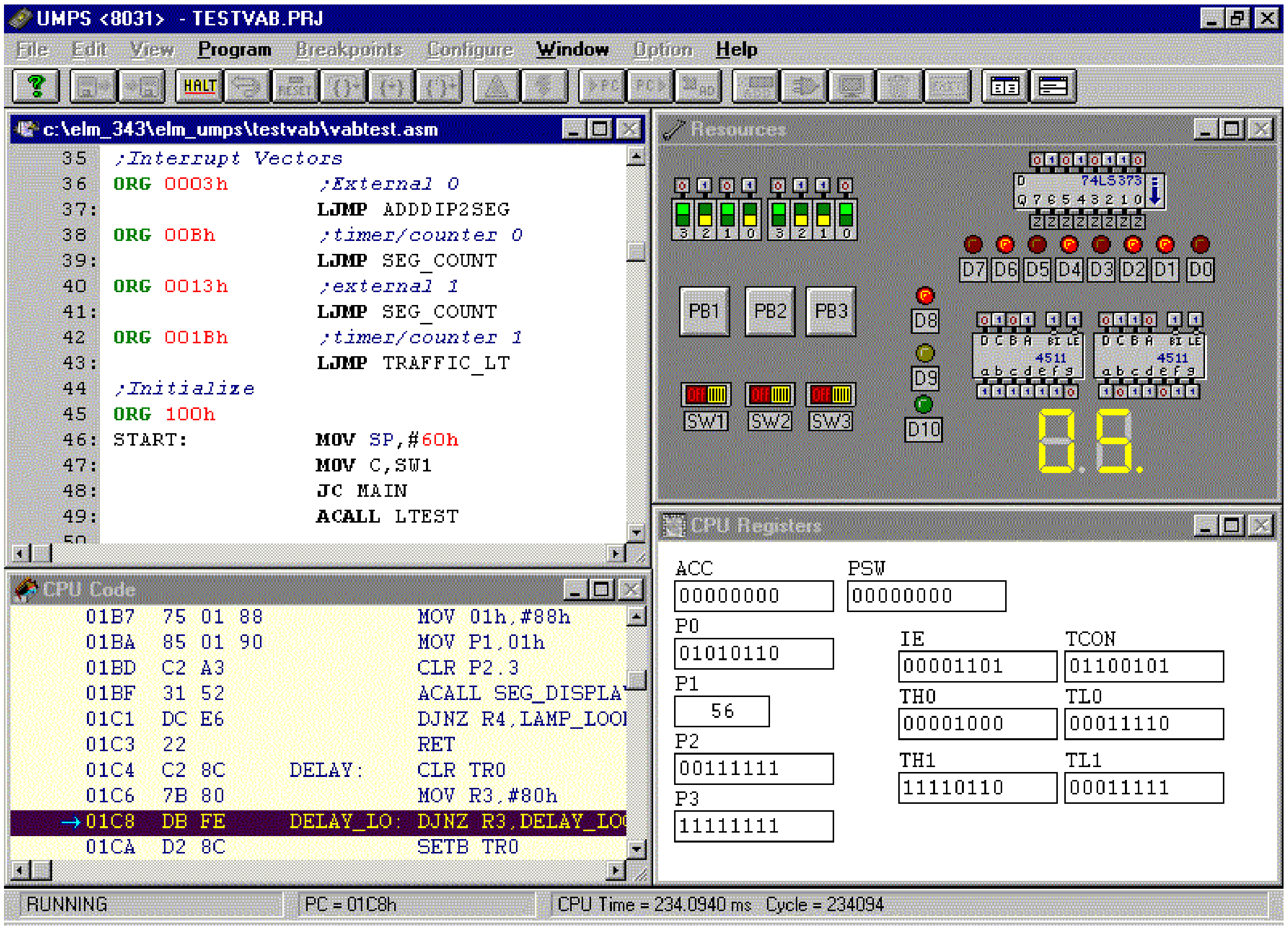
Application of Microcontrollers Labs Part II - Intel 8051 and UMPS®

