8051 Integrated Development Environment

Table of Contents

1 Initial Topics

1.1 Limited Warranty

With respect to the physical diskette and physical documentation enclosed herein, Acebus warrants the same to be free of defects in materials and workmanship for a period of 60 days from the date of purchase. In the event of notification within the warranty period of defects in material or workmanship, Acebus will replace the defective diskette or documentation. The remedy for breach of this warranty shall be limited to replacement and shall not encompass any other damages, including but not limited to loss of profit, and special, incidental, consequential, or other similar claims.

Acebus specifically disclaims all other warranties, expressed or implied, including but not limited to implied warranties of merchantability and fitness for a particular purpose with respect to defects in the diskette and documentation, and the program license granted herein in particular, and without limiting operation of the program license with respect to any particular application, use, or purpose. In no event shall Acebus be liable for any loss of profit or any other commercial damage, including but not limited to special, incidental, consequential or other damages.

Acebus reserves the right to stop supporting the software/hardware and to stop releasing updates for this software at any time after the warranty period.

1.2 Copyright

1.3 Contact Information

You can contact Acebus via one of the following methods:

By mail at: Acebus
835 Weatherhead Hollow Road
Guilford, VT 03501
Via e-mail at: michael@acebus.com
Via WWW at: www.acebus.com

1.4 Registration

Acebus is a small one-man operation. Your purchase of this program will help insure its continued development. If you find this program useful please obtain a registered version.

The unregistered evaluation version of this program has the following limitations.

- Editor text buffer limited to 16k bytes
- No include file support

Otherwise it is fully functional. It is provided for evaluation purposes only. If you continue to use it then please register it.
The Registered version adds include file support and the editor uses a dynamic text buffer size. The size of the text buffer is allocated to be 64k bytes larger than the file being loaded.

The primary difference between the evaluation and registered versions is the size of the 8051 application that can be developed.

The installation and registration key files can be obtained online using secure order processing via the following Register Now (http://www.regnow.com/) link

8051 Integrated Development Environment Secure Order Form
(https://www.regnow.com/softsell/nph-softsell.cgi?item=4322-1)


The online registration process uses electronic distribution of the IDE and registration key file. Upon completion of the on-line order process you will be given a time limited download link for a compressed file containing the IDE setup files. You will also be sent the registration key file as an e-mail attachment. This e-mail also contains instructions on using the attached key file to register the IDE. A hard copy of the program on diskette is not included with the on-line registration process.

The manual is supplied in an Adobe Acrobat (PDF) format file and is automatically copied to your computer during the installation process.

1.5 Introduction

The 8051 Integrated Development Environment (IDE) combines a text editor, assembler and software simulator into a single program. All components needed to develop 8051 programs (and its various derivatives) available and controllable from this single IDE.

Enter and modify the program source code from within the built in editor. Then assemble the source code by selecting the Assemble command. If any errors are located the appropriate source module can be automatically loaded and the cursor placed on the line containing the error. Fix the error and move to the next error (if any). Once all errors have been fixed reassemble the code.

After successfully assembling the source code use the simulator to step through your program. You can watch registers, flags, ports and memory locations change as your program progresses. You will see the flow of your program and be able to verify that it operates as intended. If it does not then return to the editor, reassemble and back to the simulator.

1.6 Software Installation

To install the 8051 IDE onto the hard disk of a Windows 2000 (or XP) based computer perform the following steps:

1. Download the 8051 for Windows IDE from the following web link

http://www.acebus.com/download/win8051.zip

2. Extract the setup files contained in the win8051.zip file to a temporary directory.

3. Once the setup files have been extracted, run the Setup.exe program to begin the installation process.

Once setup has completed a 8051 IDE program group will be added to the Start menu under Programs. A short cut to the 8051 IDE executable is located in this new file group.

Selecting its shortcut from the Start Menu runs the 8051 IDE.

1.7 Software Operation

The 8051 Integrated Development Environment (IDE) is a Windows 2000 (or XP) based application. It operates with the same look and feel of other Windows based application.

The following interface objects are present:
Menu bar: contains the main menu options from which submenus are displayed.
Tool bar: contains a row of icon providing a short cut to many of the menu commands.
Status bar: contains information about the programs current operations.
Client area: contains the editor and simulation windows.

See Also

File menu
Edit menu
View menu
Simulate menu
Options menu
Window menu
Help menu

1.8 Help Using Help Command

Use the Context Help command to obtain help on some portion of the 8051 IDE. When you choose the Toolbar's Context Help button, the mouse pointer will change to an arrow and question mark. Then click somewhere in the 8051 IDE window, such as another Toolbar button. The Help topic will be shown for the item you clicked.

Shortcut

Toolbar: 
Keys: Shift+F1

1.9 Title Bar

The title bar is located along the top of a window. It contains the name of the application and

document.

To move the window, drag the title bar. Note: You can also move dialog boxes by dragging their title bars.

A title bar may contain the following elements:
- Application Control-menu button
- Document Control-menu button
- Maximize button
- Minimize button
- Name of the application
- Name of the document
- Restore button

1.10 Scroll bars

Displayed at the right and bottom edges of the document window. The scroll boxes inside the scroll bars indicate your vertical and horizontal location in the document. You can use the mouse to scroll to other parts of the document.

1.11 Output window

The output window list single line messages which result from assembly or searching for files. Double clicking on a error message or found in file entry will cause the associated source file to be displayed with the cursor positioned on the indicated line.

See Also

Output command (View menu)

1.12 Options menu

The Options menu offers the following commands:

Defaults	Displays a dialog box, which allows you to change various program defaults.
Simulator	Displays a dialog box, which allows you to specify various simulator related parameters.

1.13 Default Settings dialog

The Default Settings dialog allows you to specify various default options for the 8051 IDE. The following is a list of options available in this dialog:

Font size

This drop down list box allows you to select the size of the font used in the editor, output, register, IRAM, DRAM, XRAM and watch windows.

Tab size

This edit box allows you to specify the tab size used by the text editor.

Default radix

This drop down list box allows you to specify the default radix used to display values in the register, IRAM, DRAM, XRAM and watch windows.

Processor type

This drop down list box allows you to specify the target processor for your application. This option allows you to specify the amount and type of resources available to your program. For example the amount of IRAM (128 or 256 bytes) that is available.

OK

Clicking on OK will result in the dialog closing and the default options changed to the specified settings.

Cancel

Clicking on Cancel will result in the dialog closing and the default options left unchanged.

2 File Menu

2.1 File menu commands

The File menu offers the following commands:

New -	Creates a new document.
Open -	Opens an existing document.
Close -	Closes an opened document.
Save -	Saves an opened document using the same file name.
Save As -	Saves an opened document to a specified file name.
Print -	Prints a document.
Print Setup -	Selects a printer and printer connection.
Exit -	Exits 8051 IDE.

2.2 File New command

Use this command to create a new document in the 8051 IDE.

You can open an existing document with the Open command.

Shortcuts

Toolbar: 
Keys: Ctrl+N

2.3 File Open command

Use this command to open an existing document in a new window. You can open multiple documents at once. Use the Window menu to switch among the multiple open documents. See Window 1, 2, ... command.

You can create new documents with the New command.

Shortcuts

Toolbar: 
Keys: Ctrl+O

2.4 File Open Dialog

The following options allow you to specify which file to open:

File Name
Type or select the filename you want to open. This box lists files with the extension you select in the List Files of Type box.

List Files of Type
Select the type of file you want to open:

Drives
Select the drive in which the 8051 IDE stores the file that you want to open.

Directories
Select the directory in which the 8051 IDE stores the file that you want to open.

Network...
Choose this button to connect to a network location, assigning it a new drive letter.

2.5 File Close command

Use this command to close all windows containing the active document. The 8051 IDE suggests that you save changes to your document before you close it. If you close a document without saving, you lose all changes made since the last time you saved it. Before closing an untitled document, the 8051 IDE displays the Save As dialog box and suggests that you name and save the document.

You can also close a document by using the Close icon on the document's window, as shown below:

2.6 File Open Workspace command

Use this command to open a previously saved workspace. The current workspace will be closed prior to opening the selected workspace.

2.7 File Save Workspace command

Use this command to save the current window configuration to the current workspace file. If the workspace has never been saved and given a name the Save As dialog box will be displayed.

2.8 File Close Workspace command

Use this command to close the current workspace. Once the current workspace has been closed the default window layout (if any will be loaded). You can then modify this layout and save it. When saving the default layout you will be asked to give it a workspace name. This allows you to create a new workspace.

2.9 Recent Workspaces 1, 2, 3, 4 command

Use the numbers and filenames listed in the Recent Workspaces sub-menu to open the last four workspaces you closed. Choose the number that corresponds with the workspace you want to open

2.10 File Save command

Use this command to save the active document to its current name and directory. When you save a document for the first time, the 8051 IDE displays the Save As dialog box so you can name your document. If you want to change the name and directory of an existing document before you save it, choose the Save As command.

Shortcuts

Toolbar: 
Keys: Ctrl+S

2.11 File Save As command

Use this command to save and name the active document. The 8051 IDE displays the Save As dialog box so you can name your document.
To save a document with its existing name and directory, use the Save command.

2.12 File Save As Dialog

The following options allow you to specify the name and location of the file you're about to save:

File Name
Type a new filename to save a document with a different name. A filename can contain up to eight characters and an extension of up to three characters. The 8051 IDE adds the extension you specify in the Save File As Type box.

Drives
Select the drive in which you want to store the document.

Directories
Select the directory in which you want to store the document.

Network...
Choose this button to connect to a network location, assigning it a new drive letter.

2.13 File Save All command

Use this command to save all open documents to their current name and directory. When you save a document for the first time, the 8051 IDE displays the Save As dialog box so you can name your document. If you want to change the name and directory of an existing document before you save it, choose the Save As command.

Shortcuts

Toolbar: 

2.14 File 1, 2, 3, 4 command

Use the numbers and filenames listed in the Recent Files sub-menu to open the last four documents you closed. Choose the number that corresponds with the document you want to open.

2.15 File Exit command

Use this command to end your 8051 IDE session. You can also use the Close command on the application Control menu. The 8051 IDE prompts you to save documents with unsaved changes.

Shortcuts

Mouse: Double-click the application's Control menu button.

Keys: Alt+F4

3 Editor Basics

3.1 Creating a New Edit Window

To create a new empty edit window, select New from the File menu.
A new edit window will be displayed into which you can enter text.

See Also

File New command

3.2 Opening an Existing Text File

To open an existing text file, select Open from the File menu.

Use the displayed dialog to locate and select the desired file.

Once selected, click on the OK button.

A new window will be displayed into which the specified file will be loaded.

See Also
File Open command

3.3 Saving Text in an Edit Window

To save the file in the currently active edit window, select Save from the File menu.

If this is new document (not loaded previously loaded or saved) then you will be prompted to enter a file name and directory to save the text in. Then click on the OK button.

If the text already has a name then it will be saved under that name automatically.

See Also
File Save command

3.4 Save a File Under a New Name

To save the text in the currently active edit window under a new file name, select Save as from the File menu.

Enter the name of the file the text buffer is to be saved in.

Click on the OK button.

The text in the active editor window will be saved to the specified file when OK is selected. If you decide not to save the text under that name click the Cancel button instead.

See Also
File Save As command

3.5 Closing an Editor Window

To close an editor, select the desired editor window by clicking on its title bar or by selecting it from the Window menu.

Select the Close command from the File menu.

The active window will be closed. If it has been modified since the last time it was saved you will be asked if you want to save it.

See Also
File Close command

3.6 Printing the Editor Text

The 8051 IDE allows you to print the text within the active editor window. This is accomplished by selecting the Print command via the File menu.

See Also

File Print command

3.7 Previewing Printed Text

The 8051 IDE allows you to preview how the active document will look when printed. This is accomplished by selecting the Print command via the File menu.

See Also

File Print Preview command

3.8 Printer Setup

To specify an alternate printer or to change the settings of the selected printer use the Print Setup command in the File menu. Selecting the Print Setup command will result in the Print Setup Dialog box being displayed. Use this dialog box to change the printer options.

See Also

File Print Setup command
Print Setup Dialog

3.9 Exiting the IDE

You can exit (quit) the 8051 IDE by selecting Exit from the File menu. You will be prompted to save any source code modified since it was last saved. If no source code has been modified since it was last saved then the 8051 IDE will exit directly without prompting.

3.10 Editor

The editor will be invoked when a file is loaded or when the New command is selected.

The editor in the IDE is a full screen editor, designed for program editing and not word processing. Features like word wrap are not available.
Many editor commands are invoked through a compound key sequence (or keystroke). A compound keystroke is accomplished by holding down one key, pressing another, and then releasing them both. Compound key sequences are indicated with a plus sign. For example pressing and holding down the Ctrl key, pressing the L key and releasing them both performs the Ctrl+L compound keystroke. Some compound key strokes require three keys. Pressing and holding the Shift key, pressing and

holding the Ctrl key, pressing the PgUp key and releasing all three keys perform for example Shift+Ctrl+PgUp.

Some editor commands utilize a compound keystroke and a single keystroke. For example the key sequence Ctrl+Q Y is composed of a compound keystroke and a single keystroke. Pressing and holding down the Ctrl key, pressing the Q key, releasing them both, then pressing and releasing the Y key performs this example. This type of key sequence requires that the Ctrl key be released before the second key is pressed. For example Ctrl+Q Ctrl+Y is not the same as Ctrl+Q Y.

See Also

Basic Editor Operations
Cursor movement
Line Editing
Word Editing
Block Editing
Clip Board
Text Search
Text Highlighting
Miscellaneous Keyboard Commands
Mouse Operations

3.11 Basic Editor Operations

Key Stroke	Description
Insert	Toggles between insert and over type mode (notice the status bar pane change from INS to OVR or from OVR to INS). In insert mode the character under the cursor will be pushed to the right and the character typed will be inserted at the cursor location. In over-type mode the character typed will replace the character under the cursor.
Delete	Removes the character under the cursor. All characters to the right will be moved one location to the left.
Tab	If no text is selected (highlighted) pressing the Tab key advances the cursor to the next tab stop in the editor. Spaces are inserted into the editor from the initial cursor location to the new location. If the selected text spans more than one line then the block of selected text is indented with spaces. The number of spaces added is equal to the current tab setting. If the selected text is contained to a single line then pressing the Tab key does nothing.
BackSpace	Deletes the character to the left of the cursor. All characters to the right and under the cursor shift left. If the cursor is at the start of a line the current line will be added to the previous one.
Ctrl+Z	Undoes last edit action.
Alt-BackSpace	Same a Ctrl+Z
Ctrl+Y	Redoes last undone edit action.

3.12 Cursor movement

Key Stroke	Description
Home	Moves the cursor to the start of the current line.
End	Move the cursor to the end of the current line.
PgUp	Displays the previous page of text. If you are at the top of the text buffer then pressing PgUp will be ignored.
PgDn	Displays the next page of text. If you are at the bottom of the text buffer then pressing PgDn will be ignored.
UpArrow	Moves the cursor up one line. If the cursor is at the top of the text buffer then pressing UpArrow will have no effect.
DownArrow	Moves the cursor down one line. If the cursor is on the last line of the text buffer pressing DownArrow will have no effect.
RightArrow	Moves the cursor one character location to the right. If the cursor is at the end of a line then it will be positioned to the start of the next line.
LeftArrow	Moves the cursor one character location to the left. If the cursor is at the start of a line then the cursor is moved to the end of the previous line.
Ctrl+PgUp	Go to top of document: Moves the cursor to the top of the text buffer.
Ctrl+PgDn	Go to end of document: Moves the cursor to the bottom of the text buffer.
Ctrl+Home	Same as Ctrl+PgUp.
Ctrl+End	Same as Ctrl+PgDn.
Ctrl+UpArrow	Scroll up: Scrolls the text window up one line. All lines will move down, the bottom line will scroll off the screen and the next line (if any) will replace the top line.
Ctrl+DownArrow	Scroll down: Scrolls the text window down one line. All lines will move up, the top line will scroll of the screen and a next line (if any) will replace the bottom line.
Shift+Tab	If no text is selected (highlighted) pressing the Shift+Tab key sequence backs up the cursor to the previous tab stop in the editor. If the selected text spans more than one line then the indent of the selected text is removed. The number of spaces removed is equal to the current tab setting. If the selected text is contained to a single line then pressing the Shift+Tab key sequence does nothing.
Ctrl+Q B	Go to beginning of block: Moves the cursor to the start of the currently marked block (if any).
Ctrl+Q K	Advance to end of block: Moves the cursor to the end of the currently marked block (if any).
Ctrl+Q E	Go to window top: Moves the cursor to the line displayed at the top of the currently

displayed text. Only the cursor moves.

Ctrl+Q X	Go to window bottom: Moves the cursor to the last line displayed on the screen. Only the cursor moves.

3.13 Line Editing

Key Stroke	Description
Ctrl+Q Y	Delete rest of line: Deletes all character from the current cursor location to the end of the current line, including the character under the cursor.
Ctrl+Q S	Go to start of line: Moves the cursor to the start of current line (same as the Home key).
Ctrl+Q D	Go to end of line: Moves cursor to the end of the current line (same as the End key).

3.14 Word Editing

Key Stroke	Description
Ctrl+K T	Highlight word: Highlights (block marks) the word the cursor is currently on.
Ctrl+LeftArrow	Word left: Moves the cursor left one word (to the next blank space).
Ctrl+RightArrow	Word right: Moves the cursor right one word (to the next blank space).
Ctrl+T	Delete word: Deletes the word the cursor is currently on.

3.15 Block Editing

Key Stroke	Description
Ctrl+K I	Indent block: Adds a blank space at the start of all lines contained in the marked block.
Ctrl+SpaceBar	Same as Ctrl+K I
Ctrl+K U	Unindent block: Removes a blank space (if any) from all lines contained in the marked block.
Shift+Ctrl+SpaceBar	Same as Ctrl+K U

3.16 Clip Board

Key Stroke	Description
Ctrl+C	Copy to clip board: Copies the currently selected block of text to the clipboard (temporary storage area).
Ctrl+X	Cut to clipboard: Copies the currently selected block of text to the clipboard. Then

	removes the marked block from the text buffer.
Ctrl+V	Paste from clipboard: Copies the previously stored text from the clipboard to the current cursor position.

3.17 Text Search

Key Stroke	Description
Ctrl+F	Find string: This command prompts you for a string of characters. The cursor will be positioned just after the first occurrence of the specified search string.
Ctrl+H	Find and replace string: This command prompts you for two strings of characters. The first is the string to locate and the second the string to replace the one located.
F3	Repeat last find or find/replace: This command positions the cursor just after the next occurrence of the string specified by the last Find or Find and replace string command (which ever was performed last).

3.18 Text Highlighting

Key Stroke	Description
Shift+LeftArrow	Extend highlight left: Adds the character to the left of the cursor to the marked block.
Shift+RightArrow	Extend highlight right: Adds the character to the right of the cursor to the marked block.
Shift+UpArrow	Extend highlight up: Adds the rest of the current line and the right side of the previous line to the marked block.
Shift+DownArrow	Extend highlight down: Adds the rest of the current line and the left side of the next line to the marked block.
Shift+Home	Highlight to start of line: Moves the start of the block marker to the beginning of the current line.
Shift+End	Highlight to end of line: Moves the end of block marker to the end of the current line.
Shift+Ctrl+PgUp	Highlight to start of buffer: Moves the start of block marker to the beginning of the text buffer.
Shift+Ctrl+Home	Highlight to start of buffer: Performs the same function as Shift+Ctrl+PgUp.
Shift+Ctrl+PgDn	Highlight to end of buffer: Moves the end of block marker to the end of the text buffer.
Shift+Ctrl+End	Highlight to end of buffer: Performs the same function as Shift+Ctrl+PgDn.
Shift+Ctrl+LeftArr	Highlight word left: Extends the highlight to include the word to the left of the cursor.

Shift+Ctrl+RightArr	Highlight word right: Extends the highlight to include the word to the right of the cursor.

