Innovative Approach to Teaching an Undergraduate Microprocessor Class

Ceeyavash (Jeff) SALEHI
Assistant Professor of Engineering, Southern Utah University, 351 West Center Street, Cedar City, UT 84720, U.S.A., salehi@suu.edu, http://www.suu.edu/faculty/salehi/

KEYWORDS: *microprocessor, microcontroller, course, PICMicro®*

ABSTRACT: *This paper investigates the advantages of teaching PICMicro® microcontrollers over the traditional teaching approach using full-blown microprocessors. Traditionally, most schools have taught the undergraduate microprocessor class using full-blown microprocessors such as the Z80, 68000, 8051, and the 68HC11. Although these microprocessors have been widely used in microcomputers and microcontrollers over the years, they are no longer being used in many products that utilize microprocessors. Some examples of such products include automobiles, sensor systems, various home appliances, and toys. With the exception of computer engineering, the likelihood of an engineering or engineering technology graduate, programming full-blown microprocessors is declining rapidly as the designers and manufacturers of electronic and electromechanical devices continue to reduce design/production time, circuit size, and device cost by using PICMicro® microcontrollers. The Electronics Engineering Technology program at Southern Utah University has recently taught a one-semester Microprocessor course using PIC 16F84 and PIC 16F876. The knowledge of PICMicro® programming and interfacing provides the students with a powerful tool that they can use in their research and senior projects to develop devices and circuits that will in turn improve their marketability in the job market.*

1 INTRODUCTION

The majority of the traditional engineering programs have been teaching their undergraduate Microprocessor Fundamentals course using full-blown microprocessors such as the Z80, 68000, 8051, or the 68HC11. As an example, the Z-80 microprocessor utilizes approximately 600 instructions [1]. To function as a microcontroller or a simple dedicated microcomputer, the Z-80 requires multiple ICs such as ROM, RAM, and serial I/O port. To build a microcontroller using a full-blown microprocessor, in effect, one has to build a simple mother-board.

PICMicro® ICs have a wide range of applications such as high-speed automotive and home appliance motor control, low-power remote transducer/sensor systems, electronic locks, and security devices. There are multiple advantages in using PICMicro® ICs as application-specific microcontrollers or dedicated microcomputers. PIC microcontrollers can offer a 4:1 operating speed improvement at 20 MHz and a 2:1 code compression compared to other typical 8-bit microcontrollers [2].

The PIC 16F84 IC is an 8-bit RISC (Reduced Instruction Set Computer) microcontroller with a total of 35 single-word instructions, the majority of which require one clock cycle for execution except for program branches which require two cycles [2]. The RISC feature reduces programming time and programming complexity. The 16F84 has two sets of I/O ports; five portA pins and eight portB pins a total of thirteen I/O pins, which can be configured to function as inputs or outputs. Other features of the 16F84 IC are onboard memory including 1 kWord of Flash program memory with a limit of 1000 erase/write cycles, 68 bytes of static data RAM, and 64 bytes of EEPROM data memory with a limit of 10,000,000 erase/write cycles [2].

PIC 16F876 has twenty two I/O pins, an onboard 10-bit Successive Approximation Register (SAR) A/D converter, and compared to the 16F84 microcontroller, it has enhanced memory size [3].

The PICMicro® 16F8X series have multiple internal and external interrupt sources and use 14-bit wide single-word instructions by utilizing the Harvard architecture. In the Harvard Architecture, opcode and operand are accessed from two separate memory locations via a command or opcode memory bus and a data or operand memory bus. In contrast to other microprocessors that utilize the traditional Von

Neumann Architecture where opcode and operand are fetched form the same memory location one at a time, the Harvard Architecture used in PIC ICs improves bandwidth [2].

With the exception of the field of computer engineering, the likelihood of an engineering or engineering technology graduate, programming full-blown microprocessors is decreasing rapidly as the manufacturers of electronic and electromechanical devices continue to reduce design/production time, circuit size, and device cost by using PICMicro® RISC microcontrollers and PIC-based microcontrollers such as BASIC Stamp®. This may suggest that the traditional approach to teaching an undergraduate microprocessor class can be modified or altered to better serve the needs of future engineering and technology graduates.

