CpE 213
Example ISM26 – Using the debugger

Purpose

This example will show you how to use the debugger in Keil's Microvision 2 IDE to assemble and execute short sequences of instructions for the 8051. The examples starting on page 26 of ISM and continuing on through chapter 3 are mainly single instructions. The idea is to look up the instruction encoding in the programmer's manual for the 8051 (or Appendix C of ISM), manually 'assemble' the symbolic instructions into machine code, and figure out what it is supposed to do by reading the instruction description. This example will supplement those examples by providing you with another tool for exploring the 8051 instruction set.

Using the µVision2 debugger to enter code

µVision2 is an integrated design environment (IDE) for the 8051 family. It is provided by Keil Software (www.keil.com). IDE means that the various tools (editor, compiler, linker, debugger, etc) are combined into a single integrated environment with a graphical user interface instead of separate programs run from a command line. The debugger is used to simulate the execution of 8051 instructions so that a regular desktop PC can be used for code development instead of requiring you to have actual hardware. Normally the debugger is used with code produced by the C51 compiler from a C source file. It can also be used to explore the 8051 instruction set by entering instructions manually into the simulator's code memory using the debugger's ASM command. The alternative is to use an actual 8051 or variant with a *monitor* program and a dumb terminal.

The debugger is installed on the PC's in the ECE 106 CLC. Alternately you can download and install a copy of Keil's evaluation software for µVision2. This example assumes you are using Version 2.06 of µVision2. If you have access to the evaluation cdrom, you should spend a few minutes looking at the debugger tutorial. Just run setup.exe on the evaluation cdrom, click on Product Information, followed by Testing and Debugging and watch the 3-4 minute video to get an overview of the debugger's features.

Start µVision2 by clicking on the uv2 icon. Before you can do anything with µVision2 you must create a project. Click on Project|New Project, navigate to a directory to which you have access, and give your project a meaningful name like *chapter2*. Right click on 'Target 1' in the project window and select Select Device for Target 'Target 1' from the popup menu. Notice the large number of vendors of 8051 variants shown in the listbox. That is one reason why we selected the 8051 family for CpE 213. The particular processor is not material for now so just select a Generic 8051 from the list. Next click on Project| Options for Target 'Target 1' or select Options from the Target 1 popup menu. The target you selected should be listed in the target tab of the dialog box that comes up. Make sure that 'Use Simulator' is selected and that 'Load application at startup' is cleared. If the latter is not clear you will get an error message when you try to start the debugger since there is no application to load. Ok the dialog, click on Debug|Start/stop debug session (or hit Ctrl-F5) and you will be off and running in the debugger. You should see a window that looks something like the one shown in Figure 1.

Use the View menu item to enable the disassembly, output, and memory windows if they aren't already displayed. You can ignore the other windows for now since they will be more useful when we start using C code.

The 8051 always starts at location 0 in code memory after a reset so that is the place to start loading instructions. Keil refers to this location as C:0. Other useful 8051 memory spaces include D (internal ram), and X (external ram if there is any). Select the address box in the memory window and enter the address C:0 and you'll see a display of the first few locations of code space. They will most likely be 0's since we haven't loaded anything yet. Notice that the disassembly window shows 0's displayed as NOP since 0 is the instruction code for a no-operation (do-nothing) instruction