Version 2.0


UMPS <8031> - TESTVAB.PRJ
File Edit View Program Breakpoints Configure Window Option Help
HALT
c:\elm_343\elm_umps\testvab\vabtest.asm
Resources
CPU Registers
CPU Code
74LS373
4511
PB1 PB2 PB3
SW1 SW2 SW3
D7 D6 D5 D4 D3 D2 D1 D0
D8 D9 D10

```
;Interrupt Vectors
ORG 0003h               ;External 0
                        LJMP ADDDIP2SEG
ORG 00Bh                ;timer/counter 0
                        LJMP SEG_COUNT
ORG 0013h               ;external 1
                        LJMP SEG_COUNT
ORG 001Bh               ;timer/counter 1
                        LJMP TRAFFIC_LT
;Initialize
ORG 100h
START:                  MOV SP,#60h
                        MOV C,SW1
                        JC MAIN
                        ACALL LTEST
```

```
  01B7  75 01 88              MOV 01h,#88h
  01BA  85 01 90              MOV P1,01h
  01BD  C2 A3                 CLR P2.3
  01BF  31 52                 ACALL SEG_DISPLA
  01C1  DC E6                 DJNZ R4,LAMP_LOO
  01C3  22                    RET
  01C4  C2 8C     DELAY:      CLR TR0
  01C6  7B 80                 MOV R3,#80h
→ 01C8  DB FE     DELAY_LO:   DJNZ R3,DELAY_LO
  01CA  D2 8C                 SETB TR0
```

```
ACC         PSW
00000000    00000000
P0
01010110    IE          TCON
P1          00001101    01100101
   56       TH0         TL0
P2          00001000    00011110
00111111
P3          TH1         TL1
11111111    11110110    00011111
```

```
RUNNING    PC = 01C8h    CPU Time = 234.0940 ms  Cycle = 234094
```

Electronics Management

Department of Information Management Systems

Office of Off-Campus Academic Programs

College of Applied Sciences and Arts

Southern Illinois University Carbondale

Discussion For Our Non-SIU Users

The *Application of Microcontrollers* Manual and Labs were designed for off-campus students at military bases. During a 16-week period, the students take 3 lecture courses, each presented over 3 weekends. The first 2 courses cover analog and digital principles. The final course is on microcontrollers using the 8051 as an example.

The students independently perform the first part of this manual and the labs during the first 2 courses. Part I introduces controllers using the BASIC Stamp from Parallax, Inc. The students are provided the equipment to program and interface a microcontroller using a relatively simple language: BASIC. Part I is intended to reinforce the lecture material on basic electronics principles using the microcontroller. It may be downloaded from Parallax's education website at: http://www.stampsinclass.com

Part II, this material, is performed by the students while participating in their final lecture course covering the 8051. In this part students learn principles of programming a microcontroller using Assembler. UMPS from Virtual Micro Design is used to simulate the 8051 allowing students to see how instructions affect registers, memory and devices. A Virtual Activity Board with I/O was designed with the UMPS resources. UMPS projects were designed for the material allowing student to quickly load samples while reading about them.

We thank Philippe Techer for his generosity in allowing use to distribute UMPS demo versions to our students, use of the logos in our documents, and for developing such a wonderful development AND learning tool.

We hope you find the material developed useful. Please feel free to contact us. We enjoy feedback and hearing how our material is being used.

Martin Hebel
Southern Illinois University Carbondale

Application of Microcontrollers

Contact Information

E-mail:

Primary developer of the manual and labs:
Martin Hebel .... mhebel@siu.edu

Contributing developer:
Will Devenport.. willd@siu.edu

Director, Off-Campus Academic Programs:
Dr. Terry Bowman.. tbowman@siu.edu

Chair, Department of Information Management Systems:
Dr. Janice Schoen Henry... jshenry@siu.edu

Mailing:

Electronics Management
College of Applied Sciences and Arts
Southern Illinois University, Carbondale
Carbondale, IL 62901-6614

The following people are thanked for their contributions: Ken Gracey and the gang at Parallax for their work helping make this possible for our students; Philippe Techer at Virtual Micro Design for designing a great simulation package and working with us; Myke Predko for his feedback and recommendations; I. Scott MacKenzie for a concise text on the 8051; students on campus and at Ft. Gordon, Cherry Point and New River for feedback; Cheri Barral for editing; and finally Terry Bowman and Jan Henry for budgeting the endeavor and wanting the best education for our students.