The knowledge of PICMicro® IC programming and interfacing provides the students with a powerful tool that they can use in their research or senior projects to design and develop practical devices and circuits that will in turn improve their marketability in the dynamic industrial and manufacturing environment.

Two different approaches can be considered in order to provide the students with a working knowledge of PICMicro® RISC microcontrollers. In a four credit hour class, approximately half or one-third of the semester can be used to introduce PIC ICs and the remaining time can be used to study a full-blown processor. Another approach is to use the entire semester to study PIC microcontrollers and also to study the fundamental principles that apply to all microprocessors. The Electronics Engineering Technology program at Southern Utah University has recently taught a one-semester three credit hour Microprocessors class using PIC 16F84 and PIC 16F876.

The objective of this paper is to discuss and to investigate an undergraduate microprocessor course based on the PICMicro® 16F84 and 16F876 single-chip microcontrollers.

2 METHODS

The first few lectures of the course were used to obtain a general understanding of the subunits inside a microprocessor such as the CPU, ALU, CU, and the MU and how these different subunits interact with each other. The 16F84 IC was then introduced and additional microprocessor concepts such as memory types, working and special-function registers, and assembly language programming were covered as the 16F84 operation was investigated further. After the completion of the introductory concepts and the study of the basic characteristics/features of PIC 16F84, the students were assigned a simple laboratory experiment/project. This experiment provided the first hands-on experience in PIC assembly language programming and wiring a simple PIC-controlled circuit.

The students were instructed to work in teams of two and to write a program to configure one of the I/O ports to function as an output that continuously turns an LED on for 0.5 seconds and off for 0.5 seconds. The students were provided with a 20-MHz ceramic resonator for the external clock, a 16F84 IC, and other necessary components. The main objectives of this first project were to use the MPLAB® assembler software, the programmer interface module, and a personal computer to compile and to upload an assembly language program into the 16F84 IC.

Furthermore, the students were instructed to refer to the IC data sheet to determine the internal clock frequency and the execution time for each instruction or command. They were also required to write flow-charts for the main program and any subroutine(s). The students then determined the number of instructions and the structure of the nested loop necessary to generate the 0.5 second on/off delay. The information shown below and the inner-loop instructions in Table 1 illustrate how the 0.5-second time delay was generated using a delay subroutine, called multiple times, from the main body of the program.

Ceramic resonator frequency is 20 MHz; this external clock frequency is divided by 4 within the PIC IC. The internal clock frequency is 20 MHz divided by 4 which is 5 MHz; the period for 1 clock cycle is the reciprocal of 5 MHz which is 200 nsec.

Table 1. Time-delay Instructions

INSTRUCTION	EXECUTION TIME
NOP	200 nsec
NOP	200 nsec
NOP	200 nsec
NOP	200 nsec

NOP	200 nsec
NOP	200 nsec
NOP	200 nsec
DECFSZ NUM_1, F	200 nsec
GOTO INNER_L	400 nsec
TOTAL DELAY	**2000 nsec or 2 μsec**

The inner loop runs 250 times; the outer loop also runs 250 times. Therefore, a delay of (250*250*2 μsec) which is 0.125 sec is generated by the Delay Subroutine. The delay routine is called four times (4*0.125 sec) from MAIN to generate 0.5 sec of delay.

