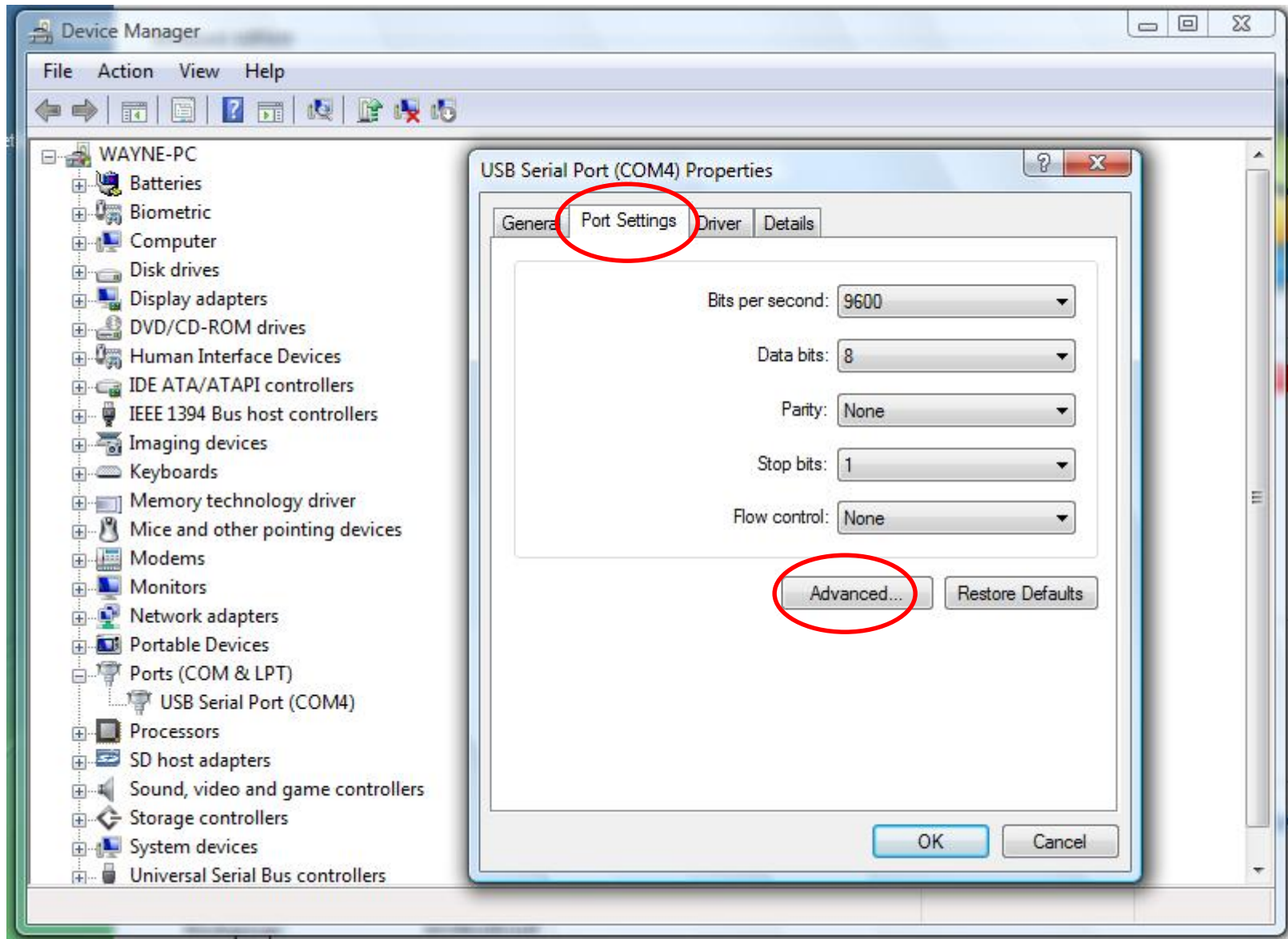
- The IDE software in your PC might not work with COM4 so you may need to reassign the COM Port of FT232RL to another port number to work with your IDE. You can double-click on the device shown on the above picture, USB Serial Com Port (COM4) to view properties as shown below:



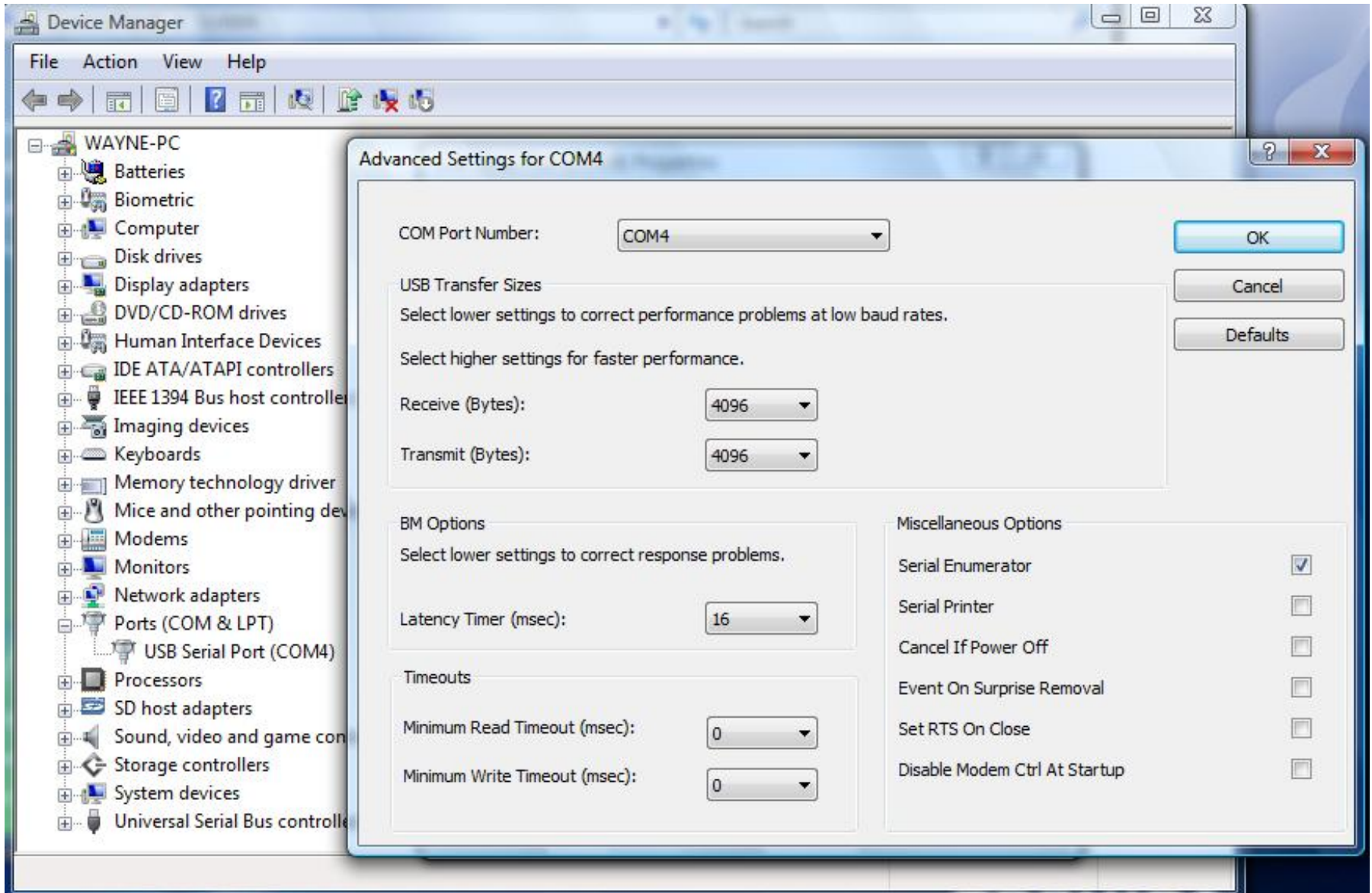
- Inside the USB-to-Serial Comm. Port Properties, click on the **Driver tab**, make sure that the driver you installed is the version **2.6.0.0 or newer**. If it's an old driver version, then click on the Update Driver tab, the Windows XP and Vista O/S will search web and download the newest driver and install it on your PC automatically. If you have a Windows 98 O/S then you may have to download the driver manually from the FT232RL web site before continuing to the next step.
- Until you successfully installed the new driver you do not need to continue to the next step.



- To reassign the COM port number, click on the Port Settings tab, then click on the Advanced button.

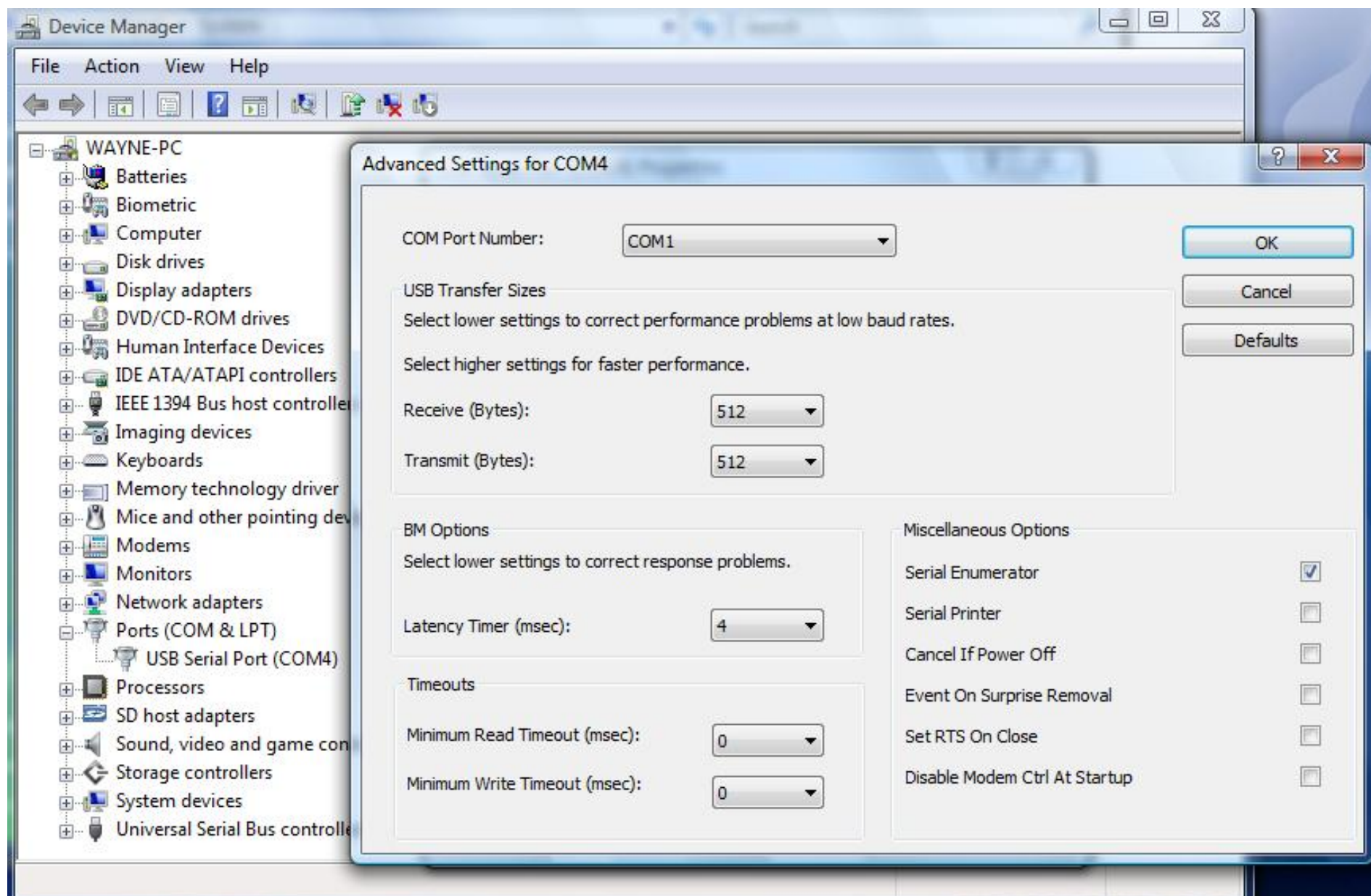


- Inside the Advanced Settings, you will see the COM port number, latency timer, settings for Receive Buffer and Transmit Buffer



- Click on the COM Port Number and check what other port numbers are unused. If you know what COM port that the IDE uses in your PC, you can just reassign the COM port of the adapter to that COM port number, otherwise you may try to reassign it to COM1 or COM2. Click OK when finished. If your PC indicates that COM1 or COM2 are not available but you know they are available, you can ignore your PC's warning and just select COM1 or COM2 anyway.
- Change the Receive and Transmit buffers from 4096 bytes to 512 bytes and latency timer from 16 msec to 4 msec, then click on the "OK" button to finish the verification.

