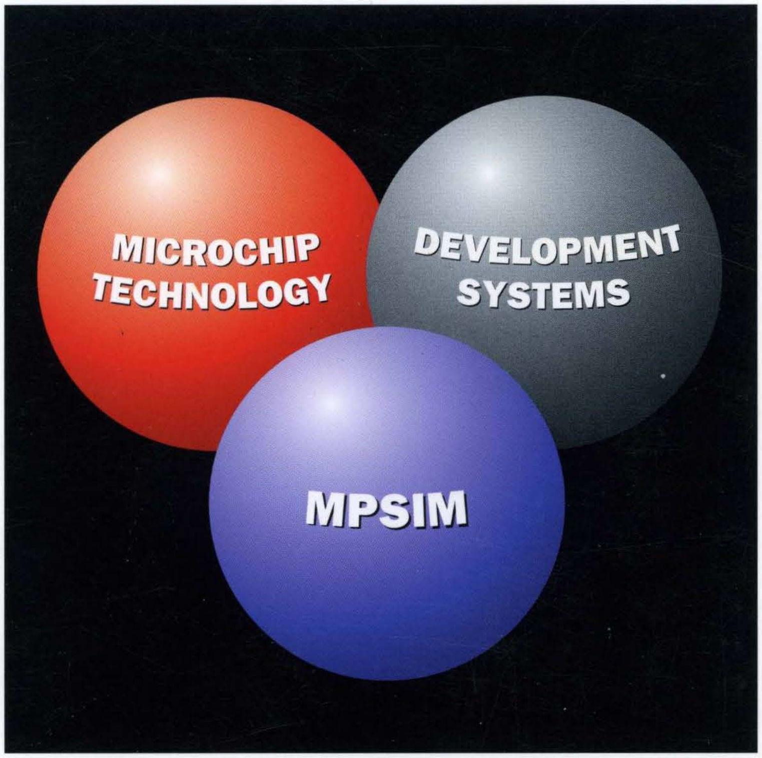
MICROCHIP MPSIM USER'S GUIDE

MPSIM SIMULATOR



USER'S GUIDE

MICROCHIP

Table of Contents

Preface

MPSIM is a discrete-event simulator designed to imitate operation of Microchip Technology's PIC16C5X and PIC16CXX families of microcontrollers. Device specific information is provided in the appendices at the end of the manual. It is a tool that has been created to assist the user in debugging software that will use any of these parts.

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 the user 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: cost, I/O timing, and execution speed.

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.

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.

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.

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.

Chapter 1. Introduction

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

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
- Documentation conventions
- Device-specific support
- Customer support information.

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 or compatible running DOS version 3.0 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
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

Break Points

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 "Getting Started" 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 v. 4.x 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, which contains a troubleshooting guide that provides some common error messages and their possible causes. Appendix B 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 in Appendix C.

Chapter 2. The MPSIM Environment

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.

The following topics will be covered:

- Layout of the User Interface
- I/O Pins
- CPU model including reset, sleep, WDT, registers and stack
- Files Used and Generated by MPSIM
- Object-Code Formats
- Invoking MPSIM

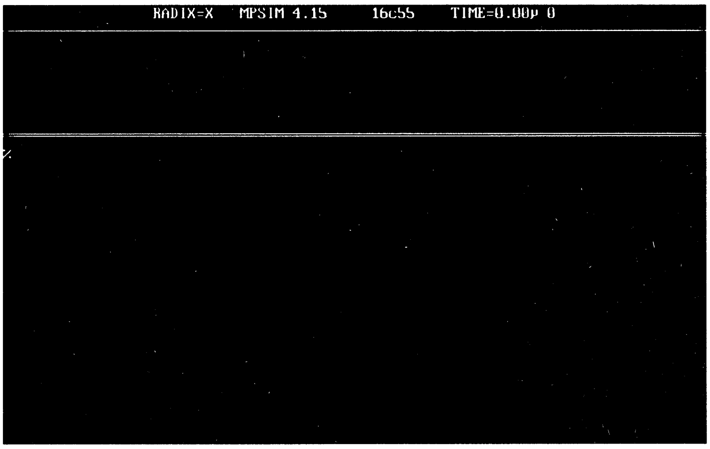
User Interface

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 object file, the radix, the MPSIM version, the controller being simulated, cycle steps and elapsed time.

Title Line

View Screen

Command Entry



```
            RADIX=X   MPSIM 4.15     16c55     TIME=0.00µ 0
```

Figure 2.1 - Start-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
-c	MPASM assembler support	MPASM
-s	MPALC assembler support	MPASM
-a	ASCII only	off

Invoking MPSIM

The '%' is MPSIM's prompt. Enter an object code filename. If you do not specify the extension, MPSIM will assume .OBJ for MPALC and .HEX for MPASM. To load a file into the simulator, use the following command:

```
%LO filename [FORMAT] <RETURN>
```

I/O Pins

There is a list of viewable and modifiable pins for each microcontroller in its appendix. These pin names 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.

* 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 RTCC/TMR0 pin is changed, the RTCC/TMR0 timer register increments in accordance with the prescaler.
- If the $\overline{\text{MCLR}}$ pin is cleared, MPSIM simulates a $\overline{\text{MCLR}}$ reset.
- The TRIS status register 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 PIC16C74 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) and WDT actually overflows. WDT time-out period is approximated at 18 ms (to closest instruction cycle multiple).

The Time-out (TO) and Power-down (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 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.

Wake-up from SLEEP through interrupt is fully simulated in the PIC16CXX 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 in the OPTION register. The basic period (with prescaler = 1) is approximated at 18 ms (to closest instruction cycle multiple).

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
TRISB	Tris register for Port B
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 family however, supports 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 (0x004). 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 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 .OBJ 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 node 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	RTCC	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 4
9	1	1	0	1	0	! Injected before cycle 9
10	0	1	0	1	1	! Injected before cycle 10
15	0	0	0	0	0	! Injected before cycle 15
16	1	0	0	0	1	! Injected before cycle 16

Figure 2.2 - Stimulus File

Files Generated by the Assembler

The MPALC and MPASM assemblers generate by default all files necessary, for use with MPSIM. To assemble a file, invoke MPALC or MPASM with the source file name as follows:

MPALC *filename*

or

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 Object File

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

Output Object File

At any time during simulation, the contents of the program memory can written to an external file with the O command. The object 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 object code formats as dictated by the LO and O commands: INHX8S or INHX8M. The default object-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 object 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

Overview

This chapter provides an introduction to MPSIM, the discrete-event simulator for Microchip Technology's PIC16C5X and PIC16CXX 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 v. 1.X and MPSIM v. 4.x 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".