The sample program shown below illustrates how the I/O configuration and the timing objectives were accomplished at the software level.

```
; code to output a TTL signal with a period of 1 second from the RB0 output

;*****************************SETUP & CINFIGURATION*******************************

        LIST P=16F84A
        #include <P16F84A.INC>

        __CONFIG _HS_OSC & _WDT_OFF & _PWRTE_ON

;*****************DEFINE VARIABLES & CHANGE PERT B0 TO AN OUTPUT***************

NUM_1 EQU h'0C'                 ;name a memory location NUM_1 for the inner loop counter
NUM_2 EQU h'0D'                 ;name a memory location NUM_2 for the outer loop counter


        BSF STATUS, RP0                 ;switch to Bank 1
        BCF TRISB, 0                    ;set bit to zero to change port to an output
        BCF STATUS, RP0                 ;switch to Bank 0

;************************************ MAIN ***************************************
;main program that sets B0 to high then calls delay routine & then to low and calls the delay routines again

MAIN:
                BSF PORTB, 0            ;make PORTB0 go high
                CALL DELAY              ;call the delay subroutine
                CALL DELAY              ;call the delay subroutine
                CALL DELAY              ;call the delay subroutine
                CALL DELAY              ;call the delay subroutine
                BCF PORTB, 0            ;make PORTB0 go low
                CALL DELAY              ;call the delay subroutine
                CALL DELAY              ;call the delay subroutine
                CALL DELAY              ;call the delay subroutine
                CALL DELAY              ;call the delay subroutine

GOTO MAIN                               ;go back to MAIN

;*******************************DELAY SUBROUTINE*********************************
;this is a routine that produces a 0.5 sec delay when it is called 4 times in MAIN

DELAY:
                        MOVLW d'250'    ;move decimal 255 to register W
                        MOVWF NUM_2     ;move content of W to NUM_2

OUTER_L:        MOVWF NUM_1             ;move content of W to NUM_1
                                        ;start of outer loop
```

```
;$$$$$$$$$$$$$$$$$$$$$$$$$$ INNER-LOOP $$$$$$$$$$$$$$$$$$$$$$$$$$

INNER_L:  NOP                    ;kill 200 nsec
          NOP                    ;kill 200 nsec
          NOP                    ;kill 200 nsec
          NOP                    ;kill 200 nsec
          NOP                    ;kill 200 nsec
          NOP                    ;kill 200 nsec
          NOP                    ;kill 200 nsec

          DECFSZ  NUM_1, F       ;decrement file & leave the number in NUM_1
                                 ;skip next line of code if file is zero

          GOTO INNER_L           ;repeat inner loop if NUM_1 is not zero
          DECFSZ NUM_2,F         ;decrement file & leave number in NUM_2
          GOTO OUTER_L           ;skip next line of code if file is zero
          RETURN                 ;return to MAIN

;$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$

END
```

All groups successfully up-loaded the assembly language program into the microcontroller, wired the circuit, and using an oscilloscope, verified the desired output by observing the LED turn on and off by a TTL signal at a frequency of 1 Hz. The exact on/off time intervals were measured to insure proper program execution and circuit operation.

PICMicro® microcontrollers have a number of features and internal sub-circuits that help reduce cost by eliminating external components, increase system reliability, provide low-power operation, and code protection [2]. Examples of such features include start-up and power-up timers which eliminate the need for external reset circuitry, SLEEP-mode operation for power conservation, and wake-up from SLEEP-mode via an external reset or through an interrupt [2]. Another important feature of PIC is the readable and writable EEPROM data memory which is indirectly-addressed through the special-function registers.

Four additional projects were completed using the 16F84; each new project was more challenging and more complex than the previous one. The second project involved the use of nine I/O ports; one portA input and eight portB outputs. A program was written to turn on one LED out of 8 LEDs connected to portB pins and to continuously shift the position of the on-LED from right to left or left to right depending on the status of a SPST decision-switch connected to one of the portA pins configured as an input. The LEDs were to be on for 50 msec and off for 60 msec.

The third project implemented software switch-debouncing, decision-making, tone generation, and data-write to Flash memory. The programs for the more challenging projects are not included in this paper due to the fact that they are very long.