Key Web Sites:

Electronics Management Home Page:	www.siu.edu/~imsasa/elm
Off-Campus Programs Home Page:	http://131.230.64.6/
Parallax Incorporated Home Page:	www.parallaxinc.com
	www.stampsinclass.com
Virtual Micro Design Home Page (UMPS):	www.vmdesign.com

Distributors & Additional Information:

Digi-Key Electronics - Stamps, components	www.digikey.com
Jameco Electronics - Stamps, components	www.jameco.com
JDR Electronics - Stamps, components	www.jdr.com
Wirz Electronics - UMPS U.S. Sales	www.wirz.com
Peter H. Anderson - General microcontroller information	www.phanderson.com
SelmaWare Solutions - Specialized interfacing software	www.selmaware.com

Texts:

The 8051 Microcontroller, 3rd ed. 1999, Scott MacKenzie. Prentice-Hall
ISBN: 0-13-780008-8

Handbook of Microcontrollers. 1999, Myke Predko. McGraw-Hill
ISBN: 0-07-913716-4

Programming and Customizing the 8051 Microcontroller. 1999, Myke Predko. McGraw-Hill.
ISBN: 0-07-134192-7

The Microcontroller Idea Book. 1994, Jan Axelson. Lakeview Research.
ISBN: 096508190-7

