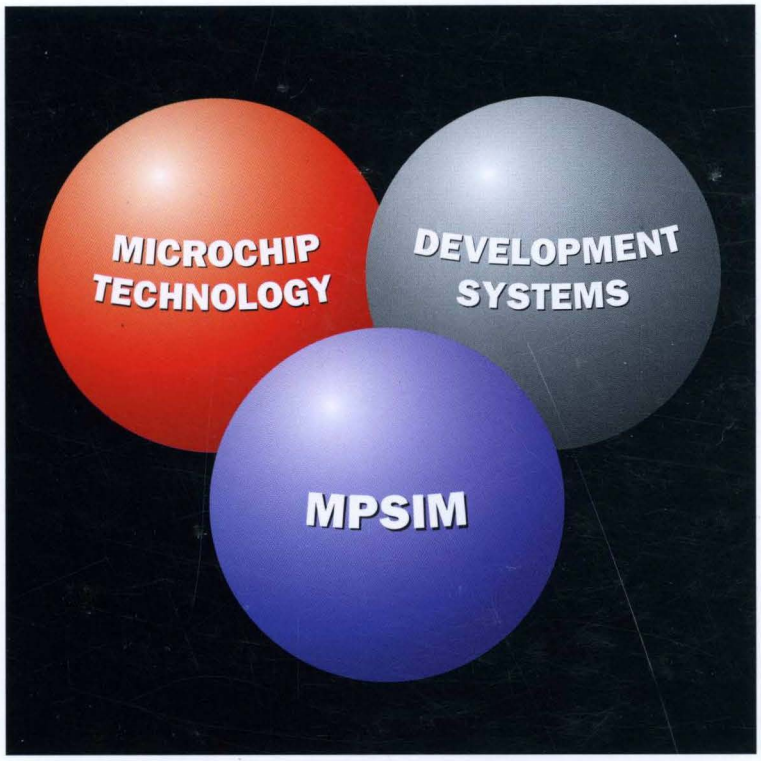
MICROCHIP MPSIM USER'S GUIDE

MPSIM
SIMULATOR

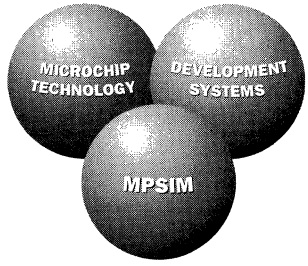


USER'S GUIDE

MICROCHIP

MPSIM
SIMULATOR

User's Guide



MPSIM USER'S GUIDE

Table of Contents

Preface

MPSIM is a discrete-event simulator tool designed to:

- Imitate operation of Microchip Technology's PIC16C5X, PIC16CXX and PIC17CXX families of microcontrollers
- Assist users in debugging software that uses Microchip microcontroller devices

A discrete-event simulator, as opposed to an in-circuit emulator, is designed to aid in the debugging of the general logic of your software. The MPSIM discrete-event simulator allows users to modify object code and immediately re-execute, inject external stimuli to the simulated processor, and trace the execution of the object code. A simulator differs from an in-circuit emulator in three important areas: I/O timing, execution speed, and cost.

This manual covers MPSIM version 5.0 and later.

I/O Timing

External timing in MPSIM is processed only once during each instruction cycle. Transient signals, such as a spike on $\overline{\text{MCLR}}$ smaller than an instruction cycle will not be simulated but may be caught by an in-circuit emulator. In MPSIM, external stimulus is injected just before the next instruction cycle execution.

Execution Speed

The execution speed of a discrete-event simulator is several orders of magnitude less than a hardware-oriented solution. Users may view slower execution speed as a handicap or a blessing. Some discrete-event simulators are unacceptably slow. MPSIM however, attempts to provide the fastest possible simulation cycle.

Cost

The cost of the debugging tool may be an issue with some developers. For this reason, Microchip Technology has developed this simulator to be a cost-effective tool for debugging application software. MPSIM does not require any external hardware to the PC, which keeps the cost at a minimum.

Debugging Tool

The simulator, however, is a great debugging tool. It is particularly suitable for optimizing algorithms. Unlike the emulator, the simulator makes many internal registers visible and can provide more complex break points.

If you are a new user, refer to Chapter 3 for a "Getting Started" tutorial.

Device specific information is provided in the appendices at the end of the manual.

Chapter 1. Introduction

Introduction

MPSIM is a discrete-event simulator designed to aid you in debugging your software applications for Microchip Technology's PIC16C5X, PIC16CXX, and PIC17CXX microcontrollers.

Highlights

Whether you are an experienced user or a beginner, we strongly suggest that you read this chapter first since it provides information about:

- Installing MPSIM
- Document Conventions
- Terminology
- Device-Specific Support
- Customer Support

If this is your first time using MPSIM we also suggest that you go through the tutorial provided in Chapter 3. This tutorial introduces all files that are used or generated by the simulator and provides a good introduction to some of the most widely-used commands.

Installing MPSIM

System Requirements

MPSIM requires an IBM® PC/AT® or compatible running DOS version 5.0 or later. The PC needs a 3 1/2 inch floppy disk drive and at least 640K main memory. We recommend a hard disk with at least 5 MB of available space.

- On the PC, create a new directory for the MPSIM software and change to that directory:

  MKDIR SIM<RETURN>

  CD SIM<RETURN>

- Copy all the files on the MPSIM diskette into the above directory:

  COPY a:*.*

After loading the software, MPSIM is ready to run.

Document Conventions

This section describes the conventions this manual uses for the data you are to enter.

TABLE 1.1- CHARACTER CONVENTIONS

Character	Represents
Square ([ ]) brackets	Optional arguments
Curly ({ }) brackets	Braces indicate group options. One or more options in the group is required.
Angle (< >) brackets	Delimiters for special keys: <TAB>, <ESC>, etc.
Pipe (|) characters	Choice of mutually exclusive arguments; an OR selection
Lower case characters	Type of data
Italic characters	A variable argument; it can be either a type of data (in lower case characters) or a specific example (in uppercase characters)
`Courier font`	User keyed data or output from the system

Terminology

Breakpoints

Source code locations where you want the code to cease execution.

Program Counter (PC)

The address in the loaded program at which execution will begin or resume.

Disassembler

Converts modified object code back into assembly-language code when a listing file wasn't loaded. Thus, mnemonic information can display even when you have made changes.

Step

A single executable instruction. You can single-step through a program by executing one instruction at a time with the SS command. A stimulus file can inject values onto specified pins at specified steps.

Symbols

Alphanumeric identifiers such as labels, constant names, bit location names and file register names. MPSIM understands both explicit data/addresses and symbols.

Trace

A trace file can be created to illustrate the execution flow of your program. Each line in the trace file contains the object code, source line, step number, elapsed time, and file registers that have changed. Trace can be limited to a range of addresses, or to a specific file register address. Please see Chapter 3 "Tutorial" for examples on the trace file. When you trace the instructions, they always display on the screen. If you previously opened a trace file and have not closed it, MPSIM also appends the trace to the file.

View screen

The portion of your monitor that dynamically displays the values in specified data areas. It is seven lines long. The V command creates a view screen; the AD command adds data areas to the display; the DV command deletes data area from the display; and the NV command deletes all data areas from the view screen.

Device-Specific Support

MPSIM provides support for more than one family of microcontrollers. Chapters 1 - 5 contain general information about MPSIM, regardless of the target processor. Device-specific information can be found in the appendices at the end of this manual.

Customer Support

If you have any questions about MPSIM, the first step is to check in **Appendix A: Troubleshooting Guide**, which contains a troubleshooting guide that provides some common error messages and their possible causes. **Appendix C: Customer Support** provides detailed information about how to connect to the Microchip Technology BBS. The BBS contains the most up-to-date development systems software, application notes, as well as a variety of other useful information. If you still cannot find the answer, contact the sales office nearest you. Information and telephone numbers are presented on the last page of the manual.

Chapter 2. The MPSIM Environment

Introduction

Chapter 2 provides an introduction to the MPSIM debugging environment. It describes all data areas that can be simulated and presents general information about using the simulator. This chapter is highly recommended for first-time users.

Highlights

The following topics will be covered:

- User Interface
- Invoking MPSIM
- I/O Pins
- CPU Model
- Hardware Stack
- Files Used and Generated By MPSIM

User Interface

The user interface consists of three areas: the title line, the view screen and a command entry/display region. The title line remains in a fixed location at the top of the screen and lists the current hex file, the radix, the MPSIM version, the controller being simulated, cycle steps and elapsed time.

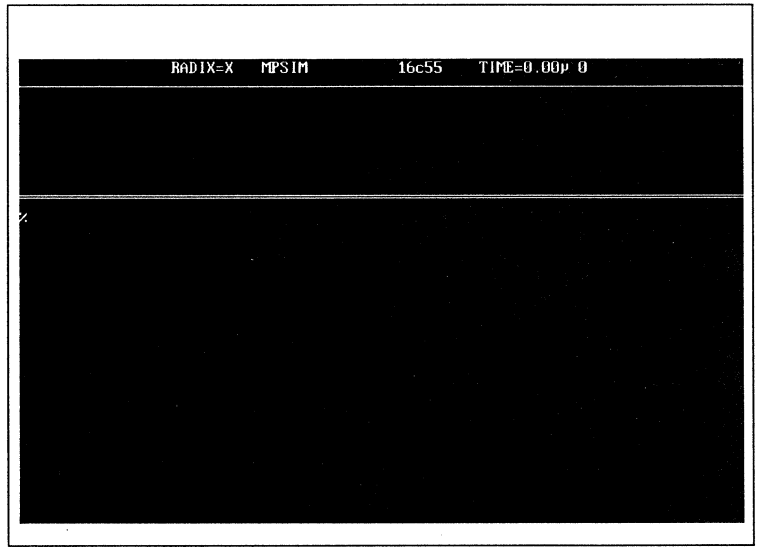


Figure 2.1Start-up

The view screen displays user selected pin and register values. This area is created by the user typically through an initialization command file. This file will be in greater detail later in this chapter in "Files Used and Generated by MPSIM".

The command entry/display region occupies the remainder of the screen. Use this area to enter commands; MPSIM enters any responses to a command on the line or lines immediately following the command.

MPSIM can be invoked with any or a combination of the following options:

Option	Description	Default
-v	verbose	off
-m	monochrome mode	off
-a	ASCII only	off

Invoking MPSIM

Invoke MPSIM by typing MPSIM at the DOS prompt, or by typing MPSIM_DP for the PIC17C42 in the extended microcontroller or microprocessor mode. MPSIM is faster than MPSIM_DP. MPSIM_DP supports the larger memory modules.

To load a file into the simulator, use the following command:

%LO *filename* [FORMAT] <RETURN>

The '%' is MPSIM's prompt. Exit MPSIM by using the AB or Q command. Obtain help with the H command.

I/O Pins

There is a list of viewable and modifiable pins for each microcontroller in its appendix. These pin names are loaded when a processor is selected and are the only ones that MPSIM recognizes as valid.

I/O Pin Modeling

Because a conflict can occur when a pin is being driven internally (via an instruction) and externally (via stimulus file), the following table is provided to illustrate the possible conditions and the order in which MPSIM processes it.

Is the pin being driven externally?	Is the pin being driven internally ?	Resolution
Yes	Yes	Chip wins.
No	No	The pins are essentially floating. The pins maintain the last external value they were driven.*
Yes	No	Simple.
No	Yes	Simple.

* Note that this does not represent the actual behavior of the circuit when the I/O pin was last driven by the chip. However, typically, a used I/O pin (especially CMOS) would not be left floating.

Pin Signals

At the end of each instruction all pins are checked for possible input or output.

- If the $\overline{\text{MCLR}}$ pin is cleared, MPSIM simulates a $\overline{\text{MCLR}}$ reset.
- The TRIS (or DDR for the PIC17CXX) status bits determine how MPSIM manipulates the port and file register bits. For example, the TRISA, RA0-RA5 and F5 registers work together; the TRISB, RB0-RB7 and F6 registers work together; and the TRISC, RC0-RC7 and F7 registers work together, etc.
  - For TRIS status register bits that are set, MPSIM reads the corresponding port bit into the corresponding file register bit.
  - For TRIS status register bits that are cleared, MPSIM writes the corresponding file register bit to the corresponding port bit (pin).
- Similarly, if any of the timer inputs are changed, the corresponding timer or its prescaler will increment.
- Any peripheral input (such as capture input) is acted upon.
- Any peripheral output (such as serial port output) is presented on the pin.

CPU Model

Reset Conditions

All reset conditions are supported by MPSIM.

A Power-On-Reset, for example, can be simulated by using the RS instruction. All special-purpose registers will be initialized to the values specified in the Microchip data sheet.

A $\overline{\text{MCLR}}$ reset during normal operation or during SLEEP, for example, can easily be simulated by driving the $\overline{\text{MCLR}}$ pin low (and then high) either via the stimulus file or by using the SE command.

A WDT time-out reset is simulated when WDT is enabled (see DW command) and proper prescaler is set (by initializing OPTION register appropriately for the PIC16CXX family or by using the FW command for the PIC17CXX family) and WDT actually overflows. WDT time-out period is approximately the "normal" time for the device being simulated (to closest instruction cycle multiple).

The Time-out ($\overline{\text{TO}}$) and Power-down ($\overline{\text{PD}}$) bits in the Status register reflect the reset condition. This feature is useful for simulating various power-up and time out forks in the user code.

Sleep

MPSIM simulates the SLEEP instruction, and will appear "asleep" until a wake-up from sleep condition occurs. For example, if the Watchdog timer has been enabled, it will wake the processor up from sleep when it times out (depending upon the prescaler setting).

Another example of a wake-up-from-sleep condition, would be Timer1 wake-up from sleep. In this case, when the processor is asleep, Timer1 would continue to increment until it overflows, and if the interrupt is enabled, will wake the processor on overflow and branch to the interrupt vector.

Wake-up from SLEEP through interrupt is fully simulated in the PIC16CXX and PIC17CXX products.

WDT

The Watchdog timer is fully simulated in the MPSIM simulator. Because it is fuse-selectable on the device, it must be enabled by a separate command (see the DW command) in MPSIM. The period of the WDT is determined by the prescaler settings. The basic period (with prescaler = 1) is approximated at 18 ms (for the PIC16C5X and PIC16CXX families and 12 ms for the PIC17CXX families).

Registers

MPSIM simulates all registers. Certain special-function registers or non-mapped registers can be added to the viewscreen or modified like any other register. Examples are timer prescaler or postscalers.

All registers are initialized appropriately at various reset conditions.

Please see the appendix of the microcontroller in question for a list of additional registers.

Register Name	Function
W	Working Register
TRISA	Tris register for Port A (PIC16C5X/PIC16CXX)
TRISB	Tris register for Port B (PIC16C5X/PIC16CXX)
TRISX (etc)*	(etc)*
OPT	Option register*

* Processor-dependent. For a complete list for a given processor, please refer to the device-specific appendix.

Hardware Stack

Push

The CALL instruction pushes the PC value + 1 to the top of the stack and loads the PC with the address of the subroutine being called. If the number of CALL instructions exceeds the depth of the stack, MPSIM will issue a "STACK OVERFLOW" warning message when executing or single-stepping through code. In the PIC16C5X family, the CALL instruction is the only instruction that causes an address to be pushed to the stack. The PIC16CXX and PIC17CXX families, however, support interrupts. When an interrupt occurs, the PC value + 1 is pushed to the stack and the PC is loaded with the address of the interrupt vector. The same error message will also be generated if too many addresses are pushed to the stack when MPSIM is executing or single-stepping through a program.

Pop

RETLW instructions in the PIC16C5X and RETLW, RETURN and RETFIE instructions in the PIC16CXX and PIC17CXX instruction set remove or "pop" the last address pushed to the stack and loads its value into the PC. If an attempt is made to pop more values than the stack contains, MPSIM will issue a "STACK UNDERFLOW" warning message when executing or single-stepping through the program.

Because stack implementation is processor-family dependent, please refer to the appendix of the processor family in question for stack simulation.

Files Used and Generated By MPSIM

MPSIM uses or creates the following I/O files.

- Command files
- Initialization files
- Journal files
- Stimulus files
- Assembler files
- HEX-Code formats

The following sections describe each of these files.

Command Files

Command files are text files containing MPSIM commands. These MPSIM commands are executed with the GE command.

There are two special command files: MPSIM.INI and MPSIM.JRN. MPSIM.INI is the initialization file that MPSIM will automatically load on start-up. MPSIM.JRN is a file containing all commands executed in the previous session.

Initialization File

When MPSIM is invoked, it automatically performs the MPSIM commands in MPSIM.INI. Common commands in this file might create a standard view screen and/or initialize data areas. Figure 3.2 in Chapter 3 lists an example initialization file and Figure 3.3 in Chapter 3 shows the resulting view screen.

Journal File

If you want to re-execute the most recent MPSIM session, LJ retrieves a list of the commands performed during the previous MPSIM session from MPSIM.JRN. This file is automatically created each time MPSIM is invoked. If you want to retain a journal file, copy it to another filename before reentering MPSIM. The first time you reenter MPSIM, the journal file is the same as you copied. However, when you exit via Q, the commands from the current MPSIM session will overwrite the previous journal file. Thereafter, you can access the copied file with GE.

As with all modern CAD/CAE tools, the concept of journal files is carried throughout MPSIM. That is, any command entered by the user is automatically stored in a journal file (named MPSIM.JRN). The journal file remains in the user's default directory regardless of the termination method (Quit or Abort). The LJ command loads and executes the journal file created during the previous simulator session. However, it doesn't store the commands from the previous journal file in the current journal file.

Performing the Q command removes the previous journal file, but using the AB (Abort) retains old journal file. The current MPSIM session commands are written over the previous journal file.

Stimulus File

This file allows you to schedule bit manipulation by forcing MPSIM to drive given pins to given values at a specified input step. This scheduling is via a text file called a stimulus file. The stimulus file can force any pin to any value at any input step during program execution. The ST command reads the stimulus file into MPSIM. When you execute the loaded file with E, each time it looks for input, it reads the next step in the stimulus file. The first line of stimulus file always consists of column headings. It lists first the word "STEP," followed by the pins that are to be manipulated. The data below STEP represents the object file's input request occurrence. The data below each pin

name is the input value. You may enter comments at the end of a line by preceding it with an exclamation mark (!). The following example illustrates the stimulus file format:

```
STEP    pin 1    pin 2     ! These are pin names
8       1        0         ! followed by values
16      0        1
24      1        0
```

Other notes on the format of stimulus file:

- The steps in the stimulus file must be decimal, regardless of the radix in which you run your simulation
- The number of spaces separating data tokens is irrelevant
- Backslash (\) is a continuation mark at the end of a line and indicates that the following line continues the statement from the current line

Step	RB2	RA3	RA2	RA1	RA0	! Column Headings
3	0	0	1	0	0	! Stimulus before cycle 3
4	1	0	1	0	1	! Injected before cycle 10
9	1	1	0	1	0	! Injected before cycle 16
10	0	1	0	1	1	! Stimulus before cycle 3
15	0	0	0	0	0	! Injected before cycle 9
16	1	0	0	0	1	! Injected before cycle 15

Figure 2.2- Stimulus File

There are three other ways to inject stimulus to the I/O pins in addition to using the stimulus file. A "clock" can be assigned to an I/O pin, Alt-function keys can be assigned to the pins (only for use in "execute" mode), and they can be modified in "single step" mode. Details and syntax for each command can be found in Chapter 5. Please see CK, DK, and SE commands.

Files Generated by the Assembler

The MPASM assembler generates by default all files necessary, for use with MPSIM. To assemble a file, invoke MPASM with the source file name as follows:

MPASM *filename*

The default assembler that MPSIM assumes is MPASM. To specify MPALC as the assembler, invoke MPSIM with the "-s" option.

Listing File

The listing file contains the source code the assembler uses to create the object code being simulated. To display the source code throughout simulation, read in the listing file with the LO command. Otherwise, MPSIM uses the disassembler.

Input Hex File

The input hex file contains the object code generated by the assembler. The LO command reads an hex file directly into program memory. The hex code format can be INHX8M or INHX8S. The default format is INHX8M.

Output Hex File

At any time during simulation, the contents of the program memory can written to an external file with the O command. The hex code format can be INHX8S or INHX8M.

Symbol File

The assembler generates the symbol file and contains a collection of symbols used in the source code. This file is used for symbolic debugging, and is automatically loaded when the LO command is used. The RA command clears the symbol file, and restores all original values.

Trace File

If you open a trace file with the TF command and later trace execution, MPSIM writes the specified trace into the trace file as well as displaying the trace on-line.

HEX Code Formats

The simulator is capable of reading or generating two different hex code formats as dictated by the LO and O commands: INHX8S or INHX8M. The default hex code format that the simulator recognizes is INHX8M, but any file format can be loaded by specifying the format when using the LO command. For example:

LO *Myfile* INHX8S

will tell the simulator to load myfile.obh and myfile.obl. (The two files necessary for INHX8S format.) Similarly, modified hex code can be saved to disk in any format by using the following command:

O *Myfile* INHX8M

The file that has been loaded into memory in any format will now be saved as a file in INHX8M format.

Chapter 3. Tutorial

Introduction

This chapter provides an introduction to MPSIM, the discrete-event simulator for Microchip Technology's PIC16C5X, PIC16CXX and PIC17CXX families of microcontrollers. It also presents a step-by-step tutorial through a sample program, SAMPLE.ASM. The tutorial is intended to familiarize you with the simulator and to provide an introduction to some of the most commonly used commands. The source code for SAMPLE.ASM and the other files used in the tutorial are available on your master disk, and can also be found in Appendix B at the end of the manual. If you do not have soft copies of the files for the tutorial, they can be created with any ASCII text editor. It is assumed that MPASM and MPSIM have been installed on your hard drive, and that all files used in the tutorial are in your working directory.

The program that is used in this tutorial, SAMPLE.ASM, is a software multiplier that takes two 8-bit numbers, "mulplr" and "mulcnd", and places the 16-bit result in "H_byte" and "L_byte" for the PIC16C54.

Because this chapter provides some background examples in addition to the tutorial, all steps that are part of the tutorial will have a step number in bold text to the left of the command in the margin.

Highlights

This chapter covers the following information:

- Assemble the Code
- Invoke the Simulator
- Load the Initialization File
- Load the Hex File
- Load the Stimulus File
- Set Up Trace Parameters
- Set Up Breakpoints
- Execute the Hex Code
- Modify the Hex Code
- Exit the MPSIM Session

Assemble the Code

Before you can begin to use the simulator, you must first assemble SAMPLE.ASM. MPASM generates a hex file in INHX8M format by default. In addition to INHX8M, the following formats can be output:

INHX8M

INHX8S

There is one default setting that the simulator assumes when it loads your code: the file format. The default file format for MPSIM is INHX8M, but any format that either assembler generates can be loaded into the simulator.

For this tutorial, we want the output file format to be INHX8M (the default format used by MPSIM), and the processor type to be PIC16C54. Type the following at the DOS prompt:

STEP 1:

MPASM *sample /p16C54* <RETURN>

Invoke the Simulator

STEP 2:

To invoke the simulator, simply type

MPSIM<RETURN> (if using the MPASM assembler)

or

MPSIM -s<RETURN> (if using the MPALC assembler)

at the DOS prompt.

The following screen will display:

```
                RADIX=X   MPSIM          16c55    TIME=0.00µ 0
W: 00  F1: 00  F2: 100  F3: 00  F4: 00  F5: 00  F6: 00  F7: 00




% SR X
% ZP
% ZR
% ZT
% RE
% V W,X,2
% AD F1,X,2
% AD F2,X,3
% AD F3,X,2
% AD F4,X,2
% AD F5,X,2
% AD F6,X,2
% AD F7,X,2
%
```

Figure 3.1- MPSIM.INI View Screen

MPSIM.INI

Observe the information in the command area and the information that is displayed in the view screen. The data areas appear in the view screen because an initialization file, MPSIM.INI is in your working directory. MPSIM.INI is simply an ASCII file that contains the same commands that appear in the command area. Every time MPSIM is invoked, it looks for a file called MPSIM.INI. If one exists on your working directory, all of the MPSIM commands appearing in that file will be executed, much like a DOS batch file. It is important to understand that an initialization file can be named anything. MPSIM.INI is unique in that it is automatically loaded when MPSIM is invoked.

Load the Initialization File

Initialization files are very useful because they allow you to choose data areas that you wish to view, display them on the viewscreen, load your program, and create break points–all in one step. In other words, you can invoke MPSIM, load your initialization file, begin debugging, exit MPSIM, and return later, easily setting up the viewscreen the same way that you had it when you quit the program, simply by loading the initialization file.

Creating an initialization file

One easy way to create an initialization file is to first invoke the simulator, type in commands that set up your viewscreen, set some break points, and then quit the simulator. When you quit, you will notice that a file "MPSIM.JRN" has been created. This "journal" file contains every command that you executed in the previous session. If the W register, or any other register was added to the viewscreen, the commands implementing this will be saved in the journal file. This file can then be edited using any text editor to remove commands such as "E" (execute) or "Q" (Quit), and then saved under another file name. It is necessary to remove commands such as "E" and "Q" because they will also be executed when you load your ANYTHING.INI file, and the simulator would set up your viewscreen, execute your code, and quit. It is also important to save the journal file under another name before invoking MPSIM a second time. Each time MPSIM is invoked, it overwrites the previous journal file, and if you did not rename the journal file, it will contain all commands executed in the current session.

STEP 3:

For this example, we will use the initialization file called "SAMPLE.INI". We will load it by using the following command:

GE *sample.ini* <RETURN>

MPSIM executes the commands in the following SAMPLE.INI file.

```
LO SAMPLE
ST SAMPLE
SR X
ZP
ZR
ZT
RE
P 54
NV
AD mulcnd
AD mulplr
AD H_byte
AD L_byte
AD count
AD portb
AD RB7,B,1
AD RB6,B,1
AD RB5,B,1
AD RB4,B,1
AD RB3,B,1
AD RB2,B,1
AD RB1,B,1
AD RB0,B,1
```

Figure 3.2–Sample .INI Initialization File

This changes the viewscreen so that it displays the data areas that SAMPLE.HEX uses, in the most useful format.

Figure 3.3– Sample.INI View Screen

The commands in this file create the viewscreen shown above and re-initialize data areas. The viewscreen now contains data areas that can be watched during the execution of SAMPLE.

Load the Hex File

Notice that the LO command is listed in the SAMPLE.INI file. Because of this, the hex file was automatically loaded when SAMPLE.INI was loaded. If the LO command were not in the SAMPLE.INI file, you could load the file by typing in the following:

LO *sample* <RETURN>

It is important to realize that because we have assembled the code in the MPSIM default format (INHX8M), we do not have to specify the format being loaded. If we had assembled filename in any format other than INHX8M, we would have had to load the file in the following way:

LO *filename format* <RETURN>

MPSIM loads the named hex file, and then looks for a source file. If the file is available, it also loads the symbol table and the listing file.

Load the Stimulus File

SAMPLE.INI has taken care of loading the stimulus file. You can see in the SAMPLE.INI file that the command:

ST *sample.sti* <RETURN>

was executed when the initialization file was loaded.

The stimulus file contains values that are to be input to the pins. When you execute the loaded program, at every instruction step specified in the stimulus file, MPSIM retrieves the input data, and injects their values to the pins.

! Stimulus file for SAMPLE.ASM

STEP	RB7	RB6	RB5	RB4	RB3	RB2	RB1	RB0	!PortB Pins
3	0	0	0	0	1	0	0	1	! 9 x 5
5	0	0	0	0	0	1	0	1	
65	0	0	0	0	1	0	1	0	! 10 x 5
67	0	0	0	0	0	1	0	1	
127	0	0	0	1	1	0	1	1	! 27 x 3
129	0	0	0	0	0	0	1	1	
191	0	0	0	1	0	0	0	1	! 17 x 7
193	0	0	0	0	0	1	1	1	
253	0	1	0	0	0	0	0	0	! 64 x 63
255	0	0	1	1	1	1	1	1	

Figure 3.4 - SAMPLE.STI Stimulus File

The stimulus file for SAMPLE in figure 3.4 writes the multiplier and multiplicand values into simulated I/O port B. Since this port allows up to eight bits of data, the maximum value of the multiplier and multiplicand is 11111111 or 0xFF.

Set Up Trace Parameters

A trace file is a file that contains executed instructions, timing information, and registers that have been modified. Using a trace file can be very helpful in determining where to inject stimulus and for creating a "hard copy" of the general execution flow of your program. There are five MPSIM commands dealing with traces:

- **TF** opens and closes a file for writing the traced data.
- **TA** traces all instructions between two specified addresses
- **TC** traces a specified number of instructions.
- **TR** traces instructions dealing with specified registers and values.
- **DX** displays the current trace parameters

Try some of the following exercises. All of the traces in these exercises will be printed to a file. If you would like to try printing your trace to a default printer, substitute "PRN" in place of the trace file name.

Exercise 1: **Trace the instructions between two labels, call_m and main, and print the instructions to a file.**

The first step is to create the trace file:

TF *trace1.trc* <RETURN>

Next, specify the range of the trace. Then, begin tracing the instructions. Hit any key to interrupt the trace.

TA *main, call_m* <RETURN>

TC <RETURN>

Exercise 2: **Trace fourteen instructions (0x0E instructions) and write the trace to the file TRACE2.trc.**

Restart the system by exiting MPSIM (q <RETURN>), and repeating steps 2 (Invoke the Simulator) and 3 (Load the Initialization File). Just as in Exercise 1, we will first open the trace file

TF *trace2.trc* <RETURN>

Then, we will trace the next fourteen instructions. Note that if the number of instructions to be traced is not specified, the trace will continue until a key is pressed.

TC *E* <RETURN>

Note: If you had specified the number of instructions to be executed as "14" instead of "E", twenty steps would have been executed since the radix is set to hexadecimal (the default radix in MPSIM).

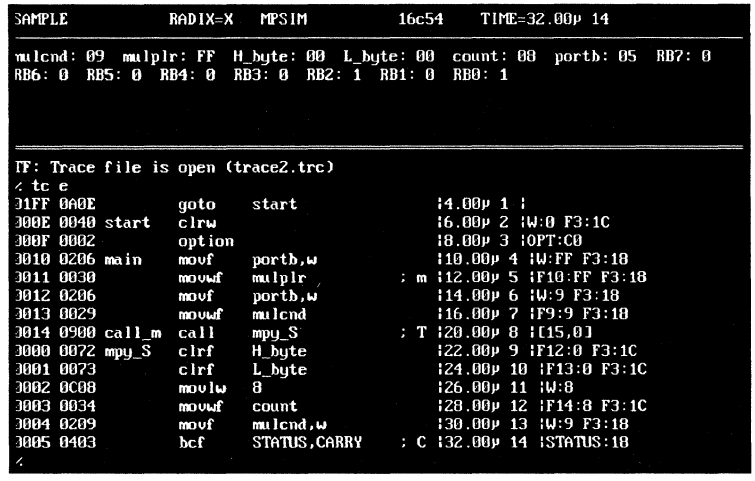


Figure 3.5 –The trace information is printed to both the screen and the trace file.