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.

Assemble the Code

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

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.

MPALC ***sample /p 16C54*** **<RETURN>**
or
MPASM ***sample /p16C54*** **<RETURN>**

Invoke the Simulator

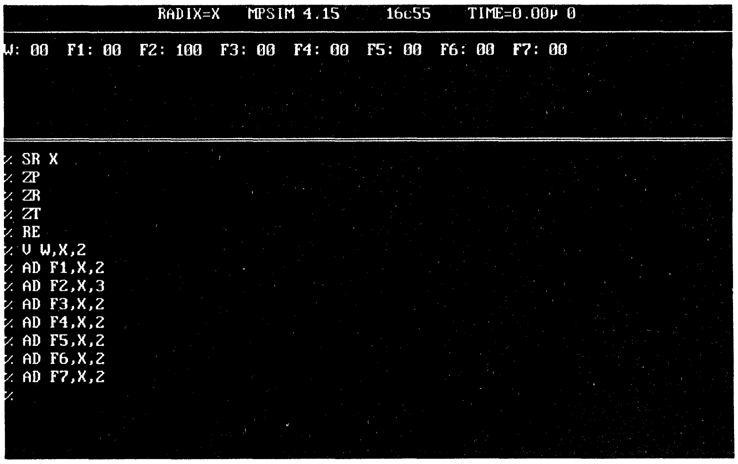
To invoke the simulator, simply type

Step 2.

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 4.15     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.

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

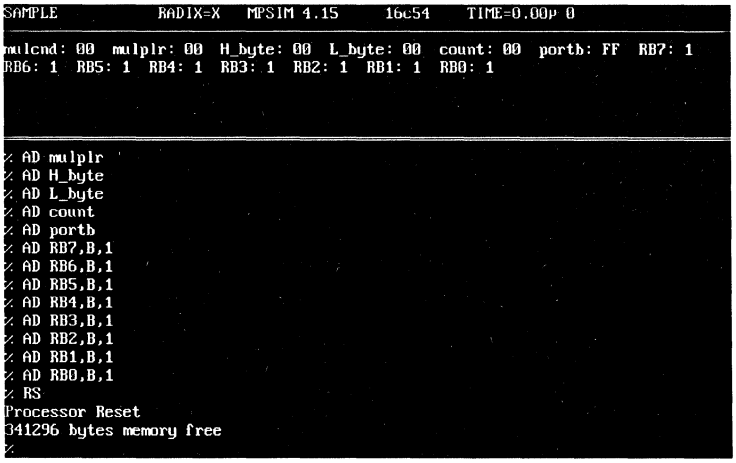
Step 3. **GE *sample.ini* <RETURN>**

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.OBJ uses, in the most useful format.



```
SAMPLE            RADIX=X   MPSIM 4.15      16c54     TIME=0.00µ 0

mulcnd: 00  mulplr: 00  H_byte: 00  L_byte: 00  count: 00  portb: FF  RB7: 1
RB6: 1  RB5: 1  RB4: 1  RB3: 1  RB2: 1  RB1: 1  RB0: 1

%: 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
Processor Reset
341296 bytes memory free
%:
```

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 object 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 object 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.

```
STEP    RB7  RB6  RB5  RB4  RB3  RB2  RB1  RB0  !PortB Pins
5       0    0    0    0    1    0    0    1    ! 9 x 5
7       0    0    0    0    0    1    0    1
84      0    0    0    0    1    0    1    0    ! 10 x 5
86      0    0    0    0    0    1    0    1
163     0    0    0    1    1    0    1    1    ! 27 x 3
165     0    0    0    0    0    0    1    1
242     0    0    0    1    0    0    0    1    ! 17 x 7
244     0    0    0    0    0    1    1    1
321     0    1    0    0    0    0    0    0    ! 64 x 63
323     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.

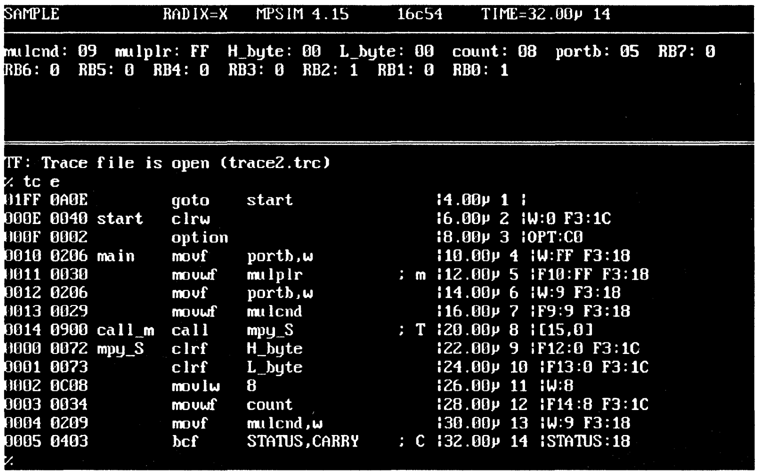
Restart the system by exiting MPSIM (q <RETURN>), and repeating steps 2 (Invoke the Simulator) and 3 (Load the Initialization File). Just as in Exercise 1, we will first open the trace file

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).



```
SAMPLE           RADIX=X   MPSIM 4.15     16c54    TIME=32.00µ 14

mulcnd: 09  mulplr: FF  H_byte: 00  L_byte: 00  count: 08  portb: 05  RB7: 0
RB6: 0  RB5: 0  RB4: 0  RB3: 0  RB2: 1  RB1: 0  RB0: 1