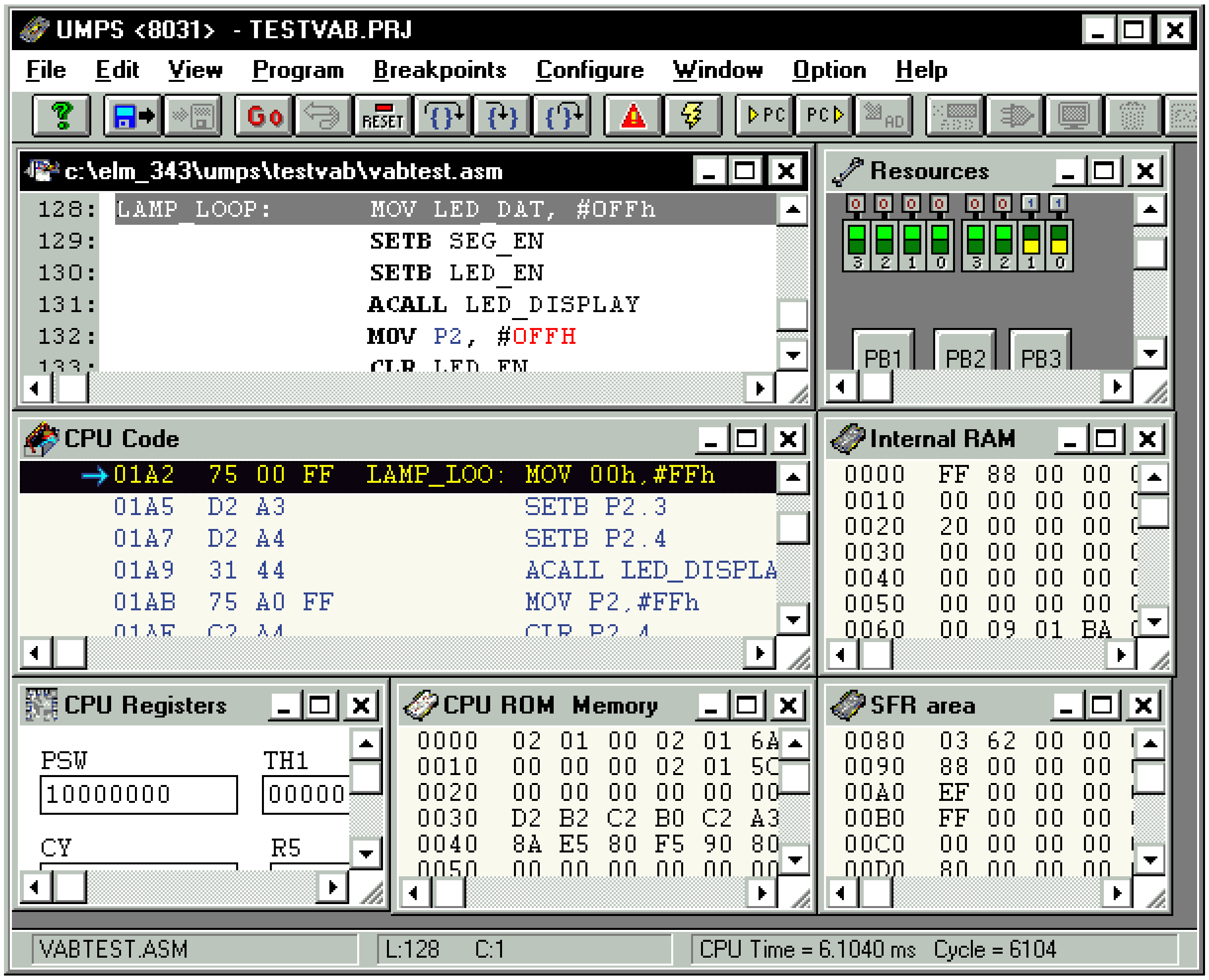
Table of Contents

Introduction: Using the UMPS® Software

The 8051 can be a complex microcontroller to program. A simulation package, such as UMPS from Virtual Micro Devices to simulate the 8051, and other controllers, can also be quite complex. In this introduction we will discuss some of the key features of UMPS.

After installing UMPS from your CD-ROM (*please see readme.txt on your CD for installation instructions*) or floppies, you will be able to execute the program from your Windows start bar: Start → Programs → UMPS 1.77- Demo → UMPS 1.77- Demo.

Figure-1 below is a sample screenshot of the VABTest project with additional windows added. Let's discuss the windows, tool bars and menus before you load your first project. Many of the features discussed will be clearer as you read the manual and perform the labs.


UMPS <8031> - TESTVAB.PRJ
File Edit View Program Breakpoints Configure Window Option Help
Go
RESET
PC
Resources
PB1
PB2
PB3
CPU Code
Internal RAM
CPU Registers
CPU ROM Memory
SFR area

c:\elm_343\umps\testvab\vabtest.asm

```
LAMP_LOOP:          MOV LED_DAT, #OFFh
                    SETB SEG_EN
                    SETB LED_EN
                    ACALL LED_DISPLAY
                    MOV P2, #OFFH
                    CLR LED_EN
```

CPU Code

```
→01A2  75 00 FF  LAMP_LOO: MOV 00h,#FFh
 01A5  D2 A3               SETB P2.3
 01A7  D2 A4               SETB P2.4
 01A9  31 44               ACALL LED_DISPLA
 01AB  75 A0 FF            MOV P2,#FFh
 01AE  C2 A4               CLR P2.4
```

Internal RAM

```
0000  FF 88 00 00
0010  00 00 00 00
0020  20 00 00 00
0030  00 00 00 00
0040  00 00 00 00
0050  00 00 00 00
0060  00 09 01 BA
```

CPU Registers

```
PSW            TH1
10000000       00000

CY             R5
```

CPU ROM Memory

```
0000  02 01 00 02 01 6A
0010  00 00 00 02 01 5C
0020  00 00 00 00 00 00
0030  D2 B2 C2 B0 C2 A3
0040  8A E5 80 F5 90 80
0050  00 00 00 00 00 00
```

SFR area

```
0080  03 62 00 00
0090  88 00 00 00
00A0  EF 00 00 00
00B0  FF 00 00 00
00C0  00 00 00 00
00D0  80 00 00 00
```

VABTEST.ASM | L:128 C:1 | CPU Time = 6.1040 ms Cycle = 6104

Figure -1: UMPS Screenshot

UMPS Windows:

1. Note first that the main UMPS window reads UMPS <8031>. The 8031, for our purposes, is functionally equivalent to the 8051. The text in this window is saved as a .txt or .asm.

2. The very top-left window titled C:\elm343\UMPS\vabtest.asm is an Assembler text window. Programs written in Assembler are compiled into machine code. When this window is highlighted, the contents may be saved as .txt or .asm.

3. The CPU Code window contains the assembled code from Assembler, or mnemonic code may be directly entered. The contents of this window may be saved in numerous formats, though for our purposes Intel HEX16 (.hex) will be used.

4. CPU Registers window shows selected special function registers of the microcontroller. These can be used to monitor operation of the microcontroller to verify program expectations. The contents of this file may be saves as a .REG file.

5. CPU ROM displays the contents of the 4K of internal ROM on the 8051 in hexadecimal and, to the far right, ASCII (not shown). The demo version of UMPS does not allow the saving of this window, but it would use the same formats as the CPU Code.

6. The Resources window shows input and output devices that can be simulated as connected to the microcontroller. The demo version of UMPS does not allow saving of this window, but resources produced for this manual are saved as .env (vab.env).