3.19 Miscellaneous Keyboard Commands

Key Stroke	Description
Ctrl+N	New: Opens a new empty editor window.
Ctrl+O	Open: Allows you to load a file into a new editor window for editing.
Ctrl+S	Save text: Saves the text currently in the buffer under the current file name. If this is a new file and has never been saved you will be prompted to enter a name.
Ctrl+F7	Assemble: Assembles the source code in the currently active editor window.
F7	Build: Assembles the current project file.
Ctrl+F5	Start Simulator: Starts and initializes the simulator engine.
Ctrl+Shift+F5	Restart Simulator: Resets the simulation engine bringing the program counter to zero.
Shift+F5	Stop Simulator: Stops the running simulation engine.
F5	Continue: Continues program execution from the current program location.
F11	Step Into: Steps into the next instruction. If this instruction is a CALL then program execution stops at the first instruction of the subroutine.
F10	Step Over: Steps over the next instruction. If this instruction is a CALL then program execution stop when it returns from the subroutine.
Shift+F11	Step Out: Steps out of the current subroutine. Execution continues until a return is executed. Program execution stops at the instruction following the CALL to the subroutine being stepped out of.
Ctrl+F10	Run to Cursor: Executes the program until the instruction at the cursor location is reached.
F9	Toggle Break Point: Sets or clears a break point at the current cursor location. If the line the cursor is one does not contain a break point then one is placed there. If the line does contain a break point then it is removed.

3.20 Mouse Operations

Many of the previously mentioned keyboard operations can also be performed using the mouse. The following lists available mouse operations.

Position Cursor: the cursor can be position in the editor windows by moving the mouse pointer to the

desired location and clicking the left mouse button.

Text Highlighting: text can be selected by moving the mouse pointer to the start of the text to be selected. Click and hold down the left mouse button. While holding down the left mouse button move the mouse pointer to the end of the text to be selected. Text will be highlighted as the mouse pointer is dragged across the window.

Highlight Word: a word (text surrounded by blanks) can be selected by positioning the mouse pointer over the desired text. While the mouse pointer is over the desired text double clicking the left mouse button.

Popup Menu: a popup menu is displayed by clicking the right mouse button with the pointer over the editor window. A context sensitive menu will be display from which commands can be selected. The menu options correspond to Edit menu commands and perform the same operation.

4 Editor Commands

4.1 Edit menu commands

The Edit menu offers the following commands:

Undo	Reverse previous editing operation.
Cut	Deletes data from the document and moves it to the clipboard.
Copy	Copies data from the document to the clipboard.
Paste	Pastes data from the clipboard into the document.
Find	Locates the specified text in the active editor window.
Find In Files	Locates the specified text in the specified file group.
Replace	Locates and replaces the specified text in the active window with the specified text.

4.2 Undo/Can't Undo command (Edit menu)

Use this command to reverse the last editing action, if possible. The Undo command is grayed if you cannot reverse your last action.

Shortcuts

Keys: Ctrl+Z or
Alt-BackSpace

4.3 Redo command (Edit menu)

Use this command to reverse the last undone editing action, if possible. The Redo command is grayed if you cannot reverse your last undone action. Undo actions can only be redone as long as no edit action other than Undo is performed. When an edit action other than Undo is performed the Redo command list is erased.

Shortcuts

Keys: Ctrl+Y

4.4 Cut command (Edit menu)

Use this command to remove the currently selected data from the document and put it on the clipboard. This command is unavailable if there is no data currently selected.

Cutting data to the clipboard replaces the contents previously stored there.

Shortcuts

Toolbar: 
Keys: Ctrl+X

4.5 Copy command (Edit menu)

Use this command to copy selected data onto the clipboard. This command is unavailable if there is no data currently selected.
Copying data to the clipboard replaces the contents previously stored there.

Shortcuts

Toolbar: 
Keys: Ctrl+C

4.6 Paste command (Edit menu)

Use this command to insert a copy of the clipboard contents at the insertion point. This command is unavailable if the clipboard is empty.

Shortcuts

Toolbar: 
Keys: Ctrl+V

4.7 Find command (Edit menu)

Use this command to search for specified text in the currently active window. Selecting this command causes the Find dialog box to be displayed.

Shortcuts

Keys: Ctrl+F

See Also

Find Dialog

4.8 Find In Files command (Edit menu)

Use this command to search for text in specified files and specified directories.

Each line of text located that match the specified search criteria will be added to the Output window. You can then double click on entries in the output window to load the associated file and have the cursor positioned to the start of the appropriate line.

Shortcuts

Toolbar: 

See Also

Find In Files Dialog

4.9 Replace command (Edit menu)

Use this command to search for and replace specified text in the currently active window. Selecting this command causes the Replace dialog box to be displayed.

See Also

Replace Dialog

4.10 Find Dialog

The following options allow you to specify the text to locate:

Find what

Type or select the text you want to search for. Clicking on the down arrow at the right of this text box will cause a list of previous search texts to be displayed. Select the text from the drop down list box or type the text you want to search for.

Match whole words only

Select this check box is you want to search for whole words only. When checked the specified text must be surrounded by blanks; or be at the start of a line followed by a space; or at the end of the line proceeded by a space; or the only word on the line. If this box is not checked then text will be found even if it is part of a word.

Match case

Select this check box if you want to perform a case sensitive search. Remove the check if you want to search regardless of case.

Direction

Up: search starting at the current cursor location towards the start of the document. If not found before the start of the document is reached then it continues at the end of the document.
Down: search starting at the current cursor location towards the end of the document. If not found before the end of the document is reached then is continues at the start of the document.

Find Next
Starts or continues the search for the specified text in the specified direction.

Cancel
Stops or cancel the search.

When Find Next is selected the current search parameters are saved for the next search operation. The Find what text is also added to the list of previous search texts. It is added to the start of the list and the last item in the list is discarded if room is not available to retain it.

4.11 Find In Files Dialog

The following options allow you to specify the text to locate:

Find what
Type or select the text you want to search for. Clicking on the down arrow at the right of this text box will cause a list of previous search texts to be displayed. Select the text from the drop down list box or type the text you want to search for.

In file type
Type or select the file filter to use when searching files. Clicking on the down arrow at the right of this text box will cause a list of previous file filters to be displayed. Select the file type from the drop down list box or type it into the text box.

In folder
Type or select the directory to look in for the specified file types. Clicking on the down arrow at the right of this text box will cause a list of previous folders to be displayed. Select the folder from the drop down list box or type it into the text box.

Match whole words only
Select this check box is you want to search for whole words only. When checked the specified text must be surrounded by blanks; or be at the start of a line followed by a space; or at the end of the line proceeded by a space; or the only word on the line. If this box is not checked then text will be found even if it is part of a word.

Match case
Select this check box if you want to perform a case sensitive search. Remove the check if you want to search regardless of case.

Search subdirectories
Select this check box if you want to search in directories within the on previously specified. If this box is not selected it will look only in the specified directory.

Find
Starts the search for the specified text in the specified files types and directories.

Cancel
Stops or cancel the search.

...
Displays the Choose Directory Dialog box.

Each line of text located that match the specified search criteria will be added to the Output window. You can then double click on entries in the output window to load the associated file and have the cursor positioned to the start of the appropriate line.

When Find is selected the current search parameters are saved for the next search operation. The Find what, In files/file types and In folder texts are also added to their associated lists of previous search texts. They are added to the start of the list and the last item in the list is discarded if room is not available to retain it.

4.12 Choose Directory Dialog

The following options allow you to specify the text to locate:

Directory name
Type or select the name of the desired directory. Double clicking on a upper level directory will close that list and display directories at the level of the selected directory. Double clicking on a lower level directory will open that directory and display the directories (if any) within it.

Drives
Is a drop down list of drives either local or network mapped that are currently available on the computer. Once another drive is selected the directory tree will be updated to reflect the directories on that drive.

OK
Closes the dialog and returning to the calling dialog changing the directory listed in that dialog.

Cancel
Closes the dialog box without changing the directory in the calling dialog.

4.13 Replace Dialog

The following options allow you to specify the text to locate:

Find what
Type or select the text you want to search for. Clicking on the down arrow at the right of this text box will cause a list of previous search texts to be displayed. Select the text from the drop down list box or type the text you want to search for.

Replace with
Type or select the text you want to replace the found text with. Clicking on the down arrow at the right of this text box will cause a list of previous search texts to be displayed. Select the text from the drop down list box or type the text into the edit box.

Match whole words only
Select this check box is you want to search for whole words only. When checked the specified text must be surrounded by blanks; or be at the start of a line followed by a space; or at the end of the line proceeded by a space; or the only word on the line. If this box is not checked then text will be found even if it is part of a word.

Match case

Select this check box if you want to perform a case sensitive search. Remove the check if you want to search regardless of case.

Direction

Up: search starting at the current cursor location towards the start of the document. If not found before the start of the document is reached then it continues at the end of the document.
Down: search starting at the current cursor location towards the end of the document. If not found before the end of the document is reached then is continues at the start of the document.

Find Next

Starts or continues the search for the specified text in the specified direction.

Replace

If the search has begun and text has been found clicking this button will cause the located text to be replaced with the specified text. If the search has not begun then clicking this button performs the same action as Find Next.

Replace All

If the search has begun and text has been found clicking this button will cause the located text to be replaced with the specified text and all remaining occurrences to be located and replaced automatically. If the search has not begun then clicking this button will search out all occurrence of the specified text and automatically replace it with the specified text.

Cancel

Stops or cancel the search.

When Find Next, Replace, or Replace All is selected the current search and replace parameters are saved for the next search and replace operation. The Find what and Replace with text is added to their corresponding lists of previous search texts. They are added to the start of the list and the last item in the list is discarded if room is not available to retain it.

5 View Menu

5.1 View menu commands

The View menu offers the following commands:

Toolbar	Shows or hides the toolbar.
Status Bar	Shows or hides the status bar.
Output	Opens an Output window.
Registers	Opens a Registers window.
Ports	Opens a Ports window.
Direct Memory	Opens a Direct Memory window.
Indirect Memory	Opens an Indirect Memory window.
External Memory	Opens an External Memory window.
Watch Window	Opens a Variable Watch window.

5.2 View Toolbar command

Use this command to display and hide the Toolbar, which includes buttons for some of the most common commands in the 8051 IDE, such as File Open. A check mark appears next to the menu item when the Toolbar is displayed.

See Toolbar for help on using the toolbar.

5.3 Toolbar

The toolbar is displayed across the top of the application window, below the menu bar. The toolbar provides quick mouse access to many tools used in the 8051 IDE,

To hide or display the Toolbar, choose Toolbar from the View menu (ALT, V, T).

Click	To
	Open a new document.
	Open an existing document. The 8051 IDE displays the Open dialog box, in which you can locate and open the desired file.
	Save the active document with its current name. If you have not named the document, the 8051 IDE displays the Save As dialog box.
	Saves all open documents with their current name. If you have not named a document, the 8051 IDE displays the Save As dialog box for each unnamed document.
	Remove selected data from the document and stores it on the clipboard.
	Copy the selection to the clipboard.
	Insert the contents of the clipboard at the insertion point.
	Searches files for specified text.
	Assembles the source code in the currently active editor window.
	Assembles the source file(s) indicated by the current project file.
	Add variable to Watch window. The text under the cursor will be presented as the variable name. You can then accept this text or edit and change it.
	Steps into the next instruction. If this instruction is a CALL then program execution stops at the first instruction of the subroutine.
	Steps over the next instruction. If this instruction is a CALL then program execution stop when it returns from the subroutine.
	Steps out of the current subroutine. Execution continues until a return is executed. Program execution stops at the instruction following the CALL to the subroutine being stepped out of.
	Executes the program until the instruction at the cursor location is reached.

Continues program execution from the current program location.

Set or clears a break point at the current cursor location. If the line the cursor is one does not contain a break point then one is placed there. If the line does contain a break point then it is removed.

Resets the simulation engine bringing the program counter to zero.

Stops the running simulation engine.

Stops program execution at its current location.

5.4 View Status Bar Command

Use this command to display and hide the Status Bar, which describes the action to be executed by the selected menu item or depressed toolbar button, and keyboard latch state. A check mark appears next to the menu item when the Status Bar is displayed.

See Status Bar for help on using the status bar.

5.5 Status bar

The status bar is displayed at the bottom of the 8051 IDE window. To display or hide the status bar, use the Status Bar command in the View menu.
The left area of the status bar describes actions of menu items as you use the arrow keys to navigate through menus. This area similarly shows messages that describe the actions of toolbar buttons as you depress them, before releasing them. If after viewing the description of the toolbar button command you wish not to execute the command, then release the mouse button while the pointer is off the toolbar button.

The right areas of the status bar indicate the following:

Indicator	**Description**
INS	Indicates the state of the Insert/Overtype mode of the editor.
Ready	Indicates the status of back ground processing (Ready or Busy).
Ln 1, Col 1	Indicates the Line and Column position of the cursor in the editor.

5.6 Output command (View menu)

Use this command to display a new output (message) window.

See Also

Output window

5.7 Registers command (View menu)

Use this command to display a new register window.

See Also
Registers window

5.8 Control Registers command (View menu)

Use this command to display a new control register window.

See Also
Control Registers window

5.9 Ports command (View menu)

Use this command to display a new ports window.

See Also
Ports window

5.10 Direct Memory command (View menu)

Use this command to display a new Direct Memory (DRAM) window.

See Also
DRAM window

5.11 Indirect Memory command (View menu)

Use this command to display a new Internal Memory (IRAM) window.

See Also
IRAM window

5.12 External Memory command (View menu)

Use this command to display a new External Memory (XRAM) window.

See Also
XRAM window

5.13 Watch Window command (View menu)

Use this command to display a new Watch window.

See Also

Watch window

6 Assembler Basics

6.1 Assembler

Selecting Assemble from the Assemble menu assembles the source in the currently active editor window. The number of errors found in the source will be displayed in the Output window.

If any errors are located double clicking on an error message will load the associated source code and position the cursor at the location of the line containing that error.

NOTE: Some errors will generate additional errors, usually the first error is the actual offense. If you fix an error and are unable to see the problem with a subsequent error try reassembling the source.

Once an error is fixed, double clicking on other listed errors will load the associated source code and position the cursor on the corresponding line If you add a new line or delete an existing line the offending line no longer has the same line number as the one in the error list. For this reason if you add or delete lines containing errors reassemble the source to regenerate an accurate error/line number list. If no errors are reported you can simulate the source.

Refer to 8051 source format, for a description of the layout of an 8031 assembly language program.

6.2 Project File

The Project option allows you to specify the main source file in a multi-file project. The project file contains include statements which list all the files in the project. This allows you to assemble your project without having to first load the main file. To specify a project file select the Project command from the Assemble menu and type in the file name or select one from the list.

Use the Close Project command to clear the specified project file name.

Refer to the section titled Multi-file Projects for an explanation of how to setup a multi-file project.

6.3 File Generation

The 8051 assembler generates a number of files when it assembles either an individual file or a project. These files have the same name as either the individual file being assembled or the name of the project being built. They will however have the following extensions

HEX Intel hex format of the assembled binary code.
LST List file containing line numbers, address, binary code and source lines of the file being assembled or the total project being built.

6.4 8051 Source Format

The assembler mnemonics, labels and symbols are not case sensitive. In other words LOOP is the same as loop to the assembler. Each line of the source code has the following format

LABEL: OPCODE OPERANDS ;COMMENT

each of these fields are defined below.

LABEL: Each line of source can contain an optional label. Labels may contain letters, digits and an under score (_) character. The label must start with a alphabetic character and terminate with a colon ":". Labels can contain as many characters as you want although the assembler only recognizes the first 32. The first time a label is encountered its name along with the current value of the program counter is added to the label table. When the label is used as an operand the value of the program counter associated with that label is substituted.

OPCODE: Opcodes can be either an 8051 mnemonic or an assembler directive. A list of mnemonics and their operands are given in appendix A. The following section contains a list of assembler directives. Opcodes must be separated from labels and operands by at least one blank.

OPERAND: Operands are added after a mnemonic or directive to indicate what the operation is to be performed on.

For example in the instruction: MOV R1,A

The operands are R1 and A. A comma, as in the example just given must separate multiple operands.

COMMENT: Comments begin with a semicolon and instruct the assembler to ignore all text from the semicolon to the end of the line.

SYMBOL: A symbol is a character string, which represents a specific value. For example the source line

```
CR  EQU  13
```

Instructs the assembler to equate the value 13 with the character sequence CR. Symbols must conform to the same requirements as labels (but they are terminated with a blank instead of a colon).

EQUATIONS: The assembler has a built in equation evaluator. Opcodes requiring an immediate value or directives requiring a value can be constructed into an equation. For example in the following source line

```
MOV  A,8*4
```

8 * 4 is the equation which will be evaluated. Equations can be comprised of numbers, labels, symbols, operators and parenthesized sub-equations.

Numbers must start with a digit between 0 and 9. Terminating with a B, H or D respectively can specify binary, hexadecimal and decimal numbers. For example 01000001b, 041h and 65d are binary, hexadecimal and decimal representations of the same value. If the radix is not specified then decimal is assumed. Although the default assembler radix can be changed with the RADIX assembler directive. Values for ASCII characters can be specified by surrounding the character with single quotes, for example 'A' will cause the ASCII value (65) for the character to be substituted. Labels and symbols can be used if previously defined or equated to a value. Labels and symbols not defined prior to their use can be used in expressions if their value is not required on the first assembler pass. For example in the source lines

```
        DB      'hello', CR, LF
CR      EQU     13
LF      EQU     10
```

the value of CR and LF are not needed on the first pass, their values can be substituted on the second pass. The only exception is the DS (Define Space) directive. An example of this is

```
        DS      BUFSIZE
BUFSIZE EQU     20
```

This code segment will generate a phase error. On the first pass the assembler must know how many bytes to allocate for the DS (Define Space) directive in order for it to continue determining address values for subsequent labels. For this reason labels and symbols used in the DS directive must be defined prior to their use.

The available operators are listed below in order of precedent

()	parenthesized sub-equations
High	returns the high byte of an integer value/result. Example: if val is equal to 0x1234 High (val) returns 0x12
Low	returns the low byte of an integer value/result. Example: if val is equal to 0x1234 Low (val) returns 0x34
HighWord	Returns the high word (two-byte) of an integer value/result. Example: if val is equal to 0x12345678 HighWord(val) returns 0x1234
LowWord	Returns the low word (two-byte) of an integer value/result. Example: if val is equal to 0x12345678 LowWord(val) returns 0x5678
^	exponential
*	multiplication
/	division
%, MOD	modular, either type can be used.
&, AND	logical AND, either type can be used.
|, OR	logical OR, either type can be used.
+	addition
-	subtraction
+, -	unary operations
.n	dot operator (20H.1=33, 20H.2=34, ... 20H.7=39)

6.5 Assembler Directives

The following is a list of directives supported by the 8051 Assembler:

ORG: Originate. Resets the program counter (at assembly time) to a specific value. For example

the source line

```
        ORG     7
```

resets the program counter to the value 7.

DATAORG: Data Originate. Resets the data address counter used by the DS directive. For example the source code segment

```
        DATAORG     10
VAR1    DS      1
VAR2    DS      1
```

sets the symbols VAR1 and VAR2 to 10 and 11 respectively.

EQU: Equates a value with a character string (Symbol). For example the source line

```
LF      EQU     10
```

adds the symbol LF to the symbol table with the associated value 10. Each time LF is used as an operand the value 10 will be substituted. A symbol can be equated only once in a project. Equated symbols are evaluated to a numeric result. The EQU directive does not perform a character substitution.

SET: Like EQU, SET sets a symbol to a specific value. A symbol's value can be modified at more than one location in the source file by the SET command.

BIT: Equates a bit address with a character string (Symbol). For example the source line

```
HoldPin BIT     80h
```

adds the symbol HoldPin to the symbol table with the associated bit address 80h (128 decimal, same as bit 0 of port 0). Each time HoldPin is used as an operand the value 80h will be substituted. A symbol can be defined using the BIT operation only once in a project. Bit defined symbols are evaluated to a numeric result. The BIT directive does not perform a character substitution.

DATA: Equates an address with a character string (Symbol). For example the source line

```
Count1  DATA    10h
```

adds the symbol Count1 to the symbol table with the associated address 10h (16 decimal). Each time Count1 is used as an operand the value 10h will be substituted. A symbol can be defined using the DATA operation only once in a project. Defined symbols are evaluated to a numeric result. The DATA directive does not perform a character substitution.

DB: Define byte. This directive places a value or string of values in program memory at the current program counter location (one per memory location). For example

```
NUM:    DB      10
MSG:    DB      'HELLO WORLD',13,10
```

are legal usage of the DB directive. Each value (or values) will occupy one memory location.