Note: Some DOS programs may only support COM1 and COM2, they may not work if the port is assigned to COM3 and higher.



Important Note!

- In order to establish a reliable USB communication, always connect the Dragon12-Plus-USB board to your PC's USB port first before invoking AsmIDE, otherwise the AsmIDE will not be able to find a COM port.
- When ending a debugging session, always close the AsmIDE first before unplugging the USB cable from the Dragon12-Plus-USB board, otherwise the AsmIDE may hang up and you need to re-establish the USB communication again.
- In case the AsmIDE hangs up, you need to close the AsmIDE first, then unplug the USB cable from your Dragon12-Plus-USB board, wait a few seconds before re-plugging the USB cable, then wait a few more seconds and allow the USB connection to be re-established.
- After cycling the USB connection, you can invoke the AsmIDE again and it may restore the USB communication. If this does not work, you need to restart your PC, but in order to avoid this, always close the AsmIDE before unplugging the USB cable.
- If restarting the PC does not solve the problem, you may need to re-install the USB driver.

Important Downloads

Download the MC9S12DG256 Device User Guide :

http://www.freescale.com/files/microcontrollers/doc/data_sheet/9S12DT256DGV3.pdf

Download the D-Bug12 Reference Guide, DB12RG4.pdf :

<http://www.ee.nmt.edu/~rison/datasheets/DB12RG4.pdf>

Download the AN2548.pdf (serial monitor) :

http://www.freescale.com/files/microcontrollers/doc/app_note/AN2548.pdf

Download the AN2153.pdf (serial bootloader) :

http://www.freescale.com/files/microcontrollers/doc/app_note/AN2153.pdf

EVB Mode Continued ...

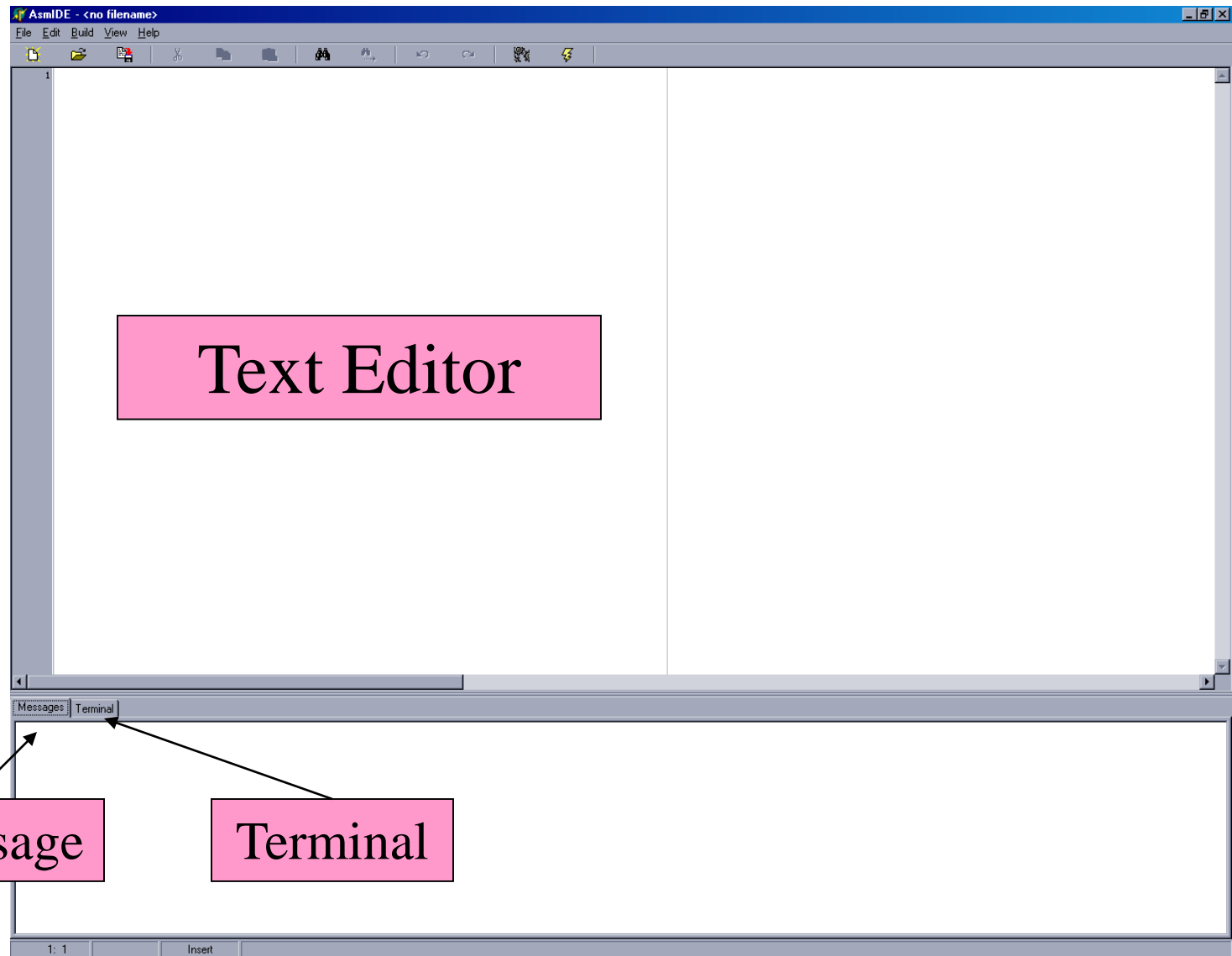
- If running program fails to return to D-Bug12, pressing the EVB's **RESET** button will cause the running program to halt execution and initiate the D-Bug12 initialization sequence.
- Using this method to regain control fails to report any information to the programmer on **why** or **how** the program may have failed.
- Dragon 12 provides an **ABORT** push button which is wired to **XIRQ**. Therefore, it generates an interrupt that causes the running program to halt and return the control back to D-Bug12 monitor and displays the contents of the registers as well.
- Note that even though the HCS12Dx256 parts contain **4 KB** of EEPROM, only the upper **3 KB** is available to the user and the lower 1 KB is overlaid with the I/O registers.
- The next slide shows only the 64-KB memory map of the MC9S12Dp256 microcontroller.
- Most of the D-Bug12 code occupies the on-chip paged flash memory beginning on page **38**.

EVB Mode Continued ...

Address Range	Description
\$0000 - \$03FF	I/O registers
\$0400 - \$0FFF	On-chip EEPROM
\$1000 - \$3BFF	On-chip SRAM (available to user)
\$3C00 - \$3FFF	On-chip SRAM (D-Bug12)
\$4000 - \$EE7F	D-Bug12 code
\$EE80 - \$EEBF	User-accessible function table
\$EEC0 - \$EEFF	Customization data
\$EF00- \$EF8B	D-Bug12 startup code
\$EF8C - \$EFFF	Secondary reset/interrupt table
\$F000 - \$FFFF	Bootloader

Starting AsmIDE:

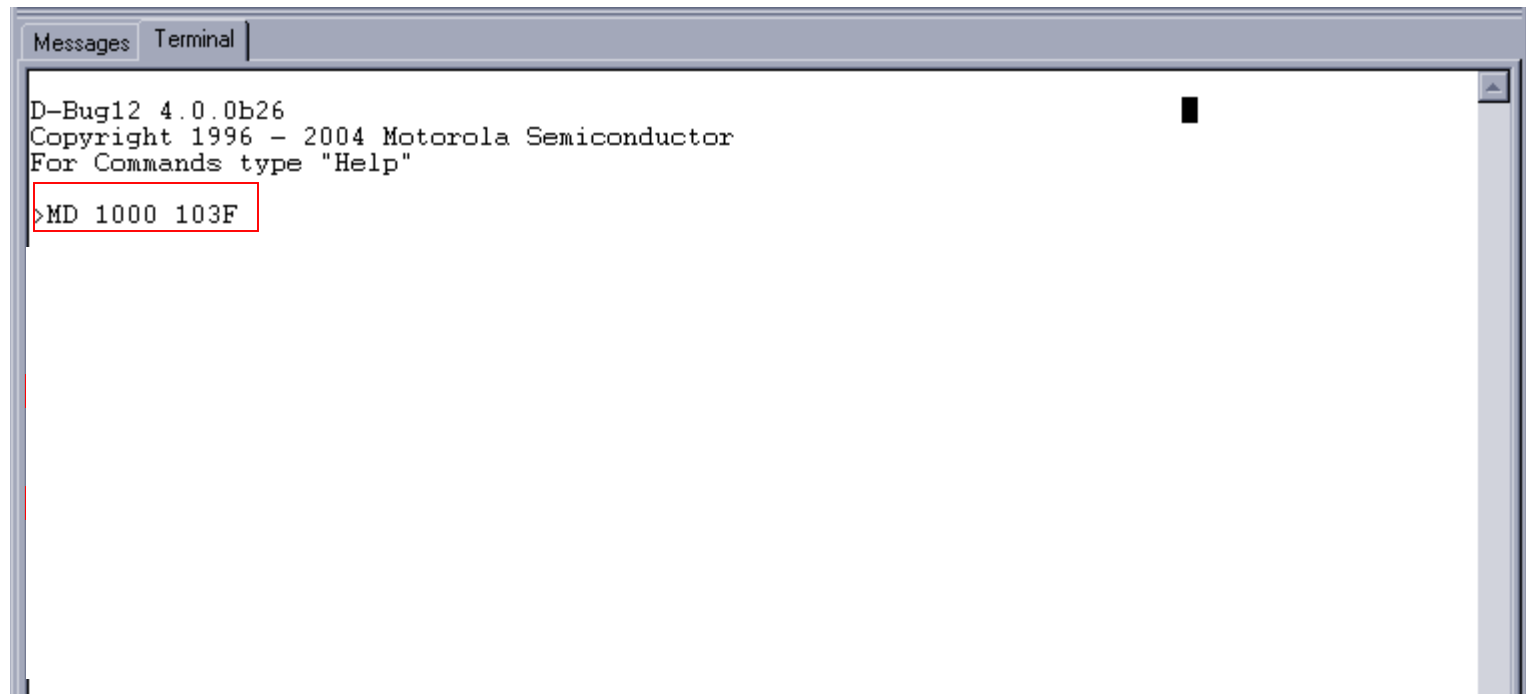
- The AsmIDE can be started by clicking on its icon.



Using the D-Bug12 Commands:

- The D-Bug12 commands provided to help the program-debugging process.
- Some of these commands can be used to set and display the content of the memory locations.

