Systems and Computer Engineering

Lab Manual

94.361

Microprocessor Systems

Fall, 2009

Table of Contents

Laboratory Marking Scheme

1. Labs are marked out of 10.
2. Prelabs consitute half (i.e. 5 marks) out of the total lab mark of 10.
3. Partially finished prelabs are penalized 2.5 marks which means that the maximum mark obtainable for that lab is 7.5/10.
4. A student showing up for a lab without the prelab done receives 0/5 for the prelab and can therefore only obtain a maximum of 5/10 for that particular lab.
5. Prelabs are marked individually. Both students in a group are expected to do the prelab to obtain full marks.
6. Late labs can be finished in subsequent weeks. Labs are penalized 2 marks per week when finished more than one week after the scheduled lab period.
7. Students can obtain at least 5/10 for a lab by just turning up at the scheduled lab period and attempting to complete the lab, even if the lab is not completed, and the prelab is not done.

LABORATORY #1

Familiarization with the SDK-86 Keypad Monitor, Downloading and the 8086 Instruction Set

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

HOME PREPARATION:

1. Read the SDK User's Guide (one of the two white books), chapters 3 and 4. Chapter 3 is summarized in Tables 3-1, 3-2, 3-5. Chapter 4 is summarized in Table 4-1. The system consists of an external SDK-86 board connected to an IBM PC. Chapter 3 describes the SDK-86 keyboard monitor. Chapter 4 describes the serial monitor which performs the same type of commands as the keyboard monitor, except it allows you to execute them remotely from the IBM PC keyboard.
2. Bring in a blank 3.5" floppy diskette for storing your programs.
3. This lab has four parts. Prepare the code for parts B and D.

LAB #1: 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. *You will have to demo your work to the T.A.*

PART A: SDK-86 AND KEYPAD MONITOR COMMANDS:

1. Page 1-1 of the SDK-86 User's Guide indicates that the systems can have either 2K or 4K of RAM. By examining the SDK-86 printed circuit board, how much RAM is in the systems in the lab ? _____K
2. What addresses does the RAM memory occupy?

   _____

   _____

3. Page 1-1 also indicates the clock frequency is 2.5 MHz or 5 MHz (jumper selectable). From page 2-11 of the SDK-86 User's Guide (or Figure 2.2 in the SDK System Design Guide) the jumper settings are either W40 or W41. What is the correct jumper setting for 2.5 MHz operation ? _____

Find the jumper on the SDK-86 board and make sure it is set for 2.5 MHz operation.

Familiarization With Keypad Instructions

For the following exercise, refer to the examples on pages 3-8, 3-9 and 3-12 of the SDK-86 User's Guide.

1. Reset the system (press SYSTEM RESET key). **NOTE:** *Hitting reset will not clear the main memory. Only the registers are reset.*
2. Examine bytes FE00:5 to FE00:9 using the SDK command EB FE00:5 , , , , , (EB is a single button "examine byte" command. pp. 3-8)

   Contents:

   FE00:5 _____H FE00:6 _____H FE00:7 _____H FE00:8 _____H FE00:9 _____H

   What physical address is represented by FE00:0 ? _____H

3. Give an EB command that accesses the same addresses as the previous one but uses different segment and offset values:

   _____

   _____

4. Move a block of data, 6 bytes long, starting from location FE000H to a new area starting at location 300H (see example on pp. 3-21).

   What is the word at location 00300H:? _____H. Use the EW command (pp. 3-9) to change its contents to ABCD H. Examine it again to confirm the change. Give the 6 bytes starting from 00300H:

   _____

   _____

5. Do a SYSTEM RESET. What are the contents of CS ? _____H

   What is the absolute address of CS:29H ? _____H

   Examine the DS register (pp. 3-12); give contents _____H

   Change its contents to 30 H and examine it again to confirm the change: _____H

6. Examine all registers starting from AX using the command ER AX , , , , , (until all shown) and write down the contents:

_____

_____

_____

PART B: 8086 OBJECT CODE:

1. In the program shown on the next page, hand assemble the two instructions whose object code is not shown. Show how each bit of the hand assembled instructions was determined.
2. Key in the program as bytes starting from location 100H. If you enter them by words, you may get the order of the bytes reversed.
3. Modify the words at location 150H to 3H and 152H to 9H.
4. Execute the program by keying the GO key with a break point set at 114H. (See pp.3-17 and 3-18 for the syntax).
5. Examine the words at the following locations: 150H _____ 152H _____ 154H _____ 160H _____

```
NAME TEST
;**
;**94.461 LAB 1 Part B.
;**
;** A simple test program that enters an infinite loop.
;**
PROG SEGMENT
ORG 00100H
ASSUME CS:PROG,DS:PROG

START:
                MOV AX,CS           8C C8
                MOV DS,AX           ________________
LOOP:           MOV BX,150H         BB 50 01
                MOV AX,[BX]         8B 07
                ADD AX,[BX+2]       ________________
                MOV [BX+4],AX       89 47 04
                MOV BX,160H         BB 60 01
                MOV [BX],AX         89 07
                JMP LOOP            EB EE
PROG            ENDS
                END                 START
```

PART C: DOWNLOADING FROM THE IBM PC TO THE SDK BOARD

1. Each PC has a sub-directory called 361 (or 461) that contains a number of files. Change to directory 461.

   NOTE: *Take copies of the two files* LAB1B.ASM *and* LAB1D.ASM *and put them on your floppy disk. Included in these programs are valuable comments about how to use the* MASM *assembler. (valuble in other labs)*