The students were then instructed to write a short EEPROM write/read program in order to become familiar with the process of indirect addressing and EEPROM data storage. The following sample program illustrates the EEPROM write/read operation.

```
;simple program to write to and read from EEPROM
;
;*********************************SETUP & CINFIGURATION*****************************

        LIST P=16F84A
        #include <P16F84A.INC>
        __CONFIG _HS_OSC & _WDT_OFF & _PWRTE_ON

;**********************DEFINE VARIABLES & SET UP PORTS*****************************

NUM_1 EQU h'0C'                  ;name RAM location 0C, NUM_1 for delay inner loop counter
NUM_2 EQU h'0D'                  ;name RAM location 0D, NUM_2 for delay outer loop counter
```

```
NUM_3 EQU h'0F'                 ;name RAM location 0F, NUM_3 for beep counter
READ EQU h'0E'                  ;name RAM location 0E, READ for EEPROM reading

        BSF STATUS, RP0         ;switch to Bank 1
        BCF TRISB, 0            ;clear TRISB bit0 to change portB0 to output
        BCF TRISB, 1            ;clear TRISB bit1 to change portB1 to output
        BCF TRISB, 2            ;clear TRISB bit2 to change portB2 to output
        BCF TRISB, 3            ;clear TRISB bit3 to change portB3 to output
        BSF TRISB, 4            ;set TRISB bit4 to change portB4 to input
        BSF TRISB, 5            ;set TRISB bit5 to change portB5 to input
        BSF TRISB, 6            ;set TRISB bit6 to change portB6 to input
        BSF TRISB, 7            ;set TRISB bit7 to change portB7 to input
        CLRF TRISA              ;clear TRISA bits to change portA bits to outputs
        BCF STATUS, RP0         ;switch to Bank 0
        CLRF PORTB                      ;clear all bits for portB
        CLRF PORTA                      ;clear all bits for portA

;*********************************************************************************

MOVLW b'0011'           ;put data in W
MOVWF EEDATA            ;put content of W in EEDATA
MOVLW h'0001'           ;put EEPROM address in W
MOVWF EEADR             ;put content of W in EEADR
BSF STATUS, RP0         ;switch to Bank1
BCF INTCON, GIE         ;disable global interrupt
BSF EECON1, WREN        ;enable EEPROM for writing
MOVLW h'55'             ;initialize/activate data write to EEPROM
MOVWF EECON2            ;initialize/activate data write to EEPROM
MOVLW h'AA'             ;initialize/activate data write to EEPROM
MOVWF EECON2            ;initialize/activate data write to EEPROM
BSF EECON1, WR          ;write data to EEPEOM
BSF INTCON, GIE         ;enable global interrupt
CLRF EEDATA             ;clear EEDATA register
BSF EECON1, RD          ;read data from EEPROM
BCF STATUS, RP0         ;change to BANK0
MOVF EEDATA, 0          ;move data from EEDATA to W
MOVWF PORTA             ;send data from W to PORTA

END
```

The objectives of the fifth 16F84 project were to implement SLEEP-mode and interrupt for power conservation as well as EEPROM data-write and data-read. This project involved scanning, storing, and playback of pressed keys on a 4 by 4 matrix keypad which can be used as a part of a digital combination lock system.

The remaining 4 weeks of the semester were used to conduct two more laboratory experiments/ projects using the 16F876 IC with the onboard 10-bit SAR A/D converter. To become familiar

with the operation of the SAR and the concept of A/D converter resolution, a simple experiment was conducted by using the 16F876 to convert and display a 0 to 5 V analog voltage, generated