Memory Display MD

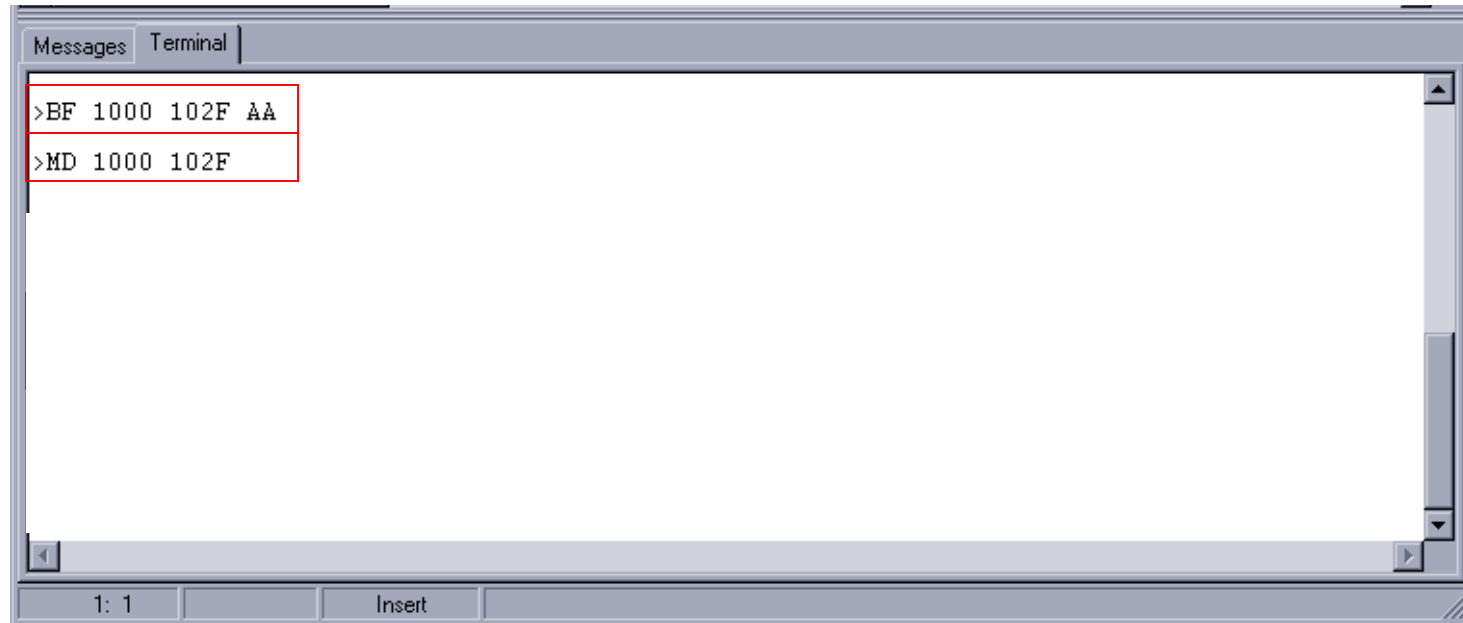


The image shows a screenshot of a terminal window titled "D-Bug12". The window has two tabs: "Messages" and "Terminal". The "Terminal" tab is active, displaying the following text:

```
D-Bug12 4.0.0b26  
Copyright 1996 - 2004 Motorola Semiconductor  
For Commands type "Help"  
>MD 1000 103F
```

The command ">MD 1000 103F" is highlighted with a red rectangular box.