DW: Define word. This directive places a value or string of values in program memory at the current program counter location one per 2 bytes of memory. For example

```
NUM:  DW     10
MSG:  DW     'HELLO WORLD',13,10
```

are legal usage of the DW directive. Each value (or values) will occupy two memory locations (in LS MS order).

DS: Define space. This directive will allocated a specified number of data memory locations starting at the current data address counter location. For example

```
BUF   DS     10
```

will allocate 10 bytes of memory starting at the location associated with BUF. This directive is used to allocate space in internal and external data memory.

PAGE rows: Rows is optional, if omitted the listing file is advanced to the top of the next page. If specified this value indicates the number of lines per page in the listing file (e.g. PAGE 66). The default lines per page is 66.

%TITLE "title string": This directive allows you to specify the title to be displayed in the listing file. You can change the title as often as you like. But since the title is displayed at the top of the first page the last defined title in your source file will be displayed. This is due to the fact that the title is displayed before any lines of the source file are processed for the second assembly pass. They have all been processed for the first pass.
The quotes around the title string are required. Every character after the first quotation mark to the character just before the next quotation mark will be included in the title.

%SUBTTL "subtitle string": This directive allows you to specify the subtitle to be displayed in the listing file. This subtitle is displayed starting will the second page and can be changed as often as you like. The last declared subtitle will be displayed in the title block of each page (except the first page).

%TOPMAR val: Allows you to specify the number of lines in the top margin of the listing file. This value must be greater than 4 and must be less than the page length minus the bottom margin (defined below).

%BOTMAR val: Allows you to specify the number of lines in the bottom margin of the listing file. This value must be greater than 0 and must be less than the page length minus the top margin (defined above).

END: Instructs the assembler to stop assembling the source file (ignore any text, which follows the END directive). The END directive is not required, the assembler will stop assembling either when it reaches an END directive or the end of the text buffer.

RADIX: Allows you to change the default radix conversion from decimal to hexadecimal or binary. The following lists the possible usage:

RADIX DEC - sets the default conversion radix to decimal.
RADIX HEX - sets the default conversion radix to hexadecimal.
RADIX BIN - sets the default conversion radix to binary.

INCLUDE: Allows you to build and simulate multi-file projects. Allowing for the creation of programs which are to large to load into the editor. The format for the include directive is:

INCLUDE "filename.ext"

The name of the included file must be enclosed in quotes. No extension is assumed and must therefor be specified. The quotation marks are optional. You can also use the traditional $ include format. This format of the include directive is:

$filename.ext

An END directive in an included file will stop the assembly of that include file but not of the project as a whole.

* The INCLUDE directive is only available in the registered version of the 8051 IDE.

BREAK: Specifies a break point in your code. This will cause the Simulator to suspend your program when this point is reached.

MEMTYPE: Specifies the type of memory subsequent variables (symbols) will be referencing. This allows the simulator to automatically select the Resource Type when adding a watch from the edit window. This directive has the following format

MEMTYPE reftype

Were: reftype is one of the following

IRAM – internal Indirect RAM
DRAM – internal Direct RAM
XRAM – external RAM

Specifying no reference type sets the default symbol reference to undefined.

HEXFILE: Specifies an alternate name for the generated hex file. The default is derived from the name of the project file. The format for the include directive is:

HEXFILE "filename.hex"

6.6 Inline Intel Hex Formatted Code

Previously assembled Intel Hex formatted code can be inserted directly into a source file. For example the following Intel Hex formatted program code

:108000007581B71282F712817320930B128242207E

can be inserted directly into your source file. You can also use the INCLUDE directives to insert an entire hex file into your project. A line that starts with a colon (:) is assumed to be an Intel Hex formatted code segment. An Intel Hex formatted code segment has the following format

:NNAAAA00DDDD.....DDCS

Were

:	signals the start of an Intel Hex formatted segment. Must be the first non-space character of the line. Otherwise the line will not be recognized as inline code.
NN	is a two-character (1-byte) ASCII hex value indicating the number of data bytes (DD).
AAAA	is a four-character (2-byte) ASCII hex value indicating the address of the code segment.
00	is a two character (1-byte) ASCII hex format identifier and must be 00 (no other format is recognized).
DD	is a two-character (1-byte) ASCII hex code (or data) byte.
CS	is a two-character (1-byte) ASCII hex value representing the 2's compliment check sum

The Intel Hex formatted code segment is added to your program code starting at the current program counter address in the following sequence.

NNAAAA00DDDD.....DD

All values (except for the check sum) are added to the program code.

The following is an example code segment that adds a Hex file named flashup.hex to the program code.

```
RelCodeSeg:     ; start of code segment to be relocated
$flashup.hex    ; include raw (already assembled) code segment
RelCodeSegEnd:  ; end of code segment to be relocated
```

The Intel Formatted code will be added to the main programs code base starting at the address associated with RelCodeSeg. The end of this added code segment is identified by the address associated with RelCodeSegEnd.

6.7 Conditional Assembly

The assembler allows you to conditionally assemble segments of your code. This allows you to assemble (or not assemble) various portions of your program. The following illustrates the usefulness of conditional assembly. In this example OFFSET is a constant, LOWBYTE and HIGHBYTE are internal memory locations. OFFSET does not change at runtime, and LOWBYTE and HIGHBYTE do.

```
OFFSET  EQU   1000H
MOV     A,LOWBYTE
ADD     A,#LOW OFFSET
MOV     LOWBYTE,A
MOV     A,HIGHBYTE
ADDC    A,#HIGH OFFSET
MOV     HIGHBYTE,A
```

In this example zero is added to the value LOWBYTE and does not change it. Therefor the code performs no useful purpose and adds overhead to the program. You could simply delete the code segment although this would cause problems if the value of OFFSET changes. You can add conditional assembly directives to help optimize the code. The following illustrates the modified code.

```
OFFSET  EQU     1000H
        CLR     C                       ; clear in case the following
                                        ; code is omitted.
IF      LOW(OFFSET) <> 0                ; if low byte is none zero
        MOV     A,LOWBYTE               ;  include code to add
        ADD     A,#LOW OFFSET           ;  low byte offset.
        MOV     LOWBYTE,A
ENDIF
        MOV     A,HIGHBYTE
        ADDC    A,#HIGH OFFSET
        MOV     HIGHBYTE,A
```

Since the low byte of OFFSET is zero the code is removed (not added) to the runtime program. Although if at a later time the value of OFFSET changes (to say 1010h) then the code would be included in the assembled program.

It is important to note that this code is added or omitted from the assembled machine code. The condition is tested at assembly time and not run time. Therefor the condition of the test must not be able to change during runtime. If the code is omitted the only way to get it back is to reassemble and reprogram the circuit.

The conditional test operators are: <, >, =, <> (less than, greater than, equal and not equal).

In the previous example we include or don't include a segment of code. Using an ELSE construct we can assemble one of two program segments. For example

```
HARDWARE   EQU    1
IF HARDWARE = 1
       MOV    R0,#10
ELSE
       MOV    R0,#20
ENDIF
```

This example allows you to maintain one version of source code for two versions of hardware. In one version you need to load R0 with the value 10 and another version with the value 20. To create firmware for each of the two versions you simply change the value HARDWARE gets equated to and reassemble.

6.8 Multi-file Projects

The 8051 assembler recognizes a number of directives, which it treats as comment lines. These directives are:
NAME, RSEG, PUBLIC, and EXTERN

These directives are used by some assemblers to allow a program to be broken into separate source modules. They allow each module to be assembled separate of each other. Since this 8051 assembler does not support true program (source) modules they have no purpose.
Use the INCLUDE directive to build multi-file projects. You can use a separate source file, which simply lists each of the modules comprising the project. The following is an example project file and has the file name PROJECT.A51

```
INCLUDE "DEFINES.A51"      ; EQUATED SYMBOLS

DATAORG 0                  ; ORIGINATE VARIABLES AT LOC 0
INCLUDE "DEFINEI.A51"      ; INTERNAL VARIABLE DEFINITIONS

DATAORG 0                  ; ORIGINATE VARIABLES AT LOC 0
INCLUDE "DEFINEX.A51"      ; EXTERNAL VARIABLE DEFINITIONS

ORG     0                  ; ORIGINATE CODE AT LOC 0
INCLUDE "VECTORS.A51"      ; STARTUP AND INTR VECTORS
INCLUDE "FONT1.A51"        ; CHARACTER FONTS
INCLUDE "FONT2.A51"
INCLUDE "MAIN.A51"         ; MAIN PROGRAM BODY
INCLUDE "XMEM.A51"         ; EXTERNAL MEMORY ROUTINES
INCLUDE "INIT.A51"         ; INITIALIZATION ROUTINES
INCLUDE "STRING.A51"       ; STRING ROUTINES
INCLUDE "MATH.A51"         ; MATH ROUTINES
INCLUDE "SERIAL.A51"       ; SERIAL I/O ROUTINES
```

This project file performs the function of a make (or link) list file. You can precede each module with an ORG directive. This provides the capability of specifying module placement. Since all modules are assembled together the EXTERN and PUBLIC statements are not needed. They are used by other assemblers which compile individual modules providing reference symbols in other modules.

Having the 8051 assembler recognize and ignore these directives in conjunction with a project file helps utilize existing code developed using other assemblers.

The project can be built two different ways. You can load the project file into the editor and invoke the assembler. Although this is a bit inconvenient. Another way is to specify the name of the project file. This option is located in the Assemble menu.

Note that the alternate include format ($filename.ext) can be used instead of the key word "include".
* The INCLUDE directive is only available in the registered version of the 8051 IDE.

7 Assembler Commands

7.1 Assemble menu commands

The Assemble menu offers the following commands:

Assemble	Assembles the source code in the active editor window.
Build	Assembles the source code file(s) indicated by the selected project.
Stop	Stops a build or assembly in progress.
Project	Specifies the name of the project file.
Close Project	Closes (clears) the project name.

7.2 Assemble command (Assemble menu)

Use this command to assemble the source code in the currently active editor window. If the simulation engine is running when this command is selected it will be shut down prior to performing the operation.

Shortcuts

Toolbar: 
Keys: Ctrl+F7

See Also

Start Simulator

7.3 Build command (Assemble menu)

Use this command to assemble the source files indicated by the current project file selection. If the simulation engine is running when this command is selected it will be shut down prior to performing the operation.

Shortcuts

Toolbar: 
Keys: F7

See Also

Project command
Start Simulator

7.4 Stop command (Assemble menu)

Use this command to stop a previously begun assembly of a specific source file.

See Also

Assemble command

7.5 Project command (Assemble menu)

Use this command to specify a project file to be used when the Build command is issued.

See Also

Close Project command (Assemble menu)
Build command (Assemble menu)

7.6 Close Project command (Assemble menu)

Use this command to clear the specified project file. Clearing the project file name will disable the Build command.

See Also

Project command (Assemble menu)Build command (Assemble menu)

8 Simulator Commands

8.1 Simulator

The simulator allows you to step through your source code watching registers, flags, ports and RAM change. You can step into, step over and execute to source lines. Selecting Start Simulator via the Simulate menu enters the simulator.

While the simulator is running it will control the display of source code and values in the associated windows (Registers, DRAM, IRAM, XRAM and Ports). The active window contains the cursor and the windows title bar is not grayed.

The Registers window contains the values currently in the processor's registers. The registers are defined as follows:

AC	Accumulator
B	Multiply, divide, and general purpose register
R0-R7	General-purpose register set
SP	Stack pointer
PC	Program counter
DPTR	Data pointer register (16 bits)

The Control Registers window contains the processor status and control registers. The registers are defined as follows:

PSW	Program Status Word (contains: CY, AC, F0, RS1, RS0, OV, F1 and P)
IP	Interrupt priority control register
IE	Interrupt enable control register
TCON	Timer/Counter control register
TMOD	Timer/Counter mode control register
PCON	Power control register
T2CON	Timer/Counter 2 control register
SCON	Serial port control register
SBUF	Serial transmit/receive buffer register
T0	Timer/Counter 0 (16 bits)
T1	Timer/Counter 1 (16 bits)
T2	Timer/Counter 2 (16 bits)
RCAP2	Timer/Counter 2 Capture register (16 bits)

The Ports window displays the values each of the processor's ports. Ports P0, P1, P2 and P3 reflect

the state on the processors external ports.

The XRAM window reflects the values in the processors external data memory. The number of RAM locations is 64k.

The IRAM window reflects the values in the processors internal memory. The number of RAM locations is dependent on the chip type specified in the options menu.

8.2 Simulate menu commands

The Simulate menu offers the following commands:

Start Simulator	Initializes and starts the simulation engine running.
Restart	Resets the simulation engine bringing the program counter to zero.
Stop	Stops the running simulation engine.
Continue	Continues program execution from the current program location.
Break	Stops program execution at its current location.
Step Into	Steps into the next instruction. If this instruction is a CALL then program execution stops at the first instruction of the subroutine.
Step Over	Steps over the next instruction. If this instruction is a CALL then program execution stop when it returns from the subroutine.
Step Out	Steps out of the current subroutine. Execution continues until a return is executed. Program execution stops at the instruction following the CALL to the subroutine being stepped out of.
Run to Cursor	Executes the program until the instruction at the cursor location is reached.
Toggle Break Point	Set or clears a break point at the current cursor location. If the line the cursor is one does not contain a break point then one is placed there. If the line does contain a break point then it is removed.

8.3 Start Simulator command (Simulate menu)

Use this command to start and initialize the simulation engine. Once started other simulation commands become available. Before this command can be selected a previous assemble or build operation must have been performed. While the simulator is running this command is unavailable (grayed).

Shortcuts

Keys: Ctrl+F5

See Also

Assemble command

Build command
Stopping the Simulator

8.4 Restart command (Simulate menu)

Use this command to restart and initialize the simulated processor to its power up/reset state.

Shortcuts

Toolbar: 
Keys: Ctrl+Shift+F5

See Also

Start Simulator

8.5 Stop command (Simulate menu)

Use this command to stop (shut down) the currently active simulation engine. This command is not available (grayed) if the simulation engine is not running. The simulator must be stop prior to assembling or building source files. Although selecting the Assemble or Build commands will automatically shut down the simulator prior to assembling the source code.

Shortcuts

Toolbar: 
Keys: Shift+F5

See Also

Start Simulator
Assemble
Build

8.6 Continue command (Simulate menu)

Use this command to continue (restart) program execution from the current Program Counter (PC) location. Execution will continue until a break point is reach or the Break command is selected.

Shortcuts

Toolbar: 
Keys: F5

See Also

Toggle Break Point
Break Program Execution

8.7 Break command (Simulate menu)

Use this command to stop the execution of the simulator. This command is grayed if program execution is not in progress.

Shortcuts

Toolbar: 

See Also

Toggle Break Point

8.8 Step Into command (Simulate menu)

Use this command to execute the instruction at the current Program Counter (PC) location. If the instruction is a CALL then simulation stops at the first instruction of the subroutine. Otherwise simulation stops at the instruction following the one being executed.

Shortcuts

Toolbar: 
Keys: F11

8.9 Step Over command (Simulate menu)

Use this command to execute the instruction at the current Program Counter (PC) location. If the instruction is a CALL then simulation stops when it returns from the subroutine. Otherwise simulation stops at the instruction following the one being executed.

Shortcuts

Toolbar: 
Keys: F10

See Also

Break Program Execution

8.10 Step Out command (Simulate menu)

Use this command to execute all of the instructions from the current Program Counter (PC) location to the remaining instructions within the current subroutine. Program execution stops at the instruction following the CALL to this subroutine (instruction to be executed after the return instruction)

Shortcuts

Toolbar: 
Keys: Shift+F11

See Also

Break Program Execution

8.11 Run To Cursor command (Simulate menu)

Use this command to execution the instruction starting at the current Program Counter (PC) location up to the instruction at the current cursor location. Program execution stops when the instruction at the current cursor location is reached.

Shortcuts

Toolbar: 
Keys: Ctrl+F10

See Also

Break Program Execution

8.12 Toggle Break Point command (Simulate menu)

Use this command to set or remove a break point for the instruction at the current cursor location. If a break point already exists at this location it is removed. If one does not exist it is placed there.

Shortcuts

Toolbar: 
Keys: F9

See Also

Remove All Break Points

8.13 Remove All Break Points (Simulate menu)

Use this command to clear all break points that are currently set. If no break points are currently set this command will be unavailable (menu option is grayed).

See Also

Toggle Break Point command

8.14 Simulator Keyboard Commands

The following lists commands invoked via key combinations while the simulator is running:

Key Combination	Command Description
Ctrl+F10	Executes to the currently highlighted source line.
F11	Executes the next instruction stepping into a CALL.
F10	Executes the next instruction stepping over a CALL.

F9	Sets break points in the Source window pane. Sets output radix in Register, IRAM, XRAM and Port window pains.

8.15 Modify Simulated Registers

The Registers, DRAM, IRAM, XRAM and Ports can be modified by moving the cursor to the desired entry and either pressing the Enter key or double clicking on it. A dialog will be displayed prompting you to enter the new value. Entering a blank line will set the item to zero.

8.16 Simulate Execution

Selecting the Step Into command via the Simulate menu, the  toolbar icon or pressing F11 will cause the next instruction to be executed. If the instruction is a CALL then the simulator will step into the subroutine stopping at the first instruction of that subroutine.

Like Step Into the Step Over command via the Simulate menu, the  toolbar icon or pressing F10 will cause the next instruction to be executed. Unlike Step Into if the instruction is a CALL the simulator will execute all instructions below (within) the subroutine being called before returning. You can break the program execution by selecting the Break command via the Simulate menu, the  toolbar icon or pressing the Esc key. Once the Break command is issued the source line being executed will be displayed with the cursor on it.

You can set break points for any instruction line in the simulator Source pane. A break point will cause the simulator to stop (break) execution at a specific instruction. If the next instruction the simulator is to execute has a break point it will stop before executing that instruction. Break points are set by highlighting the desired source line (in the Source pane) and selecting the Toggle Break Point command via the Simulate menu, the  toolbar icon or by pressing the F9 function key. Selecting the Toggle Break Point command while on a line containing a break point will remove that break point.

8.17 Counter and Interrupt Simulation

The 8051 program simulates all three counters (T0, T1 and T2). Timer modes which operate from the oscillator / 12 will increment as instructions are executed. Counter modes, which count transitions of their corresponding pin, are also implemented. Pin transitions can be simulated either by changing the port values in the simulator or by embedding instructions in your source code. For example to toggle T1 and generate a count add CPL P3.5 to your code. The simulator will register the transition and increment the T1 counter (if enabled).

The simulator will also vector on interrupts. All interrupt sources are simulated (external and internal). To generate an external interrupt, simply initialize the appropriate registers and toggle the interrupt line (INT0 or INT1). Setting the port bit low in the simulator and stepping through the code can toggle the interrupt lines. You must step through at least one instruction after setting the port pin low before setting it high again. Machine instructions which set an interrupt pin low can be placed in your code, for example CLR P3.2 (INT0). Unfortunately this type of interrupt generation is not completely

random, as it would be in actual operation.

The simulator executes code as it would on an 8032 processor. For this reason timer 2 and its associated interrupts are simulated regardless of the target processor. If you're developing code for an 8031 be sure to initialize registers such that the T2 counter is disabled. Otherwise the simulator may produce improper results for your target processor.

8.18 Serial Port Simulation

The simulator will simulate the Transmit Buffer empty flag/interrupt caused by the serial transmission of a byte. Only a single serial port (8031) is currently supported. It operates in a fixed 10 bit, mode zero. The only limitation this places on your simulation is a fixed ten-cycle count before the TI bit is set. Actual operation of the 8031 will depend on the serial mode selected in the SCON register.

To simulate serial reception, place a value in the SBUF special function register and set the RI bit in SCON. If interrupts are enable one will be generated and the simulator will vector to the appropriate program location.

8.19 Button Simulation

The simulator allows you to define and use buttons as virtual input devices. To open a virtual button select **View, Virtual, Button**. A virtual button dialog will be displayed. This dialog contains two buttons, a configuration and a user button. Selecting the Configure button displayed the Virtual Button Configuration dialog box.