Exercise 3: **Check the current trace criteria.**

DX <RETURN>

The current trace parameters display in the command entry area of the MPSIM screen.

Set Up Breakpoints

Break points are used to artificially stop program execution so that you can review how the data has been manipulated or to see the contents of the Special Function Registers. There are three instructions that deal with breakpoints:

- **DB** displays all of the breakpoints currently set.
- **BC** clears one or all of the breakpoints currently set.
- **B** sets a break point.

Exercise 1: **Initialize the breakpoints by clearing any break points currently set. Enter the following command:**

BC

Exercise 2: **Set a breakpoint at MPY_S. Enter the following command:**

B *mpy_S*<RETURN>

Exercise 3: **Review all the breakpoints. Enter the following command:**

DB<RETURN>

Exercise 4: **Delete the breakpoint at MPY_S. Enter the following command:**

BC *mpy_S*<RETURN>

Execute the Hex Code

In addition to trace, there are three instructions that you can use to execute your code.

E

SS

C

- **E** executes your code until it encounters a breakpoint or you press a key.
- **SS** single-steps through your instructions. That is, it executes one single instruction at the CPC.
- **C** Execute, ignoring "n" number of breakpoints.

Exercise 1: **Add a watch variable. Add the w register to the display.**

AD W <RETURN>

Exercise 2: **Add two breakpoints and execute until the first breakpoint is encountered.**

b main

b mpy_S

E <RETURN>

MPSIM executes until it encounters the first breakpoint or until a key is pressed. Watch the values change in the W, mulplr, H-Byte, and L-Byte registers.

Exercise 3: **Execute instructions one step at a time.**

SS <RETURN>

The SS instruction causes MPSIM to execute the instruction at the PC. Pressing <RETURN> at the MPSIM prompt re-executes the last command. Execute a second instruction by pressing <RETURN> again. Do this several times, watching how the values in the W, mulplr, H-Byte, registers change. This command can be used to single-step through your entire program to see the data values at each step, and to watch the flow of your program. If you supply an address with the SS command, MPSIM will modify the CPC to the address you specify and then will execute the instruction at that address. Remember that pressing <RETURN> will cause MPSIM to re-execute the same command, so that if you supplied an address with the command, the same address will be executed.

Exercise 4: **Execute your program and break after the second breakpoint.**

C *2* <RETURN>

MPSIM executes the instruction at the current CPC until the instruction immediately following the second break point. Watch the values change in the W, mulplr, H-Byte, and L-Byte registers.

Modify the Hex Code

MPSIM has four types of commands which allow you to modify the hex code: search commands locate code that match specified criteria, display/modify commands automatically display specified code and allow you to change it, delete commands eliminate specified code, output commands allow the modified code to be saved to a file. For the following exercises, mulplr is stored in file register F10.

Exercise 1: **Search for the next occurrence of F10, and change its contents to 0xFF.**

SF *0, 1FF, F10* <RETURN>

You will see two code lines with the "mulplr" register label.

F *F10* <RETURN>

After you type in the above command, you will see the current contents of register F10, followed by a colon. Type in the value 0xFF, and watch the contents of the file register change. You will see that the contents of "mulplr" will change since the value of "mulplr" is 0x10.

Exercise 2: **Change the value of the W register to 0x0C**

W <RETURN>

Just as in Exercise 1, you will see the current contents of W displayed on the screen, followed by a colon. Type in 0x0C, and watch the contents of the W register change.

Exercise 3: **Change the contents of program memory located at the PC to a NOP.**

Type in the following:

M 0 <RETURN>

You will see the contents of program memory displayed in hexadecimal, followed by a colon. Type in a 0 (object code for NOP), and then <RETURN>. Unlike modifying file registers, you will not immediately exit the function. Instead, you will see the contents of the next memory location followed by a colon. You can continue modifying program memory until you are finished. When you are done, type "Q". This will get you back to the MPSIM command prompt (%).

Exercise 4: **Delete program memory between address 2 and 4.**

Type in the following command:

DE *2,4* <RETURN>

This function will delete all program memory from address 2 through address 4, and will shift up remaining program memory. If you would like to only clear the program memory between two addresses, use the following command:

ZM 2,4 <RETURN>

All of program memory between addresses 2 and 4 will now contain zeros (NOP instructions). It will essentially leave a "hole" in program memory. Use the following command to view your changes:

DI 0 <RETURN>

Exercise 5: **Remove the modifications made to program memory from the object code in memory.**

Remove the modifications made to program memory from the object code in memory.

ZP <RETURN>

This instruction clears the patch table. All of the modifications made to SAMPLE.HEX program memory are removed.

Exit the MPSIM Session

There are two ways of exiting MPSIM:

AB <RETURN>

Q <RETURN>

Using the AB command causes the old journal file to remain the same. The Q command overwrites the old journal file.

You have now been introduced to some of the most commonly-used functions in the simulator, and should have an understanding of how to use them. If you need any additional information about any of the files that the simulator uses or generates, please review the information in Chapter 2. Chapter 5 provides a list of all the commands that are available in MPSIM, complete with a detailed description of their functions and syntax.

Chapter 4. Functional Categories of MPSIM Commands

Introduction

Chapter 4 is intended to be used as a quick way to help locate a MPSIM command by function. All of the commands presented in this chapter have been grouped together according to function instead of alphabetical order. Once the desired command is found, it can be looked up in Chapter Five "MPSIM Simulator Commands" if a more detailed explanation or example is required.

Highlights

All commands have been divided into the following categories:

- Loading and Saving
- Inspecting and Modifying
  - Program Memory
  - Registers
  - Display Functions
  - Patch Table
  - Clearing Memory and Registers
  - Searching Memory
  - Symbol Table
  - Restore
- Execute and Trace
  - Execution Instructions
  - Tracing Execution
  - Breakpoints
- View Screen
- Miscellaneous Commands
- MPSIM Commands

Loading and Saving

The following three commands load and save hex code and listing files.

LO *filename format*	**Load file** *filename* **with** *format* into program memory. MPSIM also loads the source file.
LS *filename*	Load ***filename*** into internal symbol table.
O *filename format*	Write modified hex code to ***filename***.

Before simulation can begin, use LO to load an hex file into program memory. Immediately after loading the object file, MPSIM tries to load the listing file using the same filename and the extension .LST. If MPSIM still can't find the listing file, the source code file cannot be loaded and displayed at break points. Instead, MPSIM disassembles the hex code and displays the disassembled instruction.

The object file can be any of two different formats: INHX85 or INHX8M.

Example: **LO *SAMPLE.OBJ INHX8M*<RETURN>**

After modifications have been made to the program memory, the user may wish to save the corrected hex code into an external file. Use the O command to output the hex code. Enter the filename including the extension.

Example: **O *SAMPLE1.HEX INHX8S*<RETURN>**

Inspecting And Modifying

MPSIM allows user to change the values of any data area or program memory any time during the simulation.

Program Memory

In the course of testing a program, you may need to modify its instructions. Both the following commands do so.

A *address*	Display/modify program memory at address using symbolic format.
M *address*	Display/modify program memory at address using the current radix format.

If you use **IA**, the source code for the address displays, followed by ':' on the next line for the new command. The new command must consist of a valid mnemonic followed by zero or more operands. Each operand must contain a single value or symbol, no expressions will be allowed. MPSIM interprets all values based on the current input radix as set with the **SR** command.

Entering '**Q**' at the prompt ends the command; entering '-' causes MPSIM to go back and inspect/modify the previous address; entering **<RETURN>** leaves the instruction alone and continues to the next address.

Example: **%IA *200* <RETURN>**

```
0200  :            CLRF   F5
      : CLRF 6
0201  :            CLRF   F7
      : -
0200  :            CLRF   6
      : Q
```

After changing the hex code, the original source code no longer displays. It is replaced by a disassembled source line.

If you use **M**, the contents of the address display in the same format as the current radix. The prompt ':' immediately follows the data. Place the new value after the prompt, using the current radix.

The **'-'**, **'Q'** and **<RETURN>** have the same affect as described above. Two additional commands that affect program memory are:

IN *address,instruction*	Insert ***instruction*** at ***address*** in symbolic format.
DE *address1,address2*	Delete program memory from ***address1*** to ***address2***.

The **IN** command places a symbolically formatted opcode at the given address, then displaces values that follow ***address*** by one location. The new command must consist of a valid mnemonic followed by zero or more operands. Each operand must contain a single value or symbol, no expressions will be allowed.

The **DE** command deletes the code within the given boundaries then shifts all data in program memory locations greater than the upper boundary down to the lower boundary.

Registers

Each register can be inspected/modified by using the following commands:

F *register*	Display/modify contents of file ***register***
W	Display/modify contents of **W** register
SC	Display/modify processor cycle time
SE *data_area*	Display/modify any ***data_area***
RE	Reset elapsed time and step count

Inspect and modify file registers with the **F** command. The value of the register displays followed by the prompt **':'**. Enter the new value after this prompt.

Example: **%F 3**

F3=20:***21*** (The value of F3 has now been changed to 21.)

To inspect and modify the W register the W command is used.

Example: **%W**

W=44:***00*** (The value of **W** has now been changed to 0.)

Inspect and modify the simulated cycle time with the SC command.

Example: **%SC**

2.0:.***2***

Display and/or modify the value of any other data area (stack, pins, status bits, all registers) with the **SE** command.

Example: **%SE *OPT***

OPT=FF:***FE***

Chapter 4. Functional Categories of MPSIM Commands

Display Functions

The display functions are provided to print formatted lists of various program variables in the command/source area on the screen.

DR	Displays the contents of all registers including W, status and the stack.
DM *addr1,addr2*	Displays the code from *address1* to *address2*. The code displays only in the current radix, not in mnemonics. *address1* must be less than *address2* and both must be in the valid range of program memory.
DI *addr1,addr2*	Displays the code from *address1* to *address2*. The code displays in both the current radix and mnemonics. *address1* must be less then *address2* and both must be in the valid range of program memory.

You can terminate the **DM** and **DI** commands at any time by pressing any key.

Example: **%DI *0, 3***

```
0000 0020  MOVWF 0    The MOVWF instruction = 20
0001 0063  CLRF 3     The CLRF instruction = 63
0002 0080  SUBWF 0,0  The SUBWF instruction = 80
0003 0069  CLRF 9     The CLRF instruction = 69
```

%DM *0, 3*

```
0000 0020
0001 0063
0002 0080
0003 0069
```

Patch Table

During the course of simulation, several changes may have been made to the hex code in order to achieve the desired results. The patch table keeps track of all changes made by maintaining the original value of the address along with the most recent change. The patch table can then be displayed out in symbolic format to aid the user in making changes to the source code. The following three commands manipulate the patch-table.

ZP	Clears the patch table and resets it to no patches made. All changes previously made to the hex code remain.
DP	Display all patches in symbolic format. Both the original hex code and new code display.
RP	Restores all patches to their original value and clears the patch table

Clearing Memory and Registers

Memory and registers can be cleared quickly by using the following commands.

ZM addr1,addr2	Zero the program memory from address1 to address2. address1 must less than address2 and both must be valid program memory addresses.
ZR	Zero all of the file registers (F0 through F31).
ZT	Zero the elapsed time counter.

Clear any of the other data areas with the SE command.

Searching Memory

It is sometimes desirable to search the program memory for specific instructions or operands. The following three commands search program memory for various patterns and display each line containing that pattern.

SI *address1,address2,instruction*

Search program memory from ***address1*** to ***address2*** for any occurrence of instruction. Instruction is in mnemonic format.

SM *address1,address2*,m

Search program memory from ***address1*** to ***address2*** for any occurrence of the value m. Specify the search criteria in the radix mode, not in mnemonics.

SF *address1,address2,register*

Search program memory from ***address1*** to ***address2*** for any instruction that accesses file ***register***. Specify the search criteria in the radix mode, not in mnemonics.

Example:

```
%SI 0, 20, NOP
0000 0000  LOOP  NOP
0006 0000        NOP
001E 0000        NOP
%SM 0, 20, 0
0000 0000
0006 0000
001E 0000
```

Symbol Table

The following commands manipulate the symbol table:

DS	Display symbol table.
DL *symbol*	Delete ***symbol*** from symbol table.
GS *symbol,value,type*	Generate ***symbol*** with a ***value*** of ***type***. ***type*** may be **file, bit(file), label** or **literal**. See the **GS** command description for the exact syntax.

Example:

```
%DS
Symbol          Value Type
START           0000    L

%GS NEWSYM,
FF, B
Symbol          Value Type
START           0000    L
NEWSYM          00FF    B
```

Restore

The Restore All command, **RA**, has the combined effect of restoring the patch table, clearing the symbol table and removing all break points.

Execute and Trace

The simulator executes in three basic modes, execute until break, single step or trace. In either of these modes you can stop execution at any time by pressing any key.

Execution Instructions

The **E** command begins execution at the specified address, or at the CPC if you don't specify an address. The loaded program executes until reaching a break point or until you press any key. If you wish to slow down execution, use the single step instruction, **SS**. **SS** executes the single instruction at the specified address or at the CPC if you don't specify an address.

Tracing Execution

In the trace mode, all addresses meeting certain conditions display as they execute. The conditions may include:

- A given instruction within address boundaries.
- Accessing a given register.
- A given register containing a value between two limits. The following trace parameters maintain trace execution.
- Register number being traced.
- Range of register values.
- Range of addresses to trace.

The following commands set up and execute the trace mode.

TC *#instructions*	Trace the next ***#instructions***. If you omit ***#instructions***, execution continues until MPSIM encounters a break point or until you press any key.
TA	Sets the upper and lower address trace limits to the full range of program memory.
TA *addr1,addr2*	Sets the lower validation limit for address trace to ***address1*** and the upper address validation limit to ***address2.***
TR	Sets the address trace to trace any file register.
TR *reg*	Sets the address trace to trace the file register.
TR *reg,min_val,max_val*	Sets the address trace to trace the file register only if the value of the register is between *min_value and max_value.*

DX Displays the current trace parameters. When in trace mode, the location, opcode, mnemonic, elapsed time, cycle steps and any changed data areas will be displayed when the given conditions are met.

Note: F2 and F3 won't display if changed, however, status bits do display.

Examples:

```
%DX
Address         0000:01FF
%TC 2
0002 0000 LOOP  NOP     |6.00u 0003 |
0003 0040 TEST  CLRW    | 8.00u 0004 | Z:1
%TR 4, 0, F
%TR 3
%TA 0, 4
%DX
Address    0000:0004
F3         0000:01FF
F4         0000:000F
%TC 40
0004 0020 CALL START    | 10.00u 0005| [005,000]
```

Stack contents always display in brackets with the top of the stack to the left.

Breakpoints

MPSIM allows the user to set up to 512 breakpoints on any valid address. It also allows conditional breakpoints on any of the data areas. When one of these breakpoints is encountered, the current address is displayed in symbolic format and control is returned to the user. The following commands control the breakpoints.

B *address*	Set breakpoint at ***address*** (symbolic address can be used).
B *data_area op val*	Break when ***data_area*** matches the condition given by the ***operator*** (=,>,<,>=,<=,!=) and ***value.***
BC *address*	Cancel breakpoint at ***address***.
BC data_area	Cancel breakpoint involving ***data_area.***
BC	Cancel all breakpoints.
C *#breakpoints*	Continue execution ignoring ***#breakpoints*** breakpoint occurrences.
DB	Display all active breakpoints.

Only one conditional breakpoint is allowed per data area.

Chapter 4. Functional Categories of MPSIM Commands

View Screen

The following commands set up and manipulate the view screen.

V data_area,radix,#digits	This command sets up the view screen. This means that the View command defines the variables (and respective formats) to constantly display on the screen. Once the view screen is set, it remains active until either a NOVIEW command or a View sets up a new view screen. The format of this command is relatively simple. Register or signal s displays in radix mode r with n digits. r defaults to hexadecimal and n defaults to 1. If n is omitted, the number of digits is 1. The radix can be binary, octal, hexadecimal or decimal.
NV	This command clears the view screen. The same effect can be achieved by redefining the view screen.
AD data_area,radix,#digits	This command adds items to the view screen. If one desires to add more display items to the view screen, use the Add command. While this command's format is identical to View, it doesn't destroy the current contents of the view screen, but simply displays additional items as well as the current ones.
DV data_area	This command simply removes display items from the view screen while leaving the display formatting intact.
TY data_area,radix,#digits	This command changes the formatting of the existing view screen. s is the signal name (if the designated signal isn't in the view screen, MPSIM gives a warning). The radix can be x, o, d or b and n is the number of spaces to reserve for this variable at the display time.

Miscellaneous Commands

SR radix	This command sets the input/output radix to Octal, hexadecimal or Decimal. The radix will be used on all inputs and outputs with the exception of file register numbers and step counts.
P {54|55|71|...}	Choose the appropriate PIC16CXX Microcontroller number n. n can be any member of the PIC16CXX or PIC16C5X microcontroller family. The default is 55.
GE filename	This command forces MPSIM to get its command stream from an external text file. When end of file is reached, the control is returned to the user interface. All the incoming commands are parsed by the same mechanism as the one supervising the on-line interface thus the syntax should follow the guidelines of this document. If the specified file is not found, the user will be notified.
Q	This command terminates the dialogue. It prints out one or two summary messages, removes the journal file and exits to the operating system.
AB	This command aborts the dialogue. It prints out one or two summary messages and exits to the operating system.
ST filename	Stimulus command allows the user to introduce an event-based stimulus injection into the model. That is, the user may want to inject certain values into certain pins or registers at some point during the simulation. The stimuli are defined in a text file whose format is described on pages 12 and 13.
H	The Help command lists the syntax and a brief summary of each command available in MPSIM. There are several screens of information. Press **SPACEBAR** to exit, any other key to display the next screen.
CK pin, high, low	This command allows you to assign a clock to an I/O pin.
DK altfxkey, pin, event	This command simulates an asynchronous event through a function keystroke, and is very useful for simulating external interrupts or resets.
FI FileNameAddr, PmemAddr, n	This command injects values into a file register when the PC = PmemAddr. Repeats n times.

Chapter 4. Functional Categories of MPSIM Commands

MPSIM Commands

The following table contains summary information grouped by function about the MPSIM commands. This information is also found on the *MPSIM Quick Reference Card.* Chapter 5 contains additional information about the MPSIM commands.

Table 4.1 MPSIM Commands by Function

MPSIM Conventions

[ ]	Brackets indicate optional items.
{ }	Braces indicate group options. One or more options in the group is required.
|	Vertical bar indicates alternative options.

System Navigation

AB	Abort Session. Aborts the dialogue, prints a summary message and exits to the operating system. AB does not overwrite the journal file.
H|Help|?	Help. The Help Command lists the syntax and gives a brief summary of each command available in MPSIM. Press the SPACEBAR to exit Help. Press any other key to display the next help screen.
Q	Quit. Terminates the dialog. Prints one or two summary messages, overwrites the old journal file and exits to the opening system.

Program Memory

DE *address1,address2*	Delete Program Memory from *address1* to *address2.*
DI *[address1[,address2]]*	Display Program Memory in Symbolic Format. Displays program memory from *address1* to *address2.* Displays in current radix and symbolic format. Omit *address2* to display next 10 lines from *address1.*
DM *[address1,address2]*	Display Program Memory in Radix Designated Format. (See SR command to set the radix.) Displays program memory from *address1* to *address2.* Data displays in current radix only. With no parameter, displays 10 lines continuing from last DM display.
FM *address1,address2,pattern*	Fill Memory. Fills program memory from *address1* to *address2* with specified HEX pattern.
IA *address*	Insert/Inspect Assembly Code (Symbolic Format). Displays or modifies program memory at address using symbolic format.

Table 4.1 MPSIM Commands by Function (Continued)

IN *address,instruction*	Insert instruction. Inserts instruction at address in symbolic format.
LO *filename[format]*	Load Object File. Loads the object file, *filename*, with *format* into program memory. MPSIM also loads the listing file and symbol file. Valid Formats: INHX8M (Default) INHX8S
M address	Display/Modify Program Memory at Address. The contents at the *address* display, and a colon (:) prompt follows. To change the value at *address*, enter a new value (in the current radix) after the prompt. Q End the command – Cause MPSIM to go back to the previous *address* <Enter↵> Continue to the next *address*
O *filename [format]*	Output Modified Object Code. Writes contents of program memory, including modifications, to the specified file in the designated format. Valid Formats: INHX8M (Default) INHX8S INHX32 (for PIC17CXX)
SF *address1,address2,register*	Search Program Memory for Register. Searches program memory from *address1* to *address2* for any instruction that accesses the specified file *register.*
SI addr1,addr2,instruction	Search Program Memory in Symbolic Format from *address1* to *address2* for any occurrence of *instruction.*
SM addr1,addr2,instruction	Designated Format. (See SR command to set the radix.) Searches program memory from *address1* to *address2* for any occurrence of *instruction.* Specify *instruction* in the current radix.
ZM address1,address2	Zero the Program Memory from *address1* to *address2.*

Breakpoints

B *address*	Set Breakpoint. Sets breakpoint at *address* (symbolic address can be used).
B *FileReg {operator value}*	Set Breakpoint. Break when *FileReg* matches the condition given by the operator and the *value.* Valid operators: { = | > | < | > = | < = | ! = }
BC	Breakpoint Clear. Cancels all breakpoints.
BC *[addr|FileReg]*	Breakpoint Clear. Cancels breakpoint at addr or *FileReg.*
DB	Display All Active Breakpoints.

Table 4.1 MPSIM Commands by Function (Continued)

Execution Instructions

C *[#breakpoints]*	Continue Executing. Continue execution ignoring *#breakpoints* encountered.
E *[address]*	Execute Program. Begins execution at the specified *address*, or at the current PC if *address* is omitted. The loaded program executes until reaching a breakpoint or until you press any key.
GO	Reset and Execute. Resets the microcontroller, initializes all registers and executes from the start.
RS	Reset Chip. Simulates a power-on reset.
SS *[address]*	Execute a Single Step. Executes a single instruction at the specified *address* or at the current PC if *address* is omitted. To single step through multiple instructions, enter SS once and press <Enter↵> at the % prompt. Then continue pressing <Enter↵>.

Tracing

DX	Display Current Trace Parameters. Displays the current trace parameters. In trace mode, the location, opcode, mnemonic, elapsed time, cycle steps, and changed file registers display when the given conditions are met.
TA *[address1,address2]*	Trace Address. Sets the lower validation limit for address trace to *address1* and the upper address validation limit to *address2*. With no arguments, MPSIM uses the full range of program memory.
TC *[#instructions]*	Trace Instructions. Trace the next *#instructions*. If you omit *#instructions*, execution continues until MPSIM encounters a breakpoint or until you press any key.
TF *[filename|PRN]*	Trace to File/Printer. Open/Close trace output file or write trace to printer. With no argument, TF closes file.
TR *register [,min_val,max_val]*	Trace Register. Sets the file register trace. With no arguments, traces any file register. To perform trace only when register value lies between *min_val* and *max_val*, specify the *min_val* and *max_val*.

Registers and Data Memory

DR	Display Registers. Displays the contents of special function registers including W, status, flags, and the stack.
EE *address*	Modify EE Memory. Modifies memory at the specified address on microcontrollers with EEPROM data memory.

Table 4.1 MPSIM Commands by Function (Continued)

F *FileReg*	File Register Display/Modify. Displays or modifies the contents of any *FileReg* (by absolute address or symbolic name).
IR *{ALL | RAM SFR}*	Initialize with Random Values. Loads random values into registers.
LR *[filename]*	Load Registers. Loads contents of registers with data from a DOS text file.
SE *[I/O_pin | port]*	Display/Modify I/O Pin. Displays or modifies an *I/O_pin* or *port.*
UR *{ALL|RAM|SFR}[filename]*	Upload Registers. Uploads contents of registers into a DOS text file.
W	Work Register Display/Modify. Displays/modifies the contents of the W register.
ZR	Zero the Registers. Zeros all file registers.

Stimulus and Timer

CK *[pin,{#hi,#low|–}*	Clock. Assigns a clock to specified I/O pin. No Argument — Displays current clock assignment. *pin,#hi,#low* — Defines clock period on pin. *pin,–* — Disables clock on specified pin.
DK [AltFxkey#,[pin,event]|[-]]	Define Key. Assigns asynchronous event to an Alt function key. No Argument: — Displays assignment of all function keys *AltFxkey#* — Displays assignment of specified function key *AltFxkey#–* — Cancels specified function. – — The dash cancels all assignments. *event = H,L,T,P* — High, Low, Toggle, Pulse
DW *[E|D]*	Enable/Disable Watchdog Timer. E = Enable D = Disable With no parameters, displays WDT state.
FI *{filename,addr,FileReg[,n]|–}*	File Input. Inserts the next value from filename into file register when current PC=addr. Repeats n times.
IP *[time|step]*	Injection Point. Injects a stimulus according to the time or step count. With no parameters, displays current mode.
RE	Reset Elapsed Time and Step Count.
SC *[cyclelength]*	Display/Set Processor Cycle Time. Displays or modifies the microcontroller's simulated cycle time.
ST *filename*	Read Stimulus File. Loads stimulus file.

Table 4.1 MPSIM Commands by Function (Continued)

`Verbose [ON/OFF]`	Echo to View Screen. Prints a line to the view screen (and to the optional trace file) when a stimulus is injected into a pin. The command, without an argument, displays the current setting.
`WP {1 /.../ 128}`	Watchdog Timer Period. Sets watchdog timer time-out period in ms. With no parameters, displays current setting. Check the device AC characteristics table for typical twdt ranges.
`ZT`	Zero the Elapsed Time Counter.

Program Memory Patch Control

`DP`	Display All Patches. Displays all patches in symbolic format. Both the original object code and new code are shown.
`RA`	Restore All. Restores patch table, clears symbol table and removes all breakpoints.
`RP`	Restore Patches. Restores all patches to original value and clears the patch table.
`ZP`	Zero the Patch Table. Clears patch table and resets to no patches made. Changes made to the object code are unaffected, and object code cannot be restored to the original.

Symbol Table

`DL symbol`	Delete Symbol from Symbol Table. Removes specified symbol from the symbol table.
`DS`	Display Symbol Table.
`GS symbol, value, type`	Generate Symbol. Generates a symbol with the value and type specified. *type:* F – File Register B – Bit L – Label K – Literal

View Screen

`LS filename`	Load Symbol File. Load filename into internal symbol table.
`AD FileReg[,radix[,#digits]]`	Add Item to View Screen. Use the AD command to add a display item to the view screen. The format of this command is identical to the V (View Screen) command. The AD command does not destroy the current contents of the view screen.
`DV FileReg`	Delete View Screen Item. Removes display items from the view screen, leaving the display formatting intact.

Table 4.1 MPSIM Commands by Function (Continued)

NV	No View Screen. Clear the view screen. The same effect can be achieved by redefining the view screen with V command.
TY *FileReg,radix,#digits*	Change View Screen. Change the formatting of the existing view screen. (If the designated signal isn't in the view screen, MPSIM gives a warning.) The radix can be x, o, d, or b. #digits is the number of spaces to reserve for this variable at display time.
V *FileReg[,radix[,#digits]]*	View Screen. Sets up the view screen. Once set, the view screen remains active until either a NV command or a V command sets up a new view screen. FileReg displays in radix mode with #digits. Radix defaults to hexadecimal and #digits to 1. The radix can be hexadecimal, octal, decimal or binary (x, o, d, or b).

System Setup and Control

EL *{0 | 1 | 2}*	Error Level. Sets current error level. 0 = Display All Messages 1 = Display Warnings & Errors Only 2 = Display Error Messages Only Enter EL with no parameters to display current level.
FW *{MC|EM|MP|RC256|RC64|RC|OSC}*	Fuse Word. Select microcontroller, extended microcontroller, or microprocessor operating mode, and set the watchdog timer fuse for the PIC17CXX simulator. Type FW with no parameters to display current modes.
GE *filename*	Get Commands from an External file. Forces MPSIM to read and perform the MPSIM commands in the named ASCII external file. Upon reaching the end of file, control returns to the user.
LJ	Load and Execute Journal File.
P *device#*	Select Microcontroller. Choose the appropriate microcontroller device#. The default is 55 which represents PIC16C55. *device#* = XX where xx is a device suffix.
SR *{O|x|D}*	Set Radix. Sets the input/output radix to octal, hexadecimal, or decimal. The radix will be used on all inputs and outputs with the exception of step counts.

Chapter 5. MPSIM Commands

Introduction

The following table gives an alphabetic summary of the commands currently available with MPSIM. Detail descriptions of each command follow the alphabetic summary.

Press <RETURN> at the % prompt to re-execute the last command entered. Thus you can use commands such as SS more easily.

Alphabetic Summary of MPSIM Commands

Alphabetic Summary of MPSIM Commands	
`[ ]`	Brackets indicate optional items.
`{ }`	Braces indicate group options. One or more options in the group is required.
|	Vertical bar indicates alternative options.
`AB`	Abort Session. Aborts the dialogue, prints a summary message and exits to the operating system. AB does not overwrite the journal file.
`AD` *`FileReg[,radix[,#digits]]`*	Add Item to View Screen. Use the AD command to add a display item to the view screen. The format of this command is identical to the V (View Screen) command. The AD command does not destroy the current contents of the view screen.
`B` *`address`*	Set Breakpoint. Sets beekeeping at address (symbolic address can be used).
`B` *`FileReg {operator value}`*	Set Breakpoint. Break when *FileReg* matches the condition given by the *operator* and the *value*. Valid operators: { = | > | < | > = | < = | ! = }
`BC`	Breakpoint Clear. Cancels all breakpoints.
`BC` *`[addr|FileReg]`*	Breakpoint Clear. Cancels breakpoint at *addr* or *FileReg*.
`C` *`[#breakpoints]`*	Continue Executing. Continue execution ignoring *#breakpoints* encountered.
`CK` *`[pin,{#hi,#low|-}`*	Clock. Assigns a clock to specified I/O pin. No Argument — Displays current clock assignment. *pin,#hi,#low* — Defines clock period on pin. *pin,–* — Disables clock on specified pin.

