**LABORATORY #4**

**USART and Numerical Coprocessor Programming**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Description | Student #1 | Student #2 | T.A. Initials | Date |
| Student Name |  |  |  |  |
| Student Number |  |  |  |  |
| Prelab/ Attendance |  |  |  |  |
| Lab #4 Completion |  |  |  |  |

# HOME PREPARATION:

1. Read section 2-11 of the SDK User’s Guide on the serial interface. Complete the programming for part A. The 8251A programming can be completed by looking in your text for the bit patterns. Appendix B of SDK User’s Guide has useful information on the 8251A.
2. The opcodes for the 8087 programming can be found on page 2-140 to 2-143 of the 1990 Microprocessor Manual. Read Chapter 5 in the 8086/8088 User’s Manual on 8087 operation.

# LAB #4: INSTRUCTIONS

Fill in the blanks on these question sheets (ONE per group). These sheets constitute your lab report. The T.A. will ask you questions about your work and you will have to demonstrate your work to the T.A. *The programs must be well documented.*

# PART A: PROGRAMMING WITH THE 8251 USART CHIP

The 8251A chip is a Universal Synchronous/Asynchronous Receiver Transmitter (USART) and is used for serial communications. On the SDK-86 board, it is used to communicate with the host PC. The use of this chip in the labs has been transparent to you this far since it is programmed by the serial monitor program in the SDK-86 ROM (i.e. by the program at ROM address FE00:0000).

In this part of the lab you are given an SDK-86 program (**LAB4A.ASM**) that receives characters typed in on the SDK-86 keyboard and transmits them to the PC via the USART.

You are required to modify this program to allow it to receive characters from the PC and display them on the SDK-86 display (in addition to what the program does now).

Note that the PC also uses a USART for serial communication and this chip is normally programmed by ASCII terminal emulator applications. The SDK program on the PC is an example of an emulation application, and will be used to receive and send characters from and to your SDK-86 program.

1. Complete the subroutine **PC2SDK** which will receive the characters from the PC and display them on the LED display of the SDK-86 board. Characters will be received from the PC when you type on the PC keyboard while the SDK program is running (you must use the “G” command from the PC keyboard to start your program on the SDK-86 board – *do not use the SDK-86 keypad*).

Note that the characters received are coded in ASCII format, and so must be translated to the 7 segment format for display. Only the characters 0 through 9 and A through F must be translated, although all others should be caught and translated to a period (‘.’).

Note that the characters received from the PC will be in upper case only since the SDK program converts characters typed on the PC keyboard to upper case.

Your **PC2SDKB** subroutine should follow the pseudo-code provided in the subroutine header. This can easily be accomplished with around 30 lines of code. The program (LAB4A.ASM) already contains the translation tables – you just have to make use of them.

The ASCII translation may be accomplished by subtracting the character zero or A as appropriate, then translating to the 7 segment code; for example:

 SUB AL, ‘0’ ;AL CONTAINS AN ASCII FORMAT NUMBER FROM 0 TO 9

 XLAT LED\_TABLE ;BX CONTAINS LED\_TABLE OFFSET (PUTS 7-SEG CODE ;IN AL)

To display the characters on the LED, you must program the LED as you have seen in previous labs. The characters should appear on the display as incrementing digits (i.e. typing ‘1 2 3’ should result in a display of ‘3 2 1’). Review the 8279 specifications to determine how this is done. Explain how you program the 8279 to accomplish this (give bit patterns):

 BIT PATTERN NOTES

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |

KBDPRG = ­­­­­\_\_\_\_\_\_\_\_\_H \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |

DISPCLR = ­­­­­\_\_\_\_\_\_\_\_\_H \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |

 DISPINI = ­­­­­\_\_\_\_\_\_\_\_\_\_H \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Note that the USART chip *required a delay between successive writes or reads.* This is due to a manufacturing design oversight and a thus a delay is required for proper chip operation. The **DELAY** subroutine should be called after each access to provide this delay.