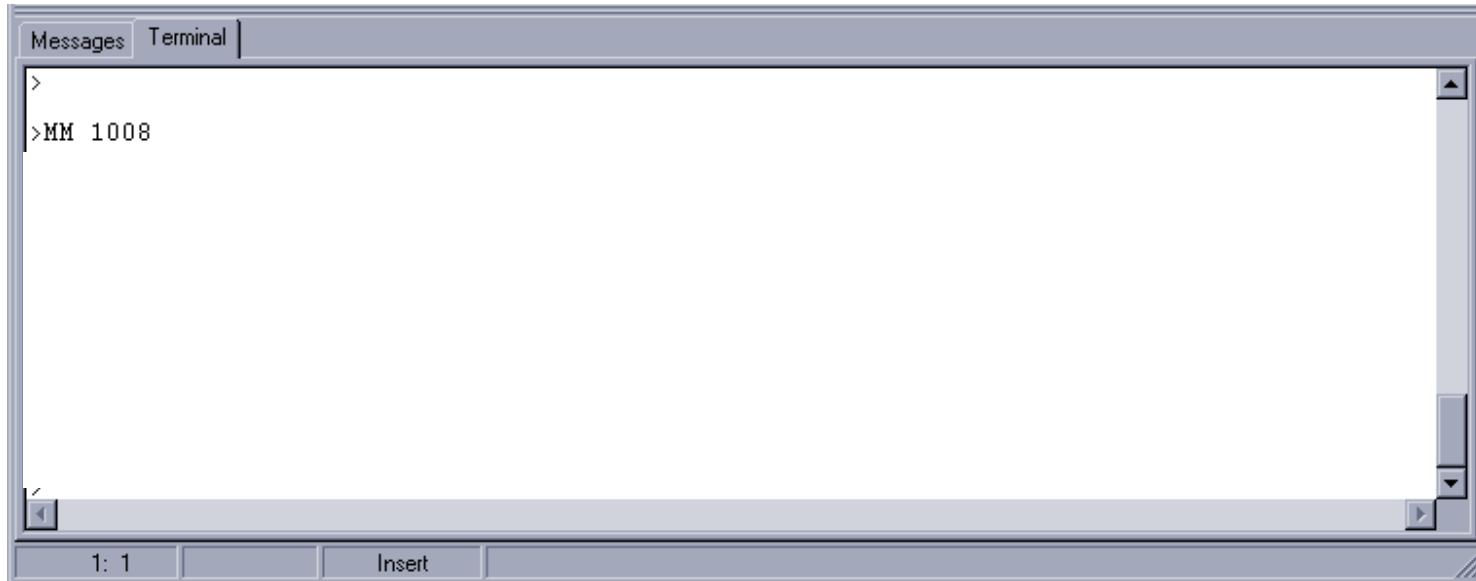
Block Fill **BF**



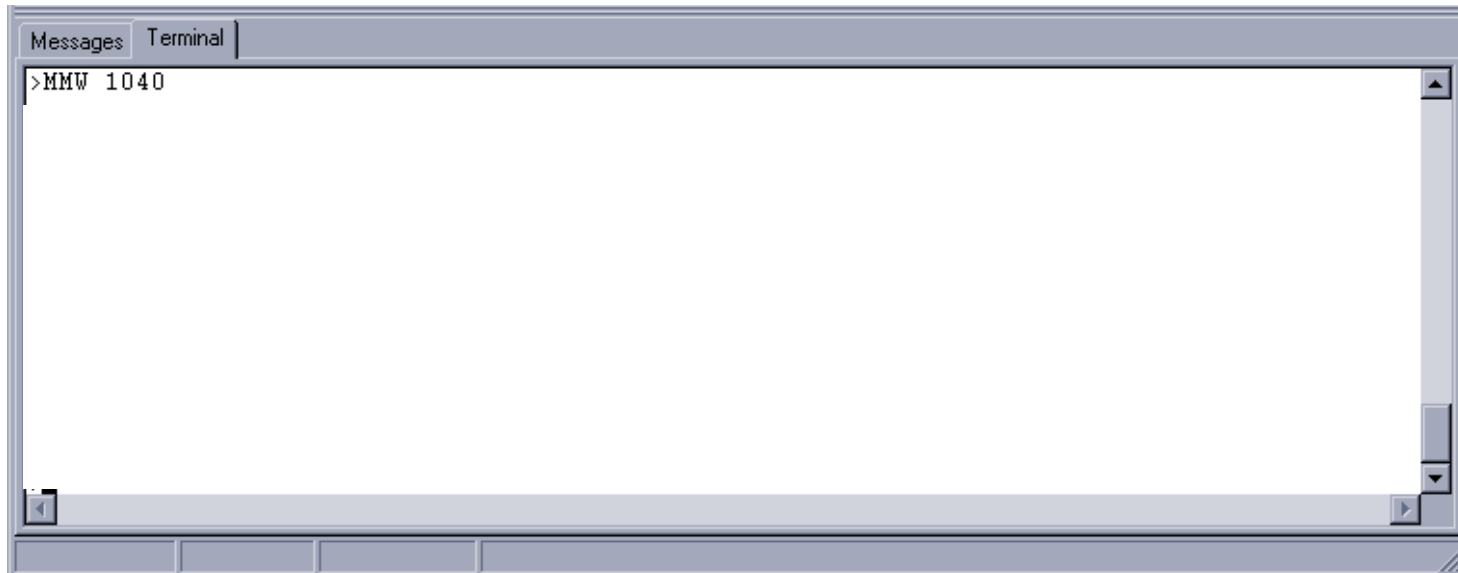
Memory Display Words **MDW**



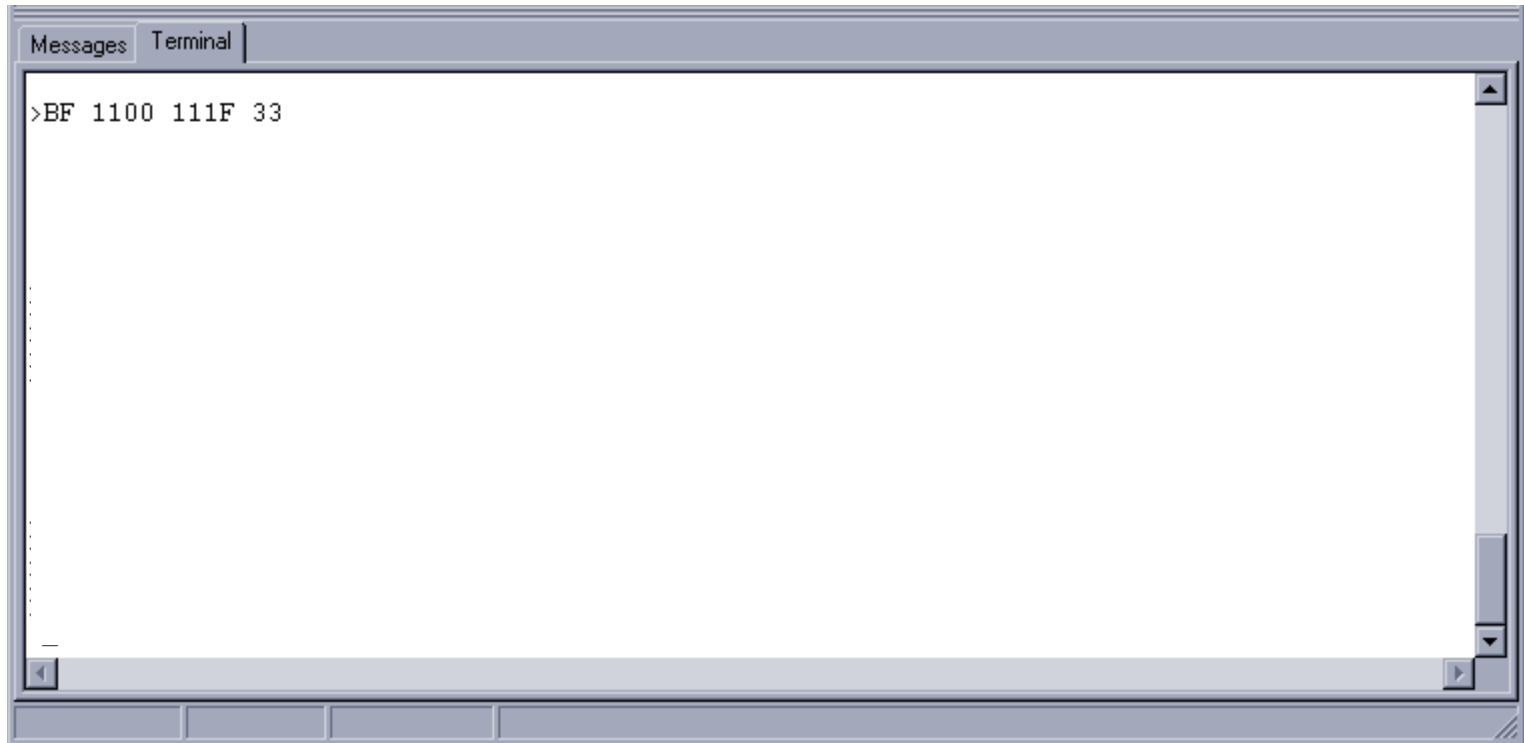
Memory Modify MM



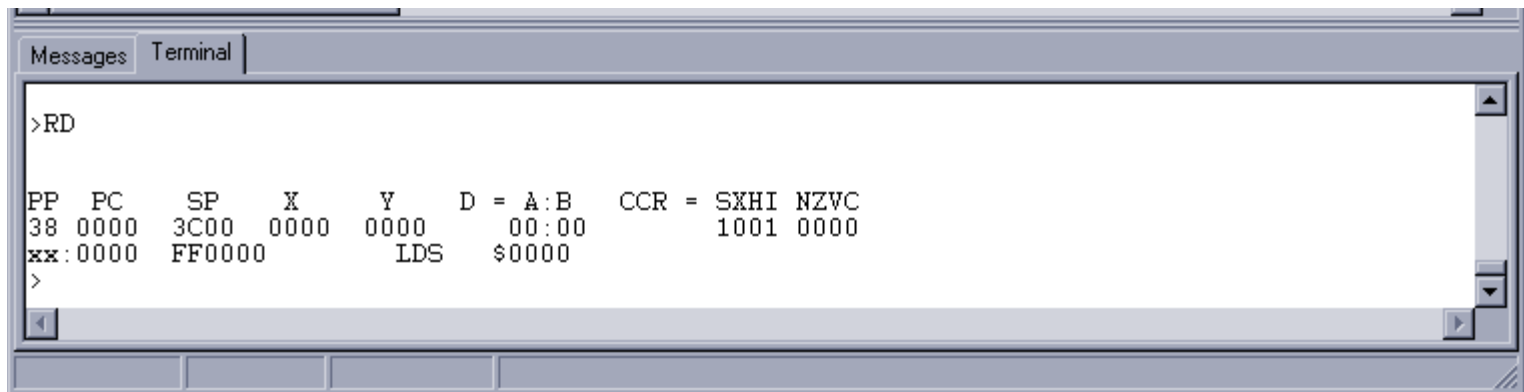
Memory Modify Word MMW



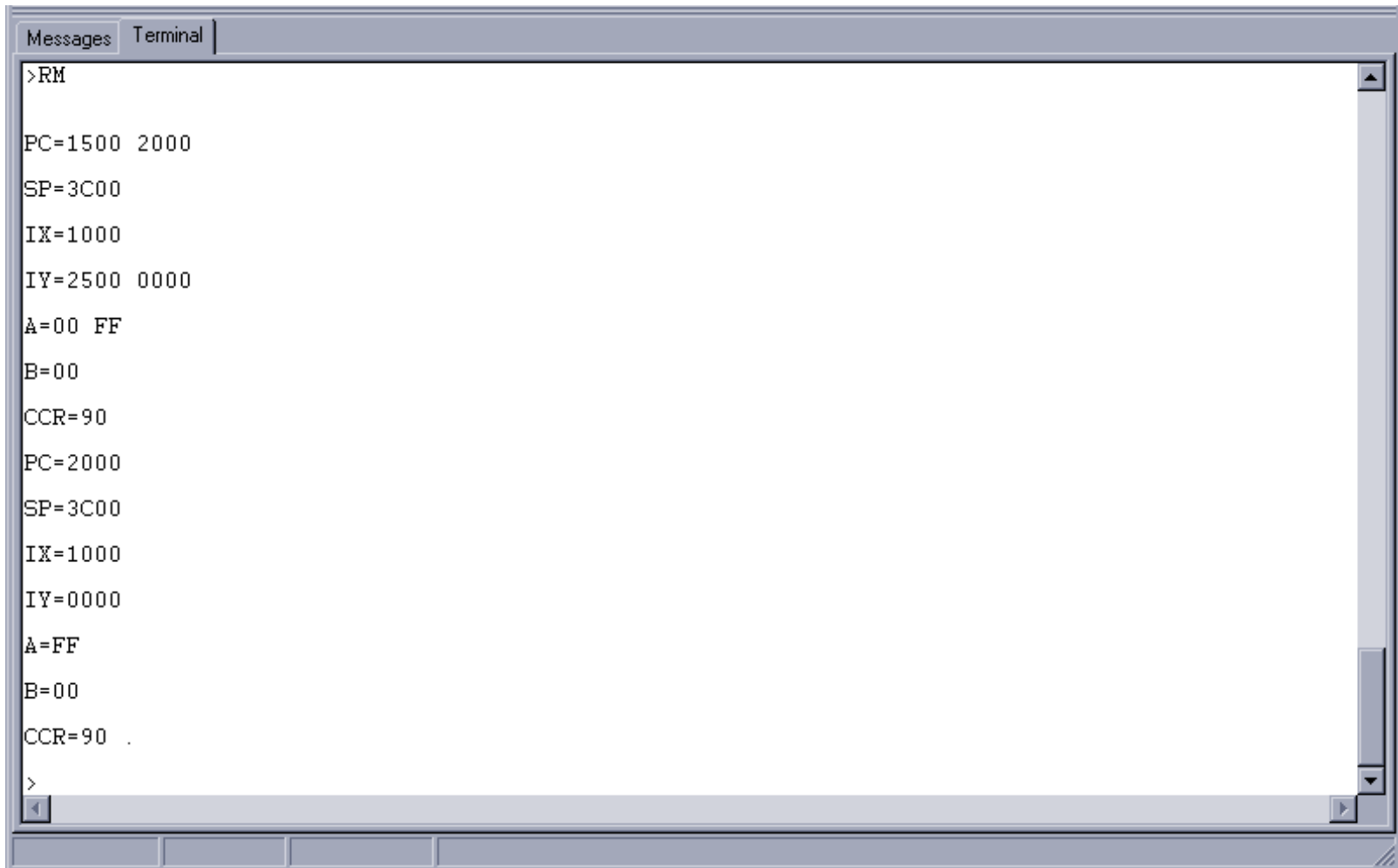
Move Memory Content Move



Register Display RD



Register Modify RM



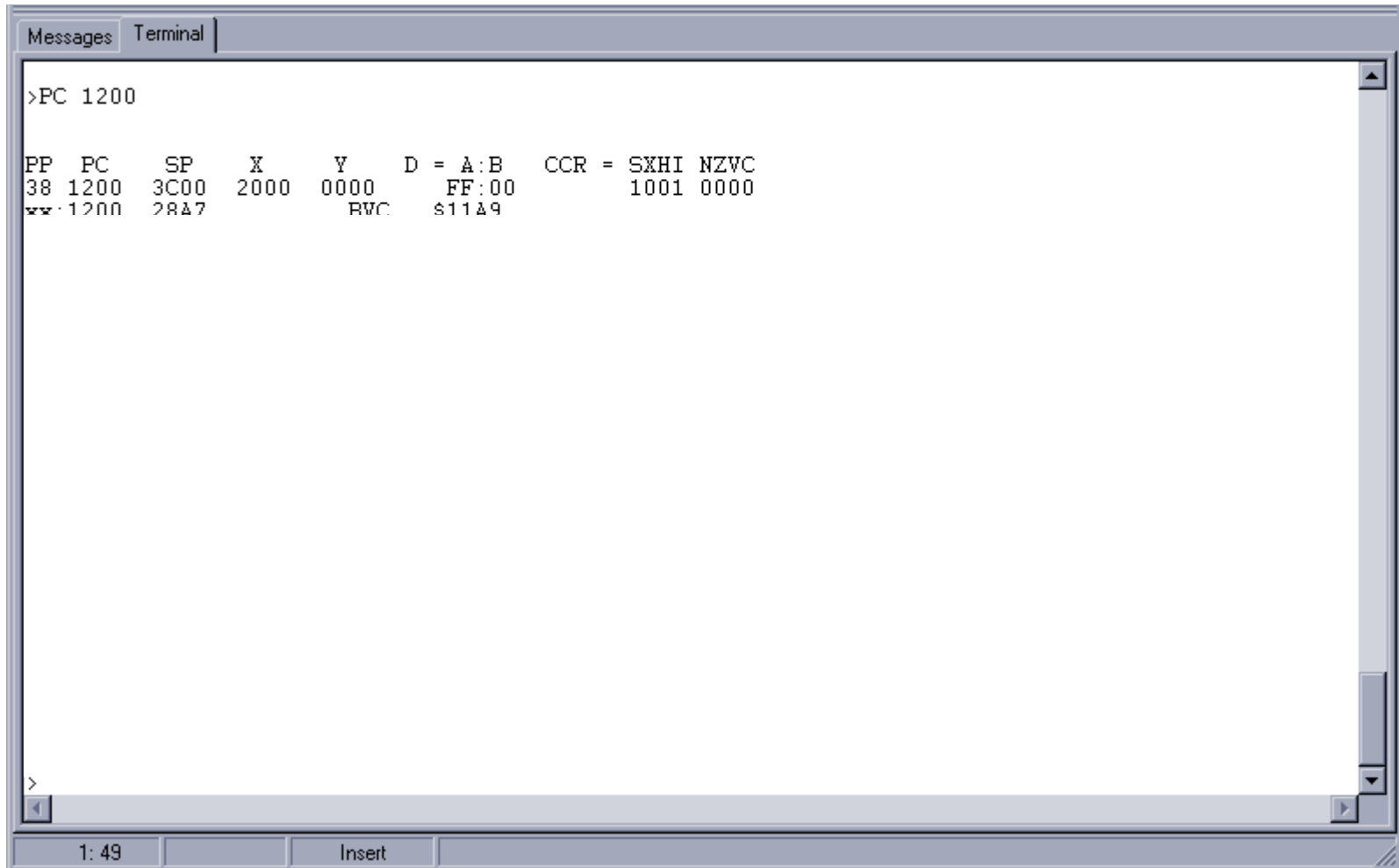
```
>RM  
  
PC=1500 2000  
SP=3C00  
IX=1000  
IY=2500 0000  
A=00 FF  
B=00  
CCR=90  
PC=2000  
SP=3C00  
IX=1000  
IY=0000  
A=FF  
B=00  
CCR=90  
>
```


<Register Name> <Register Value>

- This command allows one to change the value of any CPU register (PC, SP, X, Y, A, B, D, CCR).
- Each of the fields in the CCR may be modified by using the bit names shown in table below:

CCR bit name	Description	Legal Value
S	STOP enable	0 or 1
H	Half carry	0 or 1
N	Negative flag	0 or 1
Z	Zero flag	0 or 1
V	Two's complement over flag	0 or 1
C	Carry flag	0 or 1
IM	IRQ interrupt mask	0 or 1
XIM	XIRQ interrupt mask	0 or 1

Here are some examples of single register modify



The screenshot shows a terminal window with a title bar containing 'Messages' and 'Terminal' tabs. The terminal displays the following text:

```
>PC 1200
```

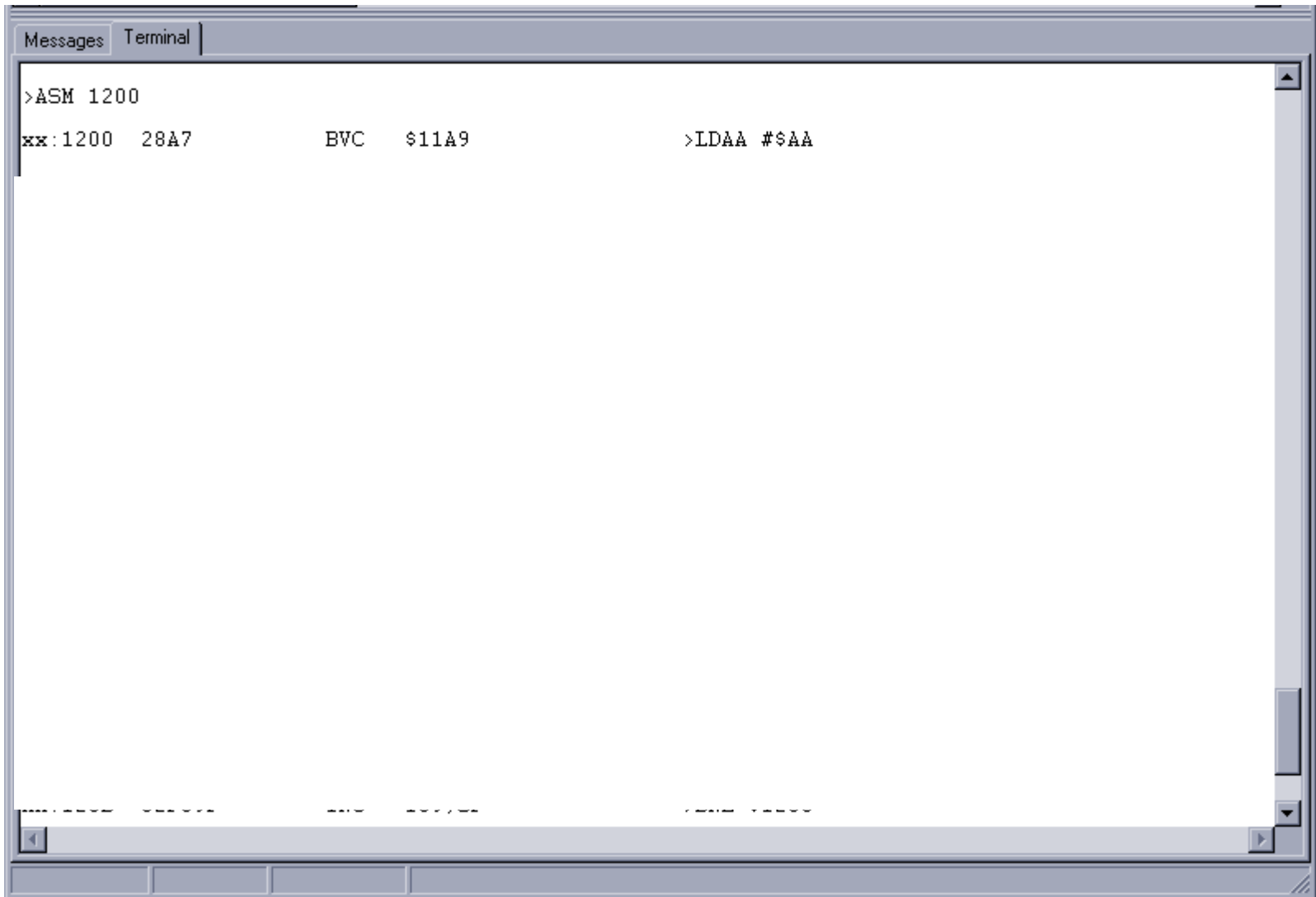
PP	PC	SP	X	Y	D = A:B	CCR =	SXHI	NZVC
38	1200	3C00	2000	0000	FF:00		1001	0000
xx	1200	28A7		RVC	811A9			

At the bottom of the terminal window, there is a status bar with a clock showing '1: 49' and a button labeled 'Insert'.