You can open as many Virtual Buttons as you wish. The number and configuration of each Virtual Button will be saved in the workspace. Saving the workspace will save the placement and configuration of the Virtual Buttons as well.

See Also

Virtual Button Configuration dialog box

8.20 Virtual Button Configuration dialog box

The Virtual Button Configuration dialog box allows you to customize the operation of each Virtual Button. The configuration of a virtual button is performed by selecting the **Configure** button in the associated Virtual Button dialog box.

To configure a virtual button you must specify the following fields

Address: specifies the corresponding port or address in external memory that is modified by the pressing of this button.

Button text: specifies the text to be displayed on your button.

Mask: specifies the mask that is to be applied to this resource when the button is pressed.

Resource Type specifies if the button is to affect a port of external memory location.

Operation: specifies if the button is to be momentary or latching.

Application: specifies how the mask is to be applied to the corresponding resource.

8.21 Simulator Display Values

You can specify an output radix for each register, port and memory location. To specify a radix other than its default, perform the following steps:
Right click on the entry whose radix is to be changed. Select Radix from the pop up menu.

Select the radix Decimal, Hexadecimal, ASCII or Binary.

Once you have changed the display radix for any of the entries they will be saved as the default.

8.22 Registers window

The register window contains the values currently in the simulated processors registers. The registers are defined as follows:

AC	Accumulator
B	Multiply, divide, and general purpose register
R0-R7	General purpose register set
SP	Stack pointer
PC	Program counter
DPTR	Data pointer register (16 bits)

Doubling clicking on a register will display a dialog box allowing you to specify a new value for that register.

See Also

Registers command (View menu)
Modify Value Dialog
Radix Popup menu

8.23 Ports window

The Ports window displays the values each of the simulated processors ports hold. Ports P0, P1, P2 and P3 reflect the state on the simulated processors external ports.

Doubling clicking on a port will display a dialog box allowing you to specify a new value for that port.

See Also

Ports command (View menu)
Modify Value Dialog
Radix Popup menu

8.24 Control Registers window

The Control Registers window contains the simulated processors status and control registers. The registers are defined as follows:

PSW	Program Status Word (contains: CY, AC, F0, RS1, RS0, OV, F1 and P)
IP	Interrupt priority control register
IE	Interrupt enable control register
TCON	Timer/Counter control register
TMOD	Timer/Counter mode control register
PCON	Power control register
T2CON	Timer/Counter 2 control register
SCON	Serial port control register
SBUF	Serial transmit/receive buffer register
T0	Timer/Counter 0 (16 bits)
T1	Timer/Counter 1 (16 bits)
T2	Timer/Counter 2 (16 bits)
RCAP2	Timer/Counter 2 Capture register (16 bits)

Doubling clicking on a Control Register entry will display a dialog box allowing you to specify a new value for that Control Register .

See Also

Control Registers command (View menu)
Modify Value Dialog
Radix Popup menu

8.25 XRAM window

The XRAM window reflects the values in the simulated processors external data memory. The number of RAM locations is 64k.

Doubling clicking on a XRAM entry will display a dialog box allowing you to specify a new value for that XRAM location.

See Also

External Memory command (View menu)
Modify Value Dialog

8.26 IRAM window

The IRAM window reflects the values in the simulated processors indirect memory. The number of memory locations is dependent on the processor type specified in the options menu.

The first 128 DRAM and IRAM locations occupy the same physical memory. The upper 128 locations of both DRAM and IRAM occupy unique physical memory locations. In addition depending on the processor type the upper 128 locations of IRAM may or may not be present. Specifying the processor type in the Options menu will result in the number of IRAM locations being changed to

reflect the selected processor.

Doubling clicking on a IRAM entry will display a dialog box allowing you to specify a new value for that IRAM location.

See Also

Indirect Memory command (View menu)
Modify Value Dialog
Radix Popup menu

8.27 DRAM window

The DRAM window reflects the values in the simulated processors direct memory. Although 256 locations are displayed in this window not all of them may be valid for a specific processor.

The first 128 DRAM and IRAM locations occupy the same physical memory. The upper 128 locations of both DRAM and IRAM occupy unique physical memory locations. In addition depending on the processor type the upper 128 locations of IRAM may or may not be present. Specifying the processor type in the Options menu will result in the number of IRAM locations being changed to reflect the selected processor.

Doubling clicking on a IRAM entry will display a dialog box allowing you to specify a new value for that IRAM location.

See Also

Direct Memory command (View menu)
Modify Value Dialog
Radix Popup menu

8.28 Watch window

The Watch window reflects the values of user specified simulator attributes. The user can specify a custom combination of entries in the other simulator display windows. These entries can be registers, Internal memory, Direct memory, and external memory.

To display a Watch window select Watch Window from the View menu. A window will be display which contains the currently defined watch variables.

To add an entry into the Watch window either place the cursor over the text of the desired entry in the editor and click on the Add Watch toolbar icon  . The text under the cursor will be presented as the name of the variable to be added to the watch list. You can then accept this name or edit and change it.

Clicking the right mouse button with the mouse pointer over a Watch window will cause a popup menu specific to the Watch window to be displayed. The following table lists commands available in this popup menu.

Add	Display an Add Watch entry dialog box. This dialog allows you to define the entry to be added to the Watch window list.

Delete	Clicking the (left or right) mouse button highlights the watch entry under the mouse pointer. The left button will result in the entry being selected. The right button will select the entry and display the popup menu. Selecting the Delete Watch option will remove the highlight entry from the list.
Modify	Displays the Modify Watch dialog box for the highlighted entry. The watch can then be redefined.
Default	Changes the highlighted entries display radix to the default radix.
Decimal	Changes the highlighted entries display radix to decimal over ridding the default display radix.
Hexadecimal	Changes the highlighted entries display radix to hexadecimal over ridding the default display radix.
ASCII	Changes the highlighted entries display radix to ASCII over ridding the default display radix.
Binary	Changes the highlighted entries display radix to binary over ridding the default display radix.

See Also

Add/Modify Watch dialog
Watch Window command (View menu)

8.29 Add Watch command

Use this command to display the Add/Modify Watch dialog box. When adding an item to the Watch window via this command (and toolbar) the text under the cursor will be presented as the name of the variable to be added to the watch list.

Shortcuts

Toolbar: 

See Also

Add/Modify Watch dialog
Watch Window command (View menu)
Watch window

8.30 Modify Value Dialog

This dialog box allows you to specify a new value for the associated window entry. The following options are available in this dialog:

New Value

Type the new value for the associated window entry.

OK

Clicking on OK will result in the associated window entry being changed to the specified value.

Cancel

Clicking on Cancel will result in the dialog being closed and the window entry being left as it was.

8.31 Radix Popup menu

Clicking the right mouse button with the mouse pointer over an entry in either the Register, Internal Memory, Direct Memory or Port windows will cause a popup menu to be displayed. This popup menu allows you to select the display radix for the selected window entry. The following table lists commands available in this popup menu.

Default	Changes the highlighted entries display radix to the default radix.
Decimal	Changes the highlighted entries display radix to decimal over ridding the default display radix.
Hexadecimal	Changes the highlighted entries display radix to hexadecimal over ridding the default display radix.
ASCII	Changes the highlighted entries display radix to ASCII over ridding the default display radix.
Binary	Changes the highlighted entries display radix to binary over ridding the default display radix.

8.32 Add/Modify Watch dialog

This dialog box allows you to define a watch variable. The following options are available in this dialog:

Address

Type the address of the CPU resource to be viewed. This address can be a predefined or user defined label (register name for example). An expression can also be specified. The value at the resolved address will be displayed.

Format

Specifies the variable type at the indicated address. Select the format from the drop down list.

Bit Index

When the Bit format is specified this field allows you to select which bit in the byte at the indicated address to display. Select the bit index from the drop down list.

Byte Order

When the word or double word format is specified this field allows you to select the ordering of the bytes, starting at the specified address. Enter the values separated by commas. For example if the bytes of a word value are store with the most significant byte first specify a byte order of 0,1

Display Radix

This option allows you to over ride the default display radix for this watch only. Select the desired radix from the drop down list.

Null terminated string

When the string format is specified this field allows you to indicate that the string is NULL terminated (ends with a zero byte). The first character of the string is assumed to be at the specified address with the next characters in the string located in sequentially increasing memory locations. Place a check in this check box if the string is NULL terminated.

String length

When the string format is specified and Null terminated string is not checked this field allows you to specify the length of the string. The first character of the string is assumed to be at the specified address with the next characters in the string located in sequentially increasing memory locations.

Resource Type

This group box allows you to specify which memory area to associate with the specified address. If the Register resource is selected the value is displayed in byte format regardless of the selected format type.

OK

Clicking on OK will result in the associated window entry being changed as specified.

Cancel

Clicking on Cancel will result in the dialog being closed and the window entry being left as it was.

9 Monitor Commands

9.1 Download code (Monitor menu)

Use this command to open a serial port and download the currently assembled program to the development board. Once the code has been downloaded the serial port is left open and is monitored for communication with the development board. While the serial port is open issuing the Download command will use the currently open serial port and not ask you to specify one.

Shortcuts

Keys: F8

9.2 Start monitor (Monitor menu)

Use this command to open a serial port and monitor it for status information sent from the development board.

9.3 Reset Monitor (Monitor menu)

Use this command to pulse the reset line on the development board.

9.4 Stop (Monitor menu)

Use this command to close the currently open serial port and stop monitoring it for status from the development board.

10 Program Options

10.1 Defaults command (Options menu)

Use this command to set various program defaults.

See Also

Default Settings dialog

10.2 Simulator command (Options menu)

Use this command to open the Simulator Options dialog box.

See Also

Simulator Options dialog

10.3 Simulator Options dialog

The Simulator Options dialog allows you to enable paged memory support. When enabled you specify the page select control address. This address (in direct memory) selects the active 64k page. This is only used in the simulator when accessing external data memory. Program memory is not paged. The additional option allows you to select the number of memory pages. The number of pages tells the simulator the number of bits at the specified address that are relevant. All other bits at that address will be masked to zero before determining the memory page.

The options in this dialog are:

Enable Paged Memory: this check box item allows you to enable and disable paged memory support. When checked paged memory is supported. When not check paged memory is disabled.

Paged Memory Control Address: this option allows you to specify the direct memory location containing the page indicator. The simulator references this memory location each time you simulate a movx instruction. The actual address calculation is done as follows

Address = 0x10000 * Page + dptr

Total amount of paged memory: this option defines how many 64k block to allocate for the simulator. It also indicates the number of significant bits in the **Paged Memory Control Address** byte to use when determining the final address. For example if the amount of paged memory is set to two, then only the least significant bit of the paged memory control value will be used in the final address calculation. All the remaining bits are don't cares. They can be one or zero, they will not affect the calculation.

As an example let us say we have a device that has been developed with 512k of memory. The lower 16 address lines of the memory are controlled normally through the address/data bus. The upper address lines are connected to the lower bits of P1. We then configure the simulator as follows

Enabled Paged Memory: (checked)
Paged Memory Control Address: 144
Total amount of paged memory: 8 pages (512k)

Note: 144 is the direct memory address for P1.

In your code you would use the instructions

```
mov     p1, 01h
movx    a,@dptr
```

to access values in the second page of the 512k memory device.

Use P2 for @Ri addressing: this option enables and disables the use of P2 as the upper address byte for accessing simulated external memory when using the movx @ri,a and movx a,@ri (were ri is r0 or r1). When disabled (not checked) the upper address is assumed to be zero. When checked the upper address is composed of the value in P2.

Enable Automatic Window Updating: this option enables and disables the automatic updating of the source edit and simulation windows while the simulator is executing code. This only applies when running a code using the step over, run to or continue commands. When enabled the simulation windows will be updated after the specified number of instructions have been executed.

Instruction count between updates: indicates the number of instructions to execute between automatic updating (when enabled) of the simulation and source edit windows.

Processor variant: specifies the variant of the processor that is to be simulated. Currently the simulator provides the following options

Standard: indicates that the simulated processor does not contain any extra features.
ADuC84x: simulates the the following features of the Analog Devices ADuC84x series of processors:
Dual data pointer
EEPROM

11 Window Menu

11.1 Window menu commands

The Window menu offers the following commands, which enable you to arrange multiple views of multiple documents in the application window:

New Window	Creates a new window that views the same document.
Cascade	Arranges windows in an overlapped fashion.
Tile	Arranges windows in non-overlapped tiles.
Arrange Icons	Arranges icons of closed windows.
Window 1, 2, ...	Goes to specified window.

11.2 New command (Window menu)

Use this command to open a new source window with the same contents as the active window. You can open multiple source windows to display different parts or views of a document at the same time. If you change the contents in one window, all other windows containing the same document reflect those changes. When you open a new window, it becomes the active window and is displayed on top of all other open windows.

11.3 Cascade command (Window menu)

Use this command to arrange multiple opened windows in an overlapped fashion.

11.4 Tile command (Window menu)

Use this command to arrange multiple opened windows in a non-overlapped fashion.

11.5 Window Arrange Icons Command

Use this command to arrange the icons for minimized windows at the bottom of the main window. If there is an open document window at the bottom of the main window, then some or all of the icons may not be visible because they will be underneath this document window.

11.6 Arrange Windows command (Window menu)

Use this command to automatically arrange open windows. The source windows will be place onto of each other in the upper left section of the application windows. The output window will be positioned below the source windows. The remaining windows will be arranged along the right side of the application.

11.7 1, 2, ... command (Window menu)

The 8051 IDE displays a list of currently open document windows at the bottom of the Window menu. A check mark appears in front of the document name of the active window. Choose a document from this list to make its window active.

12 Help Menu

12.1 Help menu commands

The Help menu offers the following commands, which provide you assistance with this application:

Help Topics	Offers you an index to topics on which you can get help.
About	Displays the version number of this application.

12.2 Help Topics command (Help menu)

Use this command to display the opening screen of Help. From the opening screen, you can jump to step-by-step instructions for using the 8051 IDE and various types of reference information.

Once you open Help, you can click the Contents button whenever you want to return to the opening screen.

12.3 About command (Help menu)

Use this command to display the copyright notice and version number of your copy of the 8051 IDE.

13 Print Operations

13.1 File Print command

Use this command to print a document. This command presents a Print dialog box, where you may specify the range of pages to be printed, the number of copies, the destination printer, and other printer setup options.

Shortcuts

Toolbar: 
Keys: Ctrl+P

13.2 File Print Preview command

Use this command to display the active document as it would appear when printed. When you choose this command, the main window will be replaced with a print preview window in which one or two pages will be displayed in their printed format. The print preview toolbar offers you options to view either one or two pages at a time; move back and forth through the document; zoom in and out of pages; and initiate a print job.

13.3 print preview toolbar

The print preview toolbar offers you the following options:

Print

Bring up the print dialog box, to start a print job.

Next Page
Preview the next printed page.

Prev Page
Preview the previous printed page.

One Page / Two Page
Preview one or two printed pages at a time.

Zoom In
Take a closer look at the printed page.

Zoom Out
Take a larger look at the printed page.

Close
Return from print preview to the editing window.

13.4 Print Dialog

The following options allow you to specify how the document should be printed:

Printer
This is the active printer and printer connection. Choose the Setup option to change the printer and printer connection.

Properties
Displays a Print Setup dialog box, so you can select a printer and printer connection.

Print Range
Specify the pages you want to print:

All Prints the entire document.

Selection Prints the currently selected text.

Pages Prints the range of pages you specify in the From and To boxes.

Copies
Specify the number of copies you want to print for the above page range.

Collate Copies
Prints copies in page number order, instead of separated multiple copies of each page.

13.5 Print Progress Dialog

The Printing dialog box is shown during the time that the 8051 IDE is sending output to the printer. The page number indicates the progress of the printing.

To abort printing, choose Cancel.

13.6 File Print Setup command

Use this command to select a printer and a printer connection. This command presents a Print Setup dialog box, where you specify the printer and its connection.

13.7 Print Setup Dialog

The following options allow you to select the destination printer and its connection.

Printer
Select the printer you want to use. Choose the Default Printer; or choose the Specific Printer option and select one of the current installed printers shown in the box. You install printers and configure ports using the Windows Control Panel.

Orientation
Choose Portrait or Landscape.

Paper Size
Select the size of paper that the document is to be printed on.

Paper Source
Some printers offer multiple trays for different paper sources. Specify the tray here.

Options
Displays a dialog box where you can make additional choices about printing, specific to the type of printer you have selected.

14 System, Control and Document Commands

14.1 Size command (System menu)

Use this command to display a four-headed arrow so you can size the active window with the arrow keys.

After the pointer changes to the four-headed arrow:
Press one of the DIRECTION keys (left, right, up, or down arrow key) to move the pointer to the border you want to move.
Press a DIRECTION key to move the border.
Press ENTER when the window is the size you want.

Note: This command is unavailable if you maximize the window.

Shortcut

Mouse: Drag the size bars at the corners or edges of the window.

14.2 Move command (Control menu)

Use this command to display a four-headed arrow so you can move the active window or dialog box with the arrow keys.

Note: This command is unavailable if you maximize the window.

Shortcut

Keys: Ctrl+F7

14.3 System Minimize Command

Use this command to reduce the 8051 IDE window to an icon.

Shortcut

Mouse: Click the minimize icon  on the title bar.
Keys: Alt+F9

14.4 Maximize command (System menu)

Use this command to enlarge the active window to fill the available space.

Shortcut

Mouse: Click the maximize icon  on the title bar; or double-click the title bar.
Keys: Ctrl+F10 enlarges a document window.

14.5 Next Window command (document Control menu)

Use this command to switch to the next open document window. The 8051 IDE determines which window is next according to the order in which you opened the windows.

Shortcut

Keys: Ctrl+F6

14.6 Previous Window command (document Control menu)

Use this command to switch to the previous open document window. The 8051 IDE determines

which window is previous according to the order in which you opened the windows.

Shortcut

Keys: Shift+Ctrl+F6

14.7 Close command (Control menus)

Use this command to close the active window or dialog box.

Double-clicking a Control-menu box is the same as choosing the Close command.

Note: If you have multiple windows open for a single document, the Close command on the document Control menu closes only one window at a time. You can close all windows at once with the Close command on the File menu.

Shortcuts

Keys: Ctrl+F4 closes a document window
Alt+F4 closes the 8051 IDE window or dialog box

14.8 Restore command (Control menu)

Use this command to return the active window to its size and position before you chose the Maximize or Minimize command.

15 Appendices

15.1 Appendix A - Flag Altering Instructions

Instruction	Flags affected
ADD	C, OV, AC
ADDC	C, OV, AC
SUBB	C, OV, AC
MUL	C=0, OV
DIV	C=0, OV
DA	C
RRC	C
RLC	C
SETB C	C
CLR C	C
CPL C	C
ANL C,bit	C
ANL C,/bit	C

ORL C,bit	C
ORL C,/bit	C
MOV C,bit	C
CJNE	C

15.2 Appendix B - Instruction Symbols

A	Accumulator
AC	Auxiliary Carry flag
C	Carry flag
addr16	16 Bit Program Memory Address
addr11	11 Bit Program Memory Address
Rn	Register R0 - R7
direct	8 bit internal data memory location address
@Ri	8 bit internal data memory location addressed indirectly through R0 or R1
#data	8 bit immediate data
#data16	16 bit immediate data
rel	Signed (two's compliment) 8 bit offset.
Bit	Direct addressed bit in internal data RAM or special function register.
#	Immediate Data Prefix
@	Indirect Address Prefix
$	Current Value of Program Counter
(X)	Contents of X
((X))	Contents of Location Addressed by X
<>	Not Equal To
←←	Is Replaced by
←← →→	Is Exchanged with
v	OR operation
^	AND operation
⊽	Exclusive-OR operation
!	Ones Compliment

15.3 Appendix C - Instruction Translations

The following table lists instructions supported by the assembler that are not part of the 8031 instruction set. This enhanced instruction set simply provides alternate names for the standard set of instructions.

Mnemonic	Translation
CALL	LCALL addr16
JMP	LJMP addr16 (if operand is not @A+DPTR)

15.4 Appendix D - Predefined Labels

The following is a list of predefined assembler labels. These labels are typically associated with

direct memory access. Although they can be used with any immediate data evaluation. Associated label values are given in hexadecimal notation.