7. The Internal RAM window displays the contents in hexadecimal and ASCII of the 128 bytes of RAM available to the programmer on the 8051. The demo version of UMPS does not allow the saving of this window, but would use the same formats as the CPU Code.

8. The SFR Area window displays the contents in hexadecimal and ASCII of the 128 bytes of Special Function Registers used to control the 8051. The demo version of UMPS does not allow the saving of this window, but it would use the same formats as the CPU Code.

Furthermore, the entire collection of windows could be saved as project file (.PRJ), though this is not functional on the demo version.

At the bottom of the UMPS window is a status bar that shows the current line (L) and character (C) being executed in the Assembler Programming window. If the CPU code window is selected the bar will show the current value of the Program Counter (PC). Also shown is current CPU time (the simulated time elapse since the CPU was reset) and the number of instruction cycles elapsed since a reset.

UMPS Demonstration Version Limitations

We've mentioned a few already, but let's list them all:

- 8031 (8051) ROM memory limited to 2K bytes.
- Limit of about 800,000 instruction cycles per reset.
- Inability to save projects, resource configurations and ROM/RAM contents. CPU code and Assembler windows may be saved.
- Reduced CPU Description help files (the limited 8031 description help file should have been replaced by the ELM 343 installation software by permission of Virtual Micro Design).
- Resources will be reset when performing a CPU reset.
- The demo version will operate for approximately 3 months.

None of these limitations should prevent you from using the projects designed for this manual, or from creating your own projects. You will be limited to saving your CPU and/or Assembly files separately and using our canned resources, such as the Virtual Activity Board (other resource files may become available depending on the needs of the ELM-343 course).

UMPS Toolbar

Let's take a look at the functions of most of the buttons on the toolbar:

Help:
Used to request the help file for a given window.

Save & Open:
Save or open a file for a given type of window. Most of our work will be done with projects that require the use of the File menu.

Go/Halt:
Instructs the microcontroller simulation to execute (go). When running, this button will change to read HALT.

Reset:
This will perform a reset of the simulated microcontroller, (PC → 0000h, and various registers.) In the demo version it will also reset all resources.

Stepping (Tracing):
This will allow single stepping through code to observe its flow and actions on registers. There are three ways to step or trace:

- *Step or trace over*: The program stepping will exclude stepping through subroutines called (F8 key).
- *Step or trace into*: The program stepping will include subroutines (F7 key).
- *Step or trace out-of*: The program stepping will cease until the current subroutine exits (Alt-F8).

Toggle Breakpoint:

A breakpoint may be set/cleared on any Assembler or CPU code line to halt execution to allow analysis of registers or single stepping for debugging.

Compile:

Compiles the specified Assembler file (.ASM or .TXT) for the current microcontroller.

Set Program Counter (PC):

Used in manipulating the Program Counter (PC):

- Set PC to the currently selected line address.
- Set the current line address to the PC.
- Set the PC to a specified address via a pop-up window.

Resource/Register Tools:

These are used when working with the Resource or CPU Register Windows to configure them:

- Add a resource or register.
- Connect a resource.
- Edit a resource or register.
- Delete a resource or register.
- Exit out of configuring.

(To enter into configuration mode, use Configure → Resources or Configure → CPU_Registers.)

Menu Items of Note

We will look at a handful of menu functions which may not have toolbar equivalents and that may be important to the completion of these labs.

File →

New - Creates a new window for Assembler programs.

Load - Used to load a file of any type: CPU (Hex), Assembler (.asm), register (.reg) and resource (.env).

Load Project **- Used to load a project files, such as the ones created for these labs and the manual.**

View →

Under view we are able to open windows to view the CPU code, resources, RAM, ROM, SFR's, etc.

Configure→
Load CPU - Allows loading of a specific CPU for testing. While we will be working with only the 8031 (8051), sometimes the simplest way to clear out the contents of a current project is to select another CPU and then go back to the 8031.

Should another CPU be loaded and you attempt to load a project, you may be asked a couple confusing questions:
CPU: 8031 is used in this project, continue? -- Click '**Yes**', you understand you are loading another CPU's project file.

Continue with current CPU ? or load CPU: 8031 (No). Click '**No**' to load the 8031 (or required) CPU.