Alphabetic Summary of MPSIM Commands (Continued)	
DB	Display All Active Breakpoints.
DE *address1,address2*	Delete Program Memory from *address1* to *address2*.
DI *[address1[,address2]]*	Display Program Memory in Symbolic Format. Displays program memory from *address1* to *address2*. Displays in current radix and symbolic format. Omit *address2* to display next 10 lines from *address1*.
DK *[AltFxkey#,[pin,event]|[-]]*	Define Key. Assigns asynchronous event to an Alt function key. No Argument: Displays assignment of all function keys. *AltFxkey#* Displays assignment of specified function key. *AltFxkey#–* Cancels specified function. – The dash cancels all assignments. *event = H,L,T,P* High, Low, Toggle, Pulse
DL *symbol*	Delete Symbol from Symbol Table. Removes specified *symbol* from the symbol table.
DM *[address1,address2]*	Display Program Memory in Radix Designated Format. (See SR command to set the radix.) Displays program memory from *address1* to *address2*. Data displays in current radix only. With no parameter, displays 10 lines continuing from last DM display.
DP	Display All Patches. Displays all patches in symbolic format. Both the original object code and new code are shown.
DR	Display Registers. Displays the contents of special function registers including W, status, flags, and the stack
DS	Display Symbol Table.
DV FileReg	Delete View Screen Item. Removes display items from the view screen, leaving the display formatting intact.
DW [E|D]	Enable/Disable Watchdog Timer. E = Enable D = Disable With no parameters, displays WDT state.

Alphabetic Summary of MPSIM Commands (Continued)	
DX	Display Current Trace Parameters. Displays the current trace parameters. In trace mode, the location, opcode, mnemonic, elapsed time, cycle steps, and changed file registers display when the given conditions are met.
E *[address]*	Execute Program. Begins execution at the specified *address*, or at the current PC if *address* is omitted. The loaded program executes until reaching a breakpoint or until you press any key.
EE *address*	Modify EE Memory. Modifies memory at the specified *address* on microcontrollers with EEPROM data memory.
EL {0 | 1 | 2}	Error Level. Sets current error level. 0 = Display All Messages 1 = Display Warnings & Errors Only 2 = Display Error Messages Only Enter EL with no parameters to display current level.
F *FileReg*	File Register Display/Modify. Displays or modifies the contents of any *FileReg* (by absolute address or symbolic name).
FI *{filename,addr,FileReg[,n]|-}*	File Input. Inserts the next value from *filename* into file register when current PC=*addr.* Repeats n times.
FM *address1,address2,pattern*	Fill Memory. Fills program memory from *address1* to *address2* with specified HEX pattern.
FW *{MC|EM|MP|RC256|RC64|RC|OSC}*	Fuse Word. Select microcontroller, extended microcontroller, or microprocessor operating mode, and set the watchdog timer fuse for the PIC17CXX simulator. Type FW with no parameters to display current modes.
GE *filename*	Get Commands from an External file. Forces MPSIM to read and perform the MPSIM commands in the named ASCII external file. Upon reaching the end of file, control returns to the user.
GO	Reset and Execute. Resets the microcontroller, initializes all registers and executes from the start.

Alphabetic Summary of MPSIM Commands (Continued)	
GS *symbol, value, type*	Generate Symbol. Generates a *symbol* with the *value* and *type* specified. *type:* F – File Register B – Bit L – Label K – Literal
H|Help|?	Help. The Help Command lists the syntax and gives a brief summary of each command available in MPSIM. Press the SPACEBAR to exit Help. Press any other key to display the next help screen.
IA *address*	Insert/Inspect Assembly Code (Symbolic Format). Displays or modifies program memory at address using symbolic format.
IN *address,instruction*	Insert instruction. Inserts instruction at address in symbolic format.
IP *[time|step]*	Injection Point. Injects a stimulus according to the *time* or *step* count. With no parameters, displays current mode.
IR *{ALL | RAM SFR}*	Initialize with Random Values. Loads random values into registers.
LJ	Load and Execute Journal File.
LO *filename[format]*	Load Hex File. Loads the hex file, *filename*, with *format* into program memory. MPSIM also loads the listing file and symbol file. Valid Formats: INHX8M (Default) INHX8S
LR *[filename]*	Load Registers. Loads contents of registers with data from a DOS text file.
LS *filename*	Load Symbol File. Load *filename* into internal symbol table.
M *address*	Display/Modify Program Memory at Address. The contents at the *address* display, and a colon (:) prompt follows. To change the value at *address*, enter a new value (in the current radix) after the prompt. Q End the command – Cause MPSIM to go back to the previous *address* <Enter↵> Continue to the next *address*

Alphabetic Summary of MPSIM Commands (Continued)	
NV	No View Screen. Clear the view screen. The same effect can be achieved by redefining the view screen with **V** command.
O *filename [format]*	Output Modified Object Code. Writes contents of program memory, including modifications, to the specified file in the designated *format.* Valid Formats: INHX8M (Default) NHX8S INHX32 (for PIC17CXX)
P *device#*	Select Microcontroller. Choose the appropriate microcontroller *device#.* The default is 55 which represents PIC16C55. *device# = XX where XX is a device suffix.*
Q	Quit. Terminates the dialog. Prints one or two summary messages, overwrites the old journal file and exits to the opening system.
RA	Restore All. Restores patch table, clears symbol table and removes all breakpoints.
RE	Reset Elapsed Time and Step Count.
RP	Restore Patches. Restores all patches to original value and clears the patch table.
RS	Reset Chip. Simulates a power-on reset.
SC *[cyclelength]*	Display/Set Processor Cycle Time. Displays or modifies the microcontroller's simulated cycle time.
SE *[I/O_pin | port]*	Display/Modify I/O Pin. Displays or modifies an *I/O_pin* or *port.*
SF *address1,address2,register*	Search Program Memory for Register. Searches program memory from *address1* to *address2* for any instruction that accesses the specified file *register.*
SI *addr1,addr2,instruction*	Search Program Memory in Symbolic Format from *address1* to *address2* for any occurrence of *instruction.*
SM *addr1,addr2,instruction*	Search Program Memory in Radix Designated Format. (See SR command to set the radix.) Searches program memory from *address1* to *address2* for any occurrence of instruction. Specify *instruction* in the current radix.
SR *{O|x|D}*	Set Radix. Sets the input/output radix to octal, hexadecimal, or decimal. The radix will be used on all inputs and outputs with the exception of step counts.

Alphabetic Summary of MPSIM Commands (Continued)	
SS *[address]*	Execute a Single Step. Executes a single instruction at the specified *address* or at the current PC if *address* is omitted. To single step through multiple instructions, enter SS once and press <Enter↵> at the % prompt. Then continue pressing <Enter↵>.
ST *filename*	Read Stimulus File. Loads stimulus file.
TA *[address1,address2]*	Trace Address. Sets the lower validation limit for address trace to *address1* and the upper address validation limit to *address2*. With no arguments, MPSIM uses the full range of program memory.
TC *[#instructions]*	Trace Instructions. Trace the next *#instructions*. If you omit *#instructions*, execution continues until MPSIM encounters a breakpoint or until you press any key.
TF *[filename|PRN]*	Trace to File/Printer. Open/Close trace output file or write trace to printer. With no argument, TF closes file.
TR *register [,min_val,max_val]*	Trace Register. Sets the file register trace. With no arguments, traces any file register. To perform trace only when register value lies between *min_val* and *max_val*, specify the *min_val* and *max_val*.
TY *FileReg,radix,#digits*	Change View Screen. Change the formatting of the existing view screen. (If the designated signal isn't in the view screen, MPSIM gives a warning.) The *radix* can be x, o, d, or b. *#digits* is the number of spaces to reserve for this variable at display time.
UR *{ALL|RAM|SFR}[filename]*	Upload Registers. Uploads contents of registers into a DOS text file.
V *FileReg[,radix[,#digits]]*	View Screen. Sets up the view screen. Once set, the view screen remains active until either a NV command or a V command sets up a new view screen. *FileReg* displays in *radix* mode with *#digits*. *Radix* defaults to hexadecimal and *#digits* to 1. The *radix* can be hexadecimal, octal, decimal or binary (x, o, d, or b).
Verbose *[ON|OFF]*	Echo to View Screen. Prints a line to the view screen (and to the optional trace file) when a stimulus is injected into a pin. The command, without an argument, displays the current setting.
W	Work Register Display/Modify. Displays/modifies the contents of the W register.

Alphabetic Summary of MPSIM Commands (Continued)	
WP {1 |...| 128}	Watchdog Timer Period. Sets watchdog timer time-out period in ms. With no parameters, displays current setting. Check the device AC characteristics table for typical twdt ranges.
ZM *address1,address2*	Zero the Program Memory from *address1* to *address2*.
ZP	Zero the Patch Table. Clears patch table and resets to no patches made. Changes made to the object code are unaffected, and object code cannot be restored to the original.
ZR	Zero the Registers. Zeros all file registers.
ZT	Zero the Elapsed Time Counter.

AB – Abort Session

Syntax

AB

Description

The abort command interrupts the MPSIM session and exits. It prints out one or two summary messages, and exits to the operating system. MPSIM retains the journal file.

Examples

MPSIM Command	Result
AB<RETURN>	MPSIM exits and displays the following message: `Elapsed CPU time: h:mm:ss`

Defaults

None.

AD – Add Item to View Screen

Syntax

AD *FileReg[,radix[,#digits]]*

Description

The Add command adds a signal or register to the view screen. Optionally, you may specify a radix different from the default and/or the number of digits.

While this command's format is identical to View, it doesn't destroy the current contents of the view screen, but simply displays additional items as well as the current ones.

Examples

MPSIM Command	Result
AD TRISA	Add Tris A to the screen.
AD RA0,B	Add the RA0 pin to the screen display with binary radix.
AD MCLR,4	Add $\overline{\text{MCLR}}$ pin to the screen display with 4 digits.
AD F3,B,8	Add the F3 register (status) to the screen display with binary radix and 8 digits.

Defaults

Digits defaults to 2. The radix **ordinarily defaults to hexadecimal**, but you can change this default with the SR command.

Radix	Digits
X	2
B	8
O	3
D	2

Related Commands

The V command displays the first signal or register you request. Subsequently, you can add display items with AD or delete them with DV. If you use a V command after AD, V replaces all previous display items on the screen with the named signal or register. The NV command wipes all display items off the screen.

The GE command can load an initialization file that sets up the view screen. Thereafter, you can use AD and DV to modify it.

Note: When referencing registers for the AD instruction use hex notation. For example, file register 10 would be written as "0A".

Example:

```
AD F0A, X, 2
```

B – Set Breakpoint

Syntax

B address

B *FileReg [operator value]*

Description

This command sets a breakpoint at the specified address or at the location where the register matches the condition set by the operator and the value.

You can designate the address either with the explicit numeric location or with a symbol.

The operator can be any of the following:

=	equal
>	greater than
<	less than
>=	greater than or equal
<=	less than or equal
!=	not equal

Examples

MPSIM Command	Result
B LOOP<RETURN>	Set breakpoint at label LOOP.
B F2 > 80<RETURN>	Break if F2 is greater than 80.

Defaults

None.

Related Commands

BC clears breakpoints previously set and DB displays them.

Note: When referencing registers for relational instructions use decimal notation.

BC – Clear Breakpoint

Syntax

BC

BC *[addr | FileReg]*

Description

This command deletes a specified breakpoint, or all breakpoints if you don't specify one by address or file register.

Examples

MPSIM Command	Result
BC LOOP	Cancel breakpoint at LOOP.
BC F3	Cancel breakpoint involving the F3 register.
BC	Cancel all breakpoints.

Defaults

None.

Related Commands

B sets breakpoints and DB displays them.

C – Continue Executing

Syntax

C *[#breakpoints]*

Description

This command continues execution from the current PC. If you specify *#breakpoints*, MPSIM ignores the first *#breakpoints* breakpoints encountered.

Examples

MPSIM Command	Result
`C`	Continue executing, break at the next breakpoint.
`C 3`	Continue executing, skip the first three breakpoints found, but break at the fourth.

Defaults

`n` defaults to 0.

Related Commands

`B` sets the breakpoints, `DB` displays them and `BC` clears breakpoints previously set.

CK – Clock

Syntax

`CK` *`[pin,{#hi,#low|-}]`*

Description

This command allows you to assign a clock to an I/O pin, defining the period of the clock by stating the number of cycles that the pin should be high, and the number of cycles that it should be low.

No Argument	Displays current clock assignment.
#hi	Defines the number of T-cycles that the pin should remain high
#low	Defines the number of T-cycles that the pin should remain low
pin,–	Disables clock on specified pin.

Examples

MPSIM Command	Result
`% CK RC0, 5, 4`	Assign a clock to RC0 with a 9 T-cycle period (5 high and 4 low cycles)
`% CK RC0, -`	Cancel clock on RC0
`% CK`	Display current clock assignment

Defaults

None

Related Commands

None

DB – Display All Active Breakpoints

Syntax

DB

Description

This command lists all active breakpoints. MPSIM allows only one conditional breakpoint per data area.

Examples

MPSIM Command	Result
B LOOP	Sets a breakpoint at LOOP.
B F2 > 80	Sets a breakpoint at the location where F2 >80.
DB	Displays all breakpoint locations via messages:
	`INFO, Break when (F2 > 0080)`
	`INFO, Break on address LOOP`

Defaults

None.

Related Commands

`B` sets the breakpoints, `DB` displays them and`BC` clears breakpoints previously set.

DE – Delete Program Memory

Syntax

DE *address1,address2*

Description

This command deletes the information stored between *address1* and *address2*, inclusively. The `DE` command deletes memory within the given boundaries then shifts those locations in program memory that are greater than the upper bound down to the lower bound.

Examples

MPSIM Command	Result
DE *0015, 0020*	Removes the code from *address 15* thru *address 20*, moving code from 21 to the end of code to *address* 16.

Defaults

None.

Related Commands

None.

DI – Display Program Memory in Symbolic Format

Syntax

DI *[address1[, address2]]*

Description

This command displays program memory in symbolic format from *address1* to *address2*. *address1* must be less then *address2* and both must be in the valid range of program memory. If no *address2*, then a screen full of lines displays from *address1*.

You can terminate DI at any time by pressing any key at the terminal.

Examples

MPSIM Command	Result
DI 0, 3	The following messages display: 0000 0020 MOVWF 0 0001 0063 CLRF 3 0002 0080 SUBWF 0, 0 0003 0069 CLRF 9

Defaults

None.

Related Commands

The DM command also displays memory between two specified addresses; however, DM displays the code in the format specified by the current radix rather than in symbolic format.

DK – Define Key

Syntax

DK *[AltFxKey#,[pin,event]|[-]]*

Description

AltFxKey#	is an integer value between 1 and 12.
AltFxKey#–	Cancels specified function.
Pin	is any valid I/O pin.
Event	is H, L, T or P (high, low, toggle or pulse)
–	Cancels all assignments.
No Argument	Displays assignments of all function keys.

This command simulates an asynchronous event through an Alt-function keystroke and is very useful for simulating external interrupts or resets.

This function is used after a "GO" or "E" command. If you want to inject a stimulus while stopped at a breakpoint, use the "SE" command.

In addition to the stated syntax, the following sequences perform the indicated operations.

DK	Displays assignment of all function keys
DK AltFxKey	Displays assignment of specified function key
DK AltFxKey, -	Cancels specified function
DK -	Cancels all assignments

Examples

MPSIM Command	Result
% DK 1,RB0,L	
% E	When MPSIM is executing, if Alt-F1 is hit, RB0 will be driven low.
% DK 12, MCLR, P	Define Alt-F12 to provide a one-cycle pulse on $\overline{MCLR}$.
% E	Now during execution (with $\overline{MCLR}$ high) hitting Alt-F12 will simulate an external reset.
% DK 3, T0CKI, T	Define Alt-F3 to toggle T0CKI input.
% E	Now during execution, every time Alt-F3 is pressed RTCC input will toggle.
% DK -	Disable all assignments.

Defaults

None

Related Commands

SE, ST

DL – Delete Symbol from Symbol Table

Syntax

DL *symbol*

Description

This command removes the specified symbol from the symbol table.

Examples

MPSIM Command	Result
DL *MULPLR*	MPSIM removes ***"mulplr"*** from the symbol table. To provide to or obtain data from this data area, you must now use the actual register number, F10. The value on the view screen, since it reads "MULPLR" isn't updated.

Defaults

None.

Related Commands

GS creates a symbol and puts it into the symbol table, LS loads a new symbol table, DS displays the current symbol table andRA restores (clears) the symbol table.

DM – Display Program Memory in Radix Designated Format

Syntax

DM [*address1,address2*]

Description

This command displays program memory from*address1* to *address2*. The data stored displays in the format designated by the current radix *address1 must* be less than *address2* and both must be in the valid range for program memory.

You can terminate DM at any time by pressing any key on the terminal.

Examples

MPSIM Command	Result
DM 0, 3	MPSIM displays the memory between locations 0 and 3. The following messages display: 0000 0020 0001 0063 0002 0080 0003 0069

Defaults

None.

Related Commands

The DI command also displays memory between two specified addresses; however, DI displays the code in symbolic format rather than in the format specified by the current radix.

DP – Display All Patches

Syntax

DP

Description

This command displays all patches in symbolic format. Both the original object code and new object code display.

Defaults

None.

Related Commands

The M and IA commands modify the object code; .IN adds commands to the object code; DE removes object code; RA and RP restore the patches; and ZP zeros the patches. The command, O, writes the modified object code as a hex file.

DR – Display All Registers

Syntax

DR

Description

This command displays the contents of all registers including the W and status registers, all flags and the stack.

Defaults

None.

Related Commands

The DP, DS, and DX commands display other MPSIM data areas and parameters.

SE sets any data area's value. W displays and optionally modifies the W register.

F displays and optionally modifies a register value.

DS – Display Symbol Table

Syntax

DS

Description

This command displays the symbol table.

Examples

MPSIM Command	Result
DS	The following messages display: Symbol Value Type START 0000 L

Defaults

None.

Related Commands

GS creates a symbol and puts it into the symbol table, LS loads a new symbol table, DL removes a symbol from the current symbol table and RA restores (clears) the symbol table.

DV – Delete View Screen Item

Syntax

DV *FileReg*

Description

This command removes display items from the view screen while leaving the display formatting intact.

This command deletes a signal or register from the view screen display.

Examples

MPSIM Command	Result
DV *RTCC*	Deletes the RTCC from the view screen.

Defaults

None.

Related Commands

The V command displays the first signal or register you request. Subsequently, you can add display items with AD or delete them with DV. If you use a V command after AD, V replaces all previous display items on the screen with the named signal or register. The NV command wipes all display items off the screen.

The GE command can load an initialization file that sets up the view screen. Thereafter, you can use AD and DV to modify it.

DW – Enable / Disable Watchdog Timer

Syntax

DW *[E|D]*

Description

This command enables or disables the watchdog timer, depending or the parameter specified. *E* enables it; *D* disables it.

Examples

MPSIM Command	Result
DW E	Enables the watchdog timer.
DW D	Disables the watchdog timer.

Defaults

None.

Related Commands

RE resets the elapsed time and step count and ZT zeros the elapsed time.

DX – Display Current Trace Parameters

Syntax

DX

Description

This command displays the current trace parameters. When in trace mode, the location, opcode, mnemonic, elapsed time, cycle steps, and any changed data areas display when the given conditions are met.

Examples

MPSIM Command	Result
DX	The following message displays: `Address 0000:01FF`

Defaults

None.

Related Commands

The TA, TC and TR commands set the trace parameters.

E – Execute Program

Syntax

E *[address]*

Description

This command executes the program from the optionally specified address or the PC.

The E command begins execution at the specified address or at the current address if no address is specified. The program continues to execute until either reaching a breakpoint or until you press a key.

Examples

MPSIM Command	Result
E 0E	MPSIM executes SAMPLE.HEX from the label START until reaching a breakpoint or until you press any key.

Defaults

None.

Related Commands

The GO command resets then executes from the start; SS executes the instruction at the current PC or at a specified address. C executes from the current PC to the specified breakpoint occurrence. TA traces execution between specified addresses, and TC traces execution from the current PC for a specified number of instructions.

EE – Modify EE Memory

Syntax

EE *address*

Description

Manually Modify EE memory address on the PIC16C84.

Examples

MPSIM Command	Result
% **EE 2**	EEMEMORY[2]:00:
23	EE memory location 2 now contains value 0x23.

Defaults

None.

EL – Error Level

Syntax

EL *{0 | 1 | 2}*

Description

This command sets the current level for displaying error messages and system warnings.

0 = Display All Messages

1 = Display Warnings and Errors Only

2 = Display Error Messages Only

Enter EL with no parameters to display current level.

Examples

MPSIM Command	Result
% EL 0	All messages will display.
% EL 1	Only warnings and Error messages will display.
% EL 2	Only error messages will display.
% EL	Current warning level will display.

Defaults

All messages are displayed (Error level of 0).

Related Commands

None

F – File Register Display/Modify

Syntax

F *FileReg*

Description

This command displays and/or modifies the contents of the specified file register. The value of the register displays, followed by the prompt ':'. Place the new value after the prompt.

Examples

MPSIM Command	Result
% F 3	The following message displays:
3: **20:**	This message shows that file register 3 contains the value '20.'
% F 3	
3: 20: 21	Change file register 3 to a value of '21.'

Defaults

None.

Related Commands

The SE command can give the same result. DR, TR and ZR display, trace and zero a specified register, respectively. M and IA modify the code at a specified address, which can affect the register's value.

FI – File Input

Syntax

FI *{filename,addr,FileReg[,n]|-}*

Description

This command inserts the next value from *filename* into file register when current *PC=addr.* If n is not specified, when the last value in the file is read, the next retrieved value will be the first value in the file. This will continue until the command is cancelled. If n is specified then the file will be read n times only.

FileName	is any valid DOS file name. The file should be an ASCII file and should contain one hex value per line.
PMemAddr	is the point in program memory at which value should be injected.
FRegAddr	File register that receives the value.
n	Number of times to go through the file. If n is not specified, file is read continuously.
FI -	Closes file and cancels command.

Examples

MPSIM Command	Result
`%FIADVAL.TXT,4,9`	When the PC=4, insert the next value from ADVAL.TXT into register 9.
`% FI -`	Close file and cancel assignment.

The `FI` command is useful when simulating devices such as the PIC16C71 and PIC16C74. Both of these devices have A/D converters (among other peripheral modules). MPSIM does not perform an A/D conversion, although the interrupt that can be generated upon its completion is supported in the software. The `FI` command allows you to inject values into a register when a certain point in program memory is reached. For example, if the target processor is the PIC16C71, you could set up your source code to branch to the interrupt vector at the end of conversion and inject a value into the ADRES register during the interrupt service routine (by using the `FI` command).

The command could be set up as follows:

```
FI        ADVAL.TXT,  0x04 0x09
```

When the Program Counter equals the interrupt vector (program memory address 0x04), inject the next value in the file (ADVAL.TXT) into the ADRES register (file register address 0x09).

```
      org       0x04
IntVct bcf      INTCON, ADIE  ;At this point, the next
       movfw    ADRES         ;value in ADVAL.TXT will
                              ;be in the ADRES register
       •
       •
       •
```

The format of the ADVAL.TXT file is one HEX value on each line. For example:

0xAA

0X55

0XAA

0X55

and so on.

Defaults

None.

Related Commands

None.

FM – Fill Memory

Syntax

FM *address1,address2,pattern*

Description

This command fills unused program memory between *address1* and *address2* with the specified HEX *pattern.*

Examples

MPSIM Commands	Result
% FM 0,30,0xFFF	Fill unused program memory between 0 and 30 with 0xFFF.

Defaults

None.

Related Commands

M

FW – Fuse Word

Syntax

FW *{MC|EM|MP|RC256|RC64|RC|OSC}*

Description

This command selects the operation mode (Microcontroller Mode, Extended Microcontroller Mode, and Microprocessor Mode) or the WDT prescale option for processors in the PIC17CXX family only (these options are hardware fuse-selectable only on the physical device).

MC = Microcontroller Mode

EM = Extended Microcontroller Mode

MP = Microprocessor Mode

Type FW with no parameters to display current modes.

Examples

MPSIM Command	Result
% **FW MC**	Processor will run in Microcontroller Mode
% **FW RC64**	WDT will have a prescale = 64
% **FW**	Current operation mode and WDT prescale option are displayed.

Defaults

Operation mode = Microprocessor

WDT Prescale = OSC

Related Commands

None.

GE – Get Commands from an External File

Syntax

GE *filename*

Description

This command reads and performs the MPSIM commands in the named ASCII file.

This command forces MPSIM to get its command stream from an external text file. After reaching the end of file, control returns to the user. Commands in the text file must conform to the same syntax as commands entered on-line. If MPSIM cannot locate the specified file, an error message displays.

Examples

MPSIM Command	Result
GE SAMPLE.INI	Reads and performs commands in the file, SAMPLE.INI.

Defaults

None.

Related Commands

The V command displays the first signal or register you request. Subsequently, you can add display items with AD or delete them with DV. If you use a V command after AD, V replaces all previous display items on the screen with the named signal or register. The NV command wipes all display items off the screen.

The GE command can load an initialization file that sets up the view screen. Thereafter, you can use AD and DV to modify it.

GO – Reset and Execute

Syntax

GO

Description

This command performs a Power-On Reset and initializes all registers as specified in the microcontroller data sheet. The PIC16/17 Microcontroller then executes the loaded object code.

Examples

MPSIM Command	Result
GO	Reset and execute.

Defaults

None.

Related Commands

The E command executes from a specified address or the current PC;SS executes the instruction at the current PC or at a specified address.C executes from the current PC to the specified breakpoint occurrence. TA traces execution between specified addresses, and TC traces execution from the current PC for a specifies number of instructions.

GS – Generate Symbol

Syntax

GS *symbol,value,type*

Description

This command generates the specified symbol with the specified value and type. The type can be file(F), bit(B), label(L), or literal(K). If the type is bit, it is a bit in the specified file.

Examples

MPSIM Command	Result
% DS	
	Symbol Value Type
	START 0000 L
% GS NEWSYM, FF, B	
% DS	
	Symbol Value Type
	START 0000 L
	NEWSYM FF B

Defaults

None.

Related Commands

DL removes a symbol from the current symbol table,LS loads a new symbol table, DS displays the current symbol table andRA restores (clears) the symbol table.

H – Help

Syntax

H | Help | ?

Description

This command lists the syntax and gives a brief summary of each command available in MPSIM. Press the SPACEBAR to exit Help. Press any other key to display the next help screen.

Examples

MPSIM Command	Result
H	The Help screen, containing command descriptions and syntax displays.

Defaults

None.

Related Commands

None.

IA – Insert/Inspect Assembly Code

Syntax

IA address

Description

This command displays or modifies the program memory at address in symbolic format. The source code for the address displays, followed by the prompt ':' on the next line for the new command.

Enter the new command as a mnemonic. It must be syntactically correct. Operands may contain only a single value or symbol; expressions are not allowed. Enter values in the current radix.

Entering '*Q*' at the prompt ends the command; entering '-' causes MPSIM to go back and inspect and/or modify the previous address; entering <RETURN> continues to the next address.

After changing the object code, MPSIM no longer displays the original source code. MPSIM replaces it with a disassembled source line.

Examples

MPSIM Command	Result
% ***IA 200***<RETURN>	The instruction line at address 200 (in current radix) displays: 0020 : CLRF F5
: ***CLRF 6***	MPSIM changes the instruction as specified and displays the next instruction line (address 201): 0201 : CLRF F7:
: -	MPSIM backs up and displays the modified instruction at address 200: 0200 : CLRF 6:
: ***Q***	MPSIM exits the IA command.

Defaults

None.

Related Commands

DE, IN, M

IN – Insert Instruction

Syntax

IN *address, instruction*

Description

This command inserts *instruction* at *address.* The *instruction* places an opcode at address then displaces each program memory value after *address* by one location. *instruction* must consist of a valid mnemonic followed by zero or more operands. Each operand must contain a single value or symbol, no expressions are allowed.

Examples

MPSIM Command	Result
% ***IN 200, NOP***	MPSIM inserts a NOP instruction at address 200 (in the current radix).

Defaults

None.

Related Commands

DE, IA, M

IP – Injection Point

Syntax

IP *[time|step]*

Description

Inject stimulus according to time or step count. The "step" heading should remain labeled as "step" in the stimulus file, but `IP TIME` will override this setting. If `IP` is typed with no parameters, the current mode (`TIME or STEP`) will be shown. With no parameters, displays current mode.

Examples

MPSIM Command	Result
% **IP time**	Stimulus will now be injected according to time (integer values only).

Defaults

Default is "step"

Related Commands

None.

IR – Initialize with Random Values

Syntax

IR *{ALL | RAM SFR}*

Description

Loads random values into registers. Will also load the power on reset values into those registers that have defined power on reset values defined in the data book.

Examples

MPSIM Command	Result
% **IR ALL**	All registers will be initialized with random values, except those that have defined values for power on reset.
% **IR RAM**	Only general-purpose registers will be initialized with random values

Defaults

ALL file registers will be loaded with random values.

Related Commands

UR, LR

LJ – Load and Execute Journal File

Syntax

LJ

Description

This command loads and executes the journal file commands. These commands are not stored in the journal file recorded from the current session; MPSIM enters only the LJ command.

When the journal file contains a program execution command, you must press a key to stop program execution or wait until a breakpoint break occurs; the journal file doesn't record premature execution breaks or exits.

Examples

MPSIM Command	Result
% **LJ**	All MPSIM commands entered during the previous MPSIM session execute. These commands are not stored in the journal file recorded from the current session.

Defaults

The default extension is '.JRN'.

Related Commands

GE, LJ, LO, ST

LO – Load Object File

Syntax

LO *filename {format}*

Description

This command loads the specified file into program memory. If the selected assembler is MPASM, MPSIM will assume a .HEX extension. After loading the HEX file, MPSIM attempts to load the listing file using the same filename and the extension '.LST'. If MPSIM cannot find the listing file then all instruction displays will be a disassembly. When found, MPSIM uses the listing file for display throughout simulation.

The following is a list of valid formats:

INHX8M

INHX8S

Examples

MPSIM Command	Result
% LO *SAMPLE*	The HEX, listing and symbol file are loaded into MPSIM in INHX8M format.
% LO *SAMPLE INHX8S*	The HEX, listing and symbol file are loaded into MPSIM in INHX8S format.

Defaults

The default extension is '.HEX' and the default format is INHX8M.

Related Commands

GE, LJ, LS, ST

LR – Load Registers

Syntax

LR *filename*

Description

This command loads the contents of registers with data from a DOS text file.