Interactive Assembler ASM <Address>

- This command invokes the one-line assembler/disassembler.
- It allows memory contents to be viewed and altered using assembly language mnemonics.
- Each entered source line is translated into object code and placed into memory at the time of entry.
- When displaying memory contents, each instruction is disassembled into its source mnemonic form and displayed along with the hex object code and any instruction operands.
- This is case insensitive assembler.
- When entering branch instructions, the number placed in the operand field should be the absolute destination address of the instruction.
- The assembler calculates the two's complement offset of the branch and places the offset in memory with the instruction.
- The assembly/disassembly process may be terminated by entering a period as the first non-space character following the assembler prompt.

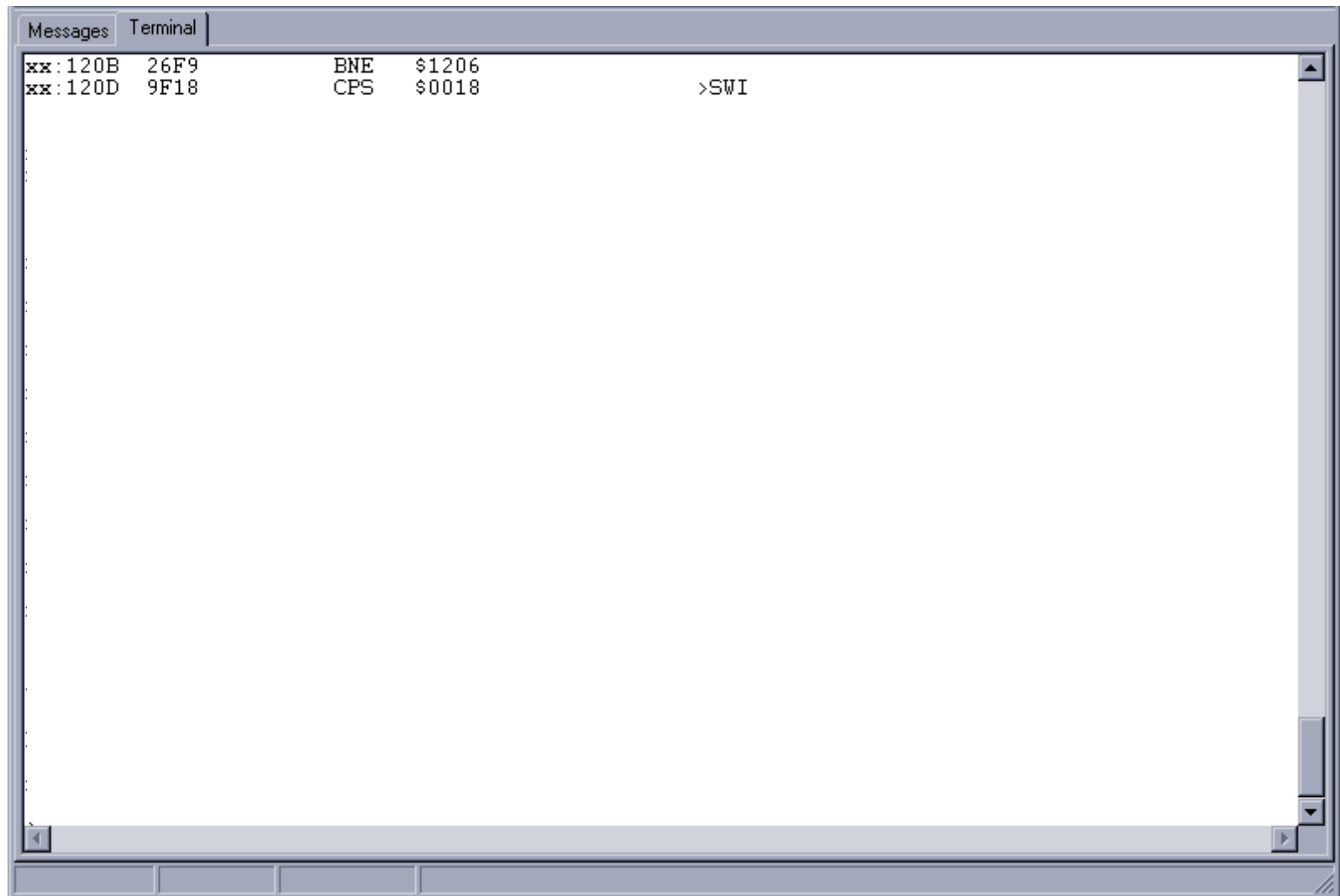
Example:



A screenshot of a terminal window with a title bar and two tabs: "Messages" and "Terminal". The "Terminal" tab is active. The terminal displays the following text:

```
>ASM 1200  
xx:1200  28A7          BVC    $11A9          >LDAA  #$AA
```

The terminal window has a standard scrollbar on the right side and a status bar at the bottom.



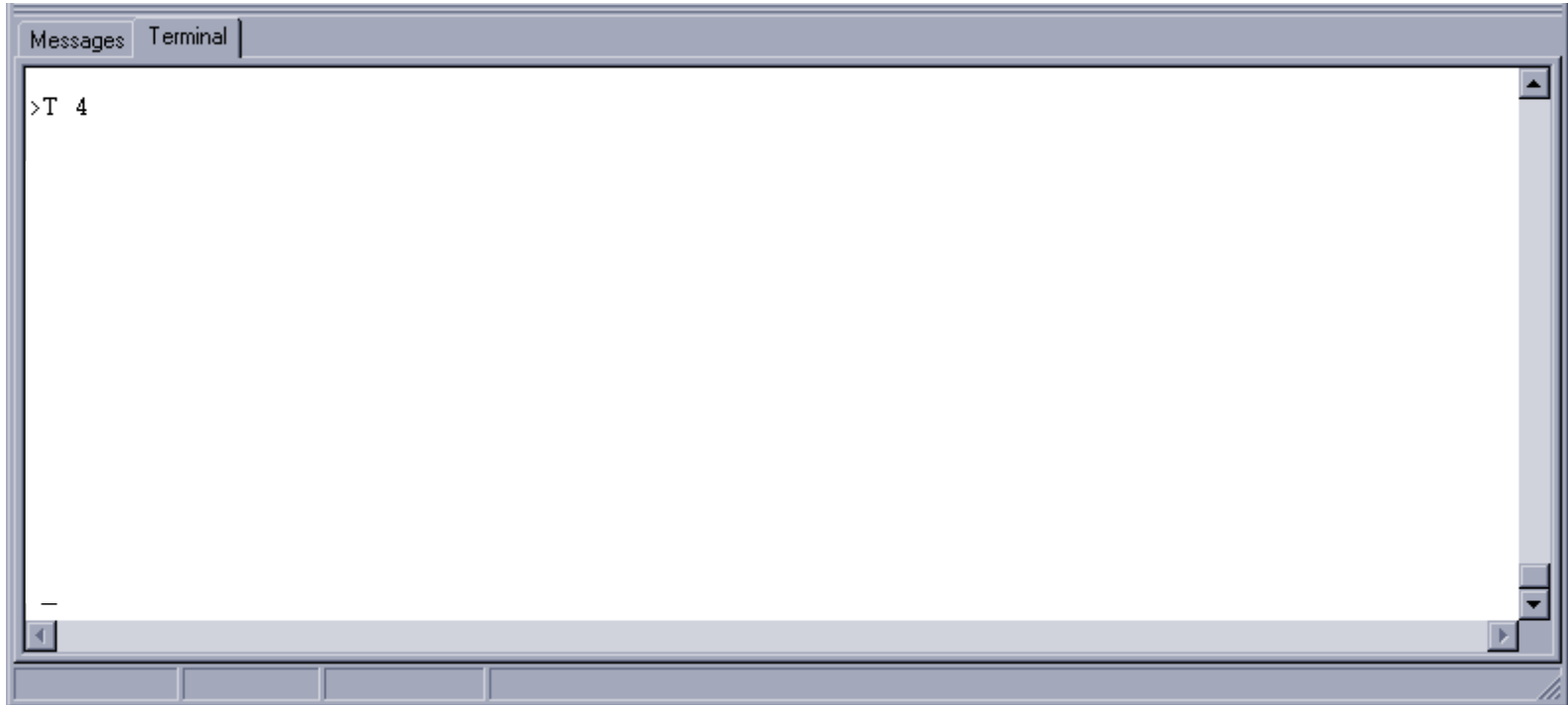
Breakpoint Set **BR**



Go Till **GT** <Address>

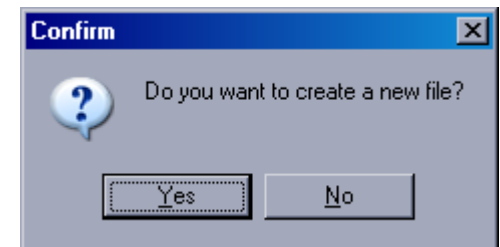


Trace T [<count>]



How to Generate a Cross Assembly Program:

- Go to **File** menu on top window and press it.
- Select New and a popup confirm box appears.
- Simply click on **Yes** to get rid of the box.
- Start to enter a new program.



Cross-assembler Line Statement Format

A definite format must be followed when writing each line of the source code file. Each line is divided into four fields. They are:

- ❶ Label Field,
- ❷ Op-code (operation code) field,
- ❸ Operand field, and
- ❹ Comment field.

This is a simple example of a source program for creating a delay loop. Notice that all four fields has been used. Furthermore, a *tab key* is used between fields to guarantee that each field has started from the same column.

LABEL FIELD	OPERATION FIELD	OPERAND FIELD	COMMENT FIELD
	LDAB	#26	;load acc. B with count = 26_{DEC}
DELAY	DECB		;decrement the counter
	BNE	DELAY	;stay in delay loop till counter = 0.

Label Field

- All labels begin in column 1 and end with the space or Colon.
- This assembler allows labels of one to fifteen characters.
- Labels could be any printable character with exception of *space*, *colon*, *comma*, and *semicolon*.
- Upper- and lowercase letters are distinct (case sensitive).
- A number can not be the first character, but can be used as part of label.
- It is a good practice to keep the length of labels under seven characters.

Operation Field

- Operation field is the second field.
- If the label field is not being used, you must leave at least one space from the left margin.
- Use *tab key* to keep all the operation instructions on the same column number.
- The operation field contains an instruction mnemonics (i.e., LDAA, STAB, DEX, etc.).
- The operation field is also used for *assembler directives* as well as *macro call*.

Operand Field

- Operand field is the third field of source line.
- At least one space must separate operand and operation fields.
- This field contains instruction operands, argument for assembler directives, or arguments for macro calls.
- The operand field may contain numbers to be used as data or address.
- There are five symbols to identify the type of number:

Prefix	Description
None	Decimal Number
\$	Hexadecimal Number
@	Octal Number
%	Binary Number
'	String of ASCII Characters

Comment Field

- The fourth and the last field of a source line is comment line.
- Comment field should be separated from last field by at least a space and *semicolon*.
- Comment field may consist of any printable ASCII character.
- This field is used for documentation of the program.