Option →
Run Mode… - Here we can designate how fast to clock the CPU, whether to break on interrupt calls, and how quickly to refresh the resources (this can dramatically affect your simulation speed).

Miscellaneous → Preferences - For the notation preference we will be using Intel (h,b) instead of Motorola ($,%) for base 16 and base 2 numbers.

Help
Under help we can view the general help files, or the current CPU Description help file. This help file is a very good reference for programming!

Well, I think that's enough discussion for now, let's test-drive it! The projects for these labs were designed to run in a screen resolution of 800x600.

Running Project Files

1) Run the UMPS Demo

2) Change the notation base to Intel (Options → Miscellaneous → Preferences)

3) Using File → Load project, locate and open **testvab.prj** in *c:/elm_343/umps/testvab*. (Note that each project has its own folder.)

4) The UMPS environment should now have an Assembler window, resource window of the Virtual Activity Board (VAB), CPU Code window and a CPU Register Window.

5) Click the "RESET" button on the toolbar. This will reset the simulated microcontroller and the resources in the demo version.

6) Click the "GO" button. The CPU code will not begin executing. The devices on the VAB will have the following effects:
   - The Red/Yellow/Green (traffic) lights will sequence.
   - Clicking the DIP switches in the upper left corner will cause to corresponding 8 LEDs to change.
   - Pressing PB1 will add the binary value of the DIPS to the 7-segment display.
   - Pressing PB2 will increment the 7-segment display count by one.
   - If SW1 is ON when the project is run following a reset, a short lamp test of all LEDs will occur.
   - If SW2 is ON, the 7-segment display will count rapidly.

7) Press the "HALT" button to stop execution.

8) Use the various stepping buttons or function-keys discussed previously to step through the program. Notice the following differences:
   - Using Step-over (F8) you can step through each line of code, but calls, such as the ACALL LED_Display, will be processed without stepping.
   - Using Step-into (F7) will step though ALL code including calls.
   - Using Step-out of (ALT-F8) will exit a called routine.

OK, now that you have the hang of the VAB and projects, let's create a program of your own to place the DIP switch data on the LEDs.

Writing in CPU Code

1) Clear out the CPU by using $\boxed{\text{Configure}}$ → $\boxed{\text{Load CPU}}$ and select **8031.cpl**. When asked about saving, click NO since you cannot save on the demo version.

2) Load up the VAB resource file using:
   $\boxed{\text{File}}$ → $\boxed{\text{Load}}$
   Under format in the load window, select **Resource**.
   In the c:/elm_343/umps directory, select **vab.env**.

3) View the CPU code by using $\boxed{\text{View}}$ → $\boxed{\text{CPU Code}}$.

4) On the right side of the CPU Code window, replace the NOP code with the following. (After each line is entered, move to the next line by pressing ENTER.)

```
SETB P2.4
Loop:  MOV P1, P0
SJMP LOOP
```

- Notice that as each line is entered, the hexadecimal on the left is updated.
- If you make a syntax mistake, an error message will appear and the code will not be assembled (try SET P2.4).

- If you make a mistake on a labeled line (Loop:), the label will be in memory and cannot be re-used. You can select a different label name, or use $\boxed{\text{Program}}$ → $\boxed{\text{Clear Labels}}$ to reset the labels.

5) Run the program and confirm that the LEDs follow the DIP settings.

6) Halt the program. With the CPU Code window selected, use $\boxed{\text{File}}$ → $\boxed{\text{Save file as}}$ to save your file. Under format select **Intel HEX16**. Open the c:/elm_343/umps/student folder. Under Filename type in **test1** and click OK.

Now let's try writing code in Assembler that will move the DIP settings to the 7-segments.

Writing Assembler Code

1) With the VAB showing and a CPU Code window open, click $\boxed{\text{File}}$ → $\boxed{\text{New}}$.

2) Reposition the window so it is out of the way of the others.

3) In the new window type the following code:

```
;Test of Assembler
DIPS equ P0
DispBus equ P1

org 0000h
            CLR P2.4      ;Disable '373 for LEDs
            CLR P2.3      ;Enable 4511's for 7 segments
Loop7:      MOV DispBus, DIPS
            SJMP Loop7
```

4) Use $\boxed{\text{File}}$ → $\boxed{\text{Save file as}}$ to save the file as a test2 under the student folder.

5) Click the Compile button (lighting bolt) and select test2.asm that you just save. A window should appear and the program should be compiled with no errors.