This command loads each file register listed on each row of *"filename"* with the specified value. If no file name is supplied, MPSIM searches for a file called "ram.dat". Each line in the file should consist of the Bank Number, File Register Number, and Value as follows:

```
BankNumber : FileRegisterNumber , Value
```

This format is also used with the "UR - Upload Registers" command. The following sample is an excerpt from a "ram.dat" file:

```
;File register values for "myfile.asm"
0, 0x0F, 0x0F
0, 0x10, 0xAA
0, 0x17, 0xFF
```

All values should be in hexadecimal radix and should begin in the first column of each row. Blank lines or lines beginning with ";" or "!" will be interpreted as comment lines and will be ignored. If an error is found in the file, a warning message will be displayed and the offending line will be ignored.

Examples

MPSIM Command	Result
% **LR**	File registers in "RAM.DAT" will be loaded with specified value
% **LR myfile**	File registers in "myfile" will be loaded with specified value.

Defaults

Registers and values from "ram.dat" file are loaded.

Related Commands

UR, IR

LS – Load Symbol File

Syntax

LS *filename*

Description

This command loads the specified symbol file into the internal symbol table. If symbolic debugging, the symbol file produced by the assembler must be loaded with the LS command or loaded through the LO command.

Examples

MPSIM Command	Result
% **LS SAMPLE**	MPSIM reads in the symbol file SAMPLE.

Defaults

The default extension is '.SYM'.

Related Commands

GS, DL, DS, RA

M – Display / Modify Program Memory at Address

Syntax

M *address*

Description

This command displays and/or modifies program memory at address. The contents of the address display in the radix designated format, and are followed immediately by a prompt ':'.

To change the value at address, place a new value after the prompt. Be sure to enter that value in the current radix.

Entering 'Q' at the prompt ends the command.

Entering '–' causes MPSIM to go back and inspect and/or modify the previous address.

Entering <RETURN> continues to the next address.

Examples

MPSIM Command	Result
% **M 0005**	MPSIM displays the instruction line at address 0005 (as determined by the current radix) in the current radix:
% **SR O**	
% **M 010**	MPSIM sets the radix to octal, then displays the instruction line at the label MAIN in octal.
% **Q**	MPSIM exits the M command.
% **SR X**	
% **M 010**	MPSIM sets the radix to hexadecimal, then displays the instruction line at the label MAIN in hexadecimal.
: –	MPSIM redisplays the instruction line at MAIN.
% **SR D**	
% **M main**	MPSIM sets the radix to decimal, then displays the instruction line at the label MAIN in decimal.

Defaults

None.

Related Commands

IA

NV – No View Screen

Syntax

NV

Description

This command deletes or clears all elements from the view screen.

The same effect can be achieved by redefining the view screen.

Examples

MPSIM Command	Result
% NV	MPSIM removes all items from the view screen.

Defaults

None.

Related Commands

AD, V

O – Output Modified Object Code

Syntax

O *filename [Format]*

Description

This command writes the contents of program memory, including any modifications to the specified file in the specified format. The program memory contains object code.

The following is a list of valid formats:

INHX8M

INHX8S

INHX32

Examples

MPSIM Command	Result
`%O`***`SAMPLE1.HEXINHX8M`***	MPSIM writes the object code, as modified, to the file SAMPLE1.HEX in the INHX16 format.

Defaults

Default output format is the same as the default input format, INHX8M.

Related Commands

None.

P – Select Microcontroller

Syntax

P *device#*

Description

Use this command to choose the appropriate microcontroller *device#*. The default is 55 which represents PIC16C55.

device# = XX where XX is a device suffix.

Examples

MPSIM Command	Result
% P 71	MPSIM sets the processor type.

Defaults

The simulated microcontroller defaults to 55.

Related Commands

None.

Q – Quit

Syntax

Q

Description

This command exits from MPSIM and returns PC control to DOS. MPSIM stores all MPSIM commands entered during this session in the journal file, MPSIM.JRN. The old MPSIM.JRN, if present, is overwritten.

Examples

MPSIM Command	Result
% **Q**	MPSIM exits and displays the following message: Elapsed CPU time: *h:mm:ss*.

Defaults

None.

Related Commands

AB

RA – Restore All

Syntax

RA

Description

This command restores the patch table, clears the symbol table of user defined symbols and removes all breakpoints.

Examples

MPSIM Command	Result
% **RA**	MPSIM restores the patch table, clears the symbol tables and removes all breakpoints.

Defaults

None.

Related Commands

RP, DL, BC

RE – Reset Elapsed Time and Step Count

Syntax

RE

Description

This command resets the elapsed time and the step count to zero.

Examples

MPSIM Command	Result
% RE	MPSIM resets the elapsed time and the step count to zeros.

Defaults

None.

Related Commands

ZT

RP – Restore Patches

Syntax

RP

Description

This command restores all patches to their original value and clears the patch table.

Examples

MPSIM Command	Result
% RP	MPSIM restores all patches.

Defaults

None.

Related Commands

RA

RS – Reset Chip

Syntax

RS

Description

Performs a Power-On Reset and initializes all registers as specified in the data sheet of the specified microcontroller.

Examples

MPSIM Command	Result
% RS	Executes a Power-On-Reset.

Defaults

None.

Related Commands

GO

SC – Display / Modify Processor Cycle Time

Syntax

SC *[cyclelength]*

Description

This command displays and/or modifies the microcontroller's simulated cycle time.

Examples

MPSIM Command	Result
% **SC**	MPSIM displays the current cycle in µs: 2.0:
2.0:.**2**	The entry '.2' changes the cycle to .2µs, or 200 ms.
% **SC 2000.0**	The cycle length is changed to 2000.0µs or 2.0 ms.

Defaults

The simulated cycle time defaults to 2 microseconds.

Related Commands

None.

SE – Display / Modify I/O Pin

Syntax

SE *[I/O_pin | port]*

Description

This command displays or modifies an *I/O_pin* or *port*.

Examples

MPSIM Command	Result
% **SE *RA0***	The following message displays: RA0=1:
RA0:1:***0***	The value of I/O pin RA0 changes from 1 to 0.

Defaults

None.

Related Commands

F, W, ZR

SF – Search Program Memory for Register

Syntax

SF *address1,address2,register*

Description

This command searches program memory from address1 to address 2 for any instruction that access the specified register. Register may be specified in literal, 'F' syntax or as a symbol.

Examples

MPSIM Command	Result
SF ***0, 22, portb***	MPSIM search all memory from 0 through 22 for instructions that reference the portb register, then displays the lines containing the specified instruction 0000 0000 main movf portb,W 0006 0000 movf portb,W

Defaults

None.

Related Commands

SI, SM

SI – Search Program Memory in Symbolic Format

Syntax

SI *address1,address2,instruction*

Description

This command searches program memory from *address1* to *address2* for any occurrence of *instruction*. *instruction* is in symbolic format. Full or partial instructions may be specified.

Examples

MPSIM Command	Result

% SI *0, 20, CLRF*	MPSIM searches all memory from 0 through 20 for CLRF instructions, then displays the lines containing the specified instruction:
	`0000 mpy_S clrf H_byte`
	`0001 clrf L_byte`

% SI 0, 20,	MPSIM searches all movwf count memory from 0 through 20 for `MOVWF COUNT` instructions, then display the lines containing the specified instruction: `0003 movwf count`

Defaults

None.

Related Commands

SF, SM

SM – Search Program Memory in Radix Designated Format

Syntax

SM *address1,address2,instruction*

Description

This command searches program memory from *address1* to *address2* for *instruction*. Specify *instruction* in the format designated by the radix.

Examples

MPSIM Command	Result
% **SM** *0, 30, C08*	MPSIM searches all memory from 0 through 20 for the specified instruction, then displays, in the current radix, the lines containing it: 0002 movlw 8

Defaults

None.

Related Commands

SF, SI

SR – Set Radix

Syntax

SR *[O|X|D]*

Description

This command sets the radix to octal, hexadecimal or decimal. Subsequently, MPSIM expects and uses this radix for all I/O including file register numbers and step counts.

Examples

MPSIM Command	Result
% **SR O**	The radix becomes octal.
% **SR X**	The radix becomes hexadecimal.
% **SR D**	The radix becomes decimal.

Defaults

None.

Related Commands

None.

SS – Execute A Single Step

Syntax

SS *[address]*

Description

This command executes a single step located at *address*. If you don't specify *address*, MPSIM executes the instruction at the current PC. Pressing <RETURN> at the % prompt re-executes the previous MPSIM command. Thus, by entering SS once and subsequently pressing simply <RETURN>, you can single step through multiple instructions easily.

Examples

MPSIM Command	Result
% **SS 01FF**	MPSIM resets the simulator code by executing the reset address (PIC16C54 and PIC16C55).
% **SS**	MPSIM executes the line of code at the PCP.
% SS **20**	MPSIM executes the line of code at address 20 (in the current radix).
% SS **LOOP**	MPSIM executes the line of code at the label LOOP.

Defaults

None.

Related Commands

None.

ST – Read Stimulus File

Syntax

ST *filename*

Description

This command inserts specified values into specified pins or registers at a specified simulation step or time. The specified values, pins/registers and steps are defined in a text file called a stimulus file. Stimulus can be injected either according to step or time. See instruction 'IP' for details.

The stimulus file allows you to schedule bit manipulation by forcing MPSIM to drive given pins to given values at a specified input step.

The ST command reads the stimulus file into MPSIM. When you execute a file with the E command, each time it looks for input, it reads the next step in the stimulus file.

The first line of stimulus file always consists of column headings. It lists first the word "STEP," followed by the pins that are to be manipulated. The data below STEP represents the object file's input request occurrence. The data below each pin name is the input value. You may enter comments at the end of a line by preceding it with an exclamation mark (!).

The following example illustrates the stimulus file format:

```
STEP    RA0     RA1     ! These are I/O pin names
8       1       0
16      0       1       ! followed by values
24      1       1
```

Other notes on the format of stimulus file:

- the number of spaces separating columns is irrelevant
- the step count must be in decimal

Examples

MPSIM Command	Result
% **ST** ***SAMPLE.STI***	MPSIM reads the specified stimulus file. Upon execution, it will retrieve input as designated in this file.

Defaults

The default injection point is "step". The default file extension is '.STI'.

Related Commands

IP

TA – Trace Address

Syntax

TA *[address1,address2]*

Description

This command sets the trace to print only those instructions located between *address1* and *address2*. If you don't specify *address1* and *address2*, MPSIM uses the full memory.

Examples

MPSIM Command	Result
% TA *main, call_m*	MPSIM will print/display only those instructions between *main* and *call_m*.

Defaults

Address range defaults to all of memory.

Related Commands

TC, TF, TR

TC – Trace Instructions

Syntax

TC [*#instructions*]

Description

This command traces the next *#instructions* instructions, displaying the instructions if they are valid. If you don't supply the *#instructions*, the trace continues indefinitely until encountering a breakpoint or until you press any key.

Examples

MPSIM Command	Result
% **TC 3**	Trace the next three instructions.

Defaults

None.

Related Commands

TA, TF, TR

TF – Trace to File/Printer

Syntax

TF *[filename | Prn]*

Description

This command opens or closes a file for writing the trace, or prints the trace. If you enter *PRN* as an argument, MPSIM prints the trace to the default printer. If you supply *filename*, MPSIM opens that file, if you don't, MPSIM closes any currently opened output trace file.

You must use the TF command BEFORE starting the trace.

Examples

MPSIM Command	Result
% **TF**	Close the output trace file.
% **TF PRN**	Print the trace to the default printer.
% **TF *SAMPLE.TRC***	Open SAMPLE.TRC and write the trace to it.

Defaults

None.

Related Commands

TA, TC, TR

TR – Trace Register

Syntax

TR *register [,min_val,max_val]*

Description

This command sets the file register trace. If you don't supply any parameters, MPSIM traces any file register. If you specify *register*, it traces that register. If you also specify *min_val* and *max_val*, it performs the trace only if the value of the specified register lies between *min_val* and *max_val.*

Examples

MPSIM Command	Result
% **TR**	Traces all registers.
% **TR** ***W***	Traces the W register.
% **TR** ***W, 2, 7***	Traces the W register when its value falls between 2 and 7 (in the current radix).

Defaults

None.

Related Commands

TA, TC, TF

TY – Change View Screen

Syntax

TY *FileReg,radix,#digits*

Description

This command changes the formatting of the existing viewscreen. (If the designated signal isn't in the viewscreen, MPSIM gives a warning.)

The radix can be hexadecimal, octal, decimal or binary, designated by X, O, D or B, respectively.

#digits is the number of spaces to reserve for this variable at display time.

Examples

MPSIM Command	Result
% TY RTCC,B,1	RTCC I/O pin displays in binary, using one digit.

Defaults

None.

Related Commands

AD, NV, V

UR – Upload Registers

Syntax

UR *{ALL|RAM|SFR}[filename]*

Description

This command uploads the contents of registers into a DOS text file.

This command uploads file registers to "filename" (or to the default file name RAM.DAT if no file name is specified). The file will be in ASCII format and will consist of multiple lines in the following format:

```
BankNumber: FileRegisterNumber, Value
```

All values will be in hexadecimal radix and will begin in column one. For example, if ALL registers are to be uploaded to a file, the special-function registers would be print first, then all of the general-purpose registers would be printed:

```
;Special Function Registers
0, 0x00, 0x00
0, 0x01, 0x09
0, 0x02, 0xB1
```

Examples

MPSIM Command	Result
% UR ALL	Upload all registers to file RAM.DAT
% UR SFR t.out	Upload all special-function registers to the file "t.out"

Defaults

File name default is RAM.DAT.

Related Commands

LR, IR

V – View Screen

Syntax

V *FileReg[,radix[,#digits]]*

This command creates a new view screen that displays the named signal or register. Optionally, you may specify a radix different from the default and/or a number of digits.

V sets up the view screen. This means that the View command defines the variables (and respective formats) to constantly display on the screen. Once the view screen is set, it remains active until either an NV command or a V command sets up a new view screen. The format of this command is relatively simple. *FileReg* displays in radix mode with *#digits*. Radix defaults to hexadecimal and *#digits* to 1. The radix can be B (binary), O (octal), X (hexadecimal) or D (decimal).

Examples

MPSIM Command	Result
% V F3,b,8	A view screen element is created with the following format: F3: 00000000
% V RB0	A view screen element is created with the following format: RB0: 00

Defaults

The radix ordinarily defaults to hexadecimal, but you can change this default with the SR command. Digits defaults according to the table below:

Table 5.1 radix default widths

Radix	Digits
X	2
B	8
O	3
D	2

Related Commands

AD, DV, NV, TY

Verbose – Echo to Screen

Syntax

Verbose *[ON|OFF]*

Description

Prints a line to the screen (and to the optional trace file) when stimulus is injected into a pin. The command, without an argument, displays the current setting.

Examples

MPSIM Command	Result
% VERBOSE ON	Print to screen when stimuli are simulated.

Defaults

None.

W – Work Register Display / Modify

Syntax

W

Description

This command displays and/or modifies the contents of W register.

Examples

MPSIM Command	Result
% W	
W=44:	The value of W is 44 as the following message shows.
W=44:*00*	Change the value by entering a different value after the ':' prompt. The W register now has a value of 0.

Defaults

None.

Related Commands

None.

WP – Watchdog Timer Period

Syntax

WP *{1 | . . . | 128}*

Description

Sets watchdog timer time-out period in milliseconds. With no parameters, displays current setting.

Examples

MPSIM Command	Result
% **WP**	Display current period
% **WP 10**	WDT period set to 10 ms.

Defaults

"Normal" period for selected Microcontroller.

Related Commands

None

ZM – Zero the Program Memory

Syntax

ZM *address1,address2*

Description

This command zeroes the program memory from*address1* to *address2*. *address1* must less than *address2* and both must be valid program memory addresses.

Examples

MPSIM Command	Result
% **ZM** *0, 1F*	Program memory from 0 to 1F is zeroed.

Defaults

None.

Related Commands

None.

ZP – Zero the Patch Table

Syntax

ZP

Description

This command clears the patch table. Clears the patch table and resets it to no patches made. Any changes made to the object code are unaffected. Thus, the object code cannot be restored to the original.

Examples

MPSIM Command	Result
% ZP	Patch table cleared.

Defaults

None.

Related Commands

O, RA, RP

ZR – Zero the Registers

Syntax

ZR

Description

This command sets all of the file registers to zero. Care should be taken with this instruction since it will zero the lower 8 bits of F2 (PC). An RS command should follow the ZR command to ensure the PC is set the expected reset value.

Examples

MPSIM Command	Result
% ZR	All registers are zeroed.

Defaults

None.

Related Commands

DR, RS, SE

ZT – Zero the Elapsed Time Counter

Syntax

ZT

Description

This command zeros the elapsed time counter.

Examples

MPSIM Command	Result
% ZT	The elapsed time counter resets to zero.

Defaults

None.

Related Commands

RE, RS

Appendix A. Troubleshooting Guide

Introduction

This Appendix consists of the following sections:

- Solutions to common problems
- The three types of messages generated by MPSIM, grouped by severity and their possible causes and solutions. Messages have been divided into the following groups:
  - Informative Messages
  - Warning Messages
  - Error Messages

Solutions to Some Common Problems

Problem 1: I keep getting strange error messages like "stack underflow" or "Illegal Opcode" when single-stepping through or executing my code.

Solution 1: Check to make sure that the processor type you selected in MPSIM is the same as the processor type you selected when you assembled your code. This is especially important when simulating the members of the PIC16CXX or PIC17CXX family since the object code for them is different from the PIC16C5X, and the default processor type for the simulator is the PIC16C54.

Problem 2: When I am trying to step through my code, MPSIM seems to execute an instruction different from the one that is displayed in the command area.

Solution 2: Check to make sure that you loaded your code into the simulator in the same format that assembled it. For example, if you assembled your code and didn't specify an output format, your hex file will be in INHX8M format. If you then load your code into the simulator in INHX8S format, the simulator will behave strangely.

Problem 3: MPSIM does not perform indirect addressing correctly.

Solution 3: Check to make sure that you do not have your indirect addr register defined as the label "F0" in your source file. There is a symbol-table conflict when you define your label as such. Rename the "F0" label in your source file to "IND0" or any other label.

Problem 4: The W register does not update on my screen.

Solution 4: You have redefined W in your source file to be equal to zero, and MPSIM now treats W as file register 0. Change the label in your source file to "Wreg" or something similar.

Messages

Informative Messages

Address Break After

Cause: The breakpoint mode has been set to break after the instruction has been executed.

Break at Address

Cause: A breakpoint has been encountered and execution has stopped.

Break at Register

Cause: A break on register condition has been encountered and execution has stopped.

Interrupt at Address

Cause: Execution has stopped at the indicated address due to a user keyboard interrupt.

Listing File Loaded

Cause: MPSIM found and read filename.LST

No Symbols Defined!

Cause: The user has requested a list of all symbols when no symbols had been defined.

Object Code Written to Disk

Cause: MPSIM successfully dumped program memory to the named object file.

Original Source Restored

Cause: MPSIM has restored the source to its original form upon user request.

Out of Memory, Not all Source Lines Loaded

Cause: MPSIM has exhausted free memory while trying to load the listing file.

Processor Reset

Cause: MPSIM has reset the processor due to a user request.

Symbol Table Loaded

Cause: MPSIM has found and read filename.SYM.

Trace File is Closed

Cause: MPSIM has successfully closed the trace file.

Trace File is Open

Cause: MPSIM has successfully opened the trace file.

Verbose is OFF

Cause: Verbose mode is currently OFF, extended user messages will not be displayed.

Verbose is ON

Cause: Verbose mode is currently ON, extended user messages will be displayed.

Watchdog Timer Disabled

Cause: MPSIM will not respond to watchdog timer time-outs.

Watchdog Timer Enabled

Cause: MPSIM will respond to watchdog timer time-outs.

Warning Messages

Address2 < Address1

Cause: When entering a starting and ending address for a command, the ending address is greater than the ending address.

Cure: The starting address must be less than or equal to the ending address.

Arg X out of Range LABEL

Cause: You have entered a operand that is out of range of the specified instruction.

Cure: Review the instruction syntax and re-enter.

Attempt to Read Nonexistent File Register

Cause: Your object code has attempted to read a file register that does not exist in the PIC16/17 Microcontroller you have specified.

Cure: Set you PIC16/17 Microcontroller type accordingly.

Attempt to Write Nonexistent File Register

Cause: Your object code has attempted to read a file register that does not exist in the PIC16/17 Microcontroller you have specified.

Cure: Set your PIC16/17 Microcontroller type accordingly.

Bad Break Value

Cause: While defining a register breakpoint, you have specified a break value that is either unrecognized in the default radix or is out of range for the file register.

Cure: Ensure the value is valid in the current radix and not out of range of the file register.

Bad Count

Cause: You have entered a break count that is unrecognized in the current radix.

Cure: Ensure that the value is correct in the current radix.

Bad Cycle Length

Cause: You have entered a cycle length that is invalid or unrecognizable.

Cure: Re-enter the cycle length.

Bad End Address

Cause: You have entered an ending address that is out of memory bounds or unrecognizable in the current radix.

Cure: Ensure that the value is valid in the current radix and re-enter.

Bad Filename

Cause: The file name you entered was not recognizable as a DOS file name.

Cure: Ensure the file name conforms to DOS naming standards.

Bad Max. Value

Cause: This maximum value you entered is not recognizable in the current radix.

Cure: Ensure the value is valid in the current radix and re-enter.

Bad Min. Value

Cause: This minimum value you entered is not recognizable in the current radix.

Cure: Ensure the value is valid in the current radix and re-enter.

Bad Opcode

Cause: While attempting to search program memory for a specified opcode, the opcode you entered is unrecognizable in the current radix.

Cure: Ensure the opcode is valid in the current radix and re-enter.

Bad Option

Cause: The option you supplied to the V command was not valid.

Cure: Valid options are on and off. Use on of the valid options.

Bad Signal Value

Cause: While attempting to modify an I/O pin's value, you have entered a value that is unrecognizable in the current radix.

Cure: Re-enter the value ensuring it is valid in the current radix.

Bad Value

Cause: You have entered a value that is out of range of the file register or unrecognized in the current radix.

Cure: Ensure the value is valid in the current radix and in range for the file register.

Bad Width

Cause: The number you specified as the width of a view screen element was not recognized.

Cure: Ensure the width is a valid number in the current radix.

Can only Break on File Registers or Addresses

Cause: You have attempted to set a break point on an I/O pin.

Cure: Break points on I/O pins are disallowed.

Cannot Add Symbol to Symbol Table

Cause: Due to memory constraints, MPSIM cannot add the specified symbol to the symbol table.

Cure: Increase the amount of free memory before entering MPSIM.

Cannot Find Command File

Cause: MPSIM cannot find the command file you specified.

Cure: Ensure that the file is present in the path that you specified in the command.

Cannot Find Command File (MPSIM.jrn)

Cause: MPSIM cannot find the old journal file.

Cure: If MPSIM.jrn was not present in the current directory, this message is informational only. If the file is present, this may signal more serious errors with your disk.

Cannot Find List File

Cause: MPSIM cannot find the list file with the same name as the hex file plus the .LST extension.

Cure: Ensure you have a list file in the same directory as the hex file you specified.

Cannot Find Symbol File

Cause: MPSIM cannot find the symbol file with the same name as the hex file plus the .SYM extension.

Cure: Ensure you have a symbol file in the same directory as the hex file you specified.

Cannot Open Trace File

Cause: MPSIM cannot open the file you specified. This may be caused by any number of DOS errors.

Cure: Ensure that the file you specified doesn't exist and is read-only, or you have exhausted the number of DOS file handles.

Cannot Parse Filename

Cause: The file name you entered was not recognizable as a DOS file name.

Cure: Ensure the file name conforms to DOS naming standards.

Cannot Search for an IO Pin or Status Bit

Cause: You have attempted to search program memory for an instruction modifying an I/O pin or a status bit.

Cure: This operation is not supported.

Cannot Trace an IO Pin or Status Bit

Cause: You have attempted to set a trace on an I/O pin or Status Bit.

Cure: This operation is not supported.

File Symbol does not Match Page at PC=XXX

Cause: MPSIM has detected a page mismatch between the file symbol and the page select bits in the FSR.

Cure: This is a software error, your code needs to be fixed.

Invalid Filename

Cause: The file name you entered was not recognizable as a DOS file name.

Cure: Ensure the file name conforms to DOS naming standards.

Illegal Number of Arguments

Cause: You have entered the wrong number of arguments for the command.

Cure: Supply all required arguments for the command.

Illegal Radix

Cause: You have given a radix modifier that is not recognized.

Cure: Valid radix modifiers are X, D, O and B. Use one of the valid types.

Missing Instruction

Cause: You have told MPSIM to assemble an instruction, but did not supply the instruction.

Cure: Re-enter the command with the desired instruction.

No Breaks Found Involving

Cause: While trying to delete a register breakpoint, you have specified a file register that has no associated break point.

Cure: Ensure that a breakpoint for the specified file register has been defined via the DB command.

No Object Code Loaded

Cause: MPSIM cannot open the object file and as a result cannot load the object code.

Cure: Ensure that the file name you specified is present in the directory you specified.

Opcode can only be used in PIC16C55/57 Mode

Cause: MPSIM has tried to execute an instruction that is valid only for the PIC16C55 or PIC16C57. Most likely a TRIS 7 instruction.

Cure: Your Microcontroller type is not set properly. Refer to the P command.

Out of Memory

Cause: While defining a register breakpoint, MPSIM has exhausted free memory.

Cure: Increase the amount of free memory before entering MPSIM or rename the list file so that MPSIM cannot find it.

Stack Overflow

Cause: You have executed one too many RETLW instructions for the contents of the Microcontroller stack.

Cure: This is a software error, your code needs to be fixed.

Stack Underflow

Cause: You have executed one too many CALL instructions for the size of the Microcontroller stack.

Cure: This is a software error, your code needs to be fixed.

Start Address Exceeds End Address

Cause: When entering a starting and ending address for a command, the ending address is greater than the ending address.

Cure: The starting address must be less than or equal to the ending address.

Symbol Already Exists

Cause: You have attempted to define a symbol that already exists.

Cure: Use a different symbol name.

Too Many Arguments

Cause: You have entered too many arguments for the command.

Cure: Review the common syntax.

Unable to Open Object File

Cause: MPSIM cannot open the object file specified.

Cure: Ensure that the file is present in the directory you specified.

Undefined Symbol

Cause: You have attempted to delete a nonexistent symbol.

Cure: Ensure that the symbol is defined. Symbols are case sensitive. If you used the case insensitivity switch in the assembler, all symbols have been mapped to uppercase.

Uninitialized Memory Location Executed

Cause: MPSIM has attempted to execute a memory location that does not have any object code loaded.

Cure: Ensure that there is object code loaded and your program is not running amuck.

Unknown Break Mode

Cause: You have specified a break mode that is unrecognized to MPSIM.

Cure: Valid break modes are before and after. Use one of the valid break modes.

Unknown File Format

Cause: MPSIM has tried to read in an object file that is does not recognize.

Cure: Ensure that the file you specified is a valid object file in the format you specified.

Unknown Instruction XXX

Cause: You have told MPSIM to assemble an instruction which is not a valid instruction.

Cure: Re-enter the instruction in valid mnemonics.

Unknown Opcode XXX

Cause: There is an invalid opcode in your object file.

Cure: Ensure that you have loaded your object file in the correct format. Default is INHX16.

Unknown Operator

Cause: While defining a register breakpoint, you have used an unrecognized logical operator.

Cure: Valid operators are <, >, <=, >=, =, !=. Use one of the valid operators.

Unknown Radix

Cause: You have attempted to modify the default radix to a value that is unrecognized by MPSIM.

Cure: Valid radix values are X, D and O. Use one of the valid values.

Unknown Symbol Type

Cause: While attempting to define a new symbol, you have entered a symbol type that is unrecognized by MPSIM.

Cure: Valid symbol types are F, L, K and B. Use one of the valid symbol types.

Use SE Command to Modify IO Pins

Cause: You have attempted to use the F command to modify an I/O pin.

Cure: Use the SE command.

Value Out of Range

Cause: You have specified a value that is out of range or unrecognized in the current radix.

Cure: Ensure that the value is valid in the current radix and valid for the current operation.

View Item not Found

Cause: You have attempted to delete or modify a nonexistent view screen element.

Cure: Ensure that the element is present on the view screen. View screen elements are case-sensitive.

ViewScreen is Full

Cause: You have attempted to add an element to the view screen when there is no more room on the screen.

Cure: Since the view screen is static in this version, there is no work-around.

WDT Time-out

Cause: The watchdog timer has timed out.

Cure: Ensure the settings for the WDT are correct and your software resets the WDT appropriately.

XXX is not an IO Pin

Cause: You have tried to use the SE command to modify a label that is not an I/O pin.

Cure: Use the F command to modify file registers, status bits and the stack.

Appendix A. Troubleshooting Guide

Error Messages

Bad Stimulus (Line X)

Cause: MPSIM has found a stimulus value other that zero or one.

Cure: All pin stimuli must be either zero or one.

Cannot Delete Old Journal File

Cause: The file MPSIM.JRN has been read protected.

Cure: If you intended for the file to be read protected then do not worry about this error otherwise read enable the file.

Cannot Find Heading Line in Stimulus File

Cause: MPSIM cannot find the heading line in the stimulus file.

Cure: Ensure that there is a line in the file which begins with STEP.

Cannot Map Stimulus, Symbol Conflict XXX

Cause: MPSIM has encountered two column headings that are identical.

Cure: Ensure your column headings are correct.

Cannot Open File for Input XXX

Cause: MPSIM cannot open the specified file for reading.

Cure: Either the file does not exist or the file is read-only.

Cannot Open Journal File

Cause: MPSIM cannot open the old journal file.

Cure: The file MPSIM.JRN has been read protected, change the DOS attribute.

Cannot Update Journal File

Cause: MPSIM cannot update the journal file with the new commands for this session.

Cure: Either the old MPSIM.JRN cannot be deleted or the new journal file does not exist. Contact your local FAE.

Duplicate Symbol in Symbol File

Cause: MPSIM has encountered a symbol in the symbol file that has already been defined.

Cure: Delete the duplicate reference. If MPSIM finds this error it will not continue to read the symbol file.

First Heading in Stimulus File MUST be STEP

Cause: The line that MPSIM interpreted as the heading line did not begin with STEP.

Cure: Make sure all comment lines begin with '!' and the heading line begins with STEP.

Out of Memory, Cannot Create Event Calendar

Cause: MPSIM exhausted free memory while trying to create the event calendar.

Cure: Increase the amount of free memory before invoking MPSIM.

Out of Memory, Cannot Create Event (Line X)

Cause: MPSIM exhausted free memory while trying to create an event.

Cure: Increase the amount of free memory before invoking MPSIM.

Out of Memory During Build of Break

Cause: MPSIM exhausted free memory while trying to define a file register breakpoint.