Label	Value	Description
P0	80	Port 0
SP	81	Stack Pointer
DPL	82	Data Pointer Low byte
DPH	83	Data Pointer High byte
PCON	87	
TCON	88	
TMOD	89	
TL0	8A	Timer/Counter 0 Low byte
TL1	8B	Timer/Counter 1 Low byte
TH0	8C	Timer/Counter 0 High byte
TH1	8D	Timer/Counter 1 High byte
P1	90	Port 1
SCON	98	
SBUF	99	
P2	A0	Port 2
IE	A8	
P3	B0	Port 3
IP	B8	
T2CON	C8	
RCAP2L	CA	
RCAP2H	CB	
TL2	CC	Timer/Counter 2 Low byte
TH2	CD	Timer/Counter 2 High byte
PSW	D0	Program Status Word
A	E0	Accumulator
ACC	E0	Accumulator
B	F0	B register

The following is a list of predefined assembler labels for bit addressable memory locations. In the following table .x represents a value in the range of 0 to 7. For example P0.x is short hand to represent P0.0, P0.1, P0.2, P0.3, P0.4, P0.5, P0.6 and P0.7. With P0.0 equal to 80h, P0.1 equal to 81h, etc. Associated label values are given in hexadecimal notation.

Label	Value	Description
ACC.x	E0 - E7	Accumulator (bits 0 through 7)
B.x	F0 - F7	B register (bits 0 through 7)
P0.x	80 - 87	Port 0 (bits 0 through 7)
P1.x	90 - 97	Port 1 (bits 0 through 7)
P2.x	A0 - A7	Port 2 (bits 0 through 7)
P3.x	B0 - B7	Port 3 (bits 0 through 7)
PSW.x	D0 - D7	Program Status Word (bits 0 through 7)
SCON.x	98 - 9F	Serial Control register (bits 0 through 7)
IE.x	A8 - AF	
IP.x	B8 - BF	
TCON.x	88 - 8F	Timer Control register (bits 0 through 7)
T2CON.x	C8 - CF	Timer 2 Control register (bits 0 through 7)
IT0	88	
IE0	89	
IT1	8A	

IE1	8B	
TR0	8C	
TF0	8D	
TR1	8E	
TF1	8F	
RI	98	Receive Interrupt flag
TI	99	Transmit Interrupt flag
RB8	9A	
TB8	9B	
REN	9C	
SM2	9D	
SM1	9E	
SM0	9F	
EX0	A8	
ET0	A9	
EX1	AA	
ET1	AB	
ES	AC	
ET2	AD	
EA	AF	
PX0	B8	
PT0	B9	
PX1	BA	
PT1	BB	
PS	BC	
PT2	BD	
P	D0	Parity flag
OV	D2	Overflow flag
RS0	D3	Register Select (bit 0)
RS1	D4	Register Select (bit 1)
F0	D5	
AC	D6	Auxiliary Carry flag
CY	D7	Carry flag

15.5 Appendix E - 8051 Instruction Set

Arithmetic Operations

Mnemonic	Description	Size	Cycles
ADD A,Rn	Add register to Accumulator (ACC).	1	1
ADD A,direct	Add direct byte to ACC.	2	1
ADD A,@Ri	Add indirect RAM to ACC.	1	1
ADD A,#data	Add immediate data to ACC.	2	1
ADDC A,Rn	Add register to ACC with carry.	1	1
ADDC A,direct	Add direct byte to ACC with carry.	2	1
ADDC A,@Ri	Add indirect RAM to ACC with carry.	1	1
ADDC A,#data	Add immediate data to ACC with carry.	2	1
SUBB A,Rn	Subtract register from ACC with borrow.	1	1
SUBB A,direct	Subtract direct byte from ACC with borrow	2	1
SUBB A,@Ri	Subtract indirect RAM from ACC with borrow.	1	1
SUBB A,#data	Subtract immediate data from ACC with borrow.	2	1

INC A	Increment ACC.	1	1
INC Rn	Increment register.	1	1
INC direct	Increment direct byte.	2	1
INC @Ri	Increment indirect RAM.	1	1
DEC A	Decrement ACC.	1	1
DEC Rn	Decrement register.	1	1
DEC direct	Decrement direct byte.	2	1
DEC @Ri	Decrement indirect RAM.	1	1
INC DPTR	Increment data pointer.	1	2
MUL AB	Multiply A and B Result: A <- low byte, B <- high byte.	1	4
DIV AB	Divide A by B Result: A <- whole part, B <- remainder.	1	4
DA A	Decimal adjust ACC.	1	1

Logical Operations

Mnemonic	Description	Size	Cycles
ANL A,Rn	AND Register to ACC.	1	1
ANL A,direct	AND direct byte to ACC.	2	1
ANL A,@Ri	AND indirect RAM to ACC.	1	1
ANL A,#data	AND immediate data to ACC.	2	1
ANL direct,A	AND ACC to direct byte.	2	1
ANL direct,#data	AND immediate data to direct byte.	3	2
ORL A,Rn	OR Register to ACC.	1	1
ORL A,direct	OR direct byte to ACC.	2	1
ORL A,@Ri	OR indirect RAM to ACC.	1	1
ORL A,#data	OR immediate data to ACC.	2	1
ORL direct,A	OR ACC to direct byte.	2	1
ORL direct,#data	OR immediate data to direct byte.	3	2
XRL A,Rn	Exclusive OR Register to ACC.	1	1
XRL A,direct	Exclusive OR direct byte to ACC.	2	1
XRL A,@Ri	Exclusive OR indirect RAM to ACC.	1	1
XRL A,#data	Exclusive OR immediate data to ACC.	2	1
XRL direct,A	Exclusive OR ACC to direct byte.	2	1
XRL direct,#data	XOR immediate data to direct byte.	3	2
CLR A	Clear ACC (set all bits to zero).	1	1
CPL A	Compliment ACC.	1	1
RL A	Rotate ACC left.	1	1
RLC A	Rotate ACC left through carry.	1	1
RR A	Rotate ACC right.	1	1
RRC A	Rotate ACC right through carry.	1	1
SWAP A	Swap nibbles within ACC.	1	1

Data Transfer

Mnemonic	Description	Size	Cycles
MOV A,Rn	Move register to ACC.	1	1
MOV A,direct	Move direct byte to ACC.	2	1
MOV A,@Ri	Move indirect RAM to ACC.	1	1
MOV A,#data	Move immediate data to ACC.	2	1
MOV Rn,A	Move ACC to register.	1	1
MOV Rn,direct	Move direct byte to register.	2	2
MOV Rn,#data	Move immediate data to register.	2	1
MOV direct,A	Move ACC to direct byte.	2	1

MOV direct,Rn	Move register to direct byte.	2	2
MOV direct,direct	Move direct byte to direct byte.	3	2
MOV direct,@Ri	Move indirect RAM to direct byte.	2	2
MOV direct,#data	Move immediate data to direct byte.	3	2
MOV @Ri,A	Move ACC to indirect RAM.	1	1
MOV @Ri,direct	Move direct byte to indirect RAM.	2	2
MOV @Ri,#data	Move immediate data to indirect RAM.	2	1
MOV DPTR,#data16	Move immediate 16 bit data to data pointer register.	3	2
MOVC A,@A+DPTR	Move code byte relative to DPTR to ACC (16 bit address).	1	2
MOVC A,@A+PC	Move code byte relative to PC to ACC (16 bit address).	1	2
MOVX A,@Ri	Move external RAM to ACC (8 bit address).	1	2
MOVX A,@DPTR	Move external RAM to ACC (16 bit address).	1	2
MOVX @Ri,A	Move ACC to external RAM (8 bit address).	1	2
MOVX @DPTR,A	Move ACC to external RAM (16 bit address).	1	2
PUSH direct	Push direct byte onto stack.	2	2
POP direct	Pop direct byte from stack.	2	2
XCH A,Rn	Exchange register with ACC.	1	1
XCH A,direct	Exchange direct byte with ACC.	2	1
XCH A,@Ri	Exchange indirect RAM with ACC.	1	1
XCHD A,@Ri	Exchange low order nibble of indirect RAM with low order nibble of ACC.	1	1

Boolean Variable Manipulation

Mnemonic	Description	Size	Cycles
CLR C	Clear carry flag.	1	1
CLR bit	Clear direct bit.	2	1
SETB C	Set carry flag.	1	1
SETB bit	Set direct bit.	2	1
CPL C	Compliment carry flag.	1	1
CPL bit	Compliment direct bit.	2	1
ANL C,bit	AND direct bit to carry flag.	2	2
ANL C,/bit	AND compliment of direct bit to carry.	2	2
ORL C,bit	OR direct bit to carry flag.	2	2
ORL C,/bit	OR compliment of direct bit to carry.	2	2
MOV C,bit	Move direct bit to carry flag.	2	1
MOV bit,C	Move carry to direct bit.	2	2
JC rel	Jump if carry is set.	2	2
JNC rel	Jump if carry is not set.	2	2
JB bit,rel	Jump if direct bit is set.	3	2
JNB bit,rel	Jump if direct bit is not set.	3	2
JBC bit,rel	Jump if direct bit is set & clear bit.	3	2

Program Branching

Mnemonic	Description	Size	Cycles
ACALL addr11	Absolute subroutine call.	2	2
LCALL addr16	Long subroutine call.	3	2
RET	Return from subroutine.	1	2
RETI	Return from interrupt.	1	2
AJMP addr11	Absolute jump.	2	2
LJMP addr16	Long jump.	3	2
SJMP rel	Short jump (relative address).	2	2

JMP @A+DPTR	Jump indirect relative to the DPTR.	1	2
JZ rel	Jump relative if ACC is zero.	2	2
JNZ rel	Jump relative if ACC is not zero.	2	2
CJNE A,direct,rel	Compare direct byte to ACC and jump if not equal.	3	2
CJNE A,#data,rel	Compare immediate byte to ACC and jump if not equal.	3	2
CJNE Rn,#data,rel	Compare immediate byte to register and jump if not equal.	3	2
CJNE @Ri,#data,rel	Compare immediate byte to indirect and jump if not equal.	3	2
DJNZ Rn,rel	Decrement register and jump if not zero.	2	2
DJNZ direct,rel	Decrement direct byte and jump if not zero.	3	2

Other Instructions

Mnemonic	Description	Size	Cycles
NOP	No operation.	1	1

15.6 Appendix F - 8051 Instructions

15.6.1 ACALL addr

ACALL addr

Function: Absolute Call

Description: ACALL unconditionally calls a subroutine located at the location at the indicated address. The instruction increments the PC twice to obtain the address of the following instruction, then pushes the 16-bit result onto the stack (low-order byte first) and increments the Stack Pointer twice. The destination address is obtained by successively concatenating the five high-order bits of the incremented PC, opcode bits 7-5, and the second byte of the instruction. The subroutine called must therefore start within the same 2k block of the program memory as the first byte of the instruction following ACALL. No flags are affected.

Example: Initially SP equals 07H. The label "SUBRTN" is at program memory location 0345H. After executing the instruction,

ACALLL SUBRTN

at location 0123H, SP will contain 09H, internal RAM locations 08H and 09H will contain 25H and 01H, respectively, and the PC will contain 0345H.

Bytes: 2

Cycles: 2

Encoding a10 a9 a8 1 0 0 0 1 a7 a6 a5 a4 a3 a2 a1 a0

Operation: ACALL

(PC)	←←	(PC) + 2
(SP)	←←	(SP) + 1
((SP))	←←	(PC_{7-0})
(SP)	←←	(SP) + 1
((SP))	←←	(PC_{15-8})

(PC_{10-0}) ←← page address

* Obtained from the 1990 edition of the Intel 8-Bit Embedded Controllers Data Book.

15.6.2 ADD A,Rn

ADD A,Rn

Function: Add register to accumulator

Description: ADD adds the byte contained in the indicated register to the Accumulator, leaving the result in the Accumulator. The carry and auxiliary-carry flags are set, respectively, if there is a carry-out from bit 7 or bit 3, and cleared otherwise. When adding unsigned integers, the carry flag indicates an overflow occurred.

OV is set if there is a carry-out of bit 6 but not out of bit 7, or a carry-out from bit 7 but not bit 6; otherwise OV is cleared. When adding signed integers, OV indicates a negative number produced as the sum of two positive operands, or a positive sum from two negative operands.

Example: The Accumulator holds 0C3H (11000011B) and register 0 holds 0AAH (10101010B). The instruction,

ADD A,R0

will leave 6DH (01101101B) in the Accumulator with the AC flag cleared and both the carry flag and OV set to 1.

Bytes: 1

Cycles: 1

Encoding: 0 0 1 0 1 r r r

Operation: ADD
(A) ←← (A) + (Rn)

* Obtained from the 1990 edition of the Intel 8-Bit Embedded Controllers Data Book.

15.6.3 ADD A,direct

ADD A,direct

Function: Add direct memory byte to accumulator

Description: ADD adds the byte contained in the indicated direct memory location to the Accumulator, leaving the result in the Accumulator. The carry and auxiliary-carry flags are set, respectively, if there is a carry-out from bit 7 or bit 3, and cleared otherwise. When adding unsigned integers, the carry flag indicates an overflow occurred.

OV is set if there is a carry-out of bit 6 but not out of bit 7, or a carry-out from bit 7 but not bit 6; otherwise OV is cleared. When adding signed integers, OV indicates a negative number produced as the sum of two positive operands, or a positive sum from

two negative operands.

Bytes: 2

Cycles: 1

Encoding: 0 0 1 0 0 1 0 1 direct address

Operation: ADD
(A) ←← (A) + (direct)

* Obtained from the 1990 edition of the Intel 8-Bit Embedded Controllers Data Book.

15.6.4 ADD A,@Ri

ADD A,@Ri

Function: Add indirect memory to accumulator

Description: ADD adds the byte contained in the indirect memory location indicated by the specified register to the Accumulator, leaving the result in the Accumulator. The carry and auxiliary-carry flags are set, respectively, if there is a carry-out from bit 7 or bit 3, and cleared otherwise. When adding unsigned integers, the carry flag indicates an overflow occurred.

OV is set if there is a carry-out of bit 6 but not out of bit 7, or a carry-out from bit 7 but not bit 6; otherwise OV is cleared. When adding signed integers, OV indicates a negative number produced as the sum of two positive operands, or a positive sum from two negative operands.

Bytes: 1

Cycles: 1

Encoding: 0 0 1 0 0 1 1 i

Operation: ADD
(A) ←← (A) + ((Ri))

* Obtained from the 1990 edition of the Intel 8-Bit Embedded Controllers Data Book.

15.6.5 ADD A,#data

ADD A,#data

Function: Add immediate data to accumulator

Description: ADD adds the second byte of the instruction to the Accumulator, leaving the result in the Accumulator. The carry and auxiliary-carry flags are set, respectively, if there is a carry-out from bit 7 or bit 3, and cleared otherwise. When adding unsigned integers, the carry flag indicates an overflow occurred.

OV is set if there is a carry-out of bit 6 but not out of bit 7, or a carry-out from bit 7 but not bit 6; otherwise OV is cleared. When adding signed integers, OV indicates a negative number produced as the sum of two positive operands, or a positive sum from two negative operands.

Bytes: 2

Cycles: 1

Encoding: 0 0 1 0 0 1 0 0 immediate data

Operation: ADD
(A) ←← (A) + #data

* Obtained from the 1990 edition of the Intel 8-Bit Embedded Controllers Data Book.

15.6.6 ADDC A,Rn

ADDC A,Rn

Function: Add register to accumulator with carry

Description: ADDC simultaneously adds the byte contained in the specified register, the carry flag and Accumulator contents, leaving the result in the Accumulator. The carry and auxiliary-carry flags are set, respectively, if there is a carry-out from bit 7 or bit 3, and cleared otherwise. When adding unsigned integers, the carry flag indicates an overflow occurred.

OV is set if there is a carry-out of bit 6 but not out of bit 7, or a carry-out from bit 7 but not bit 6; otherwise OV is cleared. When adding signed integers, OV indicates a negative number produced as the sum of two positive operands, or a positive sum from two negative operands.

Bytes: 1

Cycles: 1

Encoding: 0 0 1 1 1 r r r

Operation: ADDC
(A) ←← (A) + (C) + (Rn)

* Obtained from the 1990 edition of the Intel 8-Bit Embedded Controllers Data Book.

15.6.7 ADDC A,direct

ADDC A,direct

Function: Add direct memory to accumulator with carry

Description: ADDC simultaneously adds the byte contained in the specified direct memory location, the carry flag and Accumulator contents, leaving the result in the

Accumulator. The carry and auxiliary-carry flags are set, respectively, if there is a carry-out from bit 7 or bit 3, and cleared otherwise. When adding unsigned integers, the carry flag indicates an overflow occurred.

OV is set if there is a carry-out of bit 6 but not out of bit 7, or a carry-out from bit 7 but not bit 6; otherwise OV is cleared. When adding signed integers, OV indicates a negative number produced as the sum of two positive operands, or a positive sum from two negative operands.

Bytes: 2

Cycles: 1

Encoding: 0 0 1 1 0 1 0 1 direct address

Operation: ADDC
(A) ←← (A) + (C) + (direct)

* Obtained from the 1990 edition of the Intel 8-Bit Embedded Controllers Data Book.

15.6.8 ADDC A,@Ri

ADDC A,@Ri

Function: Add indirect memory to accumulator with carry

Description: ADDC simultaneously adds the byte contained in the indirect memory location contained in the indicated register, the carry flag and Accumulator contents, leaving the result in the Accumulator. The carry and auxiliary-carry flags are set, respectively, if there is a carry-out from bit 7 or bit 3, and cleared otherwise. When adding unsigned integers, the carry flag indicates an overflow occurred.

OV is set if there is a carry-out of bit 6 but not out of bit 7, or a carry-out from bit 7 but not bit 6; otherwise OV is cleared. When adding signed integers, OV indicates a negative number produced as the sum of two positive operands, or a positive sum from two negative operands.

Bytes: 1

Cycles: 1

Encoding: 0 0 1 1 0 1 1 i

Operation: ADDC
(A) ←← (A) + (C) + ((Ri))

* Obtained from the 1990 edition of the Intel 8-Bit Embedded Controllers Data Book.

15.6.9 ADDC A,#data

ADDC A,#data

Function: Add immediate data to accumulator with carry

Description: ADDC simultaneously adds the second byte of the instruction, the carry flag and Accumulator contents, leaving the result in the Accumulator. The carry and auxiliary-carry flags are set, respectively, if there is a carry-out from bit 7 or bit 3, and cleared otherwise. When adding unsigned integers, the carry flag indicates an overflow occurred.

OV is set if there is a carry-out of bit 6 but not out of bit 7, or a carry-out from bit 7 but not bit 6; otherwise OV is cleared. When adding signed integers, OV indicates a negative number produced as the sum of two positive operands, or a positive sum from two negative operands.

Bytes: 2

Cycles: 1

Encoding: 0 0 1 1 0 1 0 0 immediate data

Operation: ADDC
(A) ←← (A) + (C) + #data

* Obtained from the 1990 edition of the Intel 8-Bit Embedded Controllers Data Book.

15.6.10 AJMP addr11

AJMP addr11

Function: Absolute Jump

Description: AJMP transfers program execution to the indicated address, which is formed at run-time by concatenating the high-order five bits of the PC (after incrementing the PC twice), opcode bits 7-5, and the second byte of the instruction. The destination must therefore be within the same 2K block of program memory as the first byte of the instruction following AJMP.

Example: The label "JMPADR" is at program memory location 0123H. The instruction,

AJMP JMPADR

is at location 0345H and will load the PC with 0123H.

Bytes: 2

Cycles: 2

Encoding: a10 a9 a8 0 0 0 0 1 a7 a6 a5 a4 a3 a2 a1 a0

Operation: AJMP
(PC) ←← (PC) + 2
(PC_{10-0}) ←← page address

* Obtained from the 1990 edition of the Intel 8-Bit Embedded Controllers Data Book.

15.6.11 ANL A,Rn

ANL A,Rn

Function: Logical-AND accumulator with register

Description: ANL performs the bitwise logical-AND operation between the values contained in the accumulator and the specified register and stores the result in the accumulator. No flags are affected.

Bytes: 1

Cycles: 1

Encoding: 0 1 0 1 1 r r r

Operation: ANL
(A) ←← (A) ^ (Rn)

* Obtained from the 1990 edition of the Intel 8-Bit Embedded Controllers Data Book.