1. Modify the SETUP subroutine to program the 8251A properly for 2-way communication by setting the proper equate values for the MODE and COMMAND bytes. The requirements are given here, and you must show the bits selected.

MODE: 4800 baud (64x), 8 data bits, no parity, 1 stop bit.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |

= \_\_\_\_\_\_\_H

Why does 64x give 4800 baud? (Refer to Figure 2.2 and Table 2-10 of the SDK User’s Guide.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

COMMAND: transmit enable, data terminal ready, receive enable, break character normal operation, no error reset, request to send, no internal reset, no hunt mode.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |

= \_\_\_\_\_\_\_H

Note that the **SETUP** subroutine resets the USART by writing 00H to the control register address 0FFF2H three times followed by a 40H. Then the chip is programmed with the MODE and COMMAND words. Note that the **DELAY** subroutine is called *after* *each access*. The chip status is available via the control register and data is sent to/received from the chip via the data register (address 0FFF0H).

1. Run your program to make sure it works.
2. When you get your program working, you will notice that once in a while an unexpected character is written on the PC screen when you are hitting keys on the SDK keypad. *Hitting two or more keys simultaneously can cause repeating characters*. When this happens, an error flag in the 8270 FIFO status word is set which causes the program to repeat characters because it thinks there are characters in the buffer. *Fix this bug* by masking the error flags out before checking the FIFO status word. *You must demonstrate your well documented program by the end of the lab period*.

T.A. Initials: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# PART B: FLOATING POINT EMULATION

1. Compile and run the LAB4B.ASM program on the PC following these steps:
2. MASM LAB4B/r
3. LINK LAB4B
4. LAB4B

The first step assembles the program, where the ‘/r’ indicates the 8087 instructions are to be recognized. The second step performs a standard link. The third step runs the program under DOS.

This program provided a floating point multiplication routine that emulates part of the operation of the 8087 coprocessor. The routine multiplies the two 80-bit floating-point numbers that are stored in IEEE 80-bit format (see page 5-18 of the 8086/8088 User’s Manual). The result is returned in 80-bit format as well, and is printed out on the screen.

The algorithm used essentially treats the fraction part (64-bits) as four 16-bit words, multiplying each word in the multiplicand by each word in the multiplier (each resulting in a two-word result), while maintaining the positioning of each intermediate result for later addition to form the final result. See the comments to procedure FPMULT in the code. The exponent is simply an addition of the two input exponents (with some adjustment for normalization).

1. The numbers being multiplied are 178.125 and -6.0. The 80-bit coding for 178.125 is described on page 5-19 of the 8086/8088 User’s Manual. The result of the multiplication is displayed (in 80-bit format) on the PC screen after you it by typing LAB4C at the dos prompt. Write the result in the format (± x.xx…) \* 2^e where x’s are bits.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. The program calculates the result of this multiplication a number of times using both the 8087 and the emulator. Estimate the execution time for a single loop (by dividing the total time taken by the number of repetitions) for both methods. Measure it with your watch to the closest second and fill in the table below.

TABLE 1: 8087 vs. Emulation loop times

|  |  |  |
| --- | --- | --- |
|  | 8087 | Emulation Program |
| Total Time |  |  |
| Number of loops |  |  |
| Time per loop |  |  |

1. Compare these results with the values in table 5-1 on page 5-4 of the 8086/8088 User’s Manual. Give several reasons why we are not seeing a speed improvement factor on the order of approximately 85 times (as given in the table) when using an 8087? (Hint: Compare the FPMULT code with the description of 8087 characteristics in the 8086/8088 User’s Manual). \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. Table 5-1 in the 8086/8088 User’s Manual gives times according to the use of an 8087. The 8087 is approximately 85 times faster than emulation using an 8086. How does the use of a 486 chip inside the PC, which has a built-in coprocessor, change the times? \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_