Cure: Increase the amount of free memory before invoking MPSIM.

Stimulus Data does not Match Headings (Line X)

Cause: MPSIM has found a line that has too few or too many data points to match the column headings.

Cure: Ensure each data line has one data point for each column heading.

Symbol File does not Match Hex File

Cause: You have tried to load a symbol file that was not generated for the current hex file.

Cure: If you intended to load the symbol file, the embedded file name must match the file name of the symbol file.

Symbol File is Corrupt

Cause: MPSIM has encountered some unexpected formatting in the symbol file.

Cure: Regenerate the symbol file.

Symbol File Sync Error

Cause: MPSIM has gotten lost while trying to parse the symbol file. Most likely the symbol file is corrupt.

Cure: Regenerate the symbol file.

Too Many Headings in Stimulus File (MAX=40)

Cause: The stimulus file has a limit of 40 headings, enough for each I/O pin.

Cure: If there is a need for more headings, contact your local FAE.

Unknown Command

Cause: MPSIM does not recognize the command you entered.

Cure: Refer to the command summary for valid commands.

Unexpected EOF in Stimulus File

Cause: While reading the stimulus file, MPSIM encountered a line that did not have the proper number of data points.

Cure: Ensure that all data lines have the correct number of data points.

Unknown File Register X

Cause: MPSIM does not recognize the file register as an argument to the instruction.

Cure: Re-enter the mnemonic with a valid file register.

Unknown Option X

Cause: MPSIM does not recognize the command line option X.

Cure: Refer to the section on command line arguments.

Unknown Opcode (X)

Cause: MPSIM tried to execute an opcode that is not a valid opcode.

Cure: Ensure you loaded the object file in the correct format. INHX16 and INHX8M have different byte orders.

Appendix B. Sample File Listings

MPSIM.INI

```
SR X
ZP
ZR
ZT
RE
V W,X,2
AD F1,X,2
AD F2,X,3
AD F3,X,2
AD F4,X,2
AD F5,X,2
AD F6,X,2
AD F7,X,2
RS
```

PIC16C5X.INC

```
LIST
; P16C5X.INC  Standard Header File, Version 2.0     Microchip Technology, Inc.
NOLIST
; This header file defines configurations, registers, and other useful bits of
; information for the 16C5X microcontrollers.  These names are taken to match
; the data sheets as closely as possible.  The microcontrollers included
; in this file are:
;    16C54
;    16C54A
;    16C55
;    16C56
;    16C57
;    16C58A
; There is one group of symbols that is valid for all microcontrollers.
; Each microcontroller in this family also has its own section of special
; symbols.  Note that the processor must be selected before this file is
; included.  The processor may be selected the following ways:
;       1. Command line switch:
;               C:\ MPASM MYFILE.ASM /P16C54A
;       2. LIST directive in the source file
;               LIST   P=16C54A
;       3. Processor Type entry in the MPASM full-screen interface
;==========================================================================
;
;       Generic Definitions
;
;==========================================================================
   W                            EQU     H'0000'
   F                            EQU     H'0001'
;----- Register Files ------------------------------------------------------
   CBLOCK               H'0000'
      INDF
      TMR0
      PCL
      STATUS
      FSR
      PORTA
      PORTB
```

```
   ENDC
;----- STATUS Bits --------------------------------------------------------
   PA2                          EQU     H'0007'
   PA1                          EQU     H'0006'
   PA0                          EQU     H'0005'
   NOT_TO                       EQU     H'0004'
   NOT_PD                       EQU     H'0003'
   Z                            EQU     H'0002'
   DC                           EQU     H'0001'
   C                            EQU     H'0000'
;----- OPTION Bits --------------------------------------------------------
   T0CS                         EQU     H'0005'
   T0SE                         EQU     H'0004'
   PSA                          EQU     H'0003'
   PS2                          EQU     H'0002'
   PS1                          EQU     H'0001'
   PS0                          EQU     H'0000'
;=========================================================================
;
;       Processor-dependent Definitions
;
;=========================================================================
 IFDEF __16C54
   #define __CONFIG_0
 ENDIF
 IFDEF __16C54A
   #define __CONFIG_0
 ENDIF
 IFDEF __16C55
   ; Register Files
     PORTC                      EQU     H'0007'
   #define __CONFIG_0
 ENDIF
 IFDEF __16C56
   #define __CONFIG_0
 ENDIF
 IFDEF __16C57
   ; Register Files
     PORTC                      EQU     H'0007'
   #define __CONFIG_0
 ENDIF
 IFDEF __16C58A
   #define __CONFIG_1
 ENDIF


;=========================================================================
;
;       Configuration Bits
;
;=========================================================================
 IFDEF __CONFIG_0
   _CP_ON                       EQU     H'0FF7'
   _CP_OFF                      EQU     H'0FFF'
   _WDT_ON                      EQU     H'0FFF'
   _WDT_OFF                     EQU     H'0FFB'
   _LP_OSC                      EQU     H'0FFC'
   _XT_OSC                      EQU     H'0FFD'
   _HS_OSC                      EQU     H'0FFE'
   _RC_OSC                      EQU     H'0FFF'
   ;#undefine __CONFIG_0
 ENDIF

 IFDEF __CONFIG_1
   _CP_ON                       EQU     H'0007'
   _CP_OFF                      EQU     H'0FFF'
   _WDT_ON                      EQU     H'0FFF'
```

```
   _WDT_OFF                        EQU     H'0FFB'
   _LP_OSC                         EQU     H'0FFC'
   _XT_OSC                         EQU     H'0FFD'
   _HS_OSC                         EQU     H'0FFE'
   _RC_OSC                         EQU     H'0FFF'
   ;#undefine __CONFIG_1
 ENDIF
LIST
```

PIC16CXX.INC

```
LIST
; P16CXX.INC  Standard Header File, Version 2.0     Microchip Technology, Inc.
NOLIST
; This header file defines configurations, registers, and other useful bits of
; information for the 16CXX microcontrollers.  These names are taken to match
; the data sheets as closely as possible.  The microcontrollers included
; in this file are:
;    16C61
;    16C620
;    16C621
;    16C622
;    16C64
;    16C65
;    16C71
;    16C73
;    16C74
;    16C83
;    16C84
;    16C84A
; There is one group of defines that is valid for all microcontrollers.
; Each microcontroller in this family also has its own section of special
; defines.  Note that the processor must be selected before this file is
; included.  The processor may be selected the following ways:
;       1. Command line switch:
;               C:\ MPASM MYFILE.ASM /P16C71
;       2. LIST directive in the source file
;               LIST   P=16C71
;       3. Processor Type entry in the MPASM full-screen interface
;==========================================================================
;
;       Generic Definitions
;
;==========================================================================
   W                               EQU     H'0000'
   F                               EQU     H'0001'
;----- Register Files------------------------------------------------------
   INDF                            EQU     H'0000'
   TMR0                            EQU     H'0001'
   PCL                             EQU     H'0002'
   STATUS                          EQU     H'0003'
   FSR                             EQU     H'0004'
   PORTA                           EQU     H'0005'
   PORTB                           EQU     H'0006'
   PCLATH                          EQU     H'000A'
   INTCON                          EQU     H'000B'
   OPTION_REG                      EQU     H'0081'
   TRISA                           EQU     H'0085'
   TRISB                           EQU     H'0086'
;----- INTCON Bits (except ADC/Periph) -----------------------------------
   GIE                             EQU     H'0007'
   T0IE                            EQU     H'0005'
   INTE                            EQU     H'0004'
   RBIE                            EQU     H'0003'
   T0IF                            EQU     H'0002'
```

```
   INTF                          EQU     H'0001'
   RBIF                          EQU     H'0000'
;----- OPTION Bits --------------------------------------------------------
   NOT_RBPU                      EQU     H'0007'
   INTEDG                        EQU     H'0006'
   T0CS                          EQU     H'0005'
   T0SE                          EQU     H'0004'
   PSA                           EQU     H'0003'
   PS2                           EQU     H'0002'
   PS1                           EQU     H'0001'
   PS0                           EQU     H'0000'
;----- STATUS Bits --------------------------------------------------------
   IRP                           EQU     H'0007'
   RP1                           EQU     H'0006'
   RP0                           EQU     H'0005'
   NOT_TO                        EQU     H'0004'
   NOT_PD                        EQU     H'0003'
   Z                             EQU     H'0002'
   DC                            EQU     H'0001'
   C                             EQU     H'0000'
;==========================================================================
;
;       Processor-dependent Definitions
;
;==========================================================================
 IFDEF __16C61
   #define __CONFIG_0
 ENDIF
 IFDEF __16C620
   ;----- Register Files ------------------------------------------------
      PIR1                       EQU     H'000C'
      CMCON                      EQU     H'001F'
      PIE1                       EQU     H'008C'
      PCON                       EQU     H'008E'
      VRCON                      EQU     H'009F'
   #define __CONFIG_6
 ENDIF
 IFDEF __16C621
   ;----- Register Files ------------------------------------------------
      PIR1                       EQU     H'000C'
      CMCON                      EQU     H'001F'
      PIE1                       EQU     H'008C'
      PCON                       EQU     H'008E'
      VRCON                      EQU     H'009F'
   #define __CONFIG_4
 ENDIF
 IFDEF __16C622
   ;----- Register Files ------------------------------------------------
      PIR1                       EQU     H'000C'
      CMCON                      EQU     H'001F'
      PIE1                       EQU     H'008C'
      PCON                       EQU     H'008E'
      VRCON                      EQU     H'009F'
   #define __CONFIG_5
 ENDIF
 IFDEF __16C63
   ;----- Register Files ------------------------------------------------
      PORTC                      EQU     H'0007'
      PIR1                       EQU     H'000C'
      TMR1L                      EQU     H'000E'
      TMR1H                      EQU     H'000F'
      T1CON                      EQU     H'0010'
      TMR2                       EQU     H'0011'
      T2CON                      EQU     H'0012'
      SSPBUF                     EQU     H'0013'
      SSPCON                     EQU     H'0014'
      CCPR1L                     EQU     H'0015'
```

```
    CCPR1H                  EQU     H'0016'
    CCP1CON                 EQU     H'0017'
    TRISC                   EQU     H'0087'
    PIE1                    EQU     H'008C'
    PCON                    EQU     H'008E'
    PR2                     EQU     H'0092'
    SSPADD                  EQU     H'0093'
    SSPSTAT                 EQU     H'0094'
  #define __CONFIG_2
ENDIF
IFDEF __16C64
  ;----- Register Files --------------------------------------------------
    PORTC                   EQU     H'0007'
    PORTD                   EQU     H'0008'
    PORTE                   EQU     H'0009'
    PIR1                    EQU     H'000C'
    TMR1L                   EQU     H'000E'
    TMR1H                   EQU     H'000F'
    T1CON                   EQU     H'0010'
    TMR2                    EQU     H'0011'
    T2CON                   EQU     H'0012'
    SSPBUF                  EQU     H'0013'
    SSPCON                  EQU     H'0014'
    CCPR1L                  EQU     H'0015'
    CCPR1H                  EQU     H'0016'
    CCP1CON                 EQU     H'0017'
    TRISC                   EQU     H'0087'
    TRISD                   EQU     H'0088'
    TRISE                   EQU     H'0089'
    PIE1                    EQU     H'008C'
    PCON                    EQU     H'008E'
    PR2                     EQU     H'0092'
    SSPADD                  EQU     H'0093'
    SSPSTAT                 EQU     H'0094'
  #define __CONFIG_2
ENDIF
IFDEF __16C65
  ;----- Register Files --------------------------------------------------
    PORTC                   EQU     H'0007'
    PORTD                   EQU     H'0008'
    PORTE                   EQU     H'0009'
    PIR1                    EQU     H'000C'
    PIR2                    EQU     H'000D'
    TMR1L                   EQU     H'000E'
    TMR1H                   EQU     H'000F'
    T1CON                   EQU     H'0010'
    TMR2                    EQU     H'0011'
    T2CON                   EQU     H'0012'
    SSPBUF                  EQU     H'0013'
    SSPCON                  EQU     H'0014'
    CCPR1L                  EQU     H'0015'
    CCPR1H                  EQU     H'0016'
    CCP1CON                 EQU     H'0017'
    RCSTA                   EQU     H'0018'
    TXREG                   EQU     H'0019'
    RCREG                   EQU     H'001A'
    CCPR2L                  EQU     H'001B'
    CCPR2H                  EQU     H'001C'
    CCP2CON                 EQU     H'001D'
    TRISC                   EQU     H'0087'
    TRISD                   EQU     H'0088'
    TRISE                   EQU     H'0089'
    PIE1                    EQU     H'008C'
    PIE2                    EQU     H'008D'
    PCON                    EQU     H'008E'
    PR2                     EQU     H'0092'
    SSPADD                  EQU     H'0093'
```

```
    SSPSTAT                 EQU     H'0094'
    TXSTA                   EQU     H'0098'
    SPBRG                   EQU     H'0099'
  #define __CONFIG_2
ENDIF
IFDEF __16C71
  #define __ADC_CONFIG_0
  #define __CONFIG_0
ENDIF
IFDEF __16C73
  ;----- Register Files --------------------------------------------------
    PORTC                   EQU     H'0007'
    PIR1                    EQU     H'000C'
    PIR2                    EQU     H'000D'
    TMR1L                   EQU     H'000E'
    TMR1H                   EQU     H'000F'
    T1CON                   EQU     H'0010'
    TMR2                    EQU     H'0011'
    T2CON                   EQU     H'0012'
    SSPBUF                  EQU     H'0013'
    SSPCON                  EQU     H'0014'
    CCPR1L                  EQU     H'0015'
    CCPR1H                  EQU     H'0016'
    CCP1CON                 EQU     H'0017'
    RCSTA                   EQU     H'0018'
    TXREG                   EQU     H'0019'
    RCREG                   EQU     H'001A'
    CCPR2L                  EQU     H'001B'
    CCPR2H                  EQU     H'001C'
    CCP2CON                 EQU     H'001D'
    TRISC                   EQU     H'0087'
    PIE1                    EQU     H'008C'
    PIE2                    EQU     H'008D'
    PCON                    EQU     H'008E'
    PR2                     EQU     H'0092'
    SSPADD                  EQU     H'0093'
    SSPSTAT                 EQU     H'0094'
    TXSTA                   EQU     H'0098'
    SPBRG                   EQU     H'0099'
  #define __ADC_CONFIG_1
  #define __CONFIG_2
ENDIF
IFDEF __16C74
  ;----- Register Files --------------------------------------------------
    PORTC                   EQU     H'0007'
    PORTD                   EQU     H'0008'
    PORTE                   EQU     H'0009'
    PIR1                    EQU     H'000C'
    PIR2                    EQU     H'000D'
    TMR1L                   EQU     H'000E'
    TMR1H                   EQU     H'000F'
    T1CON                   EQU     H'0010'
    TMR2                    EQU     H'0011'
    T2CON                   EQU     H'0012'
    SSPBUF                  EQU     H'0013'
    SSPCON                  EQU     H'0014'
    CCPR1L                  EQU     H'0015'
    CCPR1H                  EQU     H'0016'
    CCP1CON                 EQU     H'0017'
    RCSTA                   EQU     H'0018'
    TXREG                   EQU     H'0019'
    RCREG                   EQU     H'001A'
    CCPR2L                  EQU     H'001B'
    CCPR2H                  EQU     H'001C'
    CCP2CON                 EQU     H'001D'
    TRISC                   EQU     H'0087'
    TRISD                   EQU     H'0088'
```

```
    TRISE                   EQU     H'0089'
    PIE1                    EQU     H'008C'
    PIE2                    EQU     H'008D'
    PCON                    EQU     H'008E'
    PR2                     EQU     H'0092'
    SSPADD                  EQU     H'0093'
    SSPSTAT                 EQU     H'0094'
    TXSTA                   EQU     H'0098'
    SPBRG                   EQU     H'0099'
  #define __ADC_CONFIG_1
  #define __CONFIG_2
ENDIF
IFDEF __16C83
  ;----- Register Files --------------------------------------------------
    EEDATA                  EQU     H'0008'
    EEADR                   EQU     H'0009'
    EECON1                  EQU     H'0088'
    EECON2                  EQU     H'0089'
  #define __CONFIG_3
ENDIF
IFDEF __16C84
  ;----- Register Files --------------------------------------------------
    EEDATA                  EQU     H'0008'
    EEADR                   EQU     H'0009'
    EECON1                  EQU     H'0088'
    EECON2                  EQU     H'0089'
  #define __CONFIG_0
ENDIF
IFDEF __16C84A
  ;----- Register Files --------------------------------------------------
    EEDATA                  EQU     H'0008'
    EEADR                   EQU     H'0009'
    EECON1                  EQU     H'0088'
    EECON2                  EQU     H'0089'
  #define __CONFIG_3
ENDIF
;==========================================================================
;
;       Configuration Bits
;
;==========================================================================
IFDEF __CONFIG_0
  _CP_ON                    EQU     H'3FEF'
  _CP_OFF                   EQU     H'3FFF'
  _PWRTE_ON                 EQU     H'3FFF'
  _PWRTE_OFF                EQU     H'3FF7'
  _WDT_ON                   EQU     H'3FFF'
  _WDT_OFF                  EQU     H'3FFB'
  _LP_OSC                   EQU     H'3FFC'
  _XT_OSC                   EQU     H'3FFD'
  _HS_OSC                   EQU     H'3FFE'
  _RC_OSC                   EQU     H'3FFF'
  ;#undefine __CONFIG_0
ENDIF
IFDEF __CONFIG_1
  _BODEN_ON                 EQU     H'3FFF'
  _BODEN_OFF                EQU     H'3FBF'
  _CP_ON                    EQU     H'004F'
  _CP_OFF                   EQU     H'3FFF'
  _PWRTE_ON                 EQU     H'3FFF'
  _PWRTE_OFF                EQU     H'3FF7'
  _WDT_ON                   EQU     H'3FFF'
  _WDT_OFF                  EQU     H'3FFB'
  _LP_OSC                   EQU     H'3FFC'
  _XT_OSC                   EQU     H'3FFD'
  _HS_OSC                   EQU     H'3FFE'
  _RC_OSC                   EQU     H'3FFF'
```

```
  ;#undefine __CONFIG_1
ENDIF
IFDEF __CONFIG_2
  _CP_ALL                       EQU     H'3FCF'
  _CP_75                        EQU     H'3FDF'
  _CP_50                        EQU     H'3FEF'
  _CP_OFF                       EQU     H'3FFF'
  _PWRTE_ON                     EQU     H'3FFF'
  _PWRTE_OFF                    EQU     H'3FF7'
  _WDT_ON                       EQU     H'3FFF'
  _WDT_OFF                      EQU     H'3FFB'
  _LP_OSC                       EQU     H'3FFC'
  _XT_OSC                       EQU     H'3FFD'
  _HS_OSC                       EQU     H'3FFE'
  _RC_OSC                       EQU     H'3FFF'
  ;#undefine __CONFIG_2
ENDIF
IFDEF __CONFIG_3
  _CP_ON                        EQU     H'000F'
  _CP_OFF                       EQU     H'3FFF'
  _PWRTE_ON                     EQU     H'3FFF'
  _PWRTE_OFF                    EQU     H'3FF7'
  _WDT_ON                       EQU     H'3FFF'
  _WDT_OFF                      EQU     H'3FFB'
  _LP_OSC                       EQU     H'3FFC'
  _XT_OSC                       EQU     H'3FFD'
  _HS_OSC                       EQU     H'3FFE'
  _RC_OSC                       EQU     H'3FFF'
  ;#undefine __CONFIG_3
ENDIF
IFDEF __CONFIG_4
  _BODEN_ON                     EQU     H'3FFF'
  _BODEN_OFF                    EQU     H'3FBF'
  _CP_ALL                       EQU     H'00CF'
  _CP_50                        EQU     H'15DF'
  _CP_OFF                       EQU     H'3FFF'
  _PWRTE_ON                     EQU     H'3FFF'
  _PWRTE_OFF                    EQU     H'3FF7'
  _WDT_ON                       EQU     H'3FFF'
  _WDT_OFF                      EQU     H'3FFB'
  _LP_OSC                       EQU     H'3FFC'
  _XT_OSC                       EQU     H'3FFD'
  _HS_OSC                       EQU     H'3FFE'
  _RC_OSC                       EQU     H'3FFF'
  ;#undefine __CONFIG_4
ENDIF
IFDEF __CONFIG_5
  _BODEN_ON                     EQU     H'3FFF'
  _BODEN_OFF                    EQU     H'3FBF'
  _CP_ALL                       EQU     H'00CF'
  _CP_75                        EQU     H'15DF'
  _CP_50                        EQU     H'2AEF'
  _CP_OFF                       EQU     H'3FFF'
  _PWRTE_ON                     EQU     H'3FFF'
  _PWRTE_OFF                    EQU     H'3FF7'
  _WDT_ON                       EQU     H'3FFF'
  _WDT_OFF                      EQU     H'3FFB'
  _LP_OSC                       EQU     H'3FFC'
  _XT_OSC                       EQU     H'3FFD'
  _HS_OSC                       EQU     H'3FFE'
  _RC_OSC                       EQU     H'3FFF'
  ;#undefine __CONFIG_5
ENDIF
IFDEF __CONFIG_6
  _BODEN_ON                     EQU     H'3FFF'
  _BODEN_OFF                    EQU     H'3FBF'
  _CP_ON                        EQU     H'00CF'
```

```
  _CP_OFF                       EQU     H'3FFF'
  _PWRTE_ON                     EQU     H'3FFF'
  _PWRTE_OFF                    EQU     H'3FF7'
  _WDT_ON                       EQU     H'3FFF'
  _WDT_OFF                      EQU     H'3FFB'
  _LP_OSC                       EQU     H'3FFC'
  _XT_OSC                       EQU     H'3FFD'
  _HS_OSC                       EQU     H'3FFE'
  _RC_OSC                       EQU     H'3FFF'
  ;#undefine __CONFIG_6
ENDIF
;==========================================================================
;
;       More Bit Definitions
;
;==========================================================================
IFDEF __ADC_CONFIG_0
  ;---- Register Files ------------------------------------------------
     ADCON0                     EQU     H'0008'
     ADRES                      EQU     H'0009'
     ADCON1                     EQU     H'0088'
  ;---- Finish INTCON Definition --------------------------------------
     ADIE                       EQU     H'0006'
  ;----- ADCON0 Bits ----------------------------------------------------
     ADCS1                      EQU     H'0007'
     ADCS0                      EQU     H'0006'
     CHS1                       EQU     H'0004'
     CHS0                       EQU     H'0003'
     GO                         EQU     H'0002'
     NOT_DONE                   EQU     H'0002'
     GO_DONE                    EQU     H'0002'
     ADIF                       EQU     H'0001'
     ADON                       EQU     H'0000'
  ;----- ADCON1 Bits ----------------------------------------------------
     PCFG1                      EQU     H'0001'
     PCFG0                      EQU     H'0000'
  ;#undefine __ADC_CONFIG_0
ELSE
  ;---- Finish INTCON Definition --------------------------------------
     PEIE                       EQU     H'0006'
ENDIF
IFDEF __ADC_CONFIG_1
  ;----- Register Files ------------------------------------------------
     ADRES                      EQU     H'001E'
     ADCON0                     EQU     H'001F'
     ADCON1                     EQU     H'009F'
  ;----- ADCON0 Bits ----------------------------------------------------
     ADCS1                      EQU     H'0007'
     ADCS0                      EQU     H'0006'
     CHS2                       EQU     H'0005'
     CHS1                       EQU     H'0004'
     CHS0                       EQU     H'0003'
     GO                         EQU     H'0002'
     NOT_DONE                   EQU     H'0002'
     GO_DONE                    EQU     H'0002'
     ADON                       EQU     H'0000'
  ;----- ADCON1 Bits ----------------------------------------------------
     PCFG2                      EQU     H'0002'
     PCFG1                      EQU     H'0001'
     PCFG0                      EQU     H'0000'
  ;----- PIE1 and PIR1 ADC Bits ----------------------------------------
     ADIE                       EQU     H'0006'
     ADIF                       EQU     H'0006'
  ;#undefine __ADC_CONFIG_1
ENDIF
IFDEF CCP1CON
  CCP1X                         EQU     H'0005'
```

```
  CCP1Y                         EQU     H'0004'
  CCP1M3                        EQU     H'0003'
  CCP1M2                        EQU     H'0002'
  CCP1M1                        EQU     H'0001'
  CCP1M0                        EQU     H'0000'
ENDIF
IFDEF CCP2CON
  CCP2X                         EQU     H'0005'
  CCP2Y                         EQU     H'0004'
  CCP2M3                        EQU     H'0003'
  CCP2M2                        EQU     H'0002'
  CCP2M1                        EQU     H'0001'
  CCP2M0                        EQU     H'0000'
ENDIF
IFDEF CMCON
  C2OUT                         EQU     H'0007'
  C1OUT                         EQU     H'0006'
  CIS                           EQU     H'0003'
  CM2                           EQU     H'0002'
  CM1                           EQU     H'0001'
  CM0                           EQU     H'0000'
  ;----- PIE1 and PIR1 ADC Bits and Short Cuts --------------------------
     CMIE                       EQU     H'0006'
     CMIF                       EQU     H'0006'
ENDIF
IFDEF EECON1
  EEIF                          EQU     H'0004'
  WRERR                         EQU     H'0003'
  WREN                          EQU     H'0002'
  WR                            EQU     H'0001'
  RD                            EQU     H'0000'
ENDIF
IFDEF PCON
  NOT_POR                       EQU     H'0001'
  NOT_BO                        EQU     H'0000'
ENDIF
IFDEF PIE1
  PSPIE                         EQU     H'0007'
  SSPIE                         EQU     H'0003'
  CCP1IE                        EQU     H'0002'
  TMR2IE                        EQU     H'0001'
  TMR1IE                        EQU     H'0000'
ENDIF
IFDEF PIR1
  PSPIF                         EQU     H'0007'
  SSPIF                         EQU     H'0003'
  CCP1IF                        EQU     H'0002'
  TMR2IF                        EQU     H'0001'
  TMR1IF                        EQU     H'0000'
ENDIF
IFDEF PIE2                                          ; Assumes PIE2 and PIR2
  CCP2IE                        EQU     H'0000'
  CCP2IF                        EQU     H'0000'
ENDIF
IFDEF RCSTA
  SPEN                          EQU     H'0007'
  RC9                           EQU     H'0006'
  NOT_RC8                       EQU     H'0006'
  RC8_9                         EQU     H'0006'
  SREN                          EQU     H'0005'
  CREN                          EQU     H'0004'
  FERR                          EQU     H'0002'
  OERR                          EQU     H'0001'
  RCD8                          EQU     H'0000'
  ;----- PIE1 and PIR1 RC Bits and Short Cuts ---------------------------
     RCIE                       EQU     H'0005'
     RBFL                       EQU     H'0005'
```

```
ENDIF
IFDEF SSPCON
  WCOL                    EQU     H'0007'
  SSPOV                   EQU     H'0006'
  SSPEN                   EQU     H'0005'
  CKP                     EQU     H'0004'
  SSPM3                   EQU     H'0003'
  SSPM2                   EQU     H'0002'
  SSPM1                   EQU     H'0001'
  SSPM0                   EQU     H'0000'
ENDIF
IFDEF SSPSTAT
  D                       EQU     H'0005'
  I2C_DATA                EQU     H'0005'
  NOT_A                   EQU     H'0005'
  NOT_ADDRESS             EQU     H'0005'
  D_A                     EQU     H'0005'
  DATA_ADDRESS            EQU     H'0005'
  P                       EQU     H'0004'
  I2C_STOP                EQU     H'0004'
  S                       EQU     H'0003'
  I2C_START               EQU     H'0003'
  R                       EQU     H'0002'
  I2C_READ                EQU     H'0002'
  NOT_W                   EQU     H'0002'
  NOT_WRITE               EQU     H'0002'
  R_W                     EQU     H'0002'
  READ_WRITE              EQU     H'0002'
  UA                      EQU     H'0001'
  BF                      EQU     H'0000'
ENDIF
IFDEF T1CON
  T1CKPS1                 EQU     H'0005'
  T1CKPS0                 EQU     H'0004'
  T1OSCEN                 EQU     H'0003'
  T1INSYNC                EQU     H'0002'
  TMR1CS                  EQU     H'0001'
  TMR1ON                  EQU     H'0000'
ENDIF
IFDEF T2CON
  TOUTPS3                 EQU     H'0006'
  TOUTPS2                 EQU     H'0005'
  TOUTPS1                 EQU     H'0004'
  TOUTPS0                 EQU     H'0003'
  TMR2ON                  EQU     H'0002'
  T2CKPS1                 EQU     H'0001'
  T2CKPS0                 EQU     H'0000'
ENDIF
IFDEF TRISE
  IBF                     EQU     H'0007'
  OBF                     EQU     H'0006'
  IBOV                    EQU     H'0005'
  PSPMODE                 EQU     H'0004'
  TRISE2                  EQU     H'0002'
  TRISE1                  EQU     H'0001'
  TRISE0                  EQU     H'0000'
ENDIF
IFDEF TXSTA
  CSRC                    EQU     H'0007'
  TX9                     EQU     H'0006'
  NOT_TX8                 EQU     H'0006'
  TX8_9                   EQU     H'0006'
  TXEN                    EQU     H'0005'
  SYNC                    EQU     H'0004'
  BRGH                    EQU     H'0002'
  TRMT                    EQU     H'0001'
  TXD8                    EQU     H'0000'
```

```
   ;----- PIE1 and PIR1 TX Bits and Short Cuts --------------------------
      TXIE                        EQU     H'0004'
      TXIF                        EQU     H'0004'
 ENDIF
 IFDEF VRCON
   VREN                          EQU     H'0007'
   VROE                          EQU     H'0006'
   VRR                           EQU     H'0005'
   VR3                           EQU     H'0003'
   VR2                           EQU     H'0002'
   VR1                           EQU     H'0001'
   VR0                           EQU     H'0000'
 ENDIF
LIST
```