TF: Trace file is open (trace2.trc)
% tc e
01FF 0A0E         goto    start              ¦4.00µ 1 ¦
000E 0040 start   clrw                       ¦6.00µ 2 ¦W:0 F3:1C
000F 0002         option                     ¦8.00µ 3 ¦OPT:C0
0010 0206 main    movf    portb,w            ¦10.00µ 4 ¦W:FF F3:18
0011 0030         movwf   mulplr         ; m ¦12.00µ 5 ¦F10:FF F3:18
0012 0206         movf    portb,w            ¦14.00µ 6 ¦W:9 F3:18
0013 0029         movwf   mulcnd             ¦16.00µ 7 ¦F9:9 F3:18
0014 0900 call_m  call    mpy_S          ; T ¦20.00µ 8 ¦[15,0]
0000 0072 mpy_S   clrf    H_byte             ¦22.00µ 9 ¦F12:0 F3:1C
0001 0073         clrf    L_byte             ¦24.00µ 10 ¦F13:0 F3:1C
0002 0C08         movlw   8                  ¦26.00µ 11 ¦W:8
0003 0034         movwf   count              ¦28.00µ 12 ¦F14:8 F3:1C
0004 0209         movf    mulcnd,w           ¦30.00µ 13 ¦W:9 F3:18
0005 0403         bcf     STATUS,CARRY   ; C ¦32.00µ 14 ¦STATUS:18
%
```

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 Break Points

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 Status Register. There are three instructions that deal with break points:

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

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

BC

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

B *mpy_S*<RETURN>

Exercise 3: **Review all the break points. Enter the following command:**

DB<RETURN>

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

BC *mpy_S*<RETURN>

Execute the Object 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 break point 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 break points.

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

AD W <RETURN>

Exercise 2: **Add two break points and execute until the first break point is encountered.**

b main
b mpy_S
E <RETURN>

MPSIM executes until it encounters the first break point 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 break point.**

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 Object Code

MPSIM has four types of commands which allow you to modify the object 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 between 2 and 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.

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 Two. Chapter Five 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

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. All commands have been divided into the following categories:

- Loading and Saving
- Inspecting and Modifying
  - Program Memory
  - File Registers and Pins
  - Timers
  - Display Functions
  - Patch Table
  - Symbol Table
- Executing and Tracing
- Modifying the View Screen
- Miscellaneous Commands

Loading and Saving

The following three commands load and save object 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 object code to ***filename***.

Before simulation can begin, use LO to load an object 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 object 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 the save the corrected object code into an external file. Use the O command to output the object code. Enter the filename including the extension.

Example:	**O *SAMPLE1.OBJ INHX85*<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.

IA *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>**

```
0020 0200 0145 LABEL CLRF F5
;Clear I/O register for port A
: CLRF 6
0021 0201 0147 CLRF F7
;Clear I/O register for port C
: -
0200 0146 LABEL CLRF 6
: Q
```

After changing the object 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***

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 = 2
0001 0063  CLRF 3       The CLRF instruction = 6
0002 0080  SUBWF 0,0    The SUBWF instruction = 8
0003 0069  CLRF 9       The CLRF instruction = 6
```

```
%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 object 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 object code remain.
DP	Display all patches in symbolic format. Both the original object 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.

Break Points

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

B *address*	Set break point 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 break point at ***address***.
BC *data_area*	Cancel break point involving ***data_area***.
BC	Cancel all break points.
C *#breakpoints*	Continue execution ignoring ***#breakpoints*** break point occurrences.
DB	Display all active break points.

Only one conditional break point is allowed per data area.

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.

Chapter 5. MPSIM Commands

The following table lists the commands currently available with MPSIM. The sections that follow describe each command in greater detail. This table is also available in Appendix E, Quick Reference.

TABLE 5.1 - MPSIM COMMANDS

`AB`	Abort session.
AD *data_area*,{*O*|*X*|*B*|*D*},*#digits*	Add *data_area* to view screen, showing *data_area* in Octal, heXadecimal, Binary or Decimal radix with *#digits*.
B *address*	Set breakpoint at *address*.
B *data_area* {=|>|<|>=|<=|!=} *value*	Break when *data_area* matches the condition given by the operator (=,>,<,>=,<=,!=) and *value*.
BC [*address*| *data_area*]	Clear breakpoint at *address* or involving *data_area*. If you don't supply *address* or *data_area*, clear all break points.
C *#breakpoints*	Continue executing, ignoring *#breakpoints* breakpoint occurrences.
CK *[pin [high, low] | [-]]]*	Command allows you to assign a clock to an I/O pin, and the number of cycles that it should be held high and low.
`DB`	Display all active break points.
DE *address1*,*address2*	Delete code from memory *address1* to *address2*.
DI *address1*[,*address2*]	Display program memory from *address1* to *address2*. The code displays in both the current radix and mnemonics. If you don't supply *address2*, the I/O lines display at *address1*.
DK *[[alt-key [pin, event]]|[-]]*	Assign an asynchronous event to an Alt-key.
DL *symbol*	Delete *symbol* from symbol table.

(Cont.)

`DM` *address1,address2*	Display the code from *address1* to *address2.* The code displays only in the current radix.
`DP`	Display all patches.
`DR`	Display all registers, W register, flags, stack.
`DS`	Display symbol table.
`DV` *data_area*	Delete *data_area* from view screen.
`DW` {*E*|*D*}	Enable/Disable watchdog timer.
`DX`	Display current trace parameters.
`E` *address*	Execute program from beginning *address*. If you don't supply *address*, MPSIM begins executing at the current program count.
`EE` *address*	Modify data EE memory address on PIC16C84.
`F` *register*	Display/modify contents of file *register*.
`FI` *[[file, memory_addr, file_reg [,n]]|[-]]*	Insert the next value from file into file_reg when the current program counter equals memory_addr.
`FM` *addr1, addr2, pattern*	Fill memory from addr1 to addr2 with pattern.
`GE` *filename*	Get MPSIM commands from *filename*.
`GO`	Reset the PIC16/17 Microcontroller and execute from start.
`GS` *symbol,value,type*	Generate *symbol* with *value* and *type.* *type* can be *file, bit(file), label* or *literal*.
`H`	Display help screen.
`IA` *address*	Display/modify code at *address*. The code displays in both the current radix and mnemonics.
`IN` *address,instruction*	Insert *instruction* at *address*.
`IP` *[time | step]*	Inject stimulus according to time or step count.
`LJ`	Load and execute journal file.

(Cont.)

`LO` *filename format*	Load object file *filename* with *format* into program memory. Listing file is also loaded at this time.
`LS` *filename*	Load symbol file *filename* into internal symbol table.
`M` *address*	Display/modify code at *address*. The code displays only in the current radix.
`NV`	Clears view screen.
`O` *filename format*	Output modified object code to *filename* using *format*.
`P {`*54*|*55*|*71...*`}`	Choose Microcontroller number. Default is *55*.
`Q`	Quit session.
`RA`	Restore all: patch table, symbol table and break points.
`RE`	Reset elapsed time and step count.
`RP`	Restore patches to original instructions.
`RS`	Simulates a Power-On Reset.
`SC`	Display/modify processor cycle time (the default is 2 microseconds).
`SE` *data_area*	Display/modify *data_area*.
`SF` *address1,address2,register*	Search code from *address1* to *address2* for an instruction with *register*. The search criteria must be in the current radix.
`SI` *address1,address2,instruction*	Search code from *address1* to *address2* for an instruction. The search criteria must be in mnemonics.
`SM` *address1,address2,instruction*	Search program memory from *address1* to *address2* for *instruction*. The instruction must be in the current radix.
`SR {`*O*|*X*|*D*`}`	Set Input/Output radix to Octal, heXadecimal or Decimal.

(Cont.)

SS [*address*]	Single step execution beginning at *address*. If you don't supply *address*, MPSIM executes the next sequential instruction.
ST *filename*	Load stimulus file *filename*.
TA [*address1*,*address2*]	Set trace to print only those instructions located between addresses *address1* and *address2*. If you don't supply the addresses, MPSIM assumes full memory.
TC [*#instruction*]	Trace the next *#instruction* instructions & display if valid. If you don't supply *#instruction*, MPSIM traces until encountering a break or until you press any key.
TF [*filename*|*PRN*]	Open/Close trace output file or write trace to printer.
TR [*register*]	Set trace to print only when *register* is accessed. If you don't supply *register*, MPSIM assumes any register.
TR *register*,*min_value*,*max_value*	Set trace to print only when *register* is accessed and its value is between *min_value* and *max_value*.
TY *data_area*,{*O*|*X*|*B*|*D*},*#digits*	Change the radix and/or number of digits for *data_area* on the view screen. The new representation is in Octal, heXadecimal, Binary or Decimal radix with *#digits*.
V *data_area*,{*O*|*X*|*B*|*D*},*#digits*	Create view screen, showing *data_area* in Octal, heXadecimal, Binary or Decimal radix with *#digits*.
W	Display/modify contents of W register.
ZM *address1*,*address2*	Zero program memory from *address1* to *address2*.
ZP	Clear patch table.
ZR	Set all registers to 0.
ZT	Zero elapsed time counter to 0.

Pressing <RETURN> at the % prompt reexecutes the last command entered. Thus, you can use commands such as SS more easily.

Chapter 5: MPSIM Commands

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.

Related Commands

The **Q** command gives the same result.

AD - Add Item to View Screen

Syntax

AD *signal*{*,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 IOA`	Add Tris A to the screen.
`AD RA0,B`	Add the RA0 pin to the screen display with binary radix.
AD $\overline{MCLR}$,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 Break Point

Syntax

`B [address|pin] {operator value}`

Description

This command sets a break point at the specified address or at the location where the specified pin or 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 or 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 break point at label *LOOP*.
`B F2 > 80<RETURN>`	Break if F2 is greater than 80.

Defaults

None.

Related Commands

`BC` clears break points previously set and `DB` displays them.

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

BC - Clear Break Point

Syntax

`BC {`*address*`|`*data_area*`}`

Description

This command deletes a specified break point, or all break points if you don't specify one by address or data area.

Examples

MPSIM Command	Result
`BC LOOP`	Cancel break point at LOOP.
`BC F3`	Cancel break point involving the F3 register.
`BC`	Cancel all break points.

Defaults

None.

Related Commands

`B` sets break points and `DB` displays them.

C - Continue Executing

Syntax

C {*n*}

Description

This command continues execution from the CPC. If you specify `n`, MPSIM ignores the first *n* break points encountered.

Examples

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

Defaults

`n` defaults to 0.

Related Commands

B sets the break points, DB displays them and .BC clears break points previously set.

CK - Clock

Syntax

CK pin, high-cycles, low-cycles

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.

Pin is any valid I/O pin on the selected device

High-Cycles is the number of T-cycles that the pin should remain high

Low-Cycles is the number of T-cycles that the pin should remain low

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 Break Points

Syntax