Note! It is possible to use the entire source line as a comment line. This is accomplished by inserting an asterisk (*) as the first printable character in the line.

Assembler Directives

- *Assembler directives* are instructions to the cross-assembler to perform a particular task other than creating a *machine code* for an instruction.
- The available assembler directives vary with the assembler.
- For more detail and complete list of directive assemblers for our *AsmIDE*, one should refer to the User's Manual of the assembler.
- Some of the directives have more than one option.
- There are 42 assembler directives listed in **Motorola's reference manual**.
- Some of these *directives* that are commonly encountered will be discussed.
- In the following discussion, statements enclosed in [...] are optional.

Origin (ORG):

- The assembler uses the *location counter* to keep track of memory location where the next machine code byte should be placed.
- When the programmer wants to force the program or data array to start from a certain memory location, then this directive must be used.
- Your program always starts with this directive.

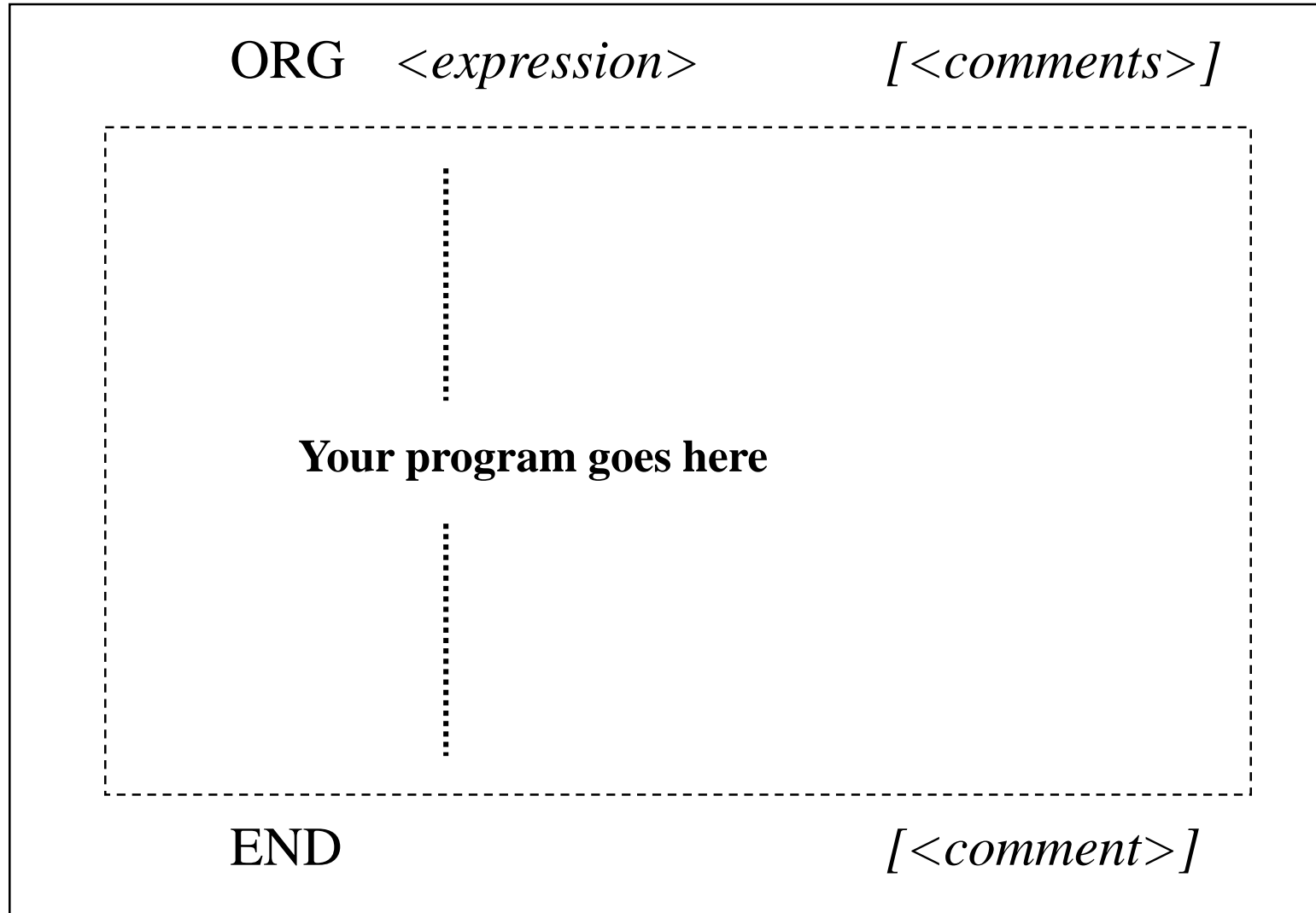
ORG \$1000

End (END):

- The end directive indicates the logical end of the source program.
- Any statement following the end directive is ignored.
- A warning message will occur if the end directive is missing from the source code; however, the program will still be assembled correctly.

END

ORG, and END Directives:



Define Byte (DB), Define Constant Byte (DC.B), Form Constant Byte (FCB):

- These three directives define the value of a byte or bytes that will be placed at a given memory location.
- The DB (or FCB, or DC.B) directive assigns the value of the expression to the memory location pointed to by the *location counter*. Then the location counter is incremented.
- Multiple bytes can be defined at a time by using commas to separate the arguments.

ORG \$1000

array DB \$11,\$22,\$33,\$44,\$55

- The assembler uses the array (which is \$1000), as address of the first byte whose initial value will be \$11.
- Therefore, the contents of memory locations \$1000 - \$1004 will be \$11, \$22, \$33, \$44, and \$55 respectively.

Define Word (DW), Define Constant Word (DC.W), Form Double Byte (FDB):

- These three directives define the value of a word or words that will be placed at a given address.

```
Vect_tbl      DW      $1234,$5678
```

- Initializes the two words starting from the current location counter to \$1234 and \$5678, respectively.
- After this statement the location counter will be incremented by 4.

Form Constant Character (FCC):

- This directive allows us to define a string of characters (a message).
- The first character in the string is used as the delimiter.
- The last character must be the same as the first character because it will be used as the delimiter.

Form Constant Character (FCC) *continued* ...

- The delimiter must not appear in the string.
- The *space character* can not be used as the delimiter.
- Each character is encoded by its corresponding ASCII code.

alpha FCC 'def'

- This directive will generate the following values in memory:

\$64,\$65,\$66

- The assembler will use the label *alpha* to refer to the address of the first letter, which is stored as the byte \$64.
- The following slide will illustrate all the printable ASCII codes for user's reference.

ASCII Characters Table

	00	10	20	30	40	50	60	70
0			spc	0	@	P	`	p
1			!	1	A	Q	a	q
2			"	2	B	R	b	r
3			#	3	C	S	c	s
4	eof		\$	4	D	T	d	t
5			%	5	E	U	e	u
6			&	6	F	V	f	v
7			'	7	G	W	g	w
8	bsp		(8	H	X	h	x
9)	9	I	Y	i	y
A	lf		*	:	J	Z	j	z
B			+	;	K	[k	{
C	ff		,	<	L	\	l	
D	cr		-	=	M]	m	}
E			.	>	N	^	n	~
F			/	?	O	_	o	Del

Fill Memory (FILL):

- This directive allows a user to fill a certain number of memory locations with a given value.
- The syntax of this directive is as follows:

[label] FILL value,count ;[comment]

where the number of bytes to be filled is indicated by *count* and the value to be filled is indicated by *value*. For example:

SpaceLine FILL \$20,40

Define Storage (DS), Define Storage Byte (DS.B), Reserve Memory Byte (RMB):

- Each of these three directives reserves a number of bytes given as the argument to the directive.
- The location counter will be incremented by the number that follows the directive mnemonic.

```
Buffer          DS      100
```

- Reserve 100 bytes in memory starting from the location represented by the label *buffer*.

Define Storage Word (DS.W), Reserve Memory Word (RMW):

- Each of these directives increments the location counter by the value indicated in the number-of-words argument multiplied by two.

```
dbuff          DS.W    20
```

- Reserves 40 bytes starting from the memory location represented by the label *dbuff*.

Equate (EQU):

- This directive assigns a value to a label. Using EQU to define constants will make a program more readable.

```
loop_cnt       EQU     40
```

(LOC):

- This directive increments and produces an internal counter used in conjunction with the backward tick mark (‘).
- By using the LOC directive and the ‘ mark, you can write a program segment like the following example without thinking of new labels:

	LOC			LOC	
	LDAA	#40		LDAA	#40
LOOP’	DECA		LOOP1	DECA	
	BNE	LOOP’		BNE	LOOP1
	LOC			LOC	
	LDX	#\$2B00		LDX	#\$2B00
LOOP’	DEX		LOOP2	DEX	
	BNE	LOOP’		BNE	LOOP2

```
*****
*   Add two N-bytes number and store the result. Least significant byte stored first. *
*   N is stored at location before the first N-byte number.                       *
*****
```

```

      ORG      $1000
SIZE  DB      6                      ;number of bytes in each number
NUM1  DB      $5B,$FC,$59,$DF,$49,$20 ;first number

      ORG      $1011                      ;starting address of second number
NUM2  DB      $22,$6B,$80,$A8,$52,$7C    ;second number

      ORG      $1100                      ;starting address of program
      LDAB     SIZE                      ;get length of each number
      LDX      #NUM1                    ;point X at first number
      CLC                                           ;no carry to begin with
LOOP  LDAA     0,X                      ;add two numbers starting with
      ADCA     $10,X                    ;least significant byte
      STAA     $20,X                    ;store the result
      INX                                           ;update pointer
      DECB                                           ;update counter
      BNE      LOOP                    ;check for end of operation
      SWI                                           ;halt

      END
```

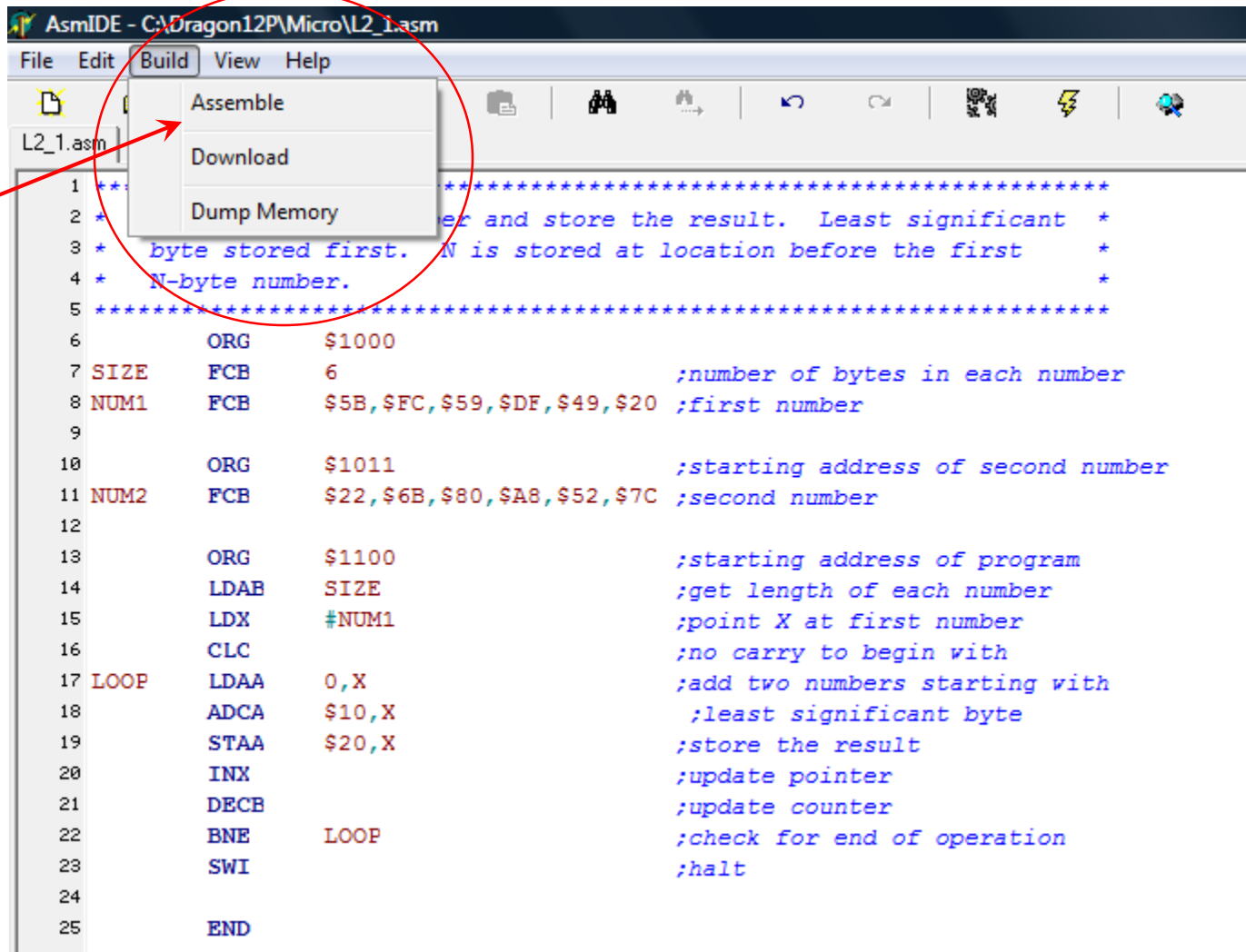
```

AsmIDE - C:\Dragon12P\Micro\L2_1.asm
File Edit Build View Help
[Icons]
L2_1.asm
1 *****
2 *   Add two N-bytes number and store the result.  Least significant *
3 *   byte stored first.  N is stored at location before the first   *
4 *   N-byte number.                                                *
5 *****
6         ORG      $1000
7 SIZE    FCB      6                      ;number of bytes in each number
8 NUM1    FCB      $5B,$FC,$59,$DF,$49,$20 ;first number
9
10        ORG      $1011                      ;starting address of second number
11 NUM2    FCB      $22,$6B,$80,$A8,$52,$7C ;second number
12
13        ORG      $1100                      ;starting address of program
14        LDAB     SIZE                      ;get length of each number
15        LDX      #NUM1                    ;point X at first number
16        CLC
17        LDAA     0,X                      ;add two numbers starting with
18        ADCA     $10,X                    ;least significant byte
19        STAA     $20,X                    ;store the result
20        INX
21        DECB
22        BNE      LOOP                    ;check for end of operation
23        SWI
24
25        END