6) Note that the code in the CPU Code window has been replaced with the compiled code of your program.

7) RESET and Run your program. (With some DIP settings, the 7-segment display may be blank)

Viewing a Register

With the following program halted, perform the following:

1) Click $\boxed{\text{Configure}}$ → $\boxed{\text{CPU Registers}}$. Resize the window.

2) Click the **Add** button on the toolbar.

3) Find and select **P0**. Click OK.

4) Perform an **Add** for **P1**.

5) Reposition the register boxes. Double-click the P0 register to see its options.

6) Click the **Exit** button on the toolbar to exit register editing.

7) Run the program and observe the registers changing as you change DIP settings.

On to your labs!

Lab I: Introduction to the 8051 and UMPS

References:
A. Application of Microcontrollers Manual V2.0. 2000, Southern Illinois University.
B. MacKenzie, I.S., 1999. The 8051 microcontroller, 3rd ed. Prentice Hall.

Objectives:
1) Discuss the relationship between mnemonics and machine code.
2) List the features of the Intel 8051 including memory and ports.
3) Identify opcodes and operands for instructions.
4) Use the 8051 RAM memory map in programming.
5) Discuss key Special Function Registers including the Accumulator, Program Status Word, Stack Pointer, Program Counter.
6) Use UMPS for CPU simulations, including device connections of the Virtual Activity Board.

- Read and perform the steps in the previous section, *Introduction: Using the UMPS Software*.
- Read Section I of Reference A and test applicable project files.
- Refer to Reference B as required for further information.

---

1. (2 pts) Machine code is typically represented in the ________________ number base. The symbolic instructions that translate directly to machine code are called ________________.

2. (3 pts) The 8051 has ______ bytes of ROM and a total of _____ bytes of RAM. Of the total amount of RAM ______ bytes are available for general use.

3. (1 pt) The last I/O bit on port 3 of the 8051 is denoted by the symbol ______.

4. (1 pt) SW3 of the Virtual Activity Board is connected to I/O ________.

5. (1 pt) Identify the opcode(s) _____________ and operand(s) ___________ in the following code:

```
MOV A, #4Fh
```

6. (1 pt) 18h is a bit that is located within the Bit Addressable RAM. This bit would be located in byte address _________.

7. (1 pt) When an add operation exceeds a value of +127, the ________ flag in the PSW would be set.

8. (2 pts) The return memory locations for a subroutine call is held in the ____________. The register that keeps track of where these locations are stored is known as the __________ __________.

9. (1 pt) In the PSW, if RS0 is 1, and RS1 is 1, R3 would refer to RAM address _______.

10. (1 pts) Load **proj_i-2**. Set a breakpoint at address 002Fh. Run the program. Record the following values when the program halts at this breakpoint.

    PC: ________ ACC: ______________ PSW:______________ SP:___________

11. (1 pt) Load project **lab_i.prj** in folder **lab_i**. JNC means to Jump to the specified address if CY is not set (no carry). Set break points at memory locations 000Ch and 000Fh. Record the number of times the microcontroller repeats the loop "LOOP" prior to continuing on to 000Fh: _____.

12. (3 pts) Why did it repeat this number of times (be specific)?

13. (2 pts) Clear the breakpoints and run **lab_i**. RAM memory location 08h changes rapidly. Why (be specific)?

Lab J: Assembler, Opcodes and Addressing.

References:

A. Application of Microcontrollers Manual V2.0. 2000, Southern Illinois University.
B. MacKenzie, I.S., 1999. The 8051 microcontroller, 3rd ed. Prentice Hall.

Objectives:

1) Discuss advantages and limitations of Assembler.
2) Write code in Assembler to use symbols, origin points, and macros.
3) Identify the differences between addressing modes.
4) Write code in Assembler to use various addressing modes.

.

- Read Section J of Reference A and test applicable project files.
- Refer to Reference B as required for further information.

---

1. (2 pts) Why can't an Assembly Language program be compiled for multiple processor families?

2. (1 pt) To instruct the compiler to originate code beginning a memory location 0100h, the instruction ______________ would be used.