```
DB
```

Description

This command lists all active break points. MPSIM allows only one conditional break point per data area.

Examples

MPSIM Command	Result
`B LOOP`	Sets a break point at LOOP.
`B F2 > 80`	Sets a break point at the location where F2 >80.

`DB`	Displays all break point locations via messages:
	`INFO, Break when (F2 > 0080)`
	`INFO, Break on address LOOP`

Defaults

None.

Related Commands

`B` sets the break points, `DB` displays them and `BC` clears break points 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 0A10`	This command removes the GOTO MAIN statement that causes the application to continuously execute.

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*, I/O lines display at *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.

Pin is any valid I/O pin.

Event is H, L, T or P (high, low, toggle or pulse)

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

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 $\overline{\text{MCLR}}$.
`%E`	Now during execution (with $\overline{\text{MCLR}}$ high) hitting Alt-F12 will simulate an external reset.
`%DK, 3, RTCC, T`	Define Alt-F3 to toggle RTCC input.
`%E`	Now during execution, every time Alt-F3 is pressed RTCC input will toggle.
`%DK -`	Disable all assignments.

Defaults

None

Related Commands

None

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 and RA 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.

Examples

MPSIM Command	Result
`DP`	

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. `O` writes the modified object code.

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 data
```

Description

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

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

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.OBJ from the label START until reaching a berakpoint 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 CPC or at a specified address. `C` executes from the CPC to the specified breakpoint occurrence. `TA` traces execution between specified addresses, and `TC` traces execution from the CPC for a specifies 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.

F - Display / Modify File Register

Syntax

`F register`

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:
F3:20:	This shows that the F3 register contains the value '20.'
F3:20:*21*	The F3 register value changes to '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, PMemAddr, FileReg Addr, [n]

Description

This command puts the next value in *FileName* in *FileReg* when the PC equals *PMemAddr.* 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.
FI -	Closes file and cancels command.