PIC17CXX.INC

```
LIST
; P17CXX.INC  Standard Header File, Version 2.0     Microchip Technology, Inc.
NOLIST
; This header file defines configurations, registers, and other useful bits of
; information for the 17CXX microcontrollers.  These names are taken to match
; the data sheets as closely as possible.  The microcontrollers included
; in this file are:
; 17C42
; 17C43
; 17C44
; There is one group of defines that is valid for all microcontrollers.
; Each microcontroller in this family also has its own section of special
; defines.  Note that the processor must be selected before this file is
; included.  The processor may be selected the following ways:
;       1. Command line switch:
;               C:\ MPASM MYFILE.ASM /P17C42
;       2. LIST directive in the source file
;               LIST   P=17C42
;       3. Processor Type entry in the MPASM full-screen interface

;==========================================================================
;
;       Generic Definitions
;
;==========================================================================
   W                             EQU     H'0000'
   F                             EQU     H'0001'
   CBLOCK              H'0000'
      BANK0
      BANK1
      BANK2
      BANK3
   ENDC
;----- Register Files ------------------------------------------------------
   CBLOCK              H'0000'                 ; Bank 0
      INDF0
      FSR0
      PCL
      PCLATH
      ALUSTA
      T0STA
      CPUSTA
      INTSTA
      INDF1
      FSR1
      WREG
      TMR0L
      TMR0H
```

```
        TBLPTRL
        TBLPTRH
        BSR
        PORTA
        DDRB
        PORTB
        RCSTA
        RCREG
        TXSTA
        TXREG
        SPBRG
    ENDC
    CBLOCK              H'0010'             ; Bank 1
        DDRC
        PORTC
        DDRD
        PORTD
        DDRE
        PORTE
        PIR
        PIE
    ENDC
    CBLOCK              H'0010'             ; Bank 2
        TMR1
        TMR2
        TMR3L
        TMR3H
        PR1
        PR2
        PR3L
        PR3H
    ENDC
    CBLOCK              H'0016'             ; Bank 2 - alternate
        CAL1L
        CAL1H
    ENDC
    CBLOCK              H'0010'             ; Bank 3
        PW1DCL
        PW2DCL
        PW1DCH
        PW2DCH
        CA2L
        CA2H
        TCON1
        TCON2
    ENDC
;----- ALUSTA Bits --------------------------------------------------------
    FS3                         EQU     H'0007'
    FS2                         EQU     H'0006'
    FS1                         EQU     H'0005'
    FS0                         EQU     H'0004'
    OV                          EQU     H'0003'
    Z                           EQU     H'0002'
    DC                          EQU     H'0001'
    C                           EQU     H'0000'
;----- CPUSTA Bits --------------------------------------------------------
    STKAV                       EQU     H'0005'
    GLINTD                      EQU     H'0004'
    NOT_TO                      EQU     H'0003'
    NOT_PD                      EQU     H'0002'
;----- INTSTA Bits --------------------------------------------------------
    PEIF                        EQU     H'0007'
    T0CKIF                      EQU     H'0006'
    T0IF                        EQU     H'0005'
    INTF                        EQU     H'0004'
    PEIE                        EQU     H'0003'
    T0CKIE                      EQU     H'0002'
```

```
   T0IE                          EQU     H'0001'
   INTE                          EQU     H'0000'
;----- PIE Bits --------------------------------------------------------
   RBIE                          EQU     H'0007'
   TMR3IE                        EQU     H'0006'
   TMR2IE                        EQU     H'0005'
   TMR1IE                        EQU     H'0004'
   CA2IE                         EQU     H'0003'
   CA1IE                         EQU     H'0002'
   TXIE                          EQU     H'0001'
   RCIE                          EQU     H'0000'
;----- PIR Bits --------------------------------------------------------
   RBIF                          EQU     H'0007'
   TMR3IF                        EQU     H'0006'
   TMR2IF                        EQU     H'0005'
   TMR1IF                        EQU     H'0004'
   CA2IF                         EQU     H'0003'
   CA1IF                         EQU     H'0002'
   TXIF                          EQU     H'0001'
   RCIF                          EQU     H'0000'
;----- PORTA Bits ------------------------------------------------------
   NOT_RBPUEQUH'0007'
   T0CKIEQUH'0001'
   INTEQUH'0000'
;----- RCSTA Bits ------------------------------------------------------
   SPEN                          EQU     H'0007'
   RC9                           EQU     H'0006'
   NOT_RC8                       EQU     H'0006'
   RC8_9                         EQU     H'0006'
   SREN                          EQU     H'0005'
   CREN                          EQU     H'0004'
   FERR                          EQU     H'0002'
   OERR                          EQU     H'0001'
   RCD8                          EQU     H'0000'
;----- T0STA Bits ------------------------------------------------------
   INTEDG                        EQU     H'0007'
   T0SE                          EQU     H'0006'
   T0CS                          EQU     H'0005'
   T0PS3                         EQU     H'0004'
   T0PS2                         EQU     H'0003'
   T0PS1                         EQU     H'0002'
   T0PS0                         EQU     H'0001'
;----- TCON1 Bits ------------------------------------------------------
   CA2ED1                        EQU     H'0007'
   CA2ED0                        EQU     H'0006'
   CA1ED1                        EQU     H'0005'
   CA1ED0                        EQU     H'0004'
   T16                           EQU     H'0003'
   TMR3CS                        EQU     H'0002'
   TMR2CS                        EQU     H'0001'
   TMR1CS                        EQU     H'0000'
;----- TCON2 Bits ------------------------------------------------------
   CA2OVF                        EQU     H'0007'
   CA1OVF                        EQU     H'0006'
   PWM2ON                        EQU     H'0005'
   PWM1ON                        EQU     H'0004'
   CA1                           EQU     H'0003'
   NOT_PR3                       EQU     H'0003'
   CA1_PR3                       EQU     H'0003'
   TMR3ON                        EQU     H'0002'
   TMR2ON                        EQU     H'0001'
   TMR1ON                        EQU     H'0000'
;----- TXSTA Bits ------------------------------------------------------
   CSRC                          EQU     H'0007'
   TX9                           EQU     H'0006'
   NOT_TX8                       EQU     H'0006'
   TX8_9                         EQU     H'0006'
```

```
   TXEN                          EQU     H'0005'
   SYNC                          EQU     H'0004'
   TRMT                          EQU     H'0001'
   TXD8                          EQU     H'0000'
;==========================================================================
;
;       Configuration Bits - Generic
;
;==========================================================================
   _XMC_MODE                     EQU     H'FFBF'
   _MC_MODE                      EQU     H'FFEF'
   _MP_MODE                      EQU     H'FFFF'
   _WDT_NORM                     EQU     H'FFF3'
   _WDT_64                       EQU     H'FFF7'
   _WDT_256                      EQU     H'FFFB'
   _WDT_1                        EQU     H'FFFF'
   _LF_OSC                       EQU     H'FFFC'
   _RC_OSC                       EQU     H'FFFD'
   _XT_OSC                       EQU     H'FFFE'
   _EC_OSC                       EQU     H'FFFF'
;==========================================================================
;
;       Processor-dependent Definitions
;
;==========================================================================
 IFDEF __17C42
   ; Nothing else needs to be defined
   #define __CONFIG_0
 ENDIF
 IFDEF __17C43
   ;----- Register Files ------------------------------------------------
      PRODL                      EQU     H'0018'
      PRODH                      EQU     H'0019'
   #define __CONFIG_1
 ENDIF
 IFDEF __17C44
   ;----- Register Files ------------------------------------------------
      PRODL                      EQU     H'0018'
      PRODH                      EQU     H'0019'
   #define __CONFIG_1
 ENDIF
;==========================================================================
;
;       Configuration Bits - Specific
;
;==========================================================================
 IFDEF __CONFIG_0
   _PMC_MODE                     EQU     H'FFAF'
   ;#undefine __CONFIG_0
 ENDIF
 IFDEF __CONFIG_1
   _PMC_MODE                     EQU     H'00AF'
   ;#undefine __CONFIG_1
 ENDIF
LIST
```

SAMPLE.ASM

```
;*******************************************************************
;                          SAMPLE.ASM
;                    8x8 Software Multiplier
;*******************************************************************
;
;   The 16 bit result is stored in 2 bytes
;
; Before calling the subroutine " mpy ", the multiplier should
; be loaded in location " mulplr ", and the multiplicand in
; " mulcnd " . The 16 bit result is stored in locations
; H_byte & L_byte.
;
;       Performance :
;                       Program Memory  :  15 locations
;                       # of cycles     :  71
;                       Scratch RAM     :   0 locations
;
;  This routine is optimized for code efficiency ( looped code )
;  For time efficiency code refer to "mult8x8F.asm" ( straight line code )
;*******************************************************************
;
        LIST    p=16C54 ; PIC16C54 is the target processor
mulcnd  equ     09      ; 8 bit multiplicand
mulplr  equ     10      ; 8 bit multiplier
H_byte  equ     12      ; High byte of the 16 bit result
L_byte  equ     13      ; Low byte of the 16 bit result
count   equ     14      ; loop counter
portb   equ     06      ; I/O register F6
STATUS  equ     03      ; STATUS register F3
CARRY   equ     0       ; Carry bit in status register
Same    equ     1       ;
;
; ****************************          Begin Multiplier Routine
mpy_S   clrf    H_byte
        clrf    L_byte
        movlw   8
        movwf   count
        movf    mulcnd,w
        bcf     STATUS,CARRY    ; Clear the carry bit in the status Reg.
loop    rrf     mulplr
        btfsc   STATUS,CARRY
        addwf   H_byte,Same
        rrf     H_byte,Same
        rrf     L_byte,Same
        decfsz  count
        goto    loop
;
        retlw   0
;
;*******************************************************************
;               Test Program
;*******************************************************************
start   clrw
        option
main    movf    portb,w
        movwf   mulplr          ; multiplier (in mulplr) = 05
        movf    portb,w
        movwf   mulcnd
;
call_m  call    mpy_S           ; The result is in locations F12 & F13
                                ; H_byte & L_byte
;
        goto    main
```

```
;
        org     01FFh
        goto    start
;
     END

list p=16C64,r=HEX
org 0
symbol_name             equ 010
symbol100_name          equ 011
symbol1000_name         equ 012
symbol123456789A        equ 013
symbol123456789ABCDEF   equ 014
device1                 equ 032
statflag                equ 02e
movf 3,w        ; set up Timer 0
iorlw 020
movwf 3
movlw 0df
movwf 01
nop
movf 3,w        ; set up Timer 1
andlw 0df
movwf 3
movlw 011
movwf 010
movlw 0fe
movwf 0e
nop
movlw 04        ; set up Timer 2
movwf 012
movf 3,w
iorlw 020
movwf 3
movlw 6
movwf 012
movf 3,w
andlw 0df
movwf 3
loop            nop
nop
nop
nop
nop
nop
nop
nop
nop
goto loop
END
```

SAMPLE.INI

```
LO SAMPLE
ST SAMPLE
SR X
ZP
ZR
ZT
RE
P  54
NV
AD mulcnd
AD mulplr
AD H_byte
```

```
AD L_byte
AD count
AD portb
AD RB7,B,1
AD RB6,B,1
AD RB5,B,1
AD RB4,B,1
AD RB3,B,1
AD RB2,B,1
AD RB1,B,1
AD RB0,B,1
RS
```

SAMPLE.STI

```
! Stimulus file for SAMPLE.ASM

STEP      RB7   RB6   RB5   RB4   RB3   RB2   RB1   RB0   ! PortB Pins
3         0     0     0     0     1     0     0     1     !  9 x 5
5         0     0     0     0     0     1     0     1
65        0     0     0     0     1     0     1     0     ! 10 x 5
67        0     0     0     0     0     1     0     1
127       0     0     0     1     1     0     1     1     ! 27 x 3
129       0     0     0     0     0     0     1     1
191       0     0     0     1     0     0     0     1     ! 17 x 7
193       0     0     0     0     0     1     1     1
253       0     1     0     0     0     0     0     0     ! 64 x 63
255       0     0     1     1     1     1     1     1
```

Appendix C. Customer Support

Keeping Current with Microchip Systems

This chapter provides a brief discussion of the Microchip BBS general services available. Because the Microchip BBS is an evolving product, details of its operation are not described here. This chapter also describes the Microchip software release numbering scheme.

Microchip Technology supports the Microchip BBS as a service to its customers. The Microchip BBS contains the most recent information regarding Microchip systems products. Microchip endeavors at all times to provide quality service and fast responsiveness to users. To accomplish this, Microchip monitors the BBS several times a week for questions. Truly urgent issues should not be left with the BBS, but referred to your local distributor, sales office or FAE.

Note: the best way to keep current with Microchip systems is to register.

Highlights

The highlighted points in this chapter include:

- Keeping Current with Microchip Systems
- Systems Information and Upgrade Hot Line
- Connecting to Microchip BBS
- Using the Bulletin Board
  - Special Interest Groups
  - Files
  - Mail
- Software Releases
  - Alpha Release
  - Intermediate Release
  - Beta Release
  - Production Release

Systems Information and Upgrade Hot Line

The Systems Information And Upgrade Line provides system users a listing of the latest versions of all of Microchip's development systems software products. Plus, this line provides information on how customers can receive any currently available upgrade kits. The Hot Line Numbers are: 1-800-755-2345 for U.S. and most of Canada, and 1-602-786-7302 for the rest of the world.

These phone numbers are also listed on the "Important Information" sheet that is shipped with all development systems. The hot line message is updated whenever a new software version is added to the Microchip BBS, or when a new upgrade kit becomes available.

Connecting to Microchip BBS

Connect worldwide to the Microchip BBS using the CompuServe® communications network. In most cases, a local call is your only expense. The Microchip BBS connection does not use CompuServe membership services, therefore, **you do not need CompuServe membership to join Microchip's BBS**.

There is **no charge** for connecting to the BBS, except for a toll charge to the CompuServe access number, where applicable. You do not need to be a CompuServe member to take advantage of this connection (you never actually log in to CompuServe).

The procedure to connect will vary slightly from country to country. Please check with your local CompuServe agent for details if you have a problem. CompuServe service allow multiple users at baud rates up to 14400 bps.

The following connect procedure applies in most locations.

1. Set your modem to 8-bit, No parity, and One stop (8N1). This is not the normal CompuServe setting which is 7E1.
2. Dial your local CompuServe access number.
3. Depress **<Enter↵>** and a garbage string will appear because CompuServe is expecting a 7E1 setting.
4. Type +, depress **<Enter↵>** and `Host Name:` will appear.
5. Type **MCHIPBBS**, depress **<Enter↵>** and you will be connected to the Microchip BBS.
6. In the United States, to find CompuServe's phone number closest to you, set your modem to 7E1 and dial (800) 848-4480 for 300-2400 baud or (800) 331-7166 for 9600-14400 baud connection. After the system responds with `Host Name:`, type

   NETWORK, depress **<Enter↵>** and follow CompuServe's directions.

For voice information (or calling from overseas), you may call (614) 457-1550 for your local CompuServe number.

Using the Bulletin Board

The bulletin board is a multifaceted tool. It can provide you with information on a number of different topics.

- Special Interest Groups
- Files
- Mail
- Bug Lists

Special Interest Groups

Special Interest Groups, or SIGs as they are commonly referred to, provide you with the opportunity to discuss issues and topics of interest with others that share your interest or questions. SIGs may provide you with information not available by any other method because of the broad background of the PIC16/17 user community.

There are SIGs for most Microchip systems, including:

- MPASM
- PICMASTER®
- PRO MATE
- Utilities
- Bugs
- MPSIM
- TRUE GAUGE™
- *fuzzy*TECH®-MP
- ASSP

These groups are monitored by the Microchip staff.

Files

Microchip regularly uses the Microchip BBS to distribute technical information, application notes, source code, errata sheets, bug reports, and interim patches for Microchip systems software products. Users can contribute files for distribution on the BBS. For each SIG, a moderator monitors, scans, and approves or disapproves files submitted to the SIG. No executable files are accepted from the user community in general to limit the spread of computer viruses.

Mail

The BBS can be used to distribute mail to other users of the service. This is one way to get answers to your questions and problems from the Microchip staff, as well as keeping in touch with fellow Microchip users worldwide.

Consider mailing the moderator of your SIG, or the SYSOP, if you have ideas or questions about Microchip products, or the operation of the BBS.

Note:	The SIGs provide you with the opportunity to discuss issues and exchange ideas. Technical support and urgent questions should be referred to your local distributor, sales representative or FAE. They are your first level of support.

Software Releases

Software products released by Microchip are referred to by version numbers. Version numbers use the form:

```
xx.yy.zz <status>
```

Where `xx` is the major release number, `yy` is the minor number, and `zz` is the intermediate number. The `status` field displays one of the following categories:

- Alpha
- Intermediate
- Beta
- Released

Production releases are numbered with major, and minor version numbers like:

```
3.04 Released
```

Alpha, Beta and Intermediate releases are numbered with the major, minor and intermediate numbers:

```
3.04.01 Alpha
```

Alpha Release

Alpha designated software is engineering software that has not been submitted to any quality assurance testing. In general, this grade of software is intended for software development team access only, but may be sent to selected individuals for conceptual evaluation. Once Alpha grade software has passed quality assurance testing, it may be upgraded to Beta or Intermediate status.

Intermediate Release

Intermediate released software represents changes to a released software system and is designated as such by adding an intermediate number to the version number. Intermediate changes are represented by:

- Bug Fixes
- Special Releases
- Feature Experiments

Intermediate released software does not represent our most tested and stable software. Typically, it will not have been subject to a thorough and rigorous test suite, unlike production released versions. Therefore, users should use these versions with care, and only in cases where the features provided by an intermediate release are required.

Intermediate releases are primarily available through the BBS.

Beta Release

Preproduction software is designated as Beta. Beta software is sent to Applications Engineers and Consultants, FAEs, and select customers. The Beta Test period is limited to a few weeks. Software that passes Beta testing without having significant flaws, will be production released. Flawed software will be evaluated, repaired, and updated with a new revision number for a subsequent Beta trial.

Production Release

Production released software is software shipped with tool products. Example products are PRO MATE, PICSTART®, and PICMASTER. The Major number is advanced when significant feature enhancements are made to the product. The minor version number is advanced for maintenance fixes and minor enhancements. Production released software represents Microchip's most stable and thoroughly tested software.

There will always be a period of time when the Production Released software is not reflected by products being shipped until stocks are rotated. You should always check the BBS for the current production release.

Appendix D. Intel INTELLEC™ Hexadecimal Format

INHX8M		INHX8S			
START	:(colon)	START	:(colon)	START	:(colon)
CHARACTER		CHARACTER			
WORD COUNT	2 Hex digits	BYTE COUNT	2 Hex digits	BYTE COUNT	2 Hex digits
ADDRESS	4 Hex digits	ADDRESS	2 Hex digits	ADDRESS	4 Hex digits
RECORD TYPE	2 Hex digits	RECORD TYPE	2 Hex digits	RECORD TYPE	2 Hex digits
HIGH BYTE	2 Hex digits	LOW or HIGH BYTE		LOW BYTE	2 Hex digits
LOW BYTE	2 Hex digits	CHECK SUM	2 Hex digits	HIGH BYTE	1 Hex digits
CHECK SUM	2 Hex digits			CHECK SUM	2 Hex digits

Figure D.1a INHX8S and INHX8M File Formats

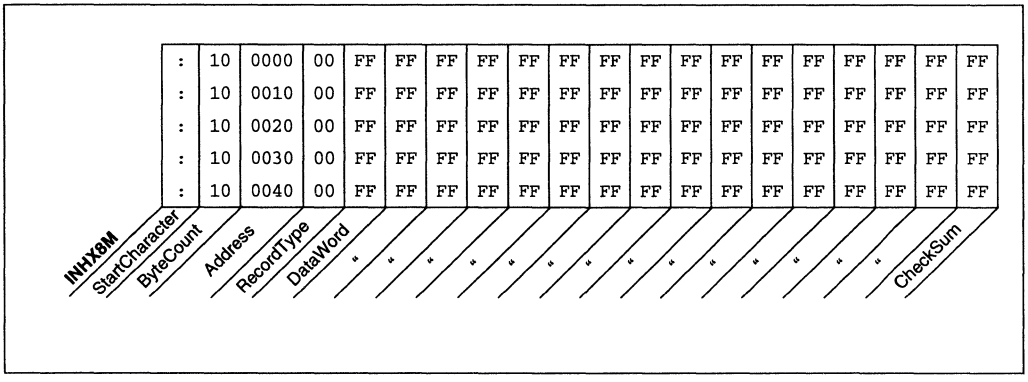


Figure D.1b - INHX8M File Formats

INHX8M

This format produces one 8-bit hexadecimal file with a low-byte/high-byte combination. Since each address can only contain 8 bits in this format, all addresses are doubled. File extensions for the object code are ".OBJ." This format is useful for transferring PIC16C5X series object code to third party EPROM programmers.

The difference between this format and Inhx16 is the word length and the high/low byte order. Inhx8m has 8-bit words (two hexadecimal digits) with the low byte first, rather than 16-bit words (four hexadecimal digits) with the high byte first.

8-Bit Hex Format:

Each data record begins with a 9 character prefix and ends with a 2 character checksum. Each record has the following format:

:BBAAAATTHHHH....HHHCC

where,

BB	a two-digit hexadecimal byte count representing the number of data words that appear on the line.
AAAA	a four-digit hexadecimal address representing the starting address for the data record.
TT	a two-digit record type that will always be '00' except for the end-of-file record which is set to '01'.
HH	a two-digit hexadecimal data word.
CC	a two-digit hexadecimal checksum that's the two's compliment of the sum of all preceding bytes in the record including the prefix.

Appendix D. Intel INTELLEC Hexadecimal Format

32-Bit Hex Format (.HEX)

The extended 32-bit address HEX format is similar to the Hex 8 format described above, except that the Intel extended linear address record is output also to establish the upper 16 bits of the data address.

Each data record begins with a 9 character prefix and ends with a 2 character checksum. Each record has the following format:

```
:BBAAAATTHHHH....HHHCC
```

where

BB - is a two digit hexadecimal byte count representing the number of data bytes that will appear on the line.

AAAA - is a four digit hexadecimal address representing the starting address of the data record.

TT - is a two digit record type record type:

00 - Data record

01 - End of File record

02 - Segment address record

04 - Linear address record

HH - is a two digit hexadecimal data word.

CC - is a two digit hexadecimal checksum that is the two's compliment of the sum of all preceding bytes in the record including the prefix.

Appendix E. PIC16C5X User's Guide Addendum

Introduction

MPSIM provides support for more than one family of Microchip microcontrollers. This section has been added as an addendum to the MPSIM user's guide to centralize PIC16C5X-specific simulator support.

I/O Pins

The PIC16C5X family consists of the PIC16C54, PIC16C55, PIC16C56, PIC16C57, and PIC16C58A. When modifying pins either manually (with the SE command) or via the stimulus file, use the following pin names only. These are the only ones that MPSIM recognizes as valid I/O pins. Because the pinout is device-specific, some pins (for example RC0 on a PIC16C54) will not be available on all parts in this family.

- $\overline{\text{MCLR}}$
- T0CKI
- RA0-RA3
- RB0-RB7
- RC0-RC7

CPU Model

Reset Conditions

All reset conditions are supported by MPSIM.

A Power-On-Reset can be simulated by using the RS instruction. All special-purpose registers will be initialized to the values specified in the data sheet.

A $\overline{\text{MCLR}}$ reset during normal operation or during SLEEP can easily be simulated by driving the MCLR pin low (and then high) via the stimulus file or by using the SE command or by using DK command.

A WDT time-out reset is simulated when WDT is enabled (see DW command) and proper prescaler is set (by initializing OPTION register appropriately) and WDT actually overflows. WDT time-out period (with prescale = 1) is approximated at 18 ms (to closest instruction cycle multiple).

The Time-out ($\overline{\text{TO}}$) and Power-down ($\overline{\text{PD}}$) bits in the Status register reflect appropriate reset condition. This feature is useful for simulating various power-up and time out forks in the user code.

Sleep

MPSIM simulates the SLEEP instruction, and will appear "asleep" until a wake-up from sleep condition occurs. For example, if the Watchdog timer has been enabled, it will wake the processor up from sleep when it times out (depending upon the prescaler setting in the OPTION register).

WDT

The Watchdog timer is fully simulated in the MPSIM simulator. Because it is fuse-selectable on the device, it must be enabled by a separate command (see the DW command) in MPSIM. The period of the WDT is determined by the prescaler settings in the OPTION register. The basic period (with prescaler = 1) is approximated at 18 ms (to closest instruction cycle multiple).

Stack

MPSIM presents an accurate simulation of the hardware stack on the PIC16C5X, and additionally provides warning messages if an underflow or overflow condition occurs. When a CALL instruction is encountered, or when an interrupt has occurred, the value of the PC+ 1 is pushed to the stack, and the stack is popped when a RETLW instruction is executed. If more than two values are pushed to the stack before it is popped, the value will be pushed to the stack, but a warning message will be issued, indicating a stack overflow condition. An error message will also be generated if the user attempts to pop an empty stack. Popping an empty stack will cause the last value popped to be put in the PC.

Special Registers

To aid in debugging this device, certain items that are normally not observable have been declared as "special" registers. For example, the W register is not directly-addressable, but can be added to the viewscreen, by adding the special label "W" or "w" with the AD command, just as any register. The following is a complete list of "special" registers that can be added to the viewscreen and observed or modified. You can add them as you normally would any other register declared in your code, specifying any radix to view them.

- W (or w)
- TRISA
- TRISB
- TRISC
- OPT (the option register)

It is important not to redefine these special labels. For example, do not define the label "W" to be equal to zero in your source code. This will cause the special label to be overridden, and "W" will now be the indirect-address register (INDF).

Peripherals

Peripherals Supported

Along with providing core support, the RTCC timer/counter module is fully supported. It is fully supported in internal and external clock modes. The prescaler is made readable and writable as 'RTCCPRE" symbol.

It is important to remember that because MPSIM executes on instruction cycle boundaries, resolutions below 1 Tcy cannot be simulated.

MPSIM is a discrete-event simulator where all stimuli are evaluated and all response generated at instruction boundaries or Tcy. One Tcy = 4 Tosc (where Tosc is input clock). Therefore, there are several events that can not be accurately simulated in MPSIM. These fall into two categories:

- Purely asynchronous events
- Synchronous events that occur at Tosc clock boundaries

Because of this, the following items are not supported in MPSIM:

- Timer0 prescaler is capable of accepting clock pulse inputs smaller than Tcy, but this can not be simulated.

In summary, the net result of instruction boundary simulation is that all events get synchronized at instruction boundary and events smaller than one instruction cycle get lost.

Appendix F. PIC16C64 User's Guide Addendum

Introduction

MPSIM provides support for more than one family of Microchip microcontrollers. This section has been added as an addendum to the MPSIM user's guide to centralize PIC16C64-specific simulator support.

I/O Pins

The PIC16C64 is a 40-pin device, with many of the I/O pins multiplexed with other peripherals (and therefore referred by more than one name). When modifying pins either manually (e.g. with the SE command) or via the stimulus file, use the following pin names only. These are the only ones that MPSIM recognizes as valid I/O pins:

- $\overline{\text{MCLR}}$
- RA0-RA5
- RB0-RB7
- RC0-RC7
- RD0-RD7
- RE0-RE2

Interrupts

MPSIM version 4.5 or greater supports all interrupts on the PIC16C64:

- Timer0 overflow
- Timer1 overflow
- Timer2
- CCP1
- SSP (in SPI mode ONLY)
- Change on Port RB <7..4>
- External interrupt from RB0/INT pin
- Parallel Slave Port

CPU Model

Reset Conditions

All reset conditions are supported by MPSIM.

A Power-On-Reset can be simulated by using the RS instruction. All special-purpose registers will be initialized to the values specified in the data sheet.

A $\overline{\text{MCLR}}$ reset during normal operation or during SLEEP can easily be simulated by driving the $\overline{\text{MCLR}}$ pin low (and then high) via the stimulus file or by using the SE command or by using DK command.

A WDT time-out reset is simulated when WDT is enabled (see DW command) and proper prescaler is set (by initializing OPTION register appropriately) and WDT actually overflows. WDT time-out period (with prescale = 1) is approximated at 18 ms (to closest instruction cycle multiple).

The Time-out ($\overline{\text{TO}}$) and Power-down ($\overline{\text{PD}}$) bits in the Status register reflect appropriate reset condition. This feature is useful for simulating various power-up and time out forks in the user code.

Sleep

MPSIM simulates the SLEEP instruction, and will appear "asleep" until a wake-up from sleep condition occurs. For example, if the Watchdog timer has been enabled, it will wake the processor up from sleep when it times out (depending upon the prescaler setting in the OPTION register). Another example of a wake-up-from-sleep condition, would be Timer1 wake-up from sleep. In this case, when the processor is asleep, Timer1 would continue to increment until it overflows, and if the interrupt is enabled, will wake the processor on overflow and branch to the interrupt vector.

WDT

The Watchdog timer is fully simulated in the MPSIM simulator. Because it is fuse-selectable on the device, it must be enabled by a separate command (see the DW command) in MPSIM. The period of the WDT is determined by the prescaler settings in the OPTION register. The basic period (with prescaler = 1) is approximated at 18 ms (to closest instruction cycle multiple).

Stack

MPSIM presents an accurate simulation of the hardware stack on the PIC16CXX, and additionally provides warning messages if an underflow or overflow condition occurs. When a CALL instruction is encountered, or when an interrupt has occurred, the value of the PC+ 1 is pushed to the stack, and the stack is popped when a RETLW, RETURN, or RETFIE instruction is executed. If more than eight values are pushed to the stack before it is popped, the value will be pushed to the stack, but a warning message will be issued, indicating a stack overflow condition. An error message will also be

generated if the user attempts to pop an empty stack. Popping an empty stack will cause the stack pointer to point to the top of a full stack, and will not generate an error message if another pop is initiated.

Special Registers

To aid in debugging this device, certain items that are normally not observable have been declared as "special" registers. Prescalers and postscalers cannot be declared in your code as "registers", so there are special labels that can be added to the view screen. You can add them as you normally would any other register declared in your code, specifying any radix to view them.

The following are special items that can be added to the view screen when the PIC16C64 has been selected:

- T0PRE - Prescaler for timer0
- T1PRE - Prescaler for timer1
- T2PRE - Prescaler for timer2
- T2POS - Postscaler for timer2
- CCP1PRE - Prescaler for CCP1
- SPIPRE - Prescaler for SPI
- SSPSR - SSP Shift register

Please remember that these labels are only available when the PIC16C64 is the target processor, and that they cannot be manually modified.

Peripherals

Peripherals Supported

Along with providing core support, the following peripheral modules (in addition to general-purpose I/O) are supported:

- Timer0
- Timer1
- Timer2
- CCP1
- Parallel Slave Port
- SSP (in SPI Mode only)

Tcycle Limitation

It is important to remember that because MPSIM executes on instruction cycle boundaries, resolutions below 1 Tcy cannot be simulated. Please see the following section for more details concerning the limitations of T-cycle simulation.