15.6.12 ANL A,direct

ANL A,direct

Function: Logical-AND accumulator with direct memory

Description: ANL performs the bitwise logical-AND operation between the values contained in the accumulator and the specified direct memory location and stores the result in the accumulator. No flags are affected.

Bytes: 2

Cycles: 1

Encoding: 0 1 0 1 0 1 0 1 direct address

Operation: ANL
(A) ←← (A) ^ (direct)

* Obtained from the 1990 edition of the Intel 8-Bit Embedded Controllers Data Book.

15.6.13 ANL A,@Ri

ANL A,@Ri

Function: Logical-AND accumulator with indirect memory

Description: ANL performs the bitwise logical-AND operation between the values contained in the

accumulator and the specified indirect memory location and stores the result in the accumulator. No flags are affected.

Bytes: 1

Cycles: 1

Encoding: 0 1 0 1 0 1 1 i

Operation: ANL
(A) ←← (A) ^ ((Ri))

* Obtained from the 1990 edition of the Intel 8-Bit Embedded Controllers Data Book.

15.6.14 ANL A,#data

ANL A,#data

Function: Logical-AND accumulator with immediate data

Description: ANL performs the bitwise logical-AND operation between the values contained in the accumulator and the second byte of the instruction and stores the result in the accumulator. No flags are affected.

Bytes: 2

Cycles: 1

Encoding: 0 1 0 1 0 1 0 0 immediate data

Operation: ANL
(A) ←← (A) ^ #data

* Obtained from the 1990 edition of the Intel 8-Bit Embedded Controllers Data Book.

15.6.15 ANL direct,A

ANL direct,A

Function: Logical-AND direct memory with accumulator

Description: ANL performs the bitwise logical-AND operation between the values contained in the specified direct memory location and accumulator and stores the result in the direct memory location. No flags are affected.

Note: When this instruction is used to modify an output port, the value used as the original port data will be read from the output data latch, not the input pins.

Bytes: 2

Cycles: 1

Encoding: 0 1 0 1 0 0 1 0 direct address

Operation: ANL
(direct) ←← (direct) ^ (A)

* Obtained from the 1990 edition of the Intel 8-Bit Embedded Controllers Data Book.

15.6.16 ANL direct,#data

ANL direct,#data

Function: Logical-AND direct memory with immediate data

Description: ANL performs the bitwise logical-AND operation between the values contained in the specified direct memory location and the second by of the instruction and stores the result in the direct memory location. No flags are affected.

Note: When this instruction is used to modify an output port, the value used as the original port data will be read from the output data latch, not the input pins.

Bytes: 3

Cycles: 2

Encoding: 0 1 0 1 0 0 1 1 direct address immediate data

Operation: ANL
(A) ←← (A) ^ (Rn)

* Obtained from the 1990 edition of the Intel 8-Bit Embedded Controllers Data Book.

15.6.17 ANL C,bit

ANL C,bit

Function: Logical-AND Carry flag with bit value

Description: If the Boolean value of the source bit is a logical 0 then clear the carry flag, otherwise leave the carry flag in its current state.

Only direct addressing is allowed for the source operand.

Bytes: 2

Cycles: 2

Encoding: 1 0 0 0 0 0 1 0 bit address

Operation: ANL
(C) ←← (C) ^ (bit)

* Obtained from the 1990 edition of the Intel 8-Bit Embedded Controllers Data Book.

15.6.18 ANL C,/bit

ANL C,/bit

Function: Logical-AND Carry flag with complement of bit value

Description: If the Boolean value of the source bit is a logical 1 then clear the carry flag, otherwise leave the carry flag in its current state.

Only direct addressing is allowed for the source operand.

Bytes: 2

Cycles: 2

Encoding: 1 0 1 1 0 0 0 0 bit address

Operation: ANL
(C) ←← (C) ^ !(bit)

* Obtained from the 1990 edition of the Intel 8-Bit Embedded Controllers Data Book.

15.6.19 CJNE A,direct,rel

CJNE A,direct,rel

Function: Compare accumulator to direct memory and Jump if Not Equal

Description: CJNE compares the magnitude of the first two operands, and branches if their values are not equal. The branch destination is compared by adding the signed relative-displacement in the last instruction byte to the PC, after incrementing the PC to the start of the next instruction. The carry flag is set if the unsigned integer value of <dest-byte> is less than the unsigned integer value of <src-byte>; otherwise, the carry is cleared. Neither operand is affected.

Bytes: 3

Cycles: 2

Encoding: 1 0 1 1 0 1 0 1 direct address rel address

Operation:
```
(PC) ←← (PC) + 3
IF (A) <> (direct) THEN
        (PC) ←← (PC) + relative offset
IF (A) < (direct) THEN
        (C) ←← 1
ELSE
        (C) ←← 0
```

* Obtained from the 1990 edition of the Intel 8-Bit Embedded Controllers Data Book.

15.6.20 CJNE A,#data,rel

CJNE A,#data,rel

Function: Compare accumulator to immediate data and Jump if Not Equal

Description: CJNE compares the magnitude of the first two operands, and branches if their values are not equal. The branch destination is compared by adding the signed relative-displacement in the last instruction byte to the PC, after incrementing the PC to the start of the next instruction. The carry flag is set if the unsigned integer value of <dest-byte> is less than the unsigned integer value of <src-byte>; otherwise, the carry is cleared. Neither operand is affected.

Bytes: 3

Cycles: 2

Encoding: 1 0 1 1 0 1 0 0 immediate data rel. address

Operation:
```
(PC) ←← (PC) + 3
IF (A) <> (direct) THEN
        (PC) ←← (PC) + relative offset
IF (A) < data THEN
        (C) ←← 1
ELSE
        (C) ←← 0
```

* Obtained from the 1990 edition of the Intel 8-Bit Embedded Controllers Data Book.

15.6.21 CJNE Rn,#data,rel

CJNE Rn,#data,rel

Function: Compare register value to immediate data and Jump if Not Equal

Description: CJNE compares the magnitude of the first two operands, and branches if their values are not equal. The branch destination is compared by adding the signed relative-displacement in the last instruction byte to the PC, after incrementing the PC to the start of the next instruction. The carry flag is set if the unsigned integer value of <dest-byte> is less than the unsigned integer value of <src-byte>; otherwise, the carry is cleared. Neither operand is affected.

Bytes: 3

Cycles: 2

Encoding: 1 0 1 1 1 r r r immediate data rel. address

Operation:
```
(PC) ←← (PC) + 3
IF (Rn) <> data THEN
        (PC) ←← (PC) + relative offset
IF (Rn) < data THEN
```

```
        (C) ←← 1
ELSE
        (C) ←← 0
```

* Obtained from the 1990 edition of the Intel 8-Bit Embedded Controllers Data Book.

15.6.22 CJNE @Ri,#data,rel

CJNE @Ri,#data,rel

Function: Compare indirect memory with immediate data and Jump if Not Equal

Description: CJNE compares the magnitude of the first two operands, and branches if their values are not equal. The branch destination is compared by adding the signed relative-displacement in the last instruction byte to the PC, after incrementing the PC to the start of the next instruction. The carry flag is set if the unsigned integer value of <dest-byte> is less than the unsigned integer value of <src-byte>; otherwise, the carry is cleared. Neither operand is affected.

Bytes: 3

Cycles: 2

Encoding: 1 0 1 1 0 1 1 i immediate data rel. address

Operation:

```
(PC) ←← (PC) + 3
IF ((Ri)) <> data THEN
        (PC) ←← (PC) + relative offset
IF ((Ri)) < data THEN
        (C) ←← 1
ELSE
        (C) ←← 0
```

* Obtained from the 1990 edition of the Intel 8-Bit Embedded Controllers Data Book.

15.6.23 CLR A

CLR A

Function: Clear Accumulator

Description: The Accumulator is cleared (all bits set to zero). No flags are affected.

Bytes: 1

Cycles: 1

Encoding: 1 1 1 0 0 1 0 0

Operation:

```
CLR
(A) ←← 0
```

* Obtained from the 1990 edition of the Intel 8-Bit Embedded Controllers Data Book.

15.6.24 CLR C

CLR C

Function: Clear Carry flag

Description: The Carry flag is cleared (reset to zero). No other flags are affected.

Bytes: 1

Cycles: 1

Encoding: 1 1 0 0 0 0 1 1

Operation: CLR
(C) ←← 0

* Obtained from the 1990 edition of the Intel 8-Bit Embedded Controllers Data Book.

15.6.25 CLR bit

CLR bit

Function: Clear bit location

Description: The specified bit addressable location is cleared (reset to zero). No other flags are affected.

Bytes: 2

Cycles: 1

Encoding: 1 1 0 0 0 0 1 0 bit address

Operation: CLR
(bit) ←← 0

* Obtained from the 1990 edition of the Intel 8-Bit Embedded Controllers Data Book.

15.6.26 CPL A

CPL A

Function: Complement Accumulator

Description: Each bit of the Accumulator is logically complemented (one's compliment). Bits, which previously contained a one, are changed to a zero, and bits, which previously contained a zero, are changed to a one. No flags are affected.

Bytes: 1

Cycles: 1

Encoding: 1 1 1 1 0 1 0 0

Operation: CPL
(A) ←← !(A)

* Obtained from the 1990 edition of the Intel 8-Bit Embedded Controllers Data Book.

15.6.27 CPL C

CPL C

Function: Complement Carry

Description: The Carry flag is complemented. A Carry which had been a one is changed to zero and vice-versa. No other flags are affected.

Bytes: 1

Cycles: 1

Encoding: 1 0 1 1 0 0 1 1

Operation: CPL
(C) ←← !(C)

* Obtained from the 1990 edition of the Intel 8-Bit Embedded Controllers Data Book.

15.6.28 CPL bit

CPL bit

Function: Complement bit location

Description: The bit addressable memory location is complemented. A bit value of one is changed to zero and vice-versa. No other flags are affected.

Note: When this instruction is used to modify an output pin, the value used as the original data will be read from the output data latch, not the input pin.

Bytes: 2

Cycles: 1

Encoding: 1 0 1 1 0 0 1 0 bit address

Operation: CPL
(bit) ←← !(bit)

* Obtained from the 1990 edition of the Intel 8-Bit Embedded Controllers Data Book.

15.6.29 DA A

DA A

Function: Decimal-adjust Accumulator for Addition

Description: DA A adjusts the eight-bit value in the Accumulator resulting from the earlier addition of two variables (each in packed-BCD format), producing two four-bit digits. Any ADD or ADDC instruction may have been used to perform the addition.

If Accumulator bits 3-0 are greater than nine (xxxx1010 - xxxx1111), or if the AC flag is one, six is added to the Accumulator producing the proper BCD digit in the low-order nibble. This internal addition would set the carry flag if a carry-out of the low-order four-bit field propagated through all high-order bits, but it would not clear the carry flag otherwise.

If the carry flag is now set, or if the four high-order bits now exceed nine (1010xxxx - 1111xxxx), these high-order bits are incremented by six, producing the proper BCD digit in the high-order nibble. Again, this would set the carry flag if there was a carry-out of the high-order bits, but wouldn't clear the carry. The carry flag thus indicates if the sum of the original two BCD variables is greater than 100, allowing multiple precision decimal addition. OV is not affected.

All of this occurs during the one instruction cycle. Essentially, this instruction performs the decimal conversion by adding 00H, 06H, 60H, or 66H to the Accumulator, depending on the initial Accumulator and PSW conditions.

Note: DAA A cannot simply convert a hexadecimal number in the Accumulator to BCD notation, nor does DA A apply to decimal subtraction.

Bytes: 1

Cycles: 1

Encoding: 1 1 0 1 0 1 0 0

Operation: DA
-contents of Accumulator are BCD
IF [[(A_{3-0}) > 9] \/ [(AC) = 1]] THEN
(A_{3-0}) ←← (A_{3-0}) + 6
AND
IF [[(A7-4) > 9] \/ [(C) = 1]] THEN
(A_{7-4}) ←← (A_{7-4}) + 6

* Obtained from the 1990 edition of the Intel 8-Bit Embedded Controllers Data Book.

15.6.30 DEC A

DEC A

Function: Decrement Accumulator

Description: The value in the Accumulator is decremented by 1. An original value of 00H will underflow to 0FFH. No flags are affected.

Bytes: 1

Cycles: 1

Encoding: 0 0 0 1 0 1 0 0

Operation: DEC
(A) ←← (A) - 1

* Obtained from the 1990 edition of the Intel 8-Bit Embedded Controllers Data Book.

15.6.31 DEC Rn

DEC Rn

Function: Decrement Register

Description: The value in the specified register is decremented by 1. An original value of 00H will underflow to 0FFH. No flags are affected.

Bytes: 1

Cycles: 1

Encoding: 0 0 0 1 1 r r r

Operation: DEC
(Rn) ←← (Rn) - 1

* Obtained from the 1990 edition of the Intel 8-Bit Embedded Controllers Data Book.

15.6.32 DEC direct

DEC direct

Function: Decrement Direct Memory

Description: The value in the specified direct memory location is decremented by 1. An original value of 00H will underflow to 0FFH. No flags are affected.

Note: When this instruction is used to modify an output port, the value used as the original port data will be read from the output data latch, not the input pins.

Bytes: 2

Cycles: 1

Encoding: 0 0 0 1 0 1 0 1 direct address

Operation: DEC
(direct) ←← (direct) - 1

* Obtained from the 1990 edition of the Intel 8-Bit Embedded Controllers Data Book.

15.6.33 DEC @Ri

DEC @Ri

Function: Decrement Indirect Memory

Description: The value in the indirect memory location (indicated by the contents of the specified register) is decremented by 1. An original value of 00H will underflow to 0FFH. No flags are affected.

Bytes: 1

Cycles: 1

Encoding: 0 0 0 1 0 1 1 i

Operation: DEC
((Ri)) ←← ((Ri)) - 1

* Obtained from the 1990 edition of the Intel 8-Bit Embedded Controllers Data Book.

15.6.34 DIV AB

DIV AB

Function: Divide

Description: DIV AB divides the unsigned eight-bit integer in the Accumulator by the unsigned eight-bit integer in register B. The Accumulator receives the integer part of the quotient; register B receives the integer remainder. The carry and OV flags are cleared.

Exception: if B had originally contained 00H, the values returned in the Accumulator and B-register will be undefined and the overflow flag will be set. The carry flag is cleared in any case.

Bytes: 1

Cycles: 4

Encoding: 1 0 0 0 0 1 0 0

Operation: DIV
$(A)_{15\text{-}8}$, $(B)_{7\text{-}0}$ ←← (A) / (B)

* Obtained from the 1990 edition of the Intel 8-Bit Embedded Controllers Data Book.

15.6.35 DJNZ Rn,rel

DJNZ Rn,rel

Function: Decrement Register and Jump if Not Zero

Description: DJNZ decrements the value contained in the specified register by 1, and branches to the address indicated by the second operand if the resulting value is not zero. An original value of 00H will underflow to 0FFH. No flags are affected. The branch destination would be computed by adding the signed relative-displacement value in the last instruction byte to the PC, after incrementing the PC to the first byte of the following instruction.

Bytes: 2

Cycles: 2

Encoding: 1 1 0 1 1 r r r rel. address

Operation: DJNZ
(PC) ←← (PC) + 2
(Rn) ←← (Rn) - 1
IF (Rn) > 0 or (Rn) < 0 THEN
(PC) ←← (PC) + rel

* Obtained from the 1990 edition of the Intel 8-Bit Embedded Controllers Data Book.

15.6.36 DJNZ direct,rel

DJNZ direct,rel

Function: Decrement Direct Memory and Jump if Not Zero

Description: DJNZ decrements the value contained in the specified direct memory location by 1, and branches to the address indicated by the second operand if the resulting value is not zero. An original value of 00H will underflow to 0FFH. No flags are affected. The branch destination would be computed by adding the signed relative-displacement value in the last instruction byte to the PC, after incrementing the PC to the first byte of the following instruction.

Note: When this instruction is used to modify an output port, the value used as the original port data will be read from the output data latch, not the input pins.

Bytes: 3

Cycles: 2

Encoding: 1 1 0 1 0 1 0 1 direct address rel. address

Operation: DJNZ

(PC) ←← (PC) + 3
(direct) ←← (direct) - 1
IF (direct) > 0 or (direct) < 0 THEN
(PC) ←← (PC) + rel

* Obtained from the 1990 edition of the Intel 8-Bit Embedded Controllers Data Book.

15.6.37 INC A

INC A

Function: Increment Accumulator

Description: INC increments the contents of the Accumulator by 1. An original value of 0FFH will overflow to 00H. No flags are affected.

Bytes: 1

Cycles: 1

Encoding: 0 0 0 0 0 1 0 0

Operation: INC
(A) ←← (A) + 1

* Obtained from the 1990 edition of the Intel 8-Bit Embedded Controllers Data Book.

15.6.38 INC Rn

INC Rn

Function: Increment Register

Description: INC increments the contents of the specified register by 1. An original value of 0FFH will overflow to 00H. No flags are affected.

Bytes: 1

Cycles: 1

Encoding: 0 0 0 0 1 r r r

Operation: INC
(Rn) ←← (Rn) + 1

* Obtained from the 1990 edition of the Intel 8-Bit Embedded Controllers Data Book.

15.6.39 INC direct

INC direct

Function: Increment Direct Memory

Description: INC increments the value contained in the direct memory location indicated by the specified address by 1. An original value of 0FFH will overflow to 00H. No flags are affected.

Note: When this instruction is used to modify an output port, the value used as the original port data will be read from the output data latch, not the input pins.

Bytes: 2

Cycles: 1

Encoding: 0 0 0 0 0 1 0 1 direct address

Operation: INC
(direct) ←← (direct) + 1

* Obtained from the 1990 edition of the Intel 8-Bit Embedded Controllers Data Book.

15.6.40 INC @Ri

INC @Ri

Function: Increment Indirect Memory

Description: INC increments the value contained in the indirect memory location indicated by the contents of the specified register. . An original value of 0FFH will overflow to 00H. No flags are affected.

Bytes: 1

Cycles: 1

Encoding: 0 0 0 0 0 1 1 i

Operation: INC
((Ri)) ←← ((Ri)) + 1

* Obtained from the 1990 edition of the Intel 8-Bit Embedded Controllers Data Book.

15.6.41 INC DPTR

INC DPTR

Function: Increment Data Pointer

Description: Increments the 16-bit data pointer by 1. A 16-bit increment (modulo 2^16) is performed; an overflow of the low-order byte of the data pointer (DPL) from 0FFH to 00H will increment the high-order byte (DPH). No flags are affected.

This is the only 16-bit register, which can be incremented.

Bytes: 1

Cycles: 2

Encoding: 1 0 1 0 0 0 1 1

Operation: INC
(DPTR) ←← (DPTR) + 1

* Obtained from the 1990 edition of the Intel 8-Bit Embedded Controllers Data Book.

15.6.42 JB bit,rel

JB bit,rel

Function: Jump if Bit set

Description: If the indicated bit is a one, jump to the address indicated; otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative-displacement in the third instruction byte to the PC, after incrementing the PC to the first byte of the next instruction. The bit tested is not modified. No flags are affected.

Bytes: 3

Cycles: 2

Encoding: 0 0 1 0 0 0 0 0 bit address rel. address

Operation: JB
(PC) ←← (PC) + 3
IF (bit) = 1 THEN
(PC) ←← (PC) + rel

* Obtained from the 1990 edition of the Intel 8-Bit Embedded Controllers Data Book.

15.6.43 JBC bit,rel

JBC bit,rel

Function: Jump if Bit is set and Clear bit

Description: If the indicated bit is one, branch to the address indicated; otherwise proceed with the next instruction. The bit will not be cleared if it is already zero. The branch destination is computed by adding the signed relative-displacement in the third instruction byte to the PC, after incrementing the PC to the first byte of the next instruction. No flags are affected

Bytes: 3

Cycles: 2

Encoding: 0 0 0 1 0 0 0 0 bit address rel. address

Operation: JBC
(PC) ←← (PC) + 3
IF (bit) = 1 THEN
(bit) ←← 0
(PC) ←← (PC) + rel

* Obtained from the 1990 edition of the Intel 8-Bit Embedded Controllers Data Book.