Examples

MPSIM Command	Result
`% FI ADVals.txt, 0x89, 4`	When the PC = interrupt vector, insert the next value in the file into the ADRES register.
`%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    ADVALUES.TXT, 0x04 0x09
```

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

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

The format of the `ADVALUES.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 addr1, addr2, pattern`

Description

This command fills *unused* program memory between **addr1** and **addr2** 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.

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

GS - Generate Symbol

Syntax

GS *symbol,value,type*[*(filename)*]

Description

This command generates the specified symbol with the specified value and type. The type can be file, bit, label or literal. 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	Symbol Value Type
	START 0000 L
NEWSYM	00FF 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 and RA restores (clears) the symbol table.

H - Help

Syntax

`H`

Description

This command displays the Help screen, which lists a brief synopsis and syntax for each MPSIM command.

Examples

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

Defaults

None.

Related Commands

None.

IA – Display / Modify Program Memory (Symbolic Format)

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 the current radix) displays: `0020 0200 0145 LABEL CLRF F5 ; Clear I/O register for port A:`
`: CLRF 6`	MPSIM changes the instruction as specified and displays the next instruction line (address 201): `0021 0201 0147 CLRF F7:`
`: -`	MPSIM backs up and displays the modified instruction at address 200: `0200 0146 LABEL 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 in the stimulus file will always remain "step" regardless of the method selected to inject stimulus.

Examples

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

Defaults

Default is "step"

Related Commands

None.

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 File

Syntax

LO *filename* {*format*}

Description

This command loads the specified file into program memory. If the assembler selected is MPALC, MPSIM will assume an .OBJ extension. 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 symbolfile 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.`

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 In Radix Designated Format

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

INHX16

INHX8S

PICICE

Examples

MPSIM Command	Result
% O *SAMPLE1.OBJ INHX8M*	MPSIM writes the object code, as modified, to the file SAMPLE1.OBJ 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 [54|55|71|...]
```

Description

Use this command to select the PIC16CXX Microcontroller you want to simulate.

Examples

MPSIM Command	Result
`% P [54|55|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 break points.

Examples

MPSIM Command	Result
`% RA`	MPSIM restores the patch table, clears the symbol tables and removes all break points.

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 POR.

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 Data Area

Syntax

```
SE data
```

Description

This command displays and/or modifies any data area.

Examples

MPSIM Command	Result
`% SE` *`RA0`*	
`RA0:1:`***`0`***	The following message displays: RA0=1: 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 *address2* 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 search 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 CPC. 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           0
```

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 - Print Trace Instructions

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`	Upon the invocation of the TC command, MPSIM will print/display only those instructions between *main* and *mpy_S*.

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 - Open Trace Output File / PRINT Trace

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_value, max_value]
```

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_value* and *max_value*, it performs the trace only if the value of the specified register lies between *min_value* and *max_value*.

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 *data_area,{X|O|D|B},#digits*

Description

This command changes the formatting of existing view screen elements. *data_area* names the pin or register. If it isn't in the view screen, 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 for this variable on the view screen.

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.`

V - View Screen

Syntax

```
V signal [,radix [,digits]]
```

Description

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. Register or signal s displays in radix mode r with n digits. 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 5.4 RADIX DEFAULT WIDTHS

Radix	Digits
X	2
B	8
O	3
D	2

Related Commands

`AD, DV, NV, TY.`

W - Display / Modify the Work Register

Syntax

`W`

Description

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

Examples

MPSIM Command	Result
`% W`	
`W=44:00`	The value of W is 44 as the following message shows.
`W=44:`	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.

ZM - Zero the Program Memory

Syntax

`ZM address1, address2`

Description

This command zeros 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. Trouble Shooting Guide

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 family since the object code for the PIC16CXX family 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, you object 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.

Appendix A: Trouble Shooting Guide

Messages

Informative Messages

Address Break After

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

Break at Address

Cause: A break point 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.

Object Code Loaded

Cause: MPSIM has found and read filename.OBJ.

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.

Watch Dog Timer Disabled

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

Watch Dog 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 reenter.

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.

Appendix A: Trouble Shooting Guide

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 break point, 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: Reenter 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 reenter.

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 reenter.

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 reenter.

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 reenter.

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: Reenter 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.

Appendix A: Trouble Shooting Guide

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 object file plus the .LST extension.

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

Cannot Find Symbol File

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

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

Invalid Object File

Cause:	The object file MPSIM is trying to load is not valid for the format you specified.
Cure:	Ensure that the file is a valid object file in the format you specified.

Missing Instruction

Cause:	You have told MPSIM to assemble an instruction, but did not supply the instruction
Cure:	Reenter the command with the desired instruction.

No Breaks Found Involving

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

Appendix A: Trouble Shooting Guide

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 break point, 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 MP-SIM.

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 PIC16C5X instruction.

Cure: Reenter the instruction in valid PIC16C5X 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 break point, you have used an unrecognized logical operator.

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

Appendix A: Trouble Shooting Guide

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.

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.

Appendix A: Trouble Shooting Guide

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 break point.
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 Object File

Cause:	You have tried to load a symbol file that was not generated for the current object 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:	Regenerated 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: Reenter 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 PIC16C5X opcode.

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

Appendix B. 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
```

MPREG.H

```
LIST L=OFF
;***************************         PIC16C5X Header
*************************
;
;
PICC54    equ      1FFH              ; Define Reset Vectors
PICC55    equ      1FFH
PICC56    equ      3FFH
PICC57    equ      7FFH
PICC71    equ      0H
;
RTCC     equ      1h
PC       equ      2h
STATUS   equ      3h                ; F3 Reg is STATUS Reg.
FSR      equ      4h
;
PORT_A   equ      5h
PORT_B   equ      6h                ; I/O Port Assignments
PORT_C   equ      7h
;
;
```

```
ADRES    equ     9h      ;16C71 Special-purpose registers
ADCON0   equ     8h      ;16C71 Special-purpose registers
ADCON1   equ     88h     ;16C71 Special-purpose registers
PCLATH   equ     0AH     ;16C71 Special-purpose registers
INTCON   equ     0BH     ;16C71 Special-purpose registers
TRISA    equ     85H     ;16C71 Special-purpose registers
TRISB    equ     86H     ;16C71 Special-purpose registers
;
;
;
;*****************************************************************
;
;                               ; STATUS REG. Bits
CARRY    equ     0h             ; Carry Bit is Bit.0 of F3
C        equ     0h
DCARRY   equ     1h
DC       equ     1h
Z_bit    equ     2h             ; Bit 2 of F3 is Zero Bit
Z        equ     2h
P_DOWN   equ     3h
PD       equ     3h
T_OUT    equ     4h
TO       equ     4h
PA0      equ     5h      ;16C5X Status bits
PA1      equ     6h      ;16C5X Status bits
PA2      equ     7h      ;16C5X Status bits
;
;
RP0      equ     5h      ;16C71 Status bits
RP1      equ     6h      ;16C71 Status bits
IRP      equ     7h      ;16C71 Status bits
GIE      equ     7h      ;16C71 INTCON register bits
ADIE     equ     6h      ;16C71 INTCON register bits
RTIE     equ     5h      ;16C71 INTCON register bits
INTE     equ     4h      ;16C71 INTCON register bits
RBIE     equ     3h      ;16C71 INTCON register bits
RTIF     equ     2h      ;16C71 INTCON register bits
INTF     equ     1h      ;16C71 INTCON register bits
RBIF     equ     0       ;16C71 INTCON register bits
ADCS1    equ     7h      ;16C71 ADCN0 register bits
ADCS0    equ     6h      ;16C71 ADCN0 register bits
CHS1     equ     4h      ;16C71 ADCN0 register bits
CHS0     equ     3h      ;16C71 ADCN0 register bits
```