3. (2 pts) The following is a macro that will place the values of PB1 - PB3 on the 'traffic light' LEDs of the VAB. Write Assembler Code that will continually call this macro in a loop (use an absolute address jump). Begin compiling at memory address 0000h.

```
MACRO PB_Traffic
    MOV C, P3.2
    MOV P2.5, C
    MOV C, P3.3
    MOV P2.6,C
    MOV C, P3.4
    MOV P2.7,C
ENDMAC
```

4. (5 pts) The following code is from Project Lab_J. For each line fill in the type of addressing being used by the instruction for the source and destination data (if either is not applicable, place an X.

	Destination	Source
org 0000h		
Start: MOV R0, #41h	__________________	_________________
SETB P2.4	__________________	_________________
Loop:		
MOV A, R0	__________________	_________________
MOV @R0, A	__________________	_________________
ACALL Sub	__________________	_________________
INC R0	__________________	_________________
CJNE R0, #5Bh, Loop	__________________	_________________
LJMP Start	__________________	_________________
Sub:		
MOV P1,R0	__________________	_________________
RET		

5. (2 pts) Load and run the project **Lab_J.prj**. What occurs in the Internal RAM window?

6. (2 pts) Discuss how the program accomplishes this.

7. (1 pt) How many bytes are required for the complete program for project Lab_J ? _________.

8. (1 pt) Reset the project. How many cycles does it require to complete one entire evolution (until it jumps back to Start, set a breakpoint at 0000h). ______________. How long does this require running at the established clock speed of 12 MHz? ___________.

9. (1 pt) How many times would this routine repeat in 1 second? ____________.

10. (3 pts) Write code for the following actions:

a. Load the accumulator immediately with the number 50h: ____________

b. Place the value of the DIP switches in the R1 register: ____________

c. Load the accumulator with the location pointed to by the R1 register: ____________

d. Jump to the absolute address of 0120h: ____________

e. Enable the '373 for the LEDs: ____________

f. Place the value of accumulator on the display bus: ____________

Lab K: Interrupts- External, Timers and Counters

References:

A. Application of Microcontrollers Manual V2.0. 2000, Southern Illinois University.
B. MacKenzie, I.S., 1999. The 8051 microcontroller, 3rd ed. Prentice Hall.

Objectives:

1) Discuss the use of interrupts in microcontrollers.
2) Discuss key elements in interrupt programming, such as enables and interrupt vectors.
3) Discuss the differences between external, timer and counter interrupts.
4) Control 8051 registers for interrupt programming.

- Read Section J of Reference A and test applicable project files.
- Refer to Reference B as required for further information.

---

1. (1 pt) In general, what is the purpose of interrupts in microcontrollers?

2. (2 pts) What is the purpose of EA? What effect will it have if it is '0'?

3. (2 pts) What is the difference between a timer interrupt and a counter interrupt?

For questions 4 & 5: Using the VAB, PB2 will be used to interrupt the process light the LEDs D0-D7, but not affect the7-segment LEDs. The interrupt will process as long as the button is held down.

4. (6 pts) The following is an assortment of interrupt register bits. Place a 1 or 0 to set up the applicable interrupt for operation. Use an 'X' if the state does not matter for this operation. (Hint, you need to set/clear 3 bits).

EA _____ ES ____ ET1 ____ ET0 ____ EX1 _____ EX0 _____

IT0 _____ IT1 ____ IE0 ____ IE1 _____ TR0 _____ TF0 _____ TR1 _____ TF0 _____

(High Order) Gate ____ C/T ______ M1______ M0 _______

(Low Order) Gate ____ C/T ______ M1______ M0 _______

5. (1 pt) When the interrupt is called, the program counter will jump to memory location _________.

For questions 6 & 7: Using the VAB, we need to read the data on SW3 and place it on LED D10 every 100 µS with the 8051 clock running at 12MHz. Use Timer 0.

6. (7 pts) The following is an assortment of interrupt register bits. Place a 1 or 0 to set up the applicable interrupt for operation. Use an 'X' if the state does not matter for this operation. (Hint: You need to set/clear 7 bits.)

EA _____ ES ____ ET1 ____ ET0 ____ EX1 _____ EX0 _____

IT0 _____ IT1 ____ IE0 ____ IE1 _____ TR0 _____ TF0 _____ TR1 _____ TF0 _____

(High Order) Gate ____ C/T ______ M1______ M0 _______

(Low Order) Gate ____ C/T ______ M1______ M0 _______

7. (1 pt) The initial value of the timer registers in decimal would be: TH0 ___________, TL0 _____________.