```

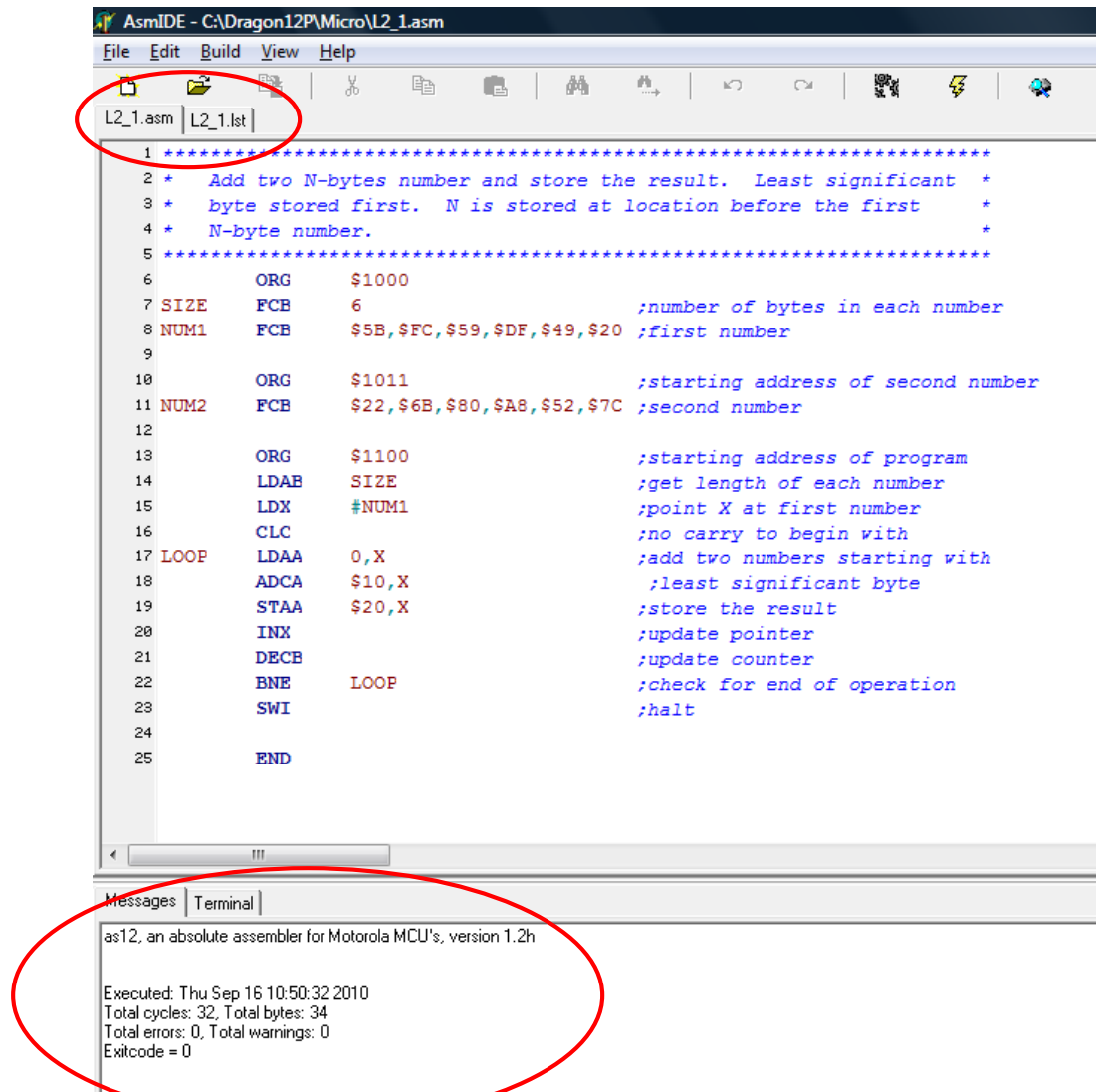
Invoking the Assembler

- To assemble an assembly program, press the **Build** menu and select **Assemble**, as shown below.



Invoking the Assembler *continued* ...

- After program is assembled successfully, the message window will show that total number of errors was zero.



The screenshot shows the AsmIDE interface with the file `L2_1.asm` open. The code is an assembly program for a Motorola MCU, version 1.2h. The program adds two 6-byte numbers stored in memory. The Messages window at the bottom displays the following execution results:

```
as12, an absolute assembler for Motorola MCU's, version 1.2h

Executed: Thu Sep 16 10:50:32 2010
Total cycles: 32, Total bytes: 34
Total errors: 0, Total warnings: 0
Exitcode = 0
```



```

1
2 as12, an absolute assembler for Motorola MCU's, version 1.2h
3
4 *****
5 *   Add two N-bytes number and store the result.  Least significant *
6 *   byte stored first.  N is stored at location before the first   *
7 *   N-byte number.                                                *
8 *****
9 1000                ORG            $1000
10 1000 06             SIZE          FCB            6                ;number of bytes in each number
11 1001 5b fc 59 df 49 20 NUM1       FCB            $5B,$FC,$59,$DF,$49,$20        ;first number
12
13 1011                ORG            $1011                ;starting address of second number
14 1011 22 6b 80 a8 52 7c NUM2      FCB            $22,$6B,$80,$A8,$52,$7C        ;second number
15
16 1100                ORG            $1100                ;starting address of program
17 1100 f6 10 00        LDAB          SIZE                ;get length of each number
18 1103 ce 10 01        LDX           #NUM1               ;point X at first number
19 1106 10 fe           CLC                                ;no carry to begin with
20 1108 a6 00           LOOP          LDAA              0,X          ;add two numbers starting with
21 110a a9 e0 10        ADCA          $10,X               ;least significant byte
22 110d 6a e0 20        STAA          $20,X               ;store the result
23 1110 08              INX                                ;update pointer
24 1111 53              DECB                               ;update counter
25 1112 26 f4           BNE           LOOP                 ;check for end of operation
26 1114 3f              SWI                                ;halt
27
28                      END
29
30 Executed: Thu Sep 16 10:50:32 2010
31 Total cycles: 32, Total bytes: 34
32 Total errors: 0, Total warnings: 0

```

Invoking the Assembler *continued* ...

- This will generate a machine code that is in **S-record** file format and has an extension of **.s19**

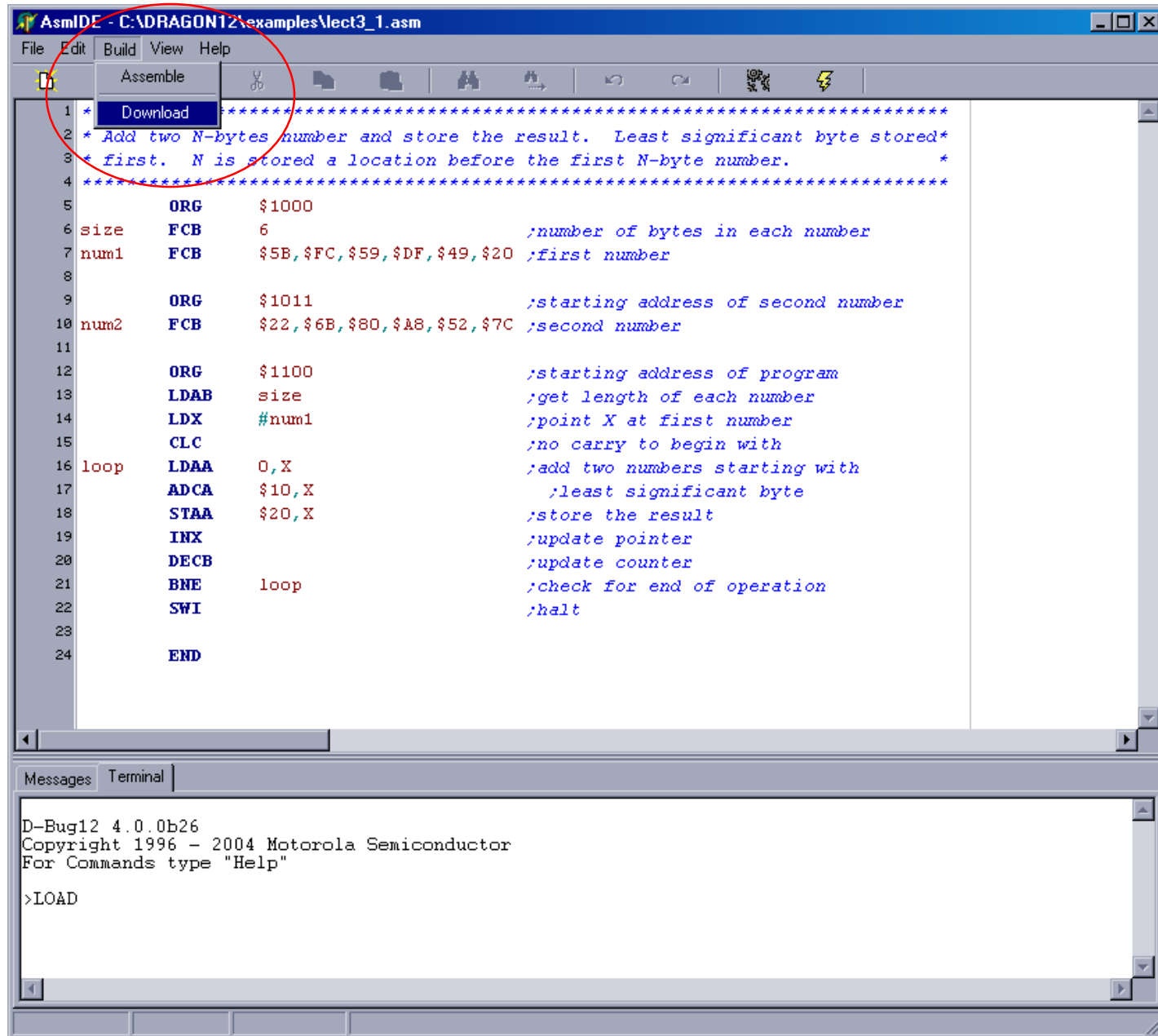
Downloading the S-Record File onto Demo Board for Execution:

- Assuming that you are connected to Demo Board, click on **Terminal** window and enter following command to download the S-record file onto the demo board:

>LOAD <Address Offset>

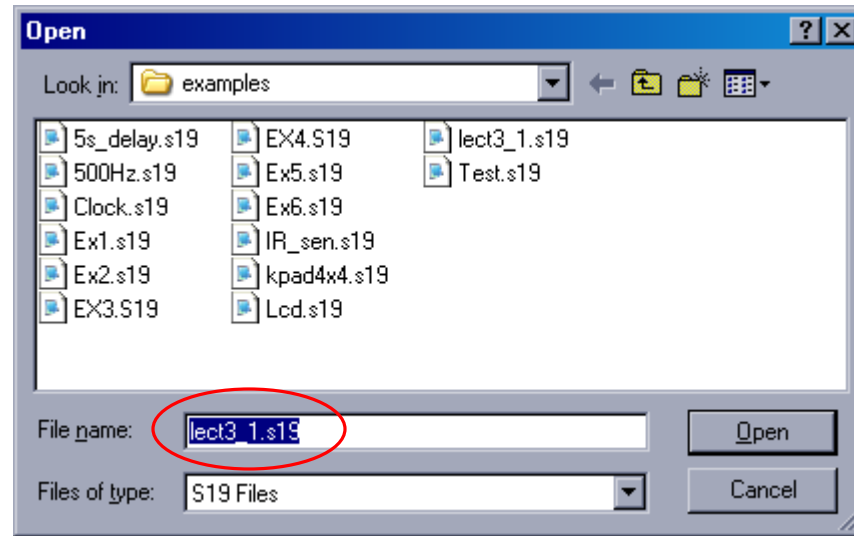
- This command is used to load S-record objects into memory from an external device.
- The address offset, if supplied, is added to the load address of each S-record before its data bytes are placed in memory.
- To start the downloading process, click on **Build** and select **Download**.