```
GO       equ     2h       ;16C71 ADCN0 register bits
ADIF     equ     1h       ;16C71 ADCN0 register bits
ADON     equ     0        ;16C71 ADCN0 register bits
PCFG1    equ     1h       ;16C71 ADCN1 register bits
PCFG0    equ     0        ;16C71 ADCN1 register bits
;
;
Same     equ     1h
;
LSB      equ     0h
MSB      equ     7h
;
TRUE     equ     1h
YES      equ     1h
FALSE    equ     0h
NO       equ     0h
;
;***********************************************************************
LIST L=ON
```

```
;*******************************************************************
;                          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 )
;*******************************************************************
;
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
;
;
     include         "mpreg.h"
;
; *****************************         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
```

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
5        0     0    0    0     1    0    0    1  !  9 x 5
7        0     0    0    0     0    1    0    1
84       0     0    0    0     1    0    1    0  ! 10 x 5
86       0     0    0    0     0    1    0    1
163      0     0    0    1     1    0    1    1  ! 27 x 3
165      0     0    0    0     0    0    1    1
242      0     0    0    1     0    0    0    1  ! 17 x 7
244      0     0    0    0     0    1    1    1
321      0     1    0    0     0    0    0    0  ! 64 x 63
323      0     0    1    1     1    1    1    1
```

Appendix C. Customer Support

Keeping Current with Microchip Systems

Microchip Technology endeavors at all times to provide the best service and responsiveness possible to it users. The Microchip Technology Systems BBS is one mechanism to facilitate this process.

The BBS is supported as a service to its customers. This is where all of the most recent information regarding systems products can be found. The BBS is monitored several times a week for questions. Truly urgent issues should not be left with the BBS, but referred to your local distributor, or Microchip sales office.

The BBS is an evolving product. Details of its operation will not be found here. This chapter provides a brief discussion of the general services available.

This chapter also describes the Microchip Systems software numbering scheme.

Highlights

The points that will be highlighted in this chapter are:

- Access to the BBS
- Special Interest Groups
- Files
- Mail
- Software Releases

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.

Bulletin Board Access

Access to the bulletin board is 24 hours per day, barring technical or mechanical difficulties. Access is gained by calling your local CompuServe® access number. Your modem should be set to 8-bits, No parity, 1 stop bit (8-1-N). The service supports baud rates from 300 to 9600 baud. To access the BBS, follow these steps:

1. Dial your local CompuServe access number.
2. Press `<ret>` and a garbage string will appear.
3. Enter `+<ret>` and `Host Name`: will appear.
4. Enter `mchipbbs<ret>` and you will be connected to the Microchip BBS.

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

Bulletin Board Usage

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
- Technical Assistance

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. They may be able to 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:

- PICMASTER
- MPASM
- Utilities
- Bugs

These groups are monitored by the Microchip staff.

Files

The Microchip BBS is used regularly 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. These files will be monitored, scanned and approved or disapproved by the moderator of the SIG to which the file is submitted. 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. Be aware, though, that the SIGs are moderated only about once per day. Truly urgent questions should be referred to your local distributor, sales representative, or FAE. They are your first line of defense.

Software Revisions

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 releasd 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 reresents 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

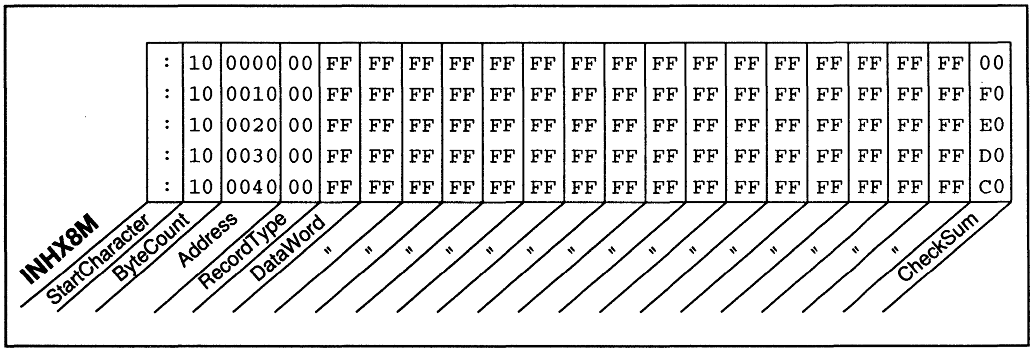
START CHARACTER	:(colon)
WORD COUNT	2 Hex digits
ADDRESS	4 Hex digits
RECORD TYPE	2 Hex digits
HIGH BYTE	2 Hex digits
LOW BYTE	2 Hex digits
CHECK SUM	2 Hex digits

INHX8S

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

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

Figure D.1a - INHX8S and INHX8M File Formats



StartCharacter	ByteCount	Address	RecordType	DataWord	"	"	"	"	"	"	"	"	"	"	"	"	"	"	"	CheckSum
:	10	0000	00	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	00
:	10	0010	00	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	F0
:	10	0020	00	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	E0
:	10	0030	00	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	D0
:	10	0040	00	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	C0

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 Word 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.

16-bit Word Format:

The 16-bit word format is basically the same as the 8-bit word format. The difference is that the hexadecimal data word is four digits. The byte count (BB), however, is based on 16-bit words.

Appendix E. Quick Reference

MPSIM Commands

The following two tables contain condensed information from Chapters 4 and 5 and are printed here for your convenience. The first table contains MPSIM commands in alphabetical order and the second table lists MPSIM commands grouped by function.