15.6.44 JC rel

JC rel

Function: Jump if Carry is set

Description: If the carry flag is set, branch to the address indicated; otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative-displacement in the third instruction byte to the PC, after incrementing the PC twice. No flags are affected.

Bytes: 2

Cycles: 2

Encoding: 0 1 0 0 0 0 0 0 rel. address

Operation: JC
(PC) ←← (PC) + 2
IF (C) = 1 THEN
(PC) ←← (PC) + rel

* Obtained from the 1990 edition of the Intel 8-Bit Embedded Controllers Data Book.

15.6.45 JMP @A+DPTR

JMP @A+DPTR

Function: Jump indirect

Description: Add the eight-bit unsigned contents of the Accumulator with the sixteen-bit data pointer, and load the resulting sum to the program counter. This will be the address for subsequent instruction fetches. Sixteen-bit addition is performed (modulo 2^16): a carry-out from the low-order eight bits propagates through the higher-order bits. Neither the Accumulator nor the Data Pointer is altered. No flags are affected.

Bytes: 1

Cycles: 2

Encoding: 0 1 1 1 0 0 1 1

Operation: JMP
(PC) ←← (A) + (DPTR)

* Obtained from the 1990 edition of the Intel 8-Bit Embedded Controllers Data Book.

15.6.46 JNB bit,rel

JNB bit,rel

Function: Jump if Bit Not set

Description: If the indicated bit is zero, branch to the indicated address; otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative-displacement in the third instruction byte to the PC, after incrementing the PC to the first byte of the next instruction. The bit tested is not modified. No flags are affected.

Bytes: 3

Cycles: 2

Encoding: 0 0 1 1 0 0 0 0 bit address rel. address

Operation: JNB
(PC) ←← (PC) + 3
IF (bit) = 0 THEN
(PC) ←← (PC) + rel

* Obtained from the 1990 edition of the Intel 8-Bit Embedded Controllers Data Book.

15.6.47 JNC rel

JNC rel

Function: Jump if Carry not set

Description: If the carry flag is a zero, branch to the indicated address; otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative-displacement in the third instruction byte to the PC, after incrementing the PC twice to point to the next instruction. The carry flag is not modified.

Bytes: 2

Cycles: 2

Encoding: 0 1 0 1 0 0 0 0 rel. address

Operation: JNC
(PC) ←← (PC) + 2
IF (C) = 0 THEN
(PC) ←← (PC) + rel

* Obtained from the 1990 edition of the Intel 8-Bit Embedded Controllers Data Book.

15.6.48 JNZ rel

JNZ rel

Function: Jump if Accumulator Not Zero

Description: If any bit of the Accumulator is a one, branch to the indicated address; otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative-displacement in the third instruction byte to the PC, after incrementing the PC twice. The Accumulator is not modified. The carry flag is not modified

Bytes: 2

Cycles: 2

Encoding: 0 1 1 1 0 0 0 0 rel. address

Operation: JNZ
(PC) ←← (PC) + 2
IF (A) <> 0 THEN
(PC) ←← (PC) + rel

* Obtained from the 1990 edition of the Intel 8-Bit Embedded Controllers Data Book.

15.6.49 JZ rel

JZ rel

Function: Jump if Accumulator Zero

Description: If all bits of the Accumulator are zero, branch to the indicated address; otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative-displacement in the third instruction byte to the PC, after incrementing the PC twice. The Accumulator is not modified. The carry flag is not modified

Bytes: 2

Cycles: 2

Encoding: 0 1 1 0 0 0 0 0 rel. address

Operation: JZ
(PC) ←← (PC) + 2
IF (A) = 0 THEN
(PC) ←← (PC) + rel

* Obtained from the 1990 edition of the Intel 8-Bit Embedded Controllers Data Book.

15.6.50 LCALL addr16

LCALL addr16

Function: Long call

Description: LCALL calls a subroutine located at the indicated address. The instruction adds three to the program counter to generate the address of the next instruction and then pushes the 16-bit result onto the stack (low byte first), incrementing the Stack Pointer by two. The high-order and low-order bytes of the PC are the loaded, respectively, with the second and third bytes of the LCALL instruction. Program execution continues with the instruction at this address. The subroutine may therefore begin anywhere in the full 64K program memory address space. No flags are affected.

Bytes: 3

Cycles: 2

Encoding: 0 0 0 1 0 0 1 0 addr15-addr8 addr7-addr0

Operation: LCALL
(PC) ←← (PC) + 3
(SP) ←← (SP) + 1
((SP)) ←← (PC_{7-0})
(SP) ←← (SP) + 1
((SP)) ←← (PC_{15-8})
(PC) ←← $addr_{15-0}$

* Obtained from the 1990 edition of the Intel 8-Bit Embedded Controllers Data Book.

15.6.51 LJMP addr16

LJMP addr16

Function: Long Jump

Description: LJMP causes an unconditional branch to the indicated address, by loading the high-order and low-order bytes of the PC (respectively) with the second and third instruction bytes. The destination may therefore be anywhere in the full 64K program memory address space. No flags are affected.

Bytes: 3

Cycles: 2

Encoding: 0 0 0 0 0 0 1 0 addr15-addr8 addr7-addr0

Operation: LJMP
(PC) ←← $addr_{15-0}$

* Obtained from the 1990 edition of the Intel 8-Bit Embedded Controllers Data Book.

15.6.52 MOV A,Rn

MOV A,Rn

Function: Move Register to Accumulator

Description: The byte value contained in the indicated register by the second operand is copied into the accumulator (the first operand). The source byte is not affected. No other register or flag is affected.

Bytes: 1

Cycles: 1

Encoding: 1 1 1 0 1 r r r

Operation: MOV
(A) ←← (Rn)

* Obtained from the 1990 edition of the Intel 8-Bit Embedded Controllers Data Book.

15.6.53 MOV A,direct

MOV A,direct

Function: Move Direct to Accumulator

Description: The byte value contained in the indicated direct memory location by the second operand is copied into the accumulator (the first operand). The source byte is not affected. No other register or flag is affected.

Bytes: 2

Cycles: 1

Encoding: 1 1 1 0 0 1 0 1 direct address

Operation: MOV
(A) ←← (direct)

* Obtained from the 1990 edition of the Intel 8-Bit Embedded Controllers Data Book.

15.6.54 MOV A,@Ri

MOV A,@Ri

Function: Move Indirect to Accumulator

Description: The byte value contained in the indicated indirect memory location, indicated by the value in the specified register by the second operand is copied into the accumulator (the first operand). The source byte is not affected. No other register or flag is affected.

Bytes: 1

Cycles: 1

Encoding: 1 1 1 0 0 1 1 i

Operation: MOV
(A) ←← ((Ri))

* Obtained from the 1990 edition of the Intel 8-Bit Embedded Controllers Data Book.

15.6.55 MOV A,#data

MOV A,#data

Function: Move Immediate to Accumulator

Description: The immediate (second instruction byte) value is copied into the accumulator (the first operand). The source byte is not affected. No other register or flag is affected.

Bytes: 2

Cycles: 1

Encoding: 0 1 1 1 0 1 0 0 immediate data

Operation: MOV
(A) ←← #data

* Obtained from the 1990 edition of the Intel 8-Bit Embedded Controllers Data Book.

15.6.56 MOV Rn,A

MOV Rn,A

Function: Move Accumulator to Register

Description: The byte value contained in the accumulator is copied into the register specified by the first operand. The source byte is not affected. No other register or flag is affected.

Bytes: 1

Cycles: 1

Encoding: 1 1 1 1 1 r r r

Operation: MOV
(Rn) ←← (A)

* Obtained from the 1990 edition of the Intel 8-Bit Embedded Controllers Data Book.

15.6.57 MOV Rn,direct

MOV Rn,direct

Function: Move Direct to Register

Description: The byte value contained in the indicated direct memory location by the second operand is copied into the specified register (the first operand). The source byte is not affected. No other register or flag is affected.

Bytes: 2

Cycles: 2

Encoding: 1 0 1 0 1 r r r direct address

Operation: MOV
(Rn) ←← (direct)

* Obtained from the 1990 edition of the Intel 8-Bit Embedded Controllers Data Book.

15.6.58 MOV Rn,#data

MOV Rn,#data

Function: Move Immediate to Register

Description: The immediate (second instruction byte) value is copied into the specified register (the first operand). The source byte is not affected. No other register or flag is affected.

Bytes: 2

Cycles: 1

Encoding: 0 1 1 1 1 r r r immediate data

Operation: MOV
(Rn) ←← #data

* Obtained from the 1990 edition of the Intel 8-Bit Embedded Controllers Data Book.

15.6.59 MOV direct,A

MOV direct,A

Function: Move Accumulator to Direct Memory

Description: The byte value contained in the accumulator is copied into the specified direct memory location. The source byte is not affected. No other register or flag is affected.

Bytes: 2

Cycles: 1

Encoding: 1 1 1 1 0 1 0 1 direct address

Operation: MOV
(direct) ←← (A)

* Obtained from the 1990 edition of the Intel 8-Bit Embedded Controllers Data Book.

15.6.60 MOV direct,Rn

MOV direct,Rn

Function: Move Register to Direct Memory

Description: The byte value contained in the indicated register is copied into the specified direct memory location. The source byte is not affected. No other register or flag is affected.

Bytes: 2

Cycles: 2

Encoding: 1 0 0 0 1 r r r direct address

Operation: MOV
(direct) ←← (Rn)

* Obtained from the 1990 edition of the Intel 8-Bit Embedded Controllers Data Book.

15.6.61 MOV direct,direct

MOV direct,direct

Function: Move Direct Memory to Direct Memory

Description: The byte contained in the indicated direct memory source (second operand) is copied into the specified direct memory location (first operand). The source byte is not affected. No other register or flag is affected.

Bytes: 3

Cycles: 2

Encoding: 1 0 0 0 0 1 0 1 dir. addr. (src) dir. addr. (dest)

Operation: MOV
(direct) ←← (direct)

* Obtained from the 1990 edition of the Intel 8-Bit Embedded Controllers Data Book.

15.6.62 MOV direct,@Ri

MOV direct,@Ri

Function: Move Indirect Memory to Direct Memory

Description: The byte contained at the indirect memory location indicated by the contents of the specified register (second operand) is copied into the direct memory (first operand). The source byte is not affected. No other register or flag is affected.

Bytes: 2

Cycles: 2

Encoding: 1 0 0 0 0 1 1 i direct address

Operation: MOV
(direct) ←← ((Ri))

* Obtained from the 1990 edition of the Intel 8-Bit Embedded Controllers Data Book.

15.6.63 MOV direct,#data

MOV direct,#data

Function: Move Immediate to Direct Memory

Description: The immediate (second instruction byte) value is copied into the specified direct memory location (the first operand). The source byte is not affected. No other register or flag is affected.

Bytes: 3

Cycles: 2

Encoding: 0 1 1 1 0 1 0 1 direct address immediate data

Operation: MOV
(direct) ←← #data

* Obtained from the 1990 edition of the Intel 8-Bit Embedded Controllers Data Book.

15.6.64 MOV @Ri,A

MOV @Ri,A

Function: Move Accumulator to Indirect Memory

Description: The byte value contained in the accumulator is copied into the indirect memory location indicated by the contents of the specified register (first operand). The source byte is not affected. No other register or flag is affected.

Bytes: 1

Cycles: 1

Encoding: 1 1 1 1 0 1 1 i

Operation: MOV
((Ri)) ←← (A)

* Obtained from the 1990 edition of the Intel 8-Bit Embedded Controllers Data Book.

15.6.65 MOV @Ri,direct

MOV @Ri,direct

Function: Move Direct Memory to Indirect Memory

Description: The byte value contained in the direct memory location (second operand) is copied into the indirect memory location indicated by the contents of the specified register (first operand). The source byte is not affected. No other register or flag is affected.

Bytes: 2

Cycles: 2

Encoding: 1 0 1 0 0 1 1 i direct address

Operation: MOV
((Ri)) ←← (direct)

* Obtained from the 1990 edition of the Intel 8-Bit Embedded Controllers Data Book.

15.6.66 MOV @Ri,#data

MOV @Ri,#data

Function: Move Immediate to Indirect Memory

Description: The immediate (second instruction byte) value is copied into the indirect memory location indicated by the contents of the specified register (first operand). The source byte is not affected. No other register or flag is affected.

Bytes: 2

Cycles: 1

Encoding: 0 1 1 1 0 1 1 i immediate data

Operation: MOV
((Ri)) ←← #data

* Obtained from the 1990 edition of the Intel 8-Bit Embedded Controllers Data Book.

15.6.67 MOV C,bit

MOV C,bit

Function: Move Bit to Carry

Description: The bit value contained in the specified bit addressable memory location (second operand) is copied into the carry flag (first operand). No other register or flag is affected.

Bytes: 2

Cycles: 1

Encoding: 1 0 1 0 0 0 1 0 bit address

Operation: MOV
(C) ←← (bit)

* Obtained from the 1990 edition of the Intel 8-Bit Embedded Controllers Data Book.

15.6.68 MOV bit,C

MOV bit,C

Function: Move Carry to Bit

Description: The bit value contained in the carry flag (second operand) is copied into the specified bit addressable memory location (second operand). No other register or flag is affected.

Bytes: 2

Cycles: 2

Encoding: 1 0 0 1 0 0 1 0 bit address

Operation: MOV
(bit) ←← (C)

* Obtained from the 1990 edition of the Intel 8-Bit Embedded Controllers Data Book.

15.6.69 MOV DPTR,#data16

MOV DPTR,#data16

Function: Load Data Pointer with a 16-bit constant

Description: The Data Pointer is loaded with the 16-bit constant indicated. The 16-bit constant is loaded into the second and third bytes of the instruction. The second byte (DPH) is

the high-order byte, while the third byte (DPL) holds the low-order byte. No flags are affected.

This is the only instruction, which moves 16 bits of data at once.

Bytes: 3

Cycles: 2

Encoding: 1 0 0 1 0 0 0 0 immed. data15-8 immed. data7-0

Operation: MOV
(DPTR) ←← #data15-0

* Obtained from the 1990 edition of the Intel 8-Bit Embedded Controllers Data Book.

15.6.70 MOVC A,@A+DPTR

MOVC A,@A+DPTR

Function: Move Code byte

Description: The MOVC instruction loads the accumulator with a code byte, or constant from program memory. The address of the byte fetched is the sum of the original unsigned eight-bit accumulator contents and the contents of the 16-bit Data Pointer. Sixteen-bit addition is performed so a carry-out from the low-order eight bits may propagate through higher-order bits. No flags are affected.

Bytes: 1

Cycles: 2

Encoding: 1 0 0 1 0 0 1 1

Operation: MOVC
(A) ←← ((A) + (DPTR))

* Obtained from the 1990 edition of the Intel 8-Bit Embedded Controllers Data Book.

15.6.71 MOVC A,@A+PC

MOVC A,@A+PC

Function: Move Code byte

Description: The MOVC instruction loads the accumulator with a code byte, or constant from program memory. The address of the byte fetched is the sum of the original unsigned eight-bit accumulator contents and the contents of the 16-bit Program Counter (PC). The PC is incremented to the address of the following instruction before being added with the accumulator. Sixteen-bit addition is performed so a carryout from the low-order eight bits may propagate through higher-order bits. No flags are affected.

Bytes: 1

Cycles: 2

Encoding: 1 0 0 0 0 0 1 1

Operation: MOVC
(PC) ←← (PC) + 1
(A) ←← ((A) + (PC))

* Obtained from the 1990 edition of the Intel 8-Bit Embedded Controllers Data Book.

15.6.72 MOVX A,@Ri

MOVX A,@Ri

Function: Move External

Description: A byte from external data memory is copied into the accumulator, hence the "X" appended to MOV. The contents of R0 or R1 in the current register bank provide an eight-bit address multiplexed with data on P0. Eight bits are sufficient for external I/O expansion decoding or for a relatively small RAM array. For somewhat larger arrays, any output port pins can be used to output higher-order address bits. These pins would be controlled by an output instruction preceding the MOVX.

Bytes: 1

Cycles: 2

Encoding: 1 1 1 0 0 0 1 i

Operation: MOVX
(A) ←← ((Ri))

* Obtained from the 1990 edition of the Intel 8-Bit Embedded Controllers Data Book.

15.6.73 MOVX A,@DPTR

MOVX A,@DPTR

Function: Move External

Description: A byte from external data memory is copied into the accumulator, hence the "X" appended to MOV. The Data Pointer generates a 16-bit address. P2 outputs the high-order eight address bits (the contents of DPH) while P0 multiplexes the low-order eight bits (DPL) with data. The P2 Special Function Register retains its previous contents while the P2 output buffers are emitting the contents of DPH. This form is faster and more efficient when accessing very large arrays (up to 64K bytes), since no additional instructions are needed to set up the output ports.

Bytes: 1

Cycles: 2

Encoding: 1 1 1 0 0 0 0 0

Operation: MOVX
(A) ←← ((Ri))

* Obtained from the 1990 edition of the Intel 8-Bit Embedded Controllers Data Book.

15.6.74 MOVX @Ri,A

MOVX @Ri,A

Function: Move External

Description: The content of the accumulator is copied to the external data memory location indicated by the specified register, hence the "X" appended to MOV. The contents of R0 or R1 in the current register bank provide an eight-bit address multiplexed with data on P0. Eight bits are sufficient for external I/O expansion decoding or for a relatively small RAM array. For somewhat larger arrays, any output port pins can be used to output higher-order address bits. These pins would be controlled by an output instruction preceding the MOVX.

Bytes: 1

Cycles: 2

Encoding: 1 1 1 1 0 0 1 i

Operation: MOVX
((Ri)) ←← (A)

* Obtained from the 1990 edition of the Intel 8-Bit Embedded Controllers Data Book.

15.6.75 MOVX @DPTR,A

MOVX @DPTR,A

Function: Move External

Description: The content of the accumulator is copied to the external memory location specified by the 16-bit Data Pointer, hence the "X" appended to MOV. The Data Pointer generates a 16-bit address. P2 outputs the high-order eight address bits (the contents of DPH) while P0 multiplexes the low-order eight bits (DPL) with data. The P2 Special Function Register retains its previous contents while the P2 output buffers are emitting the contents of DPH. This form is faster and more efficient when accessing very large arrays (up to 64K bytes), since no additional instructions are needed to set up the output ports.

Bytes: 1

Cycles: 2

Encoding: 1 1 1 1 0 0 0 0

Operation: MOVX
(DPTR) ←← (A)

* Obtained from the 1990 edition of the Intel 8-Bit Embedded Controllers Data Book.

15.6.76 MUL AB

MUL AB

Function: Multiply

Description: MUL AB multiplies the unsigned eight-bit integers in the Accumulator and the register B. The low-order byte of the sixteen-bit product is left in the Accumulator, and the high-order byte in B. If the product is greater than 255 (0FFH) the overflow flag is set; otherwise it is cleared. The carry is always cleared.

Bytes: 1

Cycles: 4

Encoding: 1 0 1 0 0 1 0 0

Operation: MUL
(A)0-7 ←← (A) X (B)
(B)15-8

* Obtained from the 1990 edition of the Intel 8-Bit Embedded Controllers Data Book.

15.6.77 NOP

NOP

Function: No Operation

Description: Execution continues at the following instruction. Other that the PC, no registers or flags are affected.

Bytes: 1

Cycles: 1

Encoding: 0 0 0 0 0 0 0 0

Operation: NOP
(PC) ←← (PC) + 1

* Obtained from the 1990 edition of the Intel 8-Bit Embedded Controllers Data Book.

15.6.78 ORL A,Rn

ORL A,Rn

Function: Logical-OR accumulator with register

Description: ORL performs the bitwise logical-OR operation between the values contained in the accumulator and the specified register and stores the result in the accumulator. No flags are affected.

Bytes: 1

Cycles: 1

Encoding: 0 1 0 0 1 r r r

Operation: ORL
(A) ←← (A) v (Rn)

* Obtained from the 1990 edition of the Intel 8-Bit Embedded Controllers Data Book.

15.6.79 ORL A,direct

ORL A,direct

Function: Logical-OR accumulator with direct memory

Description: ORL performs the bitwise logical-OR operation between the values contained in the accumulator and the value contained at the specified direct memory location and stores the result in the accumulator. No flags are affected.