Downloading the S-Record File onto Demo Board for Execution *continued* ...



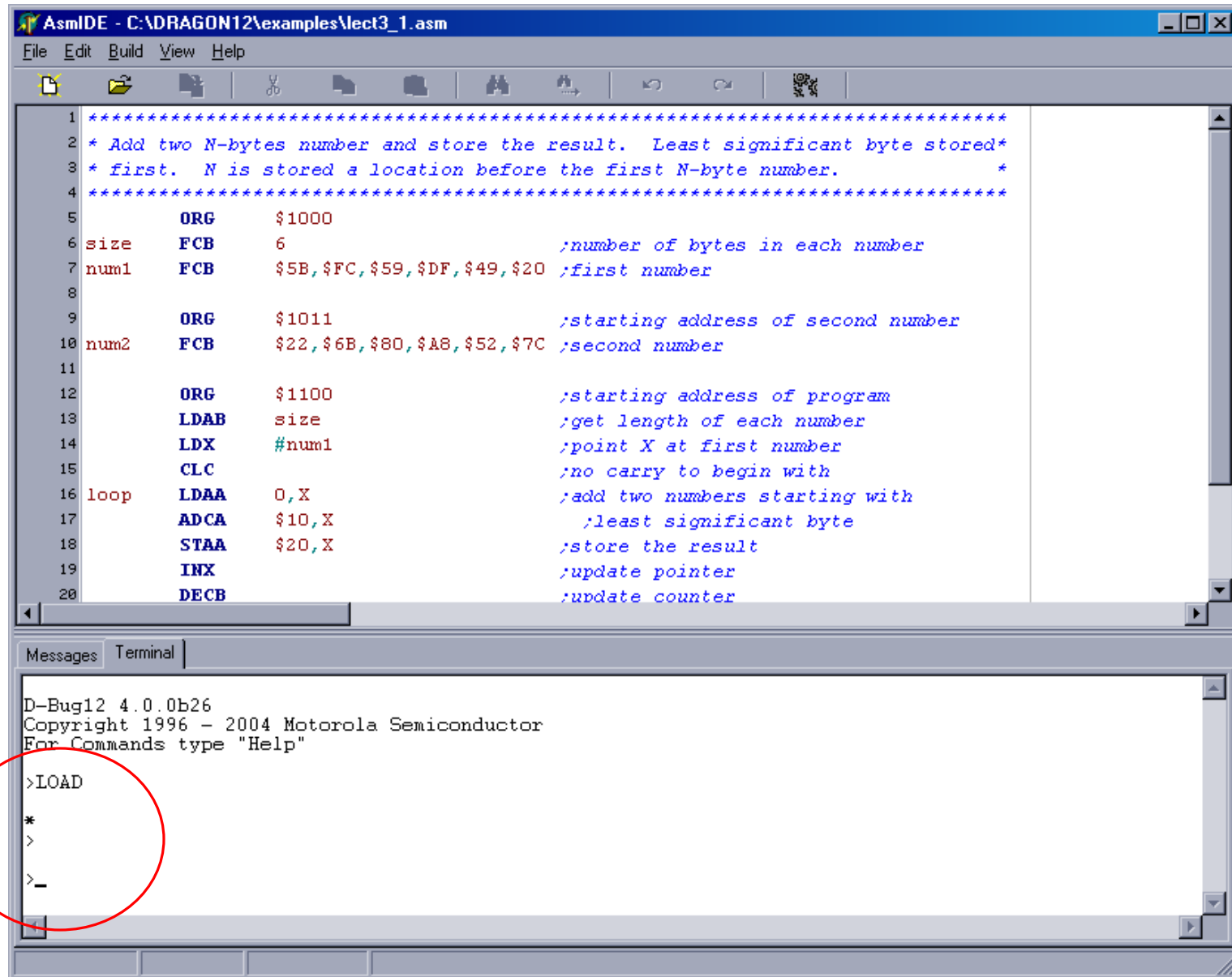
Downloading the S-Record File onto Demo Board for Execution *continued* ...

- After the Download command is selected, a popup dialog box, as shown below will appear.



- Specify the file to be downloaded and click **Open** button.
- The S-record data is not echoed to the control console. However, for each 10 S-records that are successfully loaded, an ASCII asterisk character (*) is sent to the control console.
- Once a S-record file has been successfully loaded, the D-bug12 prompt reappears on screen.

Downloading the S-Record File onto Demo Board for Execution *continued* ...



The screenshot shows the AsmIDE software interface. The main window displays assembly code for a program that adds two N-byte numbers. The code is as follows:

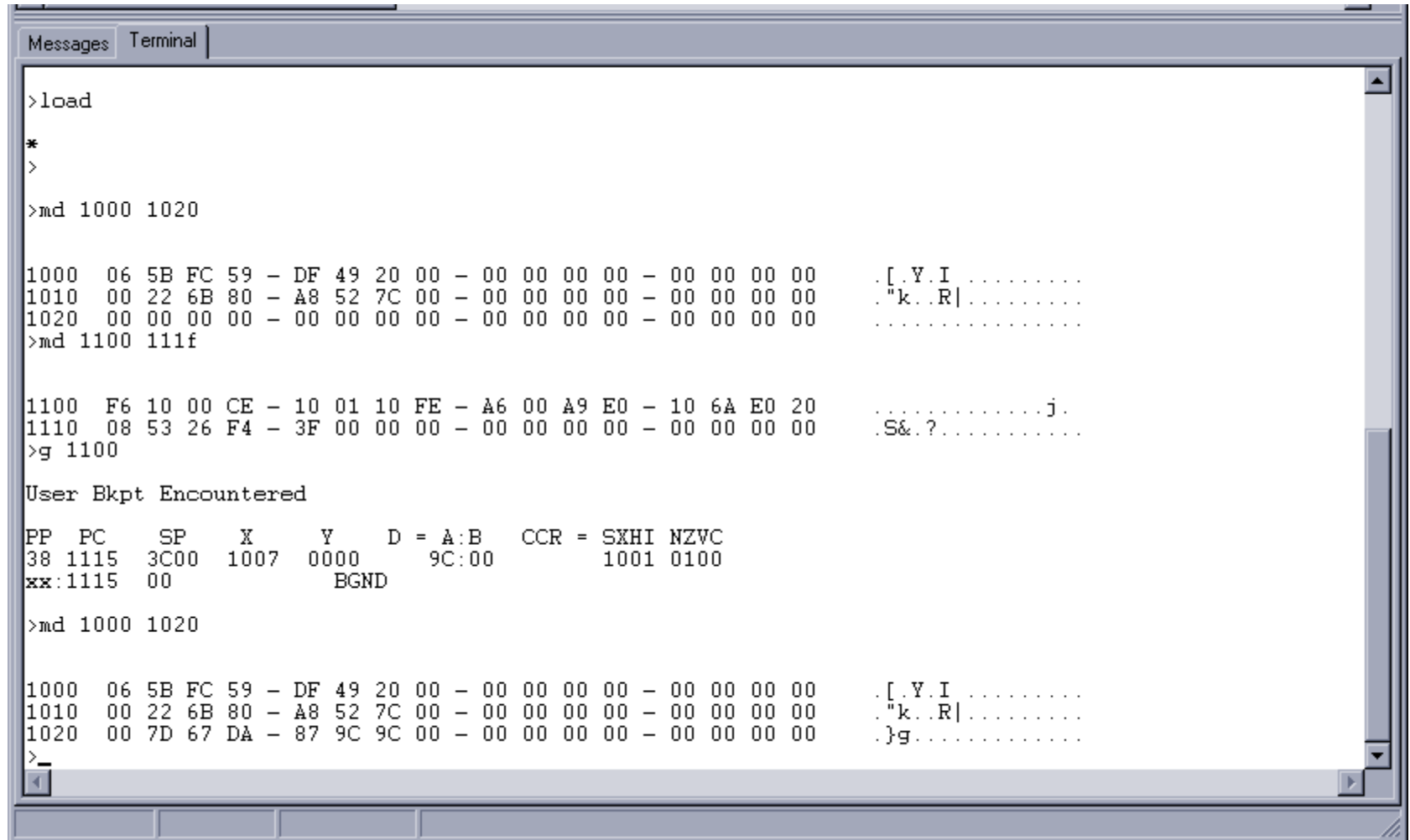
```
1 *****
2 * Add two N-bytes number and store the result. Least significant byte stored*
3 * first. N is stored a location before the first N-byte number. *
4 *****
5     ORG     $1000
6 size  FCB     6                ;number of bytes in each number
7 num1  FCB     $5B,$FC,$59,$DF,$49,$20 ;first number
8
9     ORG     $1011                ;starting address of second number
10 num2  FCB     $22,$6B,$80,$A8,$52,$7C ;second number
11
12     ORG     $1100                ;starting address of program
13 LDAB   size                ;get length of each number
14 LDX    #num1                ;point X at first number
15 CLC
16 loop  LDAA   0,X                ;add two numbers starting with
17       ADCA   $10,X                ;least significant byte
18       STAA   $20,X                ;store the result
19       INX
20       DECB
       ;update counter
```

The terminal window at the bottom shows the following text:

```
D-Bug12 4.0.0b26
Copyright 1996 - 2004 Motorola Semiconductor
For Commands type "Help"
>LOAD
*
>
>_
```

A red circle is drawn around the terminal window, highlighting the commands entered.

Running and Debugging the Program



```
Messages Terminal
>load
*
>
>md 1000 1020

1000 06 5B FC 59 - DF 49 20 00 - 00 00 00 00 - 00 00 00 00  .[.Y.I .....
1010 00 22 6B 80 - A8 52 7C 00 - 00 00 00 00 - 00 00 00 00  ."k..R| .....
1020 00 00 00 00 - 00 00 00 00 - 00 00 00 00 - 00 00 00 00  .....
>md 1100 111f

1100 F6 10 00 CE - 10 01 10 FE - A6 00 A9 E0 - 10 6A E0 20  .....j.
1110 08 53 26 F4 - 3F 00 00 00 - 00 00 00 00 - 00 00 00 00  .S&?.....
>g 1100

User Bkpt Encountered

PP PC SP X Y D = A:B CCR = SXHI NZVC
38 1115 3C00 1007 0000 9C:00 1001 0100
xx:1115 00 BGND

>md 1000 1020

1000 06 5B FC 59 - DF 49 20 00 - 00 00 00 00 - 00 00 00 00  .[.Y.I .....
1010 00 22 6B 80 - A8 52 7C 00 - 00 00 00 00 - 00 00 00 00  ."k..R| .....
1020 00 7D 67 DA - 87 9C 9C 00 - 00 00 00 00 - 00 00 00 00  .}g.....
>
```