`AB`	Abort session.
`AD` *`data_area,{O|X|B|D},#digits`*	Add *`data_area`* to view screen, showing *`data_area`* in Octal, heXadecimal, Binary or Decimal radix with *`#digits`*.
`B` *`address`*	Set breakpoint at *`address`*.
`B` *`data_area {=|>|<|>=|<=|!=} value`*	Break when *`data_area`* matches the condition given by the operator (=,>,<,>=,<=,!=) and *`value`*.
`BC` *`[address| data_area]`*	Clear breakpoint at *`address`* or involving *`data_area`*. If you don't supply *`address`* or *`data_area`*, clear all break points.
`C` *`#breakpoints`*	Continue executing, ignoring *`#breakpoints`* breakpoint occurrences.
`CK` *`[pin [high, low] | [ - ] ] ]`*	Command allows you to assign a clock to an I/O pin, and the number of cycles that it should be held high and low.
`DB`	Display all active break points.
`DE` *`address1,address2`*	Delete code from memory *`address1`* to *`address2`*.
`DI` *`address1[,address2]`*	Display program memory from *`address1`* to *`address2`*. The code displays in both the current radix and mnemonics. If you don't supply *`address2`*, the I/O lines display at *`address1`*.
`DK` *`[[alt-key [pin, event]]|[ - ]]`*	Assign an asynchronous event to an Alt-key.
`DL` *`symbol`*	Delete *`symbol`* from symbol table.

(Cont.)

DM *address1,address2*	Display the code from *address1* to *address2*. The code displays only in the current radix.
DP	Display all patches.
DR	Display all registers, W register, flags, stack.
DS	Display symbol table.
DV *data_area*	Delete *data_area* from view screen.
DW {*E*|*D*}	Enable/Disable watchdog timer.
DX	Display current trace parameters.
E *address*	Execute program from beginning *address*. If you don't supply *address*, MPSIM begins executing at the current program count.
EE *address*	Modify data EE memory address on PIC16C84.
F *register*	Display/modify contents of file *register*.
FI *[[file, memory_addr, file_reg [,n]]|[-]]*	Insert the next value from file into file_reg when the current program counter equals memory_addr.
FM *addr1, addr2, pattern*	Fill memory from addr1 to addr2 with pattern.
GE *filename*	Get MPSIM commands from *filename*.
GO	Reset the PIC16/17 Microcontroller and execute from start.
GS *symbol,value,type*	Generate *symbol* with *value* and *type*. *type* can be *file, bit(file), label* or *literal*.
H	Display help screen.
IA *address*	Display/modify code at *address*. The code displays in both the current radix and mnemonics.
IN *address,instruction*	Insert *instruction* at *address*.
IP *[time | step]*	Inject stimulus according to time or step count.
LJ	Load and execute journal file.

(Cont.)

Appendix E: Quick Reference

LO *filename format*	Load object file *filename* with *format* into program memory. Listing file is also loaded at this time.
LS *filename*	Load symbol file *filename* into internal symbol table.
M *address*	Display/modify code at *address*. The code displays only in the current radix.
NV	Clears view screen.
O *filename format*	Output modified object code to *filename* using *format*.
P {*54*|*55*|*71*...}	Choose Microcontroller number. Default is *55*.
Q	Quit session.
RA	Restore all: patch table, symbol table and break points.
RE	Reset elapsed time and step count.
RP	Restore patches to original instructions.
RS	Simulates a Power-On Reset.
SC	Display/modify processor cycle time (the default is 2 microseconds).
SE *data_area*	Display/modify *data_area*.
SF *address1,address2,register*	Search code from *address1* to *address2* for an instruction with *register*. The search criteria must be in the current radix.
SI *address1,address2,instruction*	Search code from *address1* to *address2* for an instruction. The search criteria must be in mnemonics.
SM *address1,address2,instruction*	Search program memory from *address1* to *address2* for *instruction*. The instruction must be in the current radix.
SR {*O*|*X*|*D*}	Set Input/Output radix to Octal, heXadecimal or Decimal.

(Cont.)

`SS [address]`	Single step execution beginning at *address*. If you don't supply *address*, MPSIM executes the next sequential instruction.
`ST filename`	Load stimulus file *filename*.
`TA [address1,address2]`	Set trace to print only those instructions located between addresses *address1* and *address2*. If you don't supply the addresses, MPSIM assumes full memory.
`TC [#instruction]`	Trace the next *#instruction* instructions & display if valid. If you don't supply *#instruction*, MPSIM traces until encountering a break or until you press any key.
`TF [filename|PRN]`	Open/Close trace output file or write trace to printer.
`TR [register]`	Set trace to print only when *register* is accessed. If you don't supply *register*, MPSIM assumes any register.
`TR register,min_value,max_value`	Set trace to print only when *register* is accessed and its value is between *min_value* and *max_value*.
`TY data_area,{O|X|B|D},#digits`	Change the radix and/or number of digits for *data_area* on the view screen. The new representation is in Octal, heXadecimal, Binary or Decimal radix with *#digits*.
`V data_area,{O|X|B|D},#digits`	Create view screen, showing *data_area* in Octal, heXadecimal, Binary or Decimal radix with *#digits*.
`W`	Display/modify contents of W register.
`ZM address1,address2`	Zero program memory from *address1* to *address2*.
`ZP`	Clear patch table.
`ZR`	Set all registers to 0.
`ZT`	Zero elapsed time counter to 0.

Pressing <RETURN> at the % prompt reexecutes the last command entered. Thus, you can use commands such as SS more easily.

Appendix E: Quick Reference

Loading and Saving	LO *filename format*	Load object file *filename* with *format* into program memory. Listing file is also loaded at this time.
	LS *filename*	Load symbol file *filename* into internal symbol table.
	O *filename format*	Output modified object code to *filename* using *format*.
Inspecting and Modifying Program Memory	IA *address*	Display/modify code at *address*. The code displays in both the current radix and mnemonics.
	M *address*	Display/modify code at *address*. The code displays only in the current radix.
	SI *address1, address2, instruction*	Search code from *address1* to *address2* for an instruction. The search criteria must be in mnemonics. SM *address1, address2, instruction* Search program memory from *address1* to *address2* for *instruction*. The instruction must be in the current radix. ZM *address1, address2* Zero program memory from *address1* to *address2*.
File Registers and Pins	F *register*	Display/modify contents of file *register*.
	W	Display/modify contents of W register.
	SE *data_area*	Display/modify *data_area*. SF *address1, address2, register* Search code from *address1* to *address2* for an instruction with *register*. The search criteria must be in the current radix.
	ZR	Set all registers to 0.
Timers	ZT	Zero elapsed time counter to 0.
	RE	Reset elapsed time and step count.
	SC	Display/modify processor cycle time (the default is 2 microseconds).
Display Functions	DR	Display all registers, W register, flags, stack.
	DM *address1, address2*	Display the code from *address1* to *address2*. The code displays only in the current radix.