Bytes: 2

Cycles: 1

Encoding: 0 1 0 0 0 1 0 1 direct address

Operation: ORL
(A) ←← (A) v (direct)

* Obtained from the 1990 edition of the Intel 8-Bit Embedded Controllers Data Book.

15.6.80 ORL A,@Ri

ORL A,@Ri

Function: Logical-OR accumulator with indirect memory

Description: ORL performs the bitwise logical-OR operation between the values contained in the accumulator and the value contained at the indirect memory location indicated by the contents of the specified register and stores the result in the accumulator. No flags are affected.

Bytes: 1

Cycles: 1

Encoding: 0 1 0 0 0 1 1 i

Operation: ORL
(A) ←← (A) v ((Ri))

* Obtained from the 1990 edition of the Intel 8-Bit Embedded Controllers Data Book.

15.6.81 ORL A,#data

ORL A,#data

Function: Logical-OR accumulator with immediate data

Description: ORL performs the bitwise logical-OR operation between the values contained in the accumulator and the value contained in the second byte of the instruction and stores the result in the accumulator. No flags are affected.

Bytes: 2

Cycles: 1

Encoding: 0 1 0 0 0 1 0 0 immediate data

Operation: ORL
(A) ←← (A) v #data

* Obtained from the 1990 edition of the Intel 8-Bit Embedded Controllers Data Book.

15.6.82 ORL direct,A

ORL direct,A

Function: Logical-OR direct memory with accumulator

Description: ORL performs the bitwise logical-OR operation between the values contained in the accumulator and the specified direct memory location and stores the result in the direct memory location. No flags are affected

Note: When this instruction is used to modify an output port, the value used as the original port data will be read from the output latch, not the input pins.

Bytes: 2

Cycles: 1

Encoding: 0 1 0 0 0 0 1 0 direct address

Operation: ORL
(direct) ←← (direct) v (A)

* Obtained from the 1990 edition of the Intel 8-Bit Embedded Controllers Data Book.

15.6.83 ORL direct,#data

ORL direct,#data

Function: Logical-OR direct memory with immediate data

Description: ORL performs the bitwise logical-OR operation between the values contained in the accumulator and the value contained in the second byte of the instruction and stores the result in the direct memory location. No flags are affected

Note: When this instruction is used to modify an output port, the value used as the original port data will be read from the output latch, not the input pins.

Bytes: 3

Cycles: 2

Encoding: 0 1 0 0 0 0 1 1 direct address immediate data

Operation: ORL
(direct) ←← (direct) v #data

* Obtained from the 1990 edition of the Intel 8-Bit Embedded Controllers Data Book.

15.6.84 ORL C,bit

ORL C,bit

Function: Logical-OR Carry with bit variable

Description: Set the carry flag if the state of the specified bit addressable memory location (second operand) is a 1; otherwise leave the carry in its current state. The source bit is not affected. No other flags are affected.

Bytes: 2

Cycles: 2

Encoding: 0 1 1 1 0 0 1 0 bit address

Operation: ORL
(C) ←← (C) v (bit)

* Obtained from the 1990 edition of the Intel 8-Bit Embedded Controllers Data Book.

15.6.85 ORL C,/bit

ORL C,/bit

Function: Logical-OR Carry with complement of bit variable

Description: Set the carry flag if the state of the specified bit addressable memory location (second operand) is a 0; otherwise leave the carry in its current state. The slash ("/") preceding the operand indicates that the logical complement of the addressed bit is used as the source value, but the source bit itself is not affected. No other flags are affected.

Bytes: 2

Cycles: 2

Encoding: 1 0 1 0 0 0 0 0 bit address

Operation: (ORL)
(C) ←← (C) v !(bit)

* Obtained from the 1990 edition of the Intel 8-Bit Embedded Controllers Data Book.

15.6.86 POP direct

POP direct

Function: Pop from stack

Description: The content of the internal RAM location **indirectly** addressed by the Stack Pointer is read, and the Stack Pointer is decremented by one. The value read is then transferred to the **directly** addressed byte indicated. No flags are affected.

Bytes: 2

Cycles: 2

Encoding: 1 1 0 1 0 0 0 0 direct address

Operation: POP
(direct) ←← ((SP))
(SP) ←← (SP) - 1

* Obtained from the 1990 edition of the Intel 8-Bit Embedded Controllers Data Book.

15.6.87 PUSH direct

PUSH direct

Function: Push onto stack

Description: The Stack Pointer is incremented by one. The contents of the **directly** addressable

memory location indicated, is then copied into the internal RAM location **indirectly** addressed by the Stack Pointer. No flags are affected.

Bytes: 2

Cycles: 2

Encoding: 1 1 0 0 0 0 0 0 direct address

Operation: PUSH
(SP) ←← (SP) + 1
((SP)) ←← (direct)

* Obtained from the 1990 edition of the Intel 8-Bit Embedded Controllers Data Book.

15.6.88 RET

RET

Function: Return from subroutine

Description: RET pops the high- and low-order bytes of the PC successively from the stack, decrementing the Stack Pointer by two. Program execution continues at the resulting address, generally the instruction immediately following an ACALL or LCALL. No flags are affected.

Bytes: 1

Cycles: 2

Encoding: 0 0 1 0 0 0 1 0

Operation: RET
(PC15-8) ←← ((SP))
(SP) ←← (SP) - 1
(PC7-0) ←← ((SP))
(SP) ←← (SP) - 1

* Obtained from the 1990 edition of the Intel 8-Bit Embedded Controllers Data Book.

15.6.89 RETI

RETI

Function: Return from interrupt

Description: RETI pops the high- and low-order bytes of the PC successively from the stack, and restores the interrupt logic to accept additional interrupts at the same priority level as the one just processed. The Stack Pointer is decremented by two. No other registers are affected; the PSW is not automatically restored to its pre-interrupt status. Program execution continues at the resulting address, which is generally the instruction immediately after the point at which the interrupt request was detected. If

a lower- or same-level interrupt had been pending when the RETI instruction is executed, that one instruction will be executed before the pending interrupt is processed.

Bytes: 1

Cycles: 2

Encoding: 0 0 1 1 0 0 1 0

Operation: RETI
(PC15-8) ←← ((SP))
(SP) ←← (SP) - 1
(PC7-0) ←← ((SP))
(SP) ←← (SP) - 1

* Obtained from the 1990 edition of the Intel 8-Bit Embedded Controllers Data Book.

15.6.90 RL A

RL A

Function: Rotate Accumulator Left

Description: The eight bits in the Accumulator are rotated one bit to the left. Bit 7 is rotated into the bit 0 position. No flags are affected.

Bytes: 1

Cycles: 1

Encoding: 0 0 1 0 0 0 1 1

Operation: RL
(An + 1) ←← (An) n = 0 - 6
(A0) ←← (A7)

* Obtained from the 1990 edition of the Intel 8-Bit Embedded Controllers Data Book.

15.6.91 RLC A

RLC A

Function: Rotate Accumulator Left through the Carry flag

Description: The eight bits in the Accumulator and the carry flag are together rotated one bit to the left. Bit 7 moves into the carry flag; the original state of the carry flag moves into the bit 0 position. No other flags are affected.

Bytes: 1

Cycles: 1

Encoding: 0 0 1 1 0 0 1 1

Operation: RLC
(An + 1) ←← (An) n = 0 - 6
(A0) ←← (C)
(C) ←← (A7)

* Obtained from the 1990 edition of the Intel 8-Bit Embedded Controllers Data Book.

15.6.92 RR A

RR A

Function: Rotate Accumulator Right

Description: The eight bits in the Accumulator are rotated one bit to the right. Bit 0 is rotated into the bit 7 position. No flags are affected.

Bytes: 1

Cycles: 1

Encoding: 0 0 0 0 0 0 1 1

Operation: RLC
(An) ←← (An + 1) n = 0 - 6
(A0) ←← (C)
(C) ←← (A7)

* Obtained from the 1990 edition of the Intel 8-Bit Embedded Controllers Data Book.

15.6.93 RRC A

RRC A

Function: Rotate Accumulator Right through the Carry flag

Description: The eight bits in the Accumulator and the carry flag are together rotated one bit to the right. Bit 0 moves into the carry flag; the original state of the carry flag moves into the bit 7 position. No other flags are affected.

Bytes: 1

Cycles: 1

Encoding: 0 0 0 1 0 0 1 1

Operation: RLC
(An) ←← (An + 1) n = 0 - 6
(A7) ←← (C)
(C) ←← (A0)

* Obtained from the 1990 edition of the Intel 8-Bit Embedded Controllers Data Book.

15.6.94 SETB C

SETB C

Function: Set Carry

Description: SETB C sets the carry flag to one. No other flags are affected.

Bytes: 1

Cycles: 1

Encoding: 1 1 0 1 0 0 1 1

Operation: SETB
(C) ←← 1

* Obtained from the 1990 edition of the Intel 8-Bit Embedded Controllers Data Book.

15.6.95 SETB bit

SETB bit

Function: Set Bit

Description: SETB sets the indicated directly addressable bit to one. No other flags are affected.

Bytes: 2

Cycles: 1

Encoding: 1 1 0 1 0 0 1 0 bit address

Operation: SETB
(bit) ←← 1

* Obtained from the 1990 edition of the Intel 8-Bit Embedded Controllers Data Book.

15.6.96 SJMP rel

SJMP rel

Function: Short Jump

Description: Program control branches unconditionally to the address indicated. The branch destination is computed by adding the signed displacement in the second instruction byte to the PC, after incrementing the PC twice. Therefore, the range of destinations allowed is from 128 bytes preceding this instruction to 127 bytes following it.

Bytes: 2

Cycles: 2

Encoding: 1 0 0 0 0 0 0 0 rel. address

Operation: SJMP
(PC) ←← (PC) + 2
(PC) ←← (PC) + rel

* Obtained from the 1990 edition of the Intel 8-Bit Embedded Controllers Data Book.

15.6.97 SUBB A,Rn

SUBB A,Rn

Function: Subtract Register from Accumulator with borrow

Description: SUBB subtracts the value contained in the indicated register and the carry flag together from the Accumulator. SUBB sets the carry (borrow) flag if a borrow is needed for bit 7, and clears C otherwise. (If C was set before executing a SUBB instruction, this indicates that a borrow was needed for the previous step in a multiple precision subtraction, so the carry is subtracted from the Accumulator along with the value contained in the specified register). AC is set if a borrow is needed for bit 3, and cleared otherwise. OV is set if a borrow is needed into bit 6, but not into bit 7, or into bit 7, but not bit 6.

When subtracting signed integers OV indicates a negative number produced when a negative value is subtracted from a positive value, or a positive result when a positive number is subtracted from a negative number.

Bytes: 1

Cycles: 1

Encoding: 1 0 0 1 1 r r r

Operation: SUBB
(A) ←← (A) - (C) - (Rn)

* Obtained from the 1990 edition of the Intel 8-Bit Embedded Controllers Data Book.

15.6.98 SUBB A,direct

SUBB A,direct

Function: Subtract Direct from Accumulator with borrow

Description: SUBB subtracts the value contained in the indicated directly addressable memory location and the carry flag together from the Accumulator. SUBB sets the carry (borrow) flag if a borrow is needed for bit 7, and clears C otherwise. (If C was set

before executing a SUBB instruction, this indicates that a borrow was needed for the previous step in a multiple precision subtraction, so the carry is subtracted from the Accumulator along with the value contained in the specified directly addressable memory location). AC is set if a borrow is needed for bit 3, and cleared otherwise. OV is set if a borrow is needed into bit 6, but not into bit 7, or into bit 7, but not bit 6.

When subtracting signed integers OV indicates a negative number produced when a negative value is subtracted from a positive value, or a positive result when a positive number is subtracted from a negative number.

Bytes: 2

Cycles: 1

Encoding: 1 0 0 1 0 1 0 1 direct address

Operation: SUBB
(A) ←← (A) - (C) - (direct)

* Obtained from the 1990 edition of the Intel 8-Bit Embedded Controllers Data Book.

15.6.99 SUBB A,@Ri

SUBB A,@Ri

Function: Subtract Indirect from Accumulator with borrow

Description: SUBB subtracts the value contained in the indirectly addressable memory location (specified by the contents of the indicated register) and the carry flag together from the Accumulator. SUBB sets the carry (borrow) flag if a borrow is needed for bit 7, and clears C otherwise. (If C was set before executing a SUBB instruction, this indicates that a borrow was needed for the previous step in a multiple precision subtraction, so the carry is subtracted from the Accumulator along with the value contained in the specified indirectly addressable memory location). AC is set if a borrow is needed for bit 3, and cleared otherwise. OV is set if a borrow is needed into bit 6, but not into bit 7, or into bit 7, but not bit 6.

When subtracting signed integers OV indicates a negative number produced when a negative value is subtracted from a positive value, or a positive result when a positive number is subtracted from a negative number.

Bytes: 1

Cycles: 1

Encoding: 1 0 0 1 0 1 1 i

Operation: SUBB
(A) ←← (A) - (C) - ((Ri))

* Obtained from the 1990 edition of the Intel 8-Bit Embedded Controllers Data Book.

15.6.100 SUBB A,#data

SUBB A,#data

Function: Subtract Immediate from Accumulator with borrow

Description: SUBB subtracts the immediate byte value (contained in the second instruction byte) and the carry flag together from the Accumulator. SUBB sets the carry (borrow) flag if a borrow is needed for bit 7, and clears C otherwise. (If C was set before executing a SUBB instruction, this indicates that a borrow was needed for the previous step in a multiple precision subtraction, so the carry is subtracted from the Accumulator along with the immediate byte value). AC is set if a borrow is needed for bit 3, and cleared otherwise. OV is set if a borrow is needed into bit 6, but not into bit 7, or into bit 7, but not bit 6.

When subtracting signed integers OV indicates a negative number produced when a negative value is subtracted from a positive value, or a positive result when a positive number is subtracted from a negative number.

Bytes: 2

Cycles: 1

Encoding: 1 0 0 1 0 1 0 0 immediate data

Operation: SUBB
(A) ←← (A) - (C) - #data

* Obtained from the 1990 edition of the Intel 8-Bit Embedded Controllers Data Book.

15.6.101 SWAP A

SWAP A

Function: Swap nibbles within the Accumulator

Description: SWAP A interchanges the low- and high-order nibbles (four-bit fields) of the Accumulator (bits 3-0 and bits 7-4). The operation can also be thought of as a four-bit rotate instruction. No flags are affected.

Bytes: 1

Cycles: 1

Encoding: 1 1 0 0 0 1 0 0

Operation: SWAP
(A3-0) ←← →→ (A7-4)

* Obtained from the 1990 edition of the Intel 8-Bit Embedded Controllers Data Book.

15.6.102 XCH A,Rn

XCH A,Rn

Function: Exchange Accumulator with Register

Description: XCH loads the Accumulator with the contents of the indicated register, at the same time writing the original Accumulator contents to the indicated register.

Bytes: 1

Cycles: 1

Encoding: 1 1 0 0 1 r r r

Operation: XCH
(A) ←← →→ (Rn)

* Obtained from the 1990 edition of the Intel 8-Bit Embedded Controllers Data Book.

15.6.103 XCH A,direct

XCH A,direct

Function: Exchange Accumulator with Direct Memory

Description: XCH loads the Accumulator with the contents of the indicated directly addressable memory location, at the same time writing the original Accumulator contents to the indicated directly addressable memory location.

Bytes: 2

Cycles: 1

Encoding: 1 1 0 0 0 1 0 1 direct address

Operation: XCH
(A) ←← →→ (direct)

* Obtained from the 1990 edition of the Intel 8-Bit Embedded Controllers Data Book.

15.6.104 XCH A,@Ri

XCH A,@Ri

Function: Exchange Accumulator with Indirect Memory

Description: XCH loads the Accumulator with the contents of the indirectly addressable memory location specified by the contents of the indicated register. At the same time writing the original Accumulator contents to the indirectly addressable memory location specified by the contents of the indicated register.

Bytes: 1

Cycles: 1

Encoding: 1 1 0 0 0 1 1 i

Operation: XCH
(A) ←← →→ ((Ri))

* Obtained from the 1990 edition of the Intel 8-Bit Embedded Controllers Data Book.

15.6.105 XCHD A,@Ri

XCHD A,@Ri

Function: Exchange Digit

Description: XCHD exchanges the low-order nibble of the Accumulator (bits 3-0), generally representing a hexadecimal or BCD digit, with that of the internal RAM location indirectly addressed by the specified register. The high-order nibbles (bits 7-4) of each register are not affected. No flags are altered.

Bytes: 1

Cycles: 1

Encoding: 1 1 0 1 0 1 1 i

Operation: XCHD
(A3-0) ←← →→ ((Ri3-0))

* Obtained from the 1990 edition of the Intel 8-Bit Embedded Controllers Data Book.

15.6.106 XRL A,Rn

XRL A,Rn

Function: Logical Exclusive-OR Accumulator with register

Description: XRL performs the bitwise logical Exclusive-OR operation between the values contained in the accumulator and the specified register and stores the result in the accumulator. No flags are affected.

Bytes: 1

Cycles: 1

Encoding: 0 1 1 0 1 r r r

Operation: XRL
(A) ←← (A) ∀(Rn)

* Obtained from the 1990 edition of the Intel 8-Bit Embedded Controllers Data Book.

15.6.107 XRL A,direct

XRL A,direct

Function: Logical Exclusive-OR Accumulator with direct memory

Description: XRL performs the bitwise logical Exclusive -OR operation between the values contained in the accumulator and the value contained at the specified direct memory location and stores the result in the accumulator. No flags are affected.

Bytes: 2

Cycles: 1

Encoding: 0 1 1 0 0 1 0 1 direct address

Operation: XRL
(A) ←← (A) ∀ (direct)

* Obtained from the 1990 edition of the Intel 8-Bit Embedded Controllers Data Book.

15.6.108 XRL A,@Ri

XRL A,@Ri

Function: Logical Exclusive-OR Accumulator with indirect memory

Description: XRL performs the bitwise logical Exclusive-OR operation between the values contained in the accumulator and the value contained at the indirect memory location indicated by the contents of the specified register and stores the result in the accumulator. No flags are affected.

Bytes: 1

Cycles: 1

Encoding: 0 1 1 0 0 1 1 i

Operation: XRL
(A) ←← (A) ∀ ((Ri))

* Obtained from the 1990 edition of the Intel 8-Bit Embedded Controllers Data Book.

15.6.109 XRL A,#data

XRL A,#data

Function: Logical Exclusive-OR Accumulator with immediate data

Description: XRL performs the bitwise logical Exclusive-OR operation between the values contained in the accumulator and the value contained in the second byte of the instruction and stores the result in the accumulator. No flags are affected.

Bytes: 2

Cycles: 1

Encoding: 0 1 1 0 0 1 0 0 immediate data

Operation: XRL
(A) ←← (A) ∇ #data

* Obtained from the 1990 edition of the Intel 8-Bit Embedded Controllers Data Book.

15.6.110 XRL direct,A

XRL direct,A

Function: Logical Exclusive-OR direct memory with accumulator

Description: XRL performs the bitwise logical Exclusive-OR operation between the values contained in the accumulator and the specified direct memory location and stores the result in the direct memory location. No flags are affected

Note: When this instruction is used to modify an output port, the value used as the original port data will be read from the output latch, not the input pins.

Bytes: 2

Cycles: 1

Encoding: 0 1 1 0 0 0 1 0 direct address

Operation: XRL
(direct) ←← (direct) ∇ (A)

* Obtained from the 1990 edition of the Intel 8-Bit Embedded Controllers Data Book.

15.6.111 XRL direct,#data

XRL direct,#data

Function: Logical Exclusive-OR direct memory with immediate data

Description: XRL performs the bitwise logical Exclusive-OR operation between the values contained in the accumulator and the value contained in the second byte of the instruction and stores the result in the direct memory location. No flags are affected

Note: When this instruction is used to modify an output port, the value used as the original port data will be read from the output latch, not the input pins.

Bytes: 3

Cycles: 2

Encoding: 0 1 1 0 0 0 1 1 direct address immediate data

Operation: XRL
(direct) ←← (direct) ▿ #data

* Obtained from the 1990 edition of the Intel 8-Bit Embedded Controllers Data Book.

Index

- 8 -

- A -

- B -

- C -

- D -

- E -

- F -

- H -

- I -

- L -

- M -

- N -

- O -

- P -

- R -

- S -

- T -

- V -

- W -