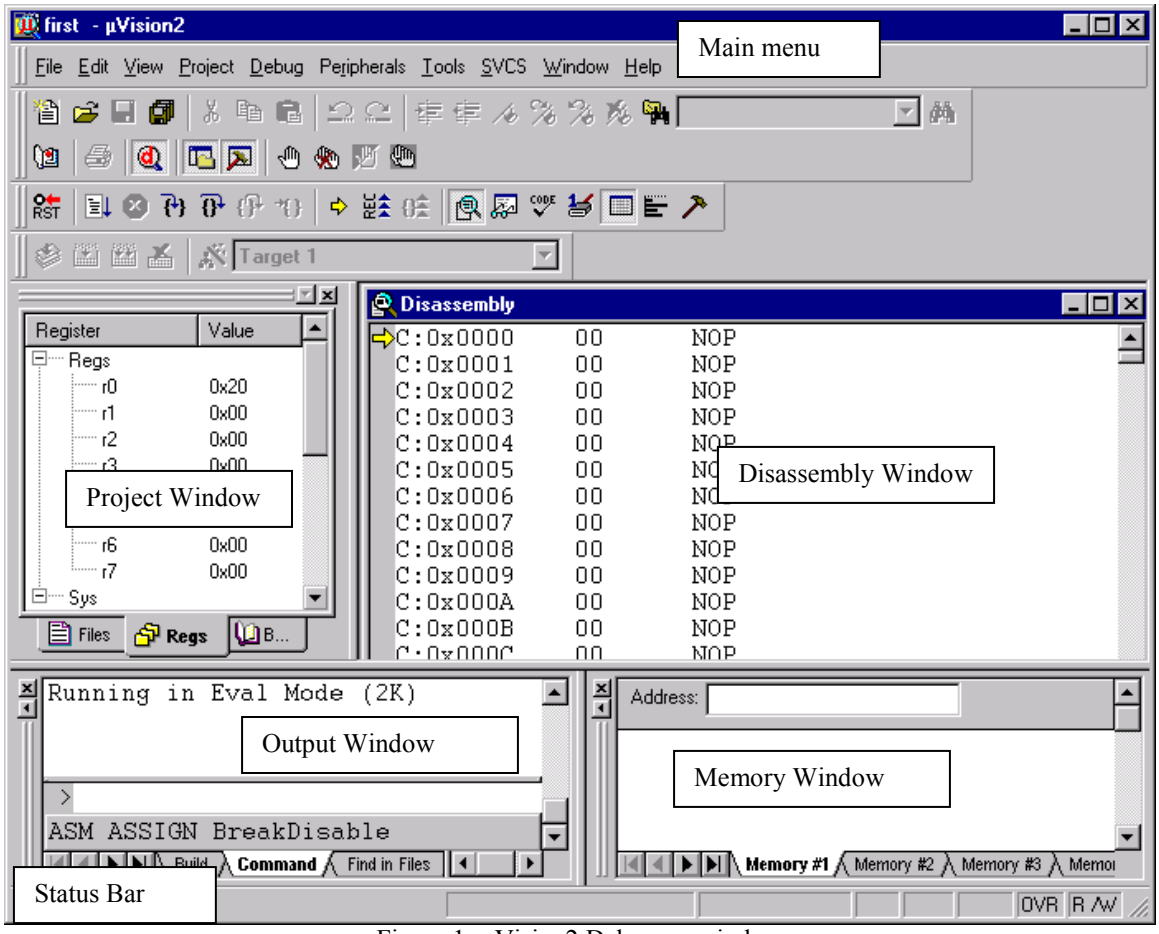

Figure 1 µVision2 Debugger window

. In the output window next to the > prompt, enter the command 'E C C:0=1,2,3'. Notice that as you type, the line underneath the command line lists the valid commands corresponding to what you type. E stands for Enter (data), the first C is the data type (character), the C:0 is location 0 in code space, and the 1,2,3 are the first 3 bytes of data. When you hit <enter> you should see the memory window update, and the disassembly window will display the mnemonic equivalent of the machine code 01 02 03.

Enter command

Now select the Memory #2 tab in the memory window and enter D:0 in the address box. This will display the first few locations of the 8051's internal ram. Enter some data here by using the command 'E C D:0=1,2,3'. This time something new happens. The register window shows that the contents of R0, R1, and R2 have changed to 1, 2, and 3 respectively. That is because the eight registers R0 through R7 are in fact the same as the first 8 storage locations in internal ram. R2 is simply a shorthand way to specify the 8-bit address 02H. It lets the 8051 access a few data locations with only a three bit address, thus saving some code space. You can access xdata in the same way. Simply substitute the address X:0 (or whatever) for C:0 or D:0.

At this point we could look up opcodes in the programmer's manual and manually enter them into code memory using the E(nter) command. That's educational, good information if you happen to be writing an assembler, but not particularly productive if you are trying to write an application program for the 8051. We can go from symbolic code to numeric code by using the ASM command. Type the command 'ASM 0' to start inserting instructions at location 0. That's the best place to start since that's where the 8051 begins executing after a restart. Now type the command ASM MOV a,#01h and see what is displayed in the disassembly window. The first line changes to C:0x0000 74 01 MOV A,#01H and the second changes to C:0x0002. That means the mov instruction is encoded into two bytes as 74 01 (in hexadecimal) and the next instruction will be located at location 2. The '#' in '#01H' means that this is an *immediate address*

ASM command

mode' instruction which means that the 01H part of the instruction is data to be loaded into the accumulator (A). What happens if you leave off the '#' (alternately called the *pound sign* or *sharp sign* depending on whether you have a commercial or musical preference)? Go ahead and try it by entering the command ASM MOV A,1. What does mov a,r1 assemble into? What is the result of executing these three instructions? If we examine the programmer's reference manual we'll see that the first, as expected, loads a 0x01 into A, the second loads the contents of location 1 in internal ram (data space or D:1) into A, and the third loads the contents of Register 1 (R1) into A. Since R1 and D:1 refer to the same location, and since we previously loaded a 0x02 into location 1, the last two instructions both load a 0x02 into A and are equivalent. The third instruction only takes a single byte and is preferred most of the time. That's *most of the time* though and not *all the time*. That's because R1 and D:1 aren't always the same location. You can make R0 through R7 refer to any of four different blocks of 8 bytes of data. The first 32 bytes of data make up four banks of eight general registers. The specific bank in use is selected by PSW bits 3 and 4 (PSW.3 and PSW.4) displayed by the debugger as the rs bits.

Reset icon


Step into icon

Let's verify what the programmer's manual tells us by executing the instructions we've just entered. Make sure the program counter (PC) is set to 0 by clicking on the RST icon. Click on the step into icon in order to execute one instruction at a time. Be careful to notice all of the registers and storage locations that change as a result of executing the instructions. Also note the *states* counter in the sys category in the project window. The states counter displays the total number of instruction cycles that have executed since reset occurred. The states counter will increment by the number of cycles each instruction takes as documented in the programmer's reference manual (or appendix C in the text).

Summary

This example has shown:

- How to set up the Microvision 2 debugger so it can be used without source code
- How to use the ASM command to enter symbolic instructions
- How to execute single or multiple instructions
- How to display the results of instruction execution.

Questions

Answer the following questions.

1. Complete Table 1 by filling in the instruction encoding in hex and the number of cycles it takes for the instructions to execute.
2. List the register and memory locations that change as a result of executing the instructions in Table 1.

Location (hex)	Machine code (hex)	Instruction	Cycles (States)
0000	74 01	MOV A,#01H	1
0002		MOV R7,A	
		ADD A,R7	
		MOV B,#2	
		MUL AB	
		MOV P1,A	
		MOV PCON,#2	
		MOV PSW,#0	