(Cont.)

Execute and Tract	E *address*	Execute program from beginning *address*. If you don't supply *address*, MPSIM begins executing at the current program count.
	C *#breakpoints*	Continue executing, ignoring *#breakpoints* break point occurrences.
	SS [*address*]	Single step execution beginning at *address*. If you don't supply *address*, MPSIM executes the next sequential instruction.
	TC [*#instruction*]	Trace the next *#instruction* instructions & display if valid. If you don't supply *#instruction*, MPSIM traces until encountering a break or until you press any key.
	TA [*address1*,*address2*]	Set trace to print only those instructions located between addresses *address1* and *address2*. If you don't supply the addresses, MPSIM assumes full memory.
	TR [*register*]	Set trace to print only when *register* is accessed. If you don't supply *register*, MPSIM assumes any register.
	TR *register*,*min_value*,*max_value*	Set trace to print only when *register* is accessed and its value is between *min_value* and *max_value*.
	DX	Display current trace parameters.
Break Points	B *address*	Set break point at *address*.
	B *data_area* {=|>|<|>=|<=|!=} *value*	Break when *data_area* matches the condition given by the operator (=,>,<,>=,<=,!=) and *value*.
	BC [*address*| *data_area*]	Clear break point at *address* or involving *data_area*. If you don't supply *address* or *data_area*, clear all break points.
	C *#breakpoints*	Continue executing, ignoring *#breakpoints* break point occurrences.
	DB	Display all active break points.
View Screen	V *data_area*,{*O*|*X*|*B*|*D*},*#digits*	Create view screen, showing *data_area* in Octal, heXadecimal, Binary or Decimal radix with *#digits*.

(Cont.)

	`NV`	Clears view screen.
	`AD` *data_area,{O|X|B|D},#digits*	Add *data_area* to view screen, showing *data_area* in Octal, heXadecimal, Binary or Decimal radix with *#digits*.
	`DV` *data_area*	Delete *data_area* from view screen.
	`TY` *data_area,{O|X|B|D},#digits*	Change the radix and/or number of digits for *data_area* on the view screen. The new representation is in Octal, heXadecimal, Binary or Decimal radix with *#digits*.
Miscellaneous	`SR` *{O|X|D}*	Set Input/Output radix to Octal, heXadecimal or Decimal.
	`P` *{54|55|71|...}*	Choose Microcontroller number. Default is *55*.
	`IP [TIME | STEP]`	Inject stimulus according to time or step count.
	`GE` *filename*	Get MPSIM commands from *filename*.
	`Q`	Quit session.
	`AB`	Abort session.
	`ST` *filename*	Load stimulus file *filename*.
	`H`	Display help screen.

Appendix F. 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.

- $\overline{\text{MCLR}}$
- RTCC
- 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 $\overline{\text{MCLR}}$ 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 (TO) and Power-down (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, RETURN, or RETFIE 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 (INDO).

Appendix F: PIC16C5X User's Guide Addendum

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 G. 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 (TO) and Power-down (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).

Appendix G: PIC16C64 User's Guide Addendum

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.

Appendix G: PIC16C64 User's Guide Addendum

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).

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 I^{2}C™ mode.

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:

- 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 (RTCC) 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 (TO) and Power-down (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: PIC16C74 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. 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 version 4.5 or greater 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 (TO) and Power-down (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.

Appendix I: PIC16C74 User's Guide Addendum

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).

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. 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

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
- 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 (TO) and Power-down (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.

Appendix J: PIC16C84 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 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.

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 wheras 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.

Index

Notes:

Notes:

Notes:

Notes:

Notes:

WORLDWIDE SALES & SERVICE

AMERICAS

Corporate Office
Microchip Technology Inc.
2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 602 786-7200 Fax: 602 899-9210

Atlanta
Microchip Technology Inc.
1521 Johnson Ferry Road NE, Suite 170
Marietta, GA 30062
Tel: 404 509-8800 Fax: 404 509-8600

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.
665 Tollgate Road, Unit C
Elgin, IL 60123-9312
Tel: 708 741-0171 Fax: 708 741-0638

Dallas
Microchip Technology Inc.
14651 Dallas Parkway, Suite 816
Dallas, TX 75240-8809
Tel: 214 991-7177 Fax: 214 991-8588

AMERICAS (continued)

Los Angeles
Microchip Technology Inc.
18201 Von Karman, Suite 455
Irvine, CA 92715
Tel: 714 263-1888 Fax: 714 263-1338

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 N First Street, Suite 590
San Jose, CA 95131
Tel: 408 436-7950 Fax: 408 436-7955

ASIA/PACIFIC
Microchip Technology Inc.
Unit No. 3002-3004, Tower 1
Metroplaza
223 Hing Fong Road
Kwai Fong, N.T. Hong Kong
Tel: 852 401 1200 Fax: 852 401 3431

EUROPE

United Kingdom
Arizona Microchip Technology Ltd.
Unit 6, The Courtyard
Meadow Bank, Furlong Road
Bourne End, Buckinghamshire SL8 5AJ
Tel: 44 0628 851 077 Fax: 44 0628 850 259

Germany
Arizona Microchip Technology GmbH
Gustav-Heinemann-Ring 125
81739 Muenchen, Germany
Tel: 49 089 627144 0 Fax: 49 089 627144 44

France
Arizona Microchip Technology SARL
2, Rue Du Buisson aux Fraises
F-91300 Massy, France
Tel: 33 01 6930 9090 Fax: 33 01 6930 9079

Italy
Arizona Microchip Technology SRL
Centro Direzionale Colleoni
Palazzo Pegaso Ingresso No. 2
Via Paracelso 23, 20041 Agrate Brianza (MI) Italy
Tel: 39 039 68 99 939 Fax: 39 039 68 99 883

JAPAN
Microchip Technology International Inc.
Shinyokohama Gotoh Bldg. 8F, 3-22-4
Shinyokohama, Kohoku-Ku, Yokohama-Shi
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