MPSIM is a discrete-event simulator where all stimuli are evaluated and all response generated at instruction boundaries or Tcy. One Tcy = 4 Tosc (where Tosc is input clock). Therefore, there are several events that can not be accurately simulated in MPSIM. These fall into two categories:

- Purely asynchronous events
- Synchronous events that occur at Tosc clock boundaries

Because of this, the following items are not supported in MPSIM:

- Timer0, Timer1, and Timer2 prescalers are capable of accepting clock pulse inputs smaller than Tcy, but these can not be simulated.
- Capture input pulses can be smaller than one Tcy, but can not be simulated.
- PWM output pulse resolution less than 1 Tcy is not supported.
- 8-bit compare will not be supported since the output resolution is limited to T cycles
- In unsynchronized counter mode, clock input smaller than Tcy is not supported
- The oscillator on RC0/RC1 pins is not supported. The user can, how ever, simply use an external clock input for simulation purposes.

In summary, the net result of instruction boundary simulation is that all events get synchronized at instruction boundary and events smaller than one instruction cycle get lost.

TIMER0

Timer0 (and the interrupt it can generate on overflow) is fully supported by MPSIM, and will increment by the internal or external clock. Clock input must have a minimum high time of 1Tcy and a minimum low time of 1Tcy due to stimulus file requirements. The prescaler for Timer0 is made accessible as T0PRE. It can be watched and modified.

TIMER1

Timer1 in its various modes is supported by MPSIM, except when running in counter mode by an external crystal. The interrupt it can be generated on overflow and wake-up from sleep through interrupt are both supported by MPSIM. The prescaler for Timer1 is viewable and modifiable as T1PRE. The external oscillator on RC0/RC1 is not simulated. The user can simply use a clock input (see CK command).

TIMER2

Timer2 and the interrupt that can be generated on overflow are fully supported by MPSIM, and both the prescaler and postscaler for Timer2 are viewable and modifiable (T2PRE and T2POS).

Appendix F. PIC16C64 User's Guide Addendum

CCP1

CAPTURE

MPSIM fully supports capture and the interrupt generated. The prescaler for the CCP module is viewable and modifiable (CCP1PRE).

COMPARE

Compare mode, its interrupt, and the special event trigger (resetting Timer1 by CCP1) are supported in this version of MPSIM.

PWM

PWM output (resolution greater than 1Tcy only) are supported in this version of MPSIM.

SSP

The Synchronous Serial Port is supported in SPI mode only. The shift register (SSPSR) can be added to the viewscreen, observed and modified. MPSIM currently does not support the I2C™ mode.

Appendix G. PIC16C65 User's Guide Addendum

Introduction

MPSIM provides support for more than one family of Microchip microcontrollers. This section has been added as an addendum to the MPSIM user's guide to centralize PIC16C65-specific simulator support.

I/O Pins

The PIC16C65 is a 40-pin device, with many of the I/O pins multiplexed with other peripherals (and therefore referred by more than one name). When modifying pins either manually (e.g. with the SE command) or via the stimulus file, use the following pin names only. These are the only ones that MPSIM recognizes as valid I/O pins:

- $\overline{\text{MCLR}}$
- RA0-RA5
- RB0-RB7
- RC0-RC7
- RD0-RD7
- RE0-RE7

Interrupts

MPSIM version 4.5 or greater supports all interrupts on the PIC16C65:

- Timer0 overflow
- Timer1 overflow
- Timer2
- CCP1
- CCP2
- SSP (in SPI mode ONLY)
- Change on Port RB <7:4>
- External interrupt from RB0/INT pin
- USART
- Parallel Slave Port

CPU Model

Reset Conditions

All reset conditions are supported by MPSIM.

A Power-On-Reset can be simulated by using the RS instruction. All special-purpose registers will be initialized to the values specified in the data sheet.

A $\overline{\text{MCLR}}$ reset during normal operation or during SLEEP can easily be simulated by driving the $\overline{\text{MCLR}}$ pin low (and then high) via the stimulus file or by using the SE command or by using DK command.

A WDT time-out reset is simulated when WDT is enabled (see DW command) and proper prescaler is set (by initializing OPTION register appropriately) and WDT actually overflows. WDT time-out period (with prescale = 1) is approximated at 18 ms (to closest instruction cycle multiple).

The Time-out ($\overline{\text{TO}}$) and Power-down ($\overline{\text{PD}}$) bits in the Status register reflect appropriate reset condition. This feature is useful for simulating various power-up and time out forks in the user code.

Sleep

MPSIM simulates the SLEEP instruction, and will appear "asleep" until a wake-up from sleep condition occurs. For example, if the Watchdog timer has been enabled, it will wake the processor up from sleep when it times out (depending upon the prescaler setting in the OPTION register). Another example of a wake-up-from-sleep condition, would be Timer1 wake-up from sleep. In this case, when the processor is asleep, Timer1 would continue to increment until it overflows, and if the interrupt is enabled, will wake the processor on overflow and branch to the interrupt vector.

WDT

The Watchdog timer is fully simulated in the MPSIM simulator. Because it is fuse-selectable on the device, it must be enabled by a separate command (see the DW command) in MPSIM. The period of the WDT is determined by the prescaler settings in the OPTION register. The basic period (with prescaler = 1) is approximated at 18 ms (to closest instruction cycle multiple).

Stack

MPSIM presents an accurate simulation of the hardware stack on the PIC16CXX, and additionally provides warning messages if an underflow or overflow condition occurs. When a CALL instruction is encountered, or when an interrupt has occurred, the value of the PC+ 1 is pushed to the stack, and the stack is popped when a RETLW, RETURN, or RETFIE instruction is executed. If more than eight values are pushed to the stack before it is popped, the value will be pushed to the stack, but a warning message will be issued, indicating a stack overflow condition. An error message will also be

generated if the user attempts to pop an empty stack. Popping an empty stack will cause the stack pointer to point to the top of a full stack, and will not generate an error message if another pop is initiated.

Special Registers

To aid in debugging this device, certain items that are normally not observable have been declared as "special" registers. Prescalers and postscalers cannot be declared in your code as "registers", so there are special labels that can be added to the view screen. You can add them as you normally would any other register declared in your code, specifying any radix to view them.

The following are special items that can be added to the view screen when the PIC16C65 has been selected:

- T0PRE - Prescaler for timer0
- T1PRE - Prescaler for timer1
- T2PRE - Prescaler for timer2
- T2POS - Postscaler for timer2
- CCP1PRE - Prescaler for CCP1
- CCP2PRE - Prescaler for CCP2
- SPIPRE - Prescaler for SPI
- SSPSR - SSP Shift register

Please remember that these labels are only available when the PIC16C65 is the target processor, and that they cannot be manually modified.

Peripherals

Peripherals Supported

Along with providing core support, the following peripheral modules (in addition to general-purpose I/O) are supported:

- Timer0
- Timer1
- Timer2
- CCP1
- CCP2
- Parallel Slave Port
- SSP (in SPI Mode only)
- USART (limited)

Tcycle Limitation

It is important to remember that because MPSIM executes on instruction cycle boundaries, resolutions below 1 Tcy cannot be simulated. Please see the following section for more details concerning the limitations of T-cycle simulation.

MPSIM is a discrete-event simulator where all stimuli are evaluated and all response generated at instruction boundaries or Tcy. One Tcy = 4 Tosc (where Tosc is input clock). Therefore, there are several events that can not be accurately simulated in MPSIM. These fall into two categories:

- Purely asynchronous events
- Synchronous events that occur at Tosc clock boundaries

Because of this, the following items are not supported in MPSIM:

- Timer0, Timer1, and Timer2 prescalers are capable of accepting clock pulse inputs smaller than Tcy, but these can not be simulated.
- Capture input pulses can be smaller than one Tcy, but can not be simulated.
- PWM output pulse resolution less than 1 Tcy is not supported.
- 8-bit compare will not be supported since the output resolution is limited to T cycles
- In unsynchronized counter mode, clock input smaller than Tcy is not supported
- The oscillator on RC0/RC1 pins is not supported. The user can, how ever, simply use an external clock input for simulation purposes.

In summary, the net result of instruction boundary simulation is that all events get synchronized at instruction boundary and events smaller than one instruction cycle get lost.

TIMER0

Timer0 (and the interrupt it can generate on overflow) is fully supported by MPSIM, and will increment by the internal or external clock. Clock input must have a minimum high time of 1Tcy and a minimum low time of 1Tcy due to stimulus file requirements. The prescaler for Timer0 is made accessible as T0PRE. It can be watched and modified.

TIMER1

Timer1 in its various modes is supported by MPSIM, except when running in counter mode by an external crystal. The interrupt it can be generated on overflow and wake-up from sleep through interrupt are both supported by MPSIM. The prescaler for Timer1 is viewable and modifiable as T1PRE. The external oscillator on RC0/RC1 is not simulated. The user can simply use a clock input (see CK command).

Appendix G. PIC16C65 User's Guide Addendum

TIMER2

Timer2 and the interrupt that can be generated on overflow are fully supported by MPSIM, and both the prescaler and postscaler for Timer2 are viewable and modifiable (T2PRE and T2POS).

CCP1 and CCP2

CAPTURE

MPSIM fully supports capture and the interrupt generated. The prescaler for the CCP module is viewable and modifiable (CCP1PRE).

COMPARE

Compare mode, its interrupt, and the special event trigger (resetting Timer1 with CCP1) are supported in this version of MPSIM.

PWM

PWM output (resolution greater than 1Tcy only) are supported in this version of MPSIM.

SSP

The Synchronous Serial Port is supported in SPI mode only. The shift register (SSPSR) can be added to the viewscreen, observed and modified. MPSIM currently does not support the I^2C mode.

USART

Timing and interrupt generation is supported. Baud rate generator is supported. Reading and writing of the registers are supported but actual receive or transmit operation is not simulated.

Appendix H. PIC16C71 User's Guide Addendum

Introduction

MPSIM provides support for more than one family of Microchip microcontrollers. This section has been added as an addendum to the MPSIM user's guide to centralize PIC16C71-specific simulator support.

I/O Pins

The PIC16C71 is an 18-pin device, with some of the I/O pins multiplexed with other peripherals (and therefore referred by more than one name). When modifying pins either manually (e.g. with the SE command) or via the stimulus file, use the following pin names only. These are the only ones that MPSIM recognizes as valid I/O pins:

- $\overline{\text{MCLR}}$
- RA0-RA4
- RB0-RB7

Additionally, RTCC is also recognized as Timer0 (previously RTCC) input, i.e. same as RA4.

Interrupts

MPSIM supports all interrupts on the PIC16C71:

- Timer0 overflow
- Change on Port RB <7..4>
- External interrupt from RB0/INT pin
- A/D interrupt complete

CPU Model

Reset Conditions

All reset conditions are supported by MPSIM.

A Power-On-Reset can be simulated by using the RS instruction. All special-purpose registers will be initialized to the values specified in the data sheet.

A $\overline{\text{MCLR}}$ reset during normal operation or during SLEEP can easily be simulated by driving the $\overline{\text{MCLR}}$ pin low (and then high) via the stimulus file or by using the SE command or by using DK command.

A WDT time-out reset is simulated when WDT is enabled (see DW command) and proper prescaler is set (by initializing OPTION register appropriately) and WDT actually overflows. WDT time-out period (with prescale = 1) is approximated at 18 ms (to closest instruction cycle multiple).

The Time-out ($\overline{\text{TO}}$) and Power-down ($\overline{\text{PD}}$) bits in the Status register reflect appropriate reset condition. This feature is useful for simulating various power-up and time out forks in the user code.

Sleep

MPSIM simulates the SLEEP instruction, and will appear "asleep" until a wake-up from sleep condition occurs. For example, if the Watchdog timer has been enabled, it will wake the processor up from sleep when it times out (depending upon the prescaler setting in the OPTION register). Another example of a wake-up-from-sleep condition, would be wake-up due to RB0/INT external interrupt.

WDT

The Watchdog timer is fully simulated in the MPSIM simulator. Because it is fuse-selectable on the device, it must be enabled by a separate command (see the DW command) in MPSIM. The period of the WDT is determined by the prescaler settings in the OPTION register. The basic period (with prescaler = 1) is approximated at 18 ms (to closest instruction cycle multiple).

Stack

MPSIM presents an accurate simulation of the hardware stack on the PIC16CXX, and additionally provides warning messages if an underflow or overflow condition occurs. When a CALL instruction is encountered, or when an interrupt has occurred, the value of the PC+ 1 is pushed to the stack, and the stack is popped when a RETLW, RETURN, or RETFIE instruction is executed. If more than eight values are pushed to the stack before it is popped, the value will be pushed to the stack, but a warning message will be issued, indicating a stack overflow condition. An error message will also be generated if the user attempts to pop an empty stack. Popping an empty stack will cause the stack pointer to point to the top of a full stack, and will not generate an error message if another pop is initiated.

Appendix H. PIC16C71 User's Guide Addendum

Special Registers

To aid in debugging this device, certain items that are normally not observable have been declared as "special" registers. Prescalers and postscalers cannot be declared in your code as "registers", so there are special labels that can be added to the view screen. You can add them as you normally would any other register declared in your code, specifying any radix to view them.

The following are special items that can be added to the view screen when the PIC16C71 has been selected:

- T0PRE - Prescaler for timer0

Please remember that these labels are only available when the PIC16C71 is the target processor, and that they cannot be manually modified.

Peripherals

Peripherals Supported

Along with providing core support, the following peripheral modules (in addition to general-purpose I/O) are supported:

- Timer0
- A/D module (limited)

Tcycle Limitation

It is important to remember that because MPSIM executes on instruction cycle boundaries, resolutions below 1 Tcy cannot be simulated. Please see the following section for more details concerning the limitations of T-cycle simulation.

MPSIM is a discrete-event simulator where all stimuli are evaluated and all response generated at instruction boundaries or Tcy. One Tcy = 4 Tosc (where Tosc is input clock). Therefore, there are several events that can not be accurately simulated in MPSIM. These fall into two categories:

- Purely asynchronous events
- Synchronous events that occur at Tosc clock boundaries

Because of this, the following items are not supported in MPSIM:

- Timer0 prescaler is capable of accepting clock pulse inputs smaller than Tcy, but this can not be simulated.

In summary, the net result of instruction boundary simulation is that all events get synchronized at instruction boundary and events smaller than one instruction cycle get lost.

TIMER0

Timer0 (and the interrupt it can generate on overflow) is fully supported by MPSIM, and will increment by the internal or external clock. Clock input must have a minimum high time of 1Tcy and a minimum low time of 1Tcy due to stimulus file requirements. The prescaler for Timer0 is made accessible as T0PRE. It can be watched and modified.

A/D Converter

All the registers, timing function and interrupt generation are implemented. The simulator, however, does not load any meaningful value into A/D result register (ADRES) at the end of a conversion. Use the FI command to load the ADRES register from a file for simulation purposes.

Appendix I. PIC16C73 User's Guide Addendum

Introduction

MPSIM provides support for more than one family of Microchip microcontrollers. This section has been added as an addendum to the MPSIM user's guide to centralize PIC16C73-specific simulator support.

I/O Pins

The PIC16C73 is a 28-pin device, with many of the I/O pins multiplexed with other peripherals (and therefore referred by more than one name). When modifying pins either manually (e.g. with the SE command) or via the stimulus file, use the following pin names only. These are the only ones that MPSIM recognizes as valid I/O pins:

- $\overline{\text{MCLR}}$
- RA0-RA5
- RB0-RB7
- RC0-RC7

Interrupts

MPSIM version 4.5 or greater supports all interrupts on the PIC16C73:

- Timer0 overflow
- Timer1 overflow
- Timer2
- CCP1
- CCP2
- SSP (in SPI mode ONLY)
- Change on Port RB <7..4>
- External interrupt from RB0/INT pin
- A/D interrupt complete
- USART

Note: Appendix O has been intentionally skipped in the numbering process.

CPU Model

Reset Conditions

All reset conditions are supported by MPSIM.

A Power-On-Reset can be simulated by using the RS instruction. All special-purpose registers will be initialized to the values specified in the data sheet.

A $\overline{\text{MCLR}}$ reset during normal operation or during SLEEP can easily be simulated by driving the $\overline{\text{MCLR}}$ pin low (and then high) via the stimulus file or by using the SE command or by using DK command.

A WDT time-out reset is simulated when WDT is enabled (see DW command) and proper prescaler is set (by initializing OPTION register appropriately) and WDT actually overflows. WDT time-out period (with prescale = 1) is approximated at 18 ms (to closest instruction cycle multiple).

The Time-out ($\overline{\text{TO}}$) and Power-down ($\overline{\text{PD}}$) bits in the Status register reflect appropriate reset condition. This feature is useful for simulating various power-up and time out forks in the user code.

Sleep

MPSIM simulates the SLEEP instruction, and will appear "asleep" until a wake-up from sleep condition occurs. For example, if the Watchdog timer has been enabled, it will wake the processor up from sleep when it times out (depending upon the prescaler setting in the OPTION register). Another example of a wake-up-from-sleep condition, would be Timer1 wake-up from sleep. In this case, when the processor is asleep, Timer1 would continue to increment until it overflows, and if the interrupt is enabled, will wake the processor on overflow and branch to the interrupt vector.

WDT

The Watchdog timer is fully simulated in the MPSIM simulator. Because it is fuse-selectable on the device, it must be enabled by a separate command (see the DW command) in MPSIM. The period of the WDT is determined by the prescaler settings in the OPTION register. The basic period (with prescaler = 1) is approximated at 18 ms (to closest instruction cycle multiple).

Stack

MPSIM presents an accurate simulation of the hardware stack on the PIC16CXX, and additionally provides warning messages if an underflow or overflow condition occurs. When a CALL instruction is encountered, or when an interrupt has occurred, the value of the PC+ 1 is pushed to the stack, and the stack is popped when a RETLW, RETURN, or RETFIE instruction is executed. If more than eight values are pushed to the stack before it is popped, the value will be pushed to the stack, but a warning message will be issued, indicating a stack overflow condition. An error message will also be

generated if the user attempts to pop an empty stack. Popping an empty stack will cause the stack pointer to point to the top of a full stack, and will not generate an error message if another pop is initiated.

Special Registers

To aid in debugging this device, certain items that are normally not observable have been declared as "special" registers. Prescalers and postscalers cannot be declared in your code as "registers", so there are special labels that can be added to the view screen. You can add them as you normally would any other register declared in your code, specifying any radix to view them.

The following are special items that can be added to the view screen when the PIC16C73 has been selected:

- T0PRE - Prescaler for timer0
- T1PRE - Prescaler for timer1
- T2PRE - Prescaler for timer2
- T2POS - Postscaler for timer2
- CCP1PRE - Prescaler for CCP1
- CCP2PRE - Prescaler for CCP2
- SPIPRE - Prescaler for SPI
- SSPSR - SSP Shift register

Please remember that these labels are only available when the PIC16C73 is the target processor, and that they cannot be manually modified.

Peripherals

Peripherals Supported

Along with providing core support, the following peripheral modules (in addition to general-purpose I/O) are supported:

- Timer0
- Timer1
- Timer2
- CCP1
- CCP2
- SSP (in SPI Mode only)
- A/D module (limited)
- USART (limited)

Tcycle Limitation

It is important to remember that because MPSIM executes on instruction cycle boundaries, resolutions below 1 Tcy cannot be simulated. Please see the following section for more details concerning the limitations of T-cycle simulation.

MPSIM is a discrete-event simulator where all stimuli are evaluated and all response generated at instruction boundaries or Tcy. One Tcy = 4 Tosc (where Tosc is input clock). Therefore, there are several events that can not be accurately simulated in MPSIM. These fall into two categories:

- Purely asynchronous events
- Synchronous events that occur at Tosc clock boundaries

Because of this, the following items are not supported in MPSIM:

- Timer0, Timer1, and Timer2 prescalers are capable of accepting clock pulse inputs smaller than Tcy, but these can not be simulated.
- Capture input pulses can be smaller than one Tcy, but can not be simulated.
- PWM output pulse resolution less than 1 Tcy is not supported.
- 8-bit compare will not be supported since the output resolution is limited to T cycles
- In unsynchronized counter mode, clock input smaller than Tcy is not supported
- The oscillator on RC0/RC1 pins is not supported. The user can, how ever, simply use an external clock input for simulation purposes.

In summary, the net result of instruction boundary simulation is that all events get synchronized at instruction boundary and events smaller than one instruction cycle get lost.

TIMER0

Timer0 (and the interrupt it can generate on overflow) is fully supported by MPSIM, and will increment by the internal or external clock. Clock input must have a minimum high time of 1Tcy and a minimum low time of 1Tcy due to stimulus file requirements. The prescaler for Timer0 is made accessible as T0PRE. It can be watched and modified.

TIMER1

Timer1 in its various modes is supported by MPSIM, except when running in counter mode by an external crystal. The interrupt it can be generated on overflow and wake-up from sleep through interrupt are both supported by MPSIM. The prescaler for Timer1 is viewable and modifiable as T1PRE. The external oscillator on RC0/RC1 is not simulated. The user can simply use a clock input (see CK command).

Appendix I. PIC16C73 User's Guide Addendum

TIMER2

Timer2 and the interrupt that can be generated on overflow are fully supported by MPSIM, and both the prescaler and postscaler for Timer2 are viewable and modifiable (T2PRE and T2POS).

CCP1 and CCP2

CAPTURE

MPSIM fully supports capture and the interrupt generated. The prescaler for the CCP module is viewable and modifiable (CCP1PRE).

COMPARE

Compare mode, its interrupt, and the special event trigger (resetting Timer1 if CCP1 and starting A/D Conversion if CCP2) are supported in this version of MPSIM.

PWM

PWM output (resolution greater than 1Tcy only) are supported in this version of MPSIM.

SSP

The Synchronous Serial Port is supported in SPI mode only. The shift register (SSPSR) can be added to the viewscreen, observed and modified. MPSIM currently does not support the I^2C mode.

USART

Timing and interrupt generation is supported. Baud rate generator is supported. Reading and writing of the registers are supported but actual receive or transmit operation is not simulated.

A/D Converter

All the registers, timing function and interrupt generation are implemented. The simulator, however, does not load any meaningful value into A/D result register (ADRES) at the end of a conversion. Use the FI command to load the ADRES register from a file for simulation purposes.

Appendix J. PIC16C74 User's Guide Addendum

Introduction

MPSIM provides support for more than one family of Microchip microcontrollers. This section has been added as an addendum to the MPSIM user's guide to centralize PIC16C74-specific simulator support.

I/O Pins

The PIC16C74 is a 40-pin device, with many of the I/O pins multiplexed with other peripherals (and therefore referred by more than one name). When modifying pins either manually (e.g. with the SE command) or via the stimulus file, use the following pin names only. These are the only ones that MPSIM recognizes as valid I/O pins:

- $\overline{\text{MCLR}}$
- RA0-RA5
- RB0-RB7
- RC0-RC7
- RD0-RD7
- RE0-RE2

Interrupts

MPSIM supports all interrupts on the PIC16C74:

- Timer0 overflow
- Timer1 overflow
- Timer2
- CCP1
- CCP2
- SSP (in SPI mode ONLY)
- Change on Port RB <7..4>
- External interrupt from RB0/INT pin
- A/D interrupt complete
- USART
- Parallel Slave Port

CPU Model

Reset Conditions

All reset conditions are supported by MPSIM.

A Power-On-Reset can be simulated by using the RS instruction. All special-purpose registers will be initialized to the values specified in the data sheet.

A $\overline{\text{MCLR}}$ reset during normal operation or during SLEEP can easily be simulated by driving the $\overline{\text{MCLR}}$ pin low (and then high) via the stimulus file or by using the SE command or by using DK command.

A WDT time-out reset is simulated when WDT is enabled (see DW command) and proper prescaler is set (by initializing OPTION register appropriately) and WDT actually overflows. WDT time-out period (with prescale = 1) is approximated at 18 ms (to closest instruction cycle multiple).

The Time-out ($\overline{\text{TO}}$) and Power-down ($\overline{\text{PD}}$) bits in the Status register reflect appropriate reset condition. This feature is useful for simulating various power-up and time out forks in the user code.

Sleep

MPSIM simulates the SLEEP instruction, and will appear "asleep" until a wake-up from sleep condition occurs. For example, if the Watchdog timer has been enabled, it will wake the processor up from sleep when it times out (depending upon the prescaler setting in the OPTION register). Another example of a wake-up-from-sleep condition, would be Timer1 wake-up from sleep. In this case, when the processor is asleep, Timer1 would continue to increment until it overflows, and if the interrupt is enabled, will wake the processor on overflow and branch to the interrupt vector.

WDT

The Watchdog timer is fully simulated in the MPSIM simulator. Because it is fuse-selectable on the device, it must be enabled by a separate command (see the DW command) in MPSIM. The period of the WDT is determined by the prescaler settings in the OPTION register. The basic period (with prescaler = 1) is approximated at 18 ms (to closest instruction cycle multiple).

Stack

MPSIM presents an accurate simulation of the hardware stack on the PIC16CXX, and additionally provides warning messages if an underflow or overflow condition occurs. When a CALL instruction is encountered, or when an interrupt has occurred, the value of the PC+ 1 is pushed to the stack, and the stack is popped when a RETLW, RETURN, or RETFIE instruction is executed. If more than eight values are pushed to the stack before it is popped, the value will be pushed to the stack, but a warning message will be issued, indicating a stack overflow condition. An error message will also be

generated if the user attempts to pop an empty stack. Popping an empty stack will cause the stack pointer to point to the top of a full stack, and will not generate an error message if another pop is initiated.

Special Registers

To aid in debugging this device, certain items that are normally not observable have been declared as "special" registers. Prescalers and postscalers cannot be declared in your code as "registers", so there are special labels that can be added to the view screen. You can add them as you normally would any other register declared in your code, specifying any radix to view them.

The following are special items that can be added to the view screen when the PIC16C74 has been selected:

- T0PRE - Prescaler for timer0
- T1PRE - Prescaler for timer1
- T2PRE - Prescaler for timer2
- T2POS - Postscaler for timer2
- CCP1PRE - Prescaler for CCP1
- CCP2PRE - Prescaler for CCP2
- SPIPRE - Prescaler for SPI
- SSPSR - SSP Shift register

Please remember that these labels are only available when the PIC16C74 is the target processor, and that they cannot be manually modified.

Peripherals

Peripherals Supported

Along with providing core support, the following peripheral modules (in addition to general-purpose I/O) are supported:

- Timer0
- Timer1
- Timer2
- CCP1
- CCP2
- Parallel Slave Port
- SSP (in SPI Mode only)
- A/D module (limited)
- USART (limited)

Tcycle Limitation

It is important to remember that because MPSIM executes on instruction cycle boundaries, resolutions below 1 Tcy cannot be simulated. Please see the following section for more details concerning the limitations of T-cycle simulation.

MPSIM is a discrete-event simulator where all stimuli are evaluated and all response generated at instruction boundaries or Tcy. One Tcy = 4 Tosc (where Tosc is input clock). Therefore, there are several events that can not be accurately simulated in MPSIM. These fall into two categories:

- Purely asynchronous events
- Synchronous events that occur at Tosc clock boundaries

Because of this, the following items are not supported in MPSIM:

- Timer0, Timer1, and Timer2 prescalers are capable of accepting clock pulse inputs smaller than Tcy, but these can not be simulated.
- Capture input pulses can be smaller than one Tcy, but can not be simulated.
- PWM output pulse resolution less than 1 Tcy is not supported.
- 8-bit compare will not be supported since the output resolution is limited to T cycles
- In unsynchronized counter mode, clock input smaller than Tcy is not supported
- The oscillator on RC0/RC1 pins is not supported. The user can, how ever, simply use an external clock input for simulation purposes.

In summary, the net result of instruction boundary simulation is that all events get synchronized at instruction boundary and events smaller than one instruction cycle get lost.

TIMER0

Timer0 (and the interrupt it can generate on overflow) is fully supported by MPSIM, and will increment by the internal or external clock. Clock input must have a minimum high time of 1Tcy and a minimum low time of 1Tcy due to stimulus file requirements. The prescaler for Timer0 is made accessible as T0PRE. It can be watched and modified.

TIMER1

Timer1 in its various modes is supported by MPSIM, except when running in counter mode by an external crystal. The interrupt it can be generated on overflow and wake-up from sleep through interrupt are both supported by MPSIM. The prescaler for Timer1 is viewable and modifiable as T1PRE. The external oscillator on RC0/RC1 is not simulated. The user can simply use a clock input (see CK command).

Appendix J. PIC16C74 User's Guide Addendum

TIMER2

Timer2 and the interrupt that can be generated on overflow are fully supported by MPSIM, and both the prescaler and postscaler for Timer2 are viewable and modifiable (T2PRE and T2POS).

CCP1 and CCP2

CAPTURE

MPSIM fully supports capture and the interrupt generated. The prescaler for the CCP module is viewable and modifiable (CCP1PRE).

COMPARE

Compare mode, its interrupt, and the special event trigger (resetting Timer1 if CCP1 and starting A/D Conversion if CCP2) are supported in this version of MPSIM.

PWM

PWM output (resolution greater than 1Tcy only) are supported in this version of MPSIM.

SSP

The Synchronous Serial Port is supported in SPI mode only. The shift register (SSPSR) can be added to the viewscreen, observed and modified. MPSIM currently does not support the I^2C mode.

USART

Timing and interrupt generation is supported. Baud rate generator is supported. Reading and writing of the registers are supported but actual receive or transmit operation is not simulated.

A/D Converter

All the registers, timing function and interrupt generation are implemented. The simulator, however, does not load any meaningful value into A/D result register (ADRES) at the end of a conversion. Use the FI command to load the ADRES register from a file for simulation purposes.

Appendix K. PIC16C84 User's Guide Addendum

Introduction

MPSIM provides support for more than one family of Microchip microcontrollers. This section has been added as an addendum to the MPSIM user's guide to centralize PIC16C74-specific simulator support.

I/O Pins

The PIC16C84 is an 18-pin device, with some of the I/O pins multiplexed with other peripherals (and therefore referred by more than one name). When modifying pins either manually (e.g. with the SE command) or via the stimulus file, use the following pin names only. These are the only ones that MPSIM recognizes as valid I/O pins:

- $\overline{\text{MCLR}}$
- RA0-RA4
- RB0-RB7

Interrupts

MPSIM supports all interrupts on the PIC16C71:

- Timer0 overflow
- Change on Port RB <7:4>
- External interrupt from RB0/INT pin
- EEPROM write complete

CPU Model

Reset Conditions

All reset conditions are supported by MPSIM.

A Power-On-Reset can be simulated by using the RS instruction. All special-purpose registers will be initialized to the values specified in the data sheet.

A $\overline{\text{MCLR}}$ reset during normal operation or during SLEEP can easily be simulated by driving the $\overline{\text{MCLR}}$ pin low (and then high) via the stimulus file or by using the SE command or by using DK command.