by a potentiometer, to a 10-bit straight-binary output. A sample program is shown below.

```
;This program allows the 16F876 to convert an analog voltage applied to portA0 to a 10-bit
;digital output which is then displayed using LEDs connected to portC and protB

;*****************************  SETUP & CINFIGURATION  ************************************

LIST P=16F876
#include <P16F876.INC>
```

```
__CONFIG _HS_OSC & _WDT_OFF & _PWRTE_ON & _WRT_ENABLE_OFF & _BODEN_OFF & _LVP_OFF &
_DEBUG_OFF
ORG 0X00

;****************************** SET UP I/O & DIGITAL INPUT ******************************

BSF STATUS, RP0          ;go to bank1
BCF STATUS, RP1
BSF TRISA, 0             ;set portA0 to be an input
BCF ADCON1, PCFG0        ;this opcode and the next three set portA0 to be an analog input
BSF ADCON1, PCFG1        ;and the rest digital, they also set the reference voltage to use
BSF ADCON1, PCFG2        ;Vdd and ground
BSF ADCON1, PCFG3
BSF ADCON1, ADFM         ;right-justify the digitized data
CLRF TRISB               ;clear all portB's to be outputs
CLRF TRISC               ;clear all portC's to be outputs

BCF STATUS, RP0          ;go to bank0
BCF STATUS, RP1
BSF ADCON0, ADCS1        ;set A/D clock to FOSC/32 for 20 MHz
BCF ADCON0, ADCS0
BCF ADCON0, CHS0         ;select A/D channel 0 (portA0)
BCF ADCON0, CHS1
BCF ADCON0, CHS2
BSF ADCON0, ADON         ;turn A/D converter on

;^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ A/D CONVERSION ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

MAIN:
        BSF ADCON0, GO              ;start A/D conversion
        TEST:
                BTFSC ADCON0, GO    ;test to see if A/D conversion is done
                GOTO TEST           ;if not, keep testing

        MOVFW ADRESH                ;get the two higher-bit digitized data
        MOVWF PORTC                 ;send them to portC for display
        BSF STATUS, RP0             ;go to bank1
        BCF STATUS, RP1
        MOVFW ADRESL                ;get the eight lower-bit digitized data
        BCF STATUS, RP0             ;go back to bank0
        BCF STATUS, RP1
        MOVWF PORTB                 ;send them to portB for display
        GOTO MAIN                   ;continue A/D conversion
END
```

A final project was assigned with the objective of using the 16F876 microcontroller in a real-life application. A simple digital thermometer circuit was constructed using an LM34 Precision Fahrenheit Temperature Sensor connected to one of the PIC I/O pins configured as an analog input. The temperature-dependent voltage, produced by the LM34 Sensor, was digitized and displayed in increments of 1 degree Fahrenheit, in decimal, by time-multiplexing the output bits and using two seven-segment displays. The conversion from straight-binary to 7-segment was accomplished within the assembly language program; therefore, eliminating the need for BCD to 7-segment decoder/driver ICs.

3 CONCLUSION

The completion of the one-semester PIC-based course demonstrated that microprocessor concepts and fundamentals as well as hands-on assembly language programming and microprocessor circuit bread-boarding can be taught in a course almost entirely dedicated to PICMicro® microcontrollers. Although the PIC based microprocessor course was taught for the first time during the fall semester of 2003, the outcomes were more than satisfactory. Some of the students who completed the class, are currently

utilizing their knowledge of PICs in their other classes and some are considering using PICs for their senior projects.

It was observed that one of the assigned projects required far more time than anticipated. Future microprocessor course contents can be refined and project objectives can be re-organized in order to optimize course outcomes as well as the use of the available time.

AKNOWLEGEMENTS

The author gratefully acknowledges the support provided by the Provost's Office at Southern Utah University (SUU) and also the support offered by the SUU faculty and staff in obtaining a grant to attend the iNEER-2004 conference.

REFERENCES

[1] Compiled by WENZLAFF, R. *Z80 8-bit Microprocessor*. Seattle, Washington, U.S.A.: University of Washington Electrical Engineering Department, Online Circuits Archive, last update April 30, 2003. Available from web: <URL:http://www.ee.washington.edu/circuit_archive/micro/z80.html>

[2] MICROCHIP®. *PIC16F8X 28/40/44-pin Enhanced Flash Microcontrollers Data Sheet*. Chandler, Arizona, U.S.A.: Microchip Technology Inc., 1998

[3] MICROCHIP®. *PIC16F87XA 18-pin Flash/EEPROM 8-Bit Microcontrollers Data Sheet*. Chandler, Arizona, U.S.A.: Microchip Technology Inc., 2003