A WDT time-out reset is simulated when WDT is enabled (see DW command) and proper prescaler is set (by initializing OPTION register appropriately) and WDT actually overflows. WDT time-out period (with prescale = 1) is approximated at 18 ms (to closest instruction cycle multiple).

The Time-out ($\overline{TO}$) and Power-down ($\overline{PD}$) bits in the Status register reflect appropriate reset condition. This feature is useful for simulating various power-up and time out forks in the user code.

Sleep

MPSIM simulates the SLEEP instruction, and will appear "asleep" until a wake-up from sleep condition occurs. For example, if the Watchdog timer has been enabled, it will wake the processor up from sleep when it times out (depending upon the prescaler setting in the OPTION register). Another example of a wake-up-from-sleep condition, would be due to RB0/INT interrupt wake-up.

WDT

The Watchdog timer is fully simulated in the MPSIM simulator. Because it is fuse-selectable on the device, it must be enabled by a separate command (see the DW command) in MPSIM. The period of the WDT is determined by the prescaler settings in the OPTION register. The basic period (with prescaler = 1) is approximated at 18 ms (to closest instruction cycle multiple).

Stack

MPSIM presents an accurate simulation of the hardware stack on the PIC16CXX, and additionally provides warning messages if an underflow or overflow condition occurs. When a CALL instruction is encountered, or when an interrupt has occurred, the value of the PC+ 1 is pushed to the stack, and the stack is popped when a RETLW, RETURN, or RETFIE instruction is executed. If more than eight values are pushed to the stack before it is popped, the value will be pushed to the stack, but a warning message will be issued, indicating a stack overflow condition. An error message will also be generated if the user attempts to pop an empty stack. Popping an empty stack will cause the stack pointer to point to the top of a full stack, and will not generate an error message if another pop is initiated.

Special Registers

To aid in debugging this device, certain items that are normally not observable have been declared as "special" registers. Prescalers and postscalers cannot be declared in your code as "registers", so there are special labels that can be added to the view screen. You can add them as you normally would any other register declared in your code, specifying any radix to view them.

The following are special items that can be added to the view screen when the PIC16C84 has been selected:

- T0PRE - Prescaler for timer0

Please remember that these labels are only available when the PIC16C84 is the target processor, and that they cannot be manually modified.

Appendix K. PIC16C84 User's Guide Addendum

Peripherals

Peripherals Supported

Along with providing core support, the following peripheral modules (in addition to general-purpose I/O) are supported:

- Timer0
- EEPROM data memory

Tcycle Limitation

It is important to remember that because MPSIM executes on instruction cycle boundaries, resolutions below 1 Tcy cannot be simulated. Please see the following section for more details concerning the limitations of T-cycle simulation.

MPSIM is a discrete-event simulator where all stimuli are evaluated and all response generated at instruction boundaries or Tcy. One Tcy = 4 Tosc (where Tosc is input clock). Therefore, there are several events that can not be accurately simulated in MPSIM. These fall into two categories:

- Purely asynchronous events
- Synchronous events that occur at Tosc clock boundaries

Because of this, the following items are not supported in MPSIM:

- Timer0 prescaler is capable of accepting clock pulse inputs smaller than Tcy, but this can not be simulated.

In summary, the net result of instruction boundary simulation is that all events get synchronized at instruction boundary and events smaller than one instruction cycle get lost.

TIMER0

Timer0 (and the interrupt it can generate on overflow) is fully supported by MPSIM, and will increment by the internal or external clock. Clock input must have a minimum high time of 1Tcy and a minimum low time of 1Tcy due to stimulus file requirements. The prescaler for Timer0 is made accessible as T0PRE. It can be watched and modified.

EEPROM Data Memory

The EEPROM data memory is fully simulated. The registers and the read/write cycles are fully implemented. The write cycle time is approximated to 10 ms (to nearest instruction cycle multiple).

Please note that whereas the write to EEPROM is supported, the simulator does not check for "the valid instruction sequence". The simulator does, however, simulate functions of WRERR and WREN control bits in the EECON1 register.

Appendix L. PIC17C42 Support

Introduction

MPSIM provides support for more than one family of Microchip microcontrollers. This section has been added as an addendum to the MPSIM user's guide to centralize PIC17C42-specific simulator support.

I/O Pins

The PIC17C42 is a 40-pin device, with many of the I/O pins multiplexed with other peripherals (and therefore referred by more than one name). When modifying pins either manually (with the SE command) or via the stimulus file, use the following pin names only. These are the only ones that MPSIM recognizes as valid I/O pins:

- $\overline{\text{MCLR}}$
- RA0-RA5
- RB0-RB7
- RC0-RC7
- RD0-RD7
- RE0-RE2

Special Function Registers

Many special-function registers in the PIC17CXX family (specifically the "peripheral registers") are located in register banks other than bank zero. To access these registers in your program, you must first select the desired bank and then specify the address within that bank (0x10 - 0x17). Because of this, the "porta" register (address 0x10 in bank 0), for example, and the "ddrc" (address 0x10 in bank 1) registers would both be defined in your source code as addresses 0x10.

In order to distinguish between labels that have the same address, MPSIM has pre-defined the following labels with file register addresses and has added them to its internal symbol table:

DDRC	TMR1	PW1DCL
PORTC	TMR2	PW2DCL
DDRD	TMR3L	PW1DCH
PORTD	TMR3H	PW2DCH
DDRE	PR1	CA2L
PORTE	PR2	CA2H
PIR	PR3L	TCON1
PIE	PR3H	TCON2

If you want to view the contents of any of these registers during your simulation session, you can add them to the viewscreen by using the "AD" command.

Interrupts

MPSIM Version 5.0 or greater supports all interrupts on the PIC17C42:

- External interrupt on INT pin
- TMR0 overflow interrupt
- External interrupt on RA0 pin
- Port B input change interrupt
- Timer/Counter1 interrupt
- Timer/Counter2 interrupt
- Timer/Counter3 interrupt
- Capture1 interrupt
- Capture2 Interrupt
- Serial port transmit interrupt*
- Serial port receive interrupt*

*Serial port timing only

CPU Model

Reset Conditions

All reset conditions are supported by MPSIM

A Power-On-Reset can be simulated by using the RS instruction. all special-purpose registers will be initialized to the values specified in the data sheet.

A $\overline{\text{MCLR}}$ reset during normal operation or during SLEEP can easily be simulated by driving the $\overline{\text{MCLR}}$ pin low (and then high) via the stimulus file, by using the SE command, or by using the DK command.

A WDT time-out reset is simulated when the WDT is enabled (see DW command) and the proper prescaler is set (see the FW command) and the WDT actually overflows. WDT time-out period is approximated at 12 ms (to closest instruction cycle multiple) but can be changed by using the WP command.

The Time out ($\overline{\text{TO}}$) and Power-Down ($\overline{\text{PD}}$) bits in the ALUSTA register reflect appropriate reset condition. This feature is useful for simulating various power-up and time-out forks in the user code.

Sleep

MPSIM simulates the SLEEP instruction and will appear "asleep" until a wake-up from sleep condition occurs. For example, if the Watchdog timer has been enabled, it will wake the processor up from sleep when it times out (depending on the fuse setting by the FW command). Another example of a wake-up-from-sleep condition, would be an input change on PORTB. If the

interrupt is enabled and the GLINTD bit is set, the processor will wake-up and will resume executing from the instruction following the SLEEP command. If the GLINTD = 0, the normal interrupt response will take place.

WDT

The Watchdog timer is fully simulated in the MPSIM simulator. Because it is fuse-selectable and fuse-configurable on the device, it must be enabled and configured by separate commands (see the DW and the FW commands) in MPSIM. The basic period of the WDT (with prescaler = 1) is approximated at 12ms (to closest instruction cycle multiple) but can also be changed via the WP command.

Stack

MPSIM presents an accurate simulation of the hardware stack on the PIC17CXX, and additionally provides warning messages if an underflow or overflow condition occurs. When a CALL or LCALL instruction is encountered or when an interrupt has occurred, the value of the PC+ 1 is pushed to the stack. The stack is popped when a RETLW, RETURN, or RETFIE instruction is executed. If more than sixteen values are pushed to the stack before it is popped, the value will be pushed to the stack, a warning message will be issued indicating a stack overflow condition, and the STAKAVL bit will be cleared until a reset condition occurs

Instruction Set

The entire PIC17CXX instruction set is supported, including pre-increment and post-increment of indirect-address registers (according to their configuration). The TABLRD and TABLWT instructions are also fully supported, including long writes for the TABLWT instruction.

Special Registers

To aid in debugging this device, certain items that are normally not observable have been declared as "special" registers. Prescalers cannot be declared in user code as "registers", so there ar special labels that can be added to the view screen. You can add them as you normally would any other register declared in your code, specifying any radix to view them.

The following special items can be added to the view screen when the PIC17C42 has been selected:

T0PRE (Prescaler for Timer 0)

WDTPRE (Prescaler for WDT)

Peripherals

Along with providing core support, the following peripheral modules (in addition to general-purpose I/O) are supported:

- Timer 0 in both internal and external clock modes
- Timer1 and Timer2 (and their respective period registers)
- Timer3
- Two Capture Modules
- Two PWM Modules
- USART (limited)

Tcycle Limitation

It is important to remember that because MPSIM executes on instruction cycle boundaries, resolutions below 1Tcy cannot be simulated. Please see the following section for more details concerning the limitations of T-cycle simulation.

MPSIM is a discrete-event simulator where all stimuli are evaluated an all response generated at instruction boundaries or Tcy. One Tcy = 4 Tosc (where Tosc is input clock). Therefor, there are several events that can not be accurately simulated in MPSIM. These fall into two categories:

- Purely asynchronous events
- Synchronous events that occur at Tosc clock boundaries

Because of this, the following items are not supported in MPSIM:

- Timer0 prescaler is capable of accepting clock pulse inputs smaller than Tcy, but these can not be simulated.
- Capture input pulses can be smaller than one Tcy, but can not be simulated.
- PWM output pulse resolution less than 1Tcy is not supported
- In unsynchronized counter mode, clock input smaller than Tcy is not supported.

In summary, the net result of instruction boundary simulation is that all events get synchronized at instruction boundary and events smaller than one instruction cycle get lost.

TIMER0

Timer0 (and the interrupt it can generate on overflow) is fully supported by MPSIM, and will increment by the internal or external clock. Delay from external clock edge to timer increment has also been simulated, as well as the interrupt latency period. Clock input must have a minimum high time of 1Tcy and a minimum low time of 1Tcy due to the stimulus file requirements. The prescaler for Timer0 is made accessible as T0PRE. It can be watched and modified.

TIMER1 and TIMER2

Timer1 and Timer2 in its various modes is fully supported by MPSIM. Delays from clock edge to increment (when configured to increment from rising or falling edge of external clock) is simulated as well as the interrupt latency periods. Clock input must have a minimum high time of 1Tcy and a minimum low time of 1Tcy due to the stimulus file requirements.

TIMER3 and Capture

MPSIM fully supports Timer3 and the Capture module in all of its modes. Delays from clock edge to increment (when configured in external mode), delay for capture and interrupt latency periods are fully supported. Clock input must have a minimum high time of 1Tcy and a minimum low time of 1Tcy due to the stimulus file requirements.

PWM

Both PWM outputs are supported (resolution greater than 1Tcy only) are supported in this version of MPSIM.

USART

Timing and interrupt generation is supported. Baud rate generator is supported. Reading and writing of the registers are supported but actual receive or transmit operation is not simulated.

Memory Modes

The following memory modes are supported by MPSIM:

- Microcontroller Mode
- Extended Microcontroller Mode
- Microprocessor Mode

The default is Microcontroller mode, which has 2K of program-memory on-chip. If you would like to use any of the other modes, you must use the FW command (since this option is fuse-selectable on-chip).

Appendix M. PIC17C43 Support

Introduction

MPSIM provides support for more than one family of Microchip microcontrollers. This section has been added as an addendum to the MPSIM user's guide to centralize PIC17C43-specific simulator support.

I/O Pins

The PIC17C43 is a 40-pin device, with many of the I/O pins multiplexed with other peripherals (and therefore referred by more than one name). When modifying pins either manually (with the SE command) or via the stimulus file, use the following pin names only. These are the only ones that MPSIM recognizes as valid I/O pins:

- $\overline{\text{MCLR}}$
- RA0-RA5
- RB0-RB7
- RC0-RC7
- RD0-RD7
- RE0-RE2

Special Function Registers

Many special-function registers in the 17CXX family (specifically the "peripheral registers") are located in register banks other than bank zero. To access these registers in your program, you must first select the desired bank and then specify the address within that bank (0x10 - 0x17). Because of this, the "porta" register (address 0x10 in bank 0), for example, and the "ddrc" (address 0x10 in bank 1) registers would both be defined in your source code as addresses 0x10.

In order to distinguish between labels that have the same address, MPSIM has pre-defined the following labels with file register addresses and has added them to its internal symbol table:

DDRC	TMR1	PW1DCL
PORTC	TMR2	PW2DCL
DDRD	TMR3L	PW1DCH
PORTD	TMR3H	PW2DCH
DDRE	PR1	CA2L
PORTE	PR2	CA2H
PIR	PR3L	TCON1
PIE	PR3H	TCON2

If you want to view the contents of any of these registers during your simulation session, you can add them to the viewscreen by using the "AD" command.

Interrupts

MPSIM Version 5.0 or greater supports all interrupts on the PIC17C43:

- External interrupt on INT pin
- TMR0 overflow interrupt
- External interrupt on RA0 pin
- Port B input change interrupt
- Timer/Counter1 interrupt
- Timer/Counter2 interrupt
- Timer/Counter3 interrupt
- Capture1 interrupt
- Capture2 Interrupt
- Serial port transmit interrupt*
- Serial port receive interrupt*

*Serial port timing only

CPU Model

Reset Conditions

All reset conditions are supported by MPSIM

A Power-On-Reset can be simulated by using the RS instruction. all special-purpose registers will be initialized to the values specified in the data sheet.

A $\overline{\text{MCLR}}$ reset during normal operation or during SLEEP can easily be simulated by driving the $\overline{\text{MCLR}}$ pin low (and then high) via the stimulus file, by using the SE command, or by using the DK command.

A WDT time-out reset is simulated when the WDT is enabled (see DW command) and the proper prescaler is set (see the FW command) and the WDT actually overflows. WDT time-out period is approximated at 12 ms (to closest instruction cycle multiple) but can be changed by using the WP command.

The Time out ($\overline{\text{TO}}$) and Power-Down ($\overline{\text{PD}}$) bits in the ALUSTA register reflect appropriate reset condition. This feature is useful for simulating various power-up and time-out forks in the user code.

Sleep

MPSIM simulates the SLEEP instruction and will appear "asleep" until a wake-up from sleep condition occurs. For example, if the Watchdog timer has been enabled, it will wake the processor up from sleep when it times out (depending on the fuse setting by the FW command). Another example of a wake-up-from-sleep condition, would be an input change on PORT B. If the interrupt is enabled and the GLINTD bit is set, the processor will wake-up and will resume executing from the instruction following the SLEEP command. If the GLINTD = 0, the normal interrupt response will take place.

WDT

The Watchdog timer is fully simulated in the MPSIM simulator. Because it is fuse-selectable and fuse-configurable on the device, it must be enabled and configured by separate commands (see the DW and the FW commands) in MPSIM. The basic period of the WDT (with prescaler = 1) is approximated at 12ms (to closest instruction cycle multiple) but can also be changed via the WP command.

Stack

MPSIM presents an accurate simulation of the hardware stack on the PIC17CXX, and additionally provides warning messages if an underflow or overflow condition occurs. When a CALL or LCALL instruction is encountered or when an interrupt has occurred, the value of the PC+ 1 is pushed to the stack. The stack is popped when a RETLW, RETURN, or RETFIE instruction is executed. If more than sixteen values are pushed to the stack before it is

popped, the value will be pushed to the stack, a warning message will be issued indicating a stack overflow condition, and the STAKAVL bit will be cleared until a reset condition occurs

Instruction Set

The entire PIC17CXX instruction set is supported, including pre-increment and post-increment of indirect-address registers (according to their configuration). The TABLRD and TABLWT instructions are also fully supported, including long writes for the TABLWT instruction. The hardware multiply instructions, MULLW and MULLWF are both fully supported as is the MOVLP instruction.

Special Registers

To aid in debugging this device, certain items that are normally not observable have been declared as "special" registers. Prescalers cannot be declared in user code as "registers", so there ar special labels that can be added to the view screen. You can add them as you normally would any other register declared in your code, specifying any radix to view them.

The following special item can be added to the view screen when the PIC17C43 has been selected:

T0PRE (Prescaler for Timer 0)

WDTPRE (Prescaler for WDT)

Peripherals

Along with providing core support, the following peripheral modules (in addition to general-purpose I/O) are supported:

- Timer 0 in both internal and external clock modes
- Timer1 and Timer2 (and their respective period registers)
- Timer3
- Two Capture Modules
- Two PWM Modules
- USART (limited)

Tcycle Limitation

It is important to remember that because MPSIM executes on instruction cycle boundaries, resolutions below 1Tcy cannot be simulated. Please see the following section for more details concerning the limitations of T-cycle simulation.

MPSIM is a discrete-event simulator where all stimuli are evaluated an all response generated at instruction boundaries or Tcy. One Tcy = 4 Tosc (where Tosc is input clock). Therefore, there are several events that can not be accurately simulated in MPSIM. These fall into two categories:

- Purely asynchronous events
- Synchronous events that occur at Tosc clock boundaries

Because of this, the following items are not supported in MPSIM:

- Timer0 prescaler is capable of accepting clock pulse inputs smaller than Tcy, but these can not be simulated.
- Capture input pulses can be smaller than one Tcy, but can not be simulated.
- PWM output pulse resolution less than 1Tcy is not supported
- In unsynchronized counter mode, clock input smaller than Tcy is not supported.

In summary, the net result of instruction boundary simulation is that all events get synchronized at instruction boundary and events smaller than one instruction cycle get lost.

TIMER0

Timer0 (and the interrupt it can generate on overflow) is fully supported by MPSIM, and will increment by the internal or external clock. Delay from external clock edge to timer increment has also been simulated, as well as the interrupt latency period. Clock input must have a minimum high time of 1Tcy and a minimum low time of 1Tcy due to the stimulus file requirements. The prescaler for Timer0 is made accessible as T0PRE. It can be watched and modified.

TIMER1 and TIMER2

Timer1 and Timer2 in its various modes is fully supported by MPSIM. Delays from clock edge to increment (when configured to increment from rising or falling edge of external clock) is simulated as well as the interrupt latency periods. Clock input must have a minimum high time of 1Tcy and a minimum low time of 1Tcy due to the stimulus file requirements.

TIMER3 and Capture

MPSIM fully supports Timer3 and the Capture module in all of its modes. Delays from clock edge to increment (when configured in external mode), delay for capture and interrupt latency periods are fully supported. Clock input must have a minimum high time of 1Tcy and a minimum low time of 1Tcy due to the stimulus file requirements.

PWM

Both PWM outputs are supported (resolution greater than 1Tcy only) are supported in this version of MPSIM.

USART

Timing and interrupt generation is supported. Baud rate generator is supported. Reading and writing of the registers are supported but actual receive or transmit operation is not simulated.

Memory Modes

The following memory modes are supported by MPSIM:

- Microcontroller Mode
- Extended Microcontroller Mode
- Microprocessor Mode

The default is Microcontroller mode, which has 4K of program-memory on-chip. If you would like to use any of the other modes, you must use the FW command (since this option is fuse-selectable on-chip).

Appendix N. PIC17C44 Support

Introduction

MPSIM provides support for more than one family of Microchip microcontrollers. This section has been added as an addendum to the MPSIM user's guide to centralize PIC17C44-specific simulator support.

I/O Pins

The PIC17C44 is a 40-pin device, with many of the I/O pins multiplexed with other peripherals (and therefore referred by more than one name). When modifying pins either manually (with the SE command) or via the stimulus file, use the following pin names only. These are the only ones that MPSIM recognizes as valid I/O pins:

- $\overline{\text{MCLR}}$
- RA0-RA5
- RB0-RB7
- RC0-RC7
- RD0-RD7
- RE0-RE2

Special Function Registers

Many special-function registers in the PIC17CXX family (specifically the "peripheral registers") are located in register banks other than bank zero. To access these registers in your program, you must first select the desired bank and then specify the address within that bank (0x10 - 0x17). Because of this, the "porta" register (address 0x10 in bank 0), for example, and the "ddrc" (address 0x10 in bank 1) registers would both be defined in your source code as addresses 0x10.

In order to distinguish between labels that have the same address, MPSIM has pre-defined the following labels with file register addresses and has added them to its internal symbol table:

DDRC	TMR1	PW1DCL
PORTC	TMR2	PW2DCL
DDRD	TMR3L	PW1DCH
PORTD	TMR3H	PW2DCH
DDRE	PR1	CA2L
PORTE	PR2	CA2H
PIR	PR3L	TCON1
PIE	PR3H	TCON2

If you want to view the contents of any of these registers during your simulation session, you can add them to the viewscreen by using the "AD" command.

Interrupts

MPSIM Version 5.0 or greater supports all interrupts on the PIC17C44:

- External interrupt on INT pin
- TMR0 overflow interrupt
- External interrupt on RT pin
- Port B input change interrupt
- Timer/Counter1 interrupt
- Timer/Counter2 interrupt
- Timer/Counter3 interrupt
- Capture1 interrupt
- Capture2 Interrupt
- Serial port transmit interrupt*
- Serial port receive interrupt*

*Serial port timing only

CPU Model

Reset Conditions

All reset conditions are supported by MPSIM

A Power-On-Reset can be simulated by using the RS instruction. all special-purpose registers will be initialized to the values specified in the data sheet.

A $\overline{\text{MCLR}}$ reset during normal operation or during SLEEP can easily be simulated by driving the $\overline{\text{MCLR}}$ pin low (and then high) via the stimulus file, by using the SE command, or by using the DK command.

A WDT time-out reset is simulated when the WDT is enabled (see DW command) and the proper prescaler is set (see the FW command) and the WDT actually overflows. WDT time-out period is approximated at 12 ms (to closest instruction cycle multiple) but can be changed by using the WP command.

The Time out ($\overline{\text{TO}}$) and Power-Down ($\overline{\text{PD}}$) bits in the ALUSTA register reflect appropriate reset condition. This feature is useful for simulating various power-up and time-out forks in the user code.

Sleep

MPSIM simulates the SLEEP instruction and will appear "asleep" until a wake-up from sleep condition occurs. For example, if the Watchdog timer has been enabled, it will wake the processor up from sleep when it times out (depending on the fuse setting by the FW command). Another example of a

wake-up-from-sleep condition, would be an input change on PORT B. If the interrupt is enabled and the GLINTD bit is set, the processor will wake-up and will resume executing from the instruction following the SLEEP command. If the GLINTD = 0, the normal interrupt response will take place.

WDT

The Watchdog timer is fully simulated in the MPSIM simulator. Because it is fuse-selectable and fuse-configurable on the device, it must be enabled and configured by separate commands (see the DW and the FW commands) in MPSIM. The basic period of the WDT (with prescaler = 1) is approximated at 12ms (to closest instruction cycle multiple) but can also be changed via the WP command.

Stack

MPSIM presents an accurate simulation of the hardware stack on the PIC17CXX, and additionally provides warning messages if an underflow or overflow condition occurs. When a CALL or LCALL instruction is encountered or when an interrupt has occurred, the value of the PC+ 1 is pushed to the stack. The stack is popped when a RETLW, RETURN, or RETFIE instruction is executed. If more than sixteen values are pushed to the stack before it is popped, the value will be pushed to the stack, a warning message will be issued indicating a stack overflow condition, and the STAKAVL bit will be cleared until a reset condition occurs

Instruction Set

The entire PIC17CXX instruction set is supported, including pre-increment and post-increment of indirect-address registers (according to their configuration). The TABLRD and TABLWT instructions are also fully supported, including long writes for the TABLWT instruction. The hardware multiply instructions, MULLW and MULLWF are both fully supported as is the MOVLP instruction.

Special Registers

To aid in debugging this device, certain items that are normally not observable have been declared as "special" registers. Prescalers cannot be declared in user code as "registers", so there ar special labels that can be added to the view screen. You can add them as you normally would any other register declared in your code, specifying any radix to view them.

The following special item can be added to the view screen when the PIC17C44 has been selected:

T0PRE (Prescaler for Timer 0)

WDTPRE (Prescaler for WDT)

Peripherals

Along with providing core support, the following peripheral modules (in addition to general-purpose I/O) are supported:

- Timer 0 in both internal and external clock modes
- Timer1 and Timer2 (and their respective period registers)
- Timer3
- Two Capture Modules
- Two PWM Modules
- USART (limited)

Tcycle Limitation

It is important to remember that because MPSIM executes on instruction cycle boundaries, resolutions below 1Tcy cannot be simulated. Please see the following section for more details concerning the limitations of T-cycle simulation.

MPSIM is a discrete-event simulator where all stimuli are evaluated an all response generated at instruction boundaries or Tcy. One Tcy = 4 Tosc (where Tosc is input clock). Therefor, there are several events that can not be accurately simulated in MPSIM. These fall into two categories:

- Purely asynchronous events
- Synchronous events that occur at Tosc clock boundaries

Because of this, the following items are not supported in MPSIM:

- Timer0 prescaler is capable of accepting clock pulse inputs smaller than Tcy, but these can not be simulated.
- Capture input pulses can be smaller than one Tcy, but can not be simulated.
- PWM output pulse resolution less than 1Tcy is not supported
- In unsynchronized counter mode, clock input smaller than Tcy is not supported.

In summary, the net result of instruction boundary simulation is that all events get synchronized at instruction boundary and events smaller than one instruction cycle get lost.

TIMER0

Timer0 (and the interrupt it can generate on overflow) is fully supported by MPSIM, and will increment by the internal or external clock. Delay from external clock edge to timer increment has also been simulated, as well as the interrupt latency period. Clock input must have a minimum high time of 1Tcy and a minimum low time of 1Tcy due to the stimulus file requirements. The prescaler for Timer0 is made accessible as T0PRE. It can be watched and modified.

TIMER1 and TIMER2

Timer1 and Timer2 in its various modes is fully supported by MPSIM. Delays from clock edge to increment (when configured to increment from rising or falling edge of external clock) is simulated as well as the interrupt latency periods. Clock input must have a minimum high time of 1Tcy and a minimum low time of 1Tcy due to the stimulus file requirements.

TIMER3 and Capture

MPSIM fully supports Timer3 and the Capture module in all of its modes. Delays from clock edge to increment (when configured in external mode), delay for capture and interrupt latency periods are fully supported. Clock input must have a minimum high time of 1Tcy and a minimum low time of 1Tcy due to the stimulus file requirements.

PWM

Both PWM outputs are supported (resolution greater than 1Tcy only) are supported in this version of MPSIM.

USART

Timing and interrupt generation is supported. Baud rate generator is supported. Reading and writing of the registers are supported but actual receive or transmit operation is not simulated.

Memory Modes

The following memory modes are supported by MPSIM:

- Microcontroller Mode
- Extended Microcontroller Mode
- Microprocessor Mode

The default is Microcontroller mode, which has 8K of program-memory on-chip. If you would like to use any of the other modes, you must use the FW command (since this option is fuse-selectable on chip).

WORLDWIDE SALES & SERVICE

AMERICAS

Corporate Office
Microchip Technology Inc.
2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 602 786-7200
Fax: 602 786-7277

Atlanta
Microchip Technology Inc.
500 Sugar Mill Road, Suite 200B
Atlanta, GA 30350
Tel: 404 640-0034
Fax: 404 640-0307

Boston
Microchip Technology Inc.
Five The Mountain Road, Suite 120
Framingham, MA 01701
Tel: 508 820-3334
Fax: 508 820-4326

Chicago
Microchip Technology Inc.
333 Pierce Road, Suite 180
Itasca, IL 60143
Tel: 708 285-0071
Fax: 708 285-0075

Dallas
Microchip Technology Inc.
14651 Dallas Parkway, Suite 816
Dallas, TX 75240-8809
Tel: 214 991-7177
Fax: 214 991-8588

Los Angeles
Microchip Technology Inc.
18201 Von Karman, Suite 455
Irvine, CA 92715
Tel: 714 263-1888
Fax: 714 263-1338

AMERICAS (continued)

New York
Microchip Technology Inc.
150 Motor Parkway, Suite 416
Hauppauge, NY 11788
Tel: 516 273-5305
Fax: 516 273-5335

San Jose
Microchip Technology Inc.
2107 North First Street, Suite 590
San Jose, CA 95131
Tel: 408 436-7950
Fax: 408 436-7955

ASIA/PACIFIC

Hong Kong
Microchip Technology Inc.
Unit No. 3002-3004, Tower 1
Metroplaza
223 Hing Fong Road
Kwai Fong, N.T. Hong Kong
Tel: 852 2 401 1200
Fax: 852 2 401 3431

Korea
Microchip Technology Korea
168-1, Youngbo Bldg. 3 Floor
Samsung-Dong, Kangnam-Ku,
Seoul, Korea
Tel: 82 2 554 7200
Fax: 82 2 558 5934

Taiwan
Microchip Technology Taiwan
10F-1C 207
Tung Hua North Road
Taipei, Taiwan, ROC
Tel: 886 2 717 7175
Fax: 886 2 545 0139

EUROPE

United Kingdom
Arizona Microchip Technology Ltd.
Unit 6, The Courtyard
Meadow Bank, Furlong Road
Bourne End, Buckinghamshire SL8 5AJ
Tel: 44 0 1628 851077
Fax: 44 0 1628 850259

France
Arizona Microchip Technology SARL
2 Rue du Buisson aux Fraises
91300 Massy - France
Tel: 33 1 69 53 63 20
Fax: 33 1 69 30 90 79

Germany
Arizona MicrochipTechnology GmbH
Gustav-Heinemann-Ring 125
D-81739 Muenchen, Germany
Tel: 49 89 627 144 0
Fax: 49 89 627 144 44

Italy
Arizona Microchip Technology SRL
Centro Direzionale Colleoni
Palazzo Pegaso Ingresso No. 2
Via Paracelso 23, 20041 Agrate Brianza
(MI) Italy
Tel: 39 039 689 9939
Fax: 39 039 689 9883

JAPAN

Microchip Technology Intl. Inc.
Benex S-1 6F
3-18-20, Shin Yokohama
Kohoku-Ku, Yokohama
Kanagawa 222 Japan
Tel: 81 45 471 6166
Fax: 81 45 471 6122

MICROCHIP

MICROCHIP

Microchip Technology Inc.
2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 602.786.7200 Fax: 602.899.9210

DS30027