2. Copy these files onto your floppy disk and then from drive A:\ (or B:\). Assemble the program by typing:MASM LAB1B. It produces LAB1B.OBJ

3. Link the program by typing: LINK LAB1B. It produces LAB1B.EXE.

4. Convert the executable code to hexadecimal format for downloading by typing: EXE2HEX LAB1B.EXE LAB1B.HEX

NOTE: *There is a batch program called* ASSEM *that will do steps 2 through 4 for you. For example, type* ASSEM LAB1B. *It should be noted though that in case of an error, it is wise to execute* MASM *and specify a .LST file name which can be examined for errors.*

5. Reset the SDK board and execute the SDK-86 command: GO FE00:0 . (don't forget the period at the end). The SDK will display **86 1.2** to show that the serial monitor is operating.

6. Execute the PC command:SDK

7. The prompt for the SDK program is a dot. To download the program enter the PC command: L LAB1B.HEX

8. You can run the 8086 program either from the PC keyboard or directly on the SDK keyboard. To run it from the PC enter the command '**G**' and hit return. If you check the program you will see that it is in an infinite loop and therefore nothing will show on the screen or keypad monitor.

9. To stop it, press the SDK-86 SYSTEM RESET and repeat commands 5 and 7. **NOTE:** *Hitting reset will not clear the main memory. Only the registers are reset.*

10. To run the program from the SDK-86, press the SDK-86 SYSTEM RESET followed by: GO 100,114. (don't forget the dot)

    This results in a breakpoint shown on the SDK display as: -br

11. Examine the words (use "EW") at the following locations:

    150H _____ 152H _____ 154H _____ 160H _____

PART D: FAMILIARIZATION WITH THE 8086 INSTRUCTION SET

In the 361 (or 461) directory is a file called LAB1D.ASM It is an 8086 assembler program that simulates a clock. It displays hours, minutes and seconds on the SDK-86 LED display.

1. Browse through the code. You will find that the subroutine to wait one second (WAIT) only contains a return statement.

2. Finish the WAIT subroutine by writing the code to delay one second. Ignore the times taken by other subroutines such as UPDATE. We just want you to write the code for a one second delay.

   -Remember that you set the clock frequency to 2.5 MHz.

   -Make use of Table 3-14 (p 3-57) in the 8086/8088 User's Manual as well as the number of clock cycles taken by each instruction.

3. Write the code for the WAIT subroutine before you come to the lab. For this first lab, it is not required to understand the rest of LAB1D.ASM However, you will have to understand the concepts of the software-hardware interface later in the course. It can't hurt to browse through it now.

4. During the LAB, add your code for the WAIT routine to LAB1D.ASM and assemble/link/download it to the SDK. Run it and demonstrate it to the T.A.

The following code is part of LAB1D.ASM

```
;** WAIT ** (Please write code below in space provided).
;**
;** Contains loops which run for approximately one second (within 25%).
;** Does not take into account the run time of the other procedures
;** in the CLOCK program (they are negligible with respect to the
;** processor speed variance).

WAIT            PROC NEAR




                RET

WAIT            ENDP
```

Final Remarks

Do not leave any of your files on the hard disk. Copy all your files to your floppy disk , since all files on the hard disk will be lost when you logout.

LABORATORY #2

Signal Waveforms and Parallel Port Programming

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

HOME PREPARATION:

1. Review the 8086 Hardware Reference Manual to familiarize yourself with the CPU bus signalling waveforms. Note that the 8086 on the SDK board is set up in Minimum Mode. Review the waveform diagram provided in Table 2.

   Review: Appendix A, logic analyzer operation

   Appendix B, in particular the 8255A and DAC extensions.

2. Calculate the number of clock cycles expected in the program of PART A.
3. Write the program required in PART B. Ensure that it is well documented.
4. As in LAB #1, and for all following labs, bring in a 3.5" diskette.

LAB #2: INSTRUCTIONS:

Fill in the blanks on these question sheets, (one per group) . These sheets constitute your lab report. The T.A. will ask questions about your work and you will have to demonstrate your work to the T.A.

PART A: BUS SIGNALS AND WAVEFORMS

In this part of the lab you will us a logic analyzer to monitor the bus signals on the CPU as it executes a small program. You must understand the relationship between the program and what appears on the bus. See APPENDIX A for settings and use of the logic analyzer.

1. Find the 50 pin male connector on the SDK board labelled 'DATA BUS' and plug the logic analyzer pod into it. **NOTE:** *The logic analyzer should be powered down while inserting and removing the pod into its front panel.*

2. Assemble, link and convert to executable code the LAB2A.ASM file provided in the 361 (or 461) directory, then download this and run it on the SDK board (follow the downloading instructions provided in LAB #1, PART C).
3. Complete the timing diagram in Table 2 by copying the waveforms as observed on the logic analyzer.
4. Determine what the processor is doing at each clock cycle by looking at the address and data values on your timing diagram. The bus has either a 20 bit address or 16-bit data on it at one time, not both. Determine which one is on the bus and write down the events which are occuring and what each one means in Table 1 below. The first entry is given for you.

TABLE 1. Timing Diagram Event Description

Clock Cycle	Event (Read/ Write/Latch)	Address or Data Value	Explanation
1-3	Write data	A500	Write A5H from AL into location MEM
4			
5-7			
8-13			
14			
15-17			
18			
19-21			
22			
23-25			
26-27			
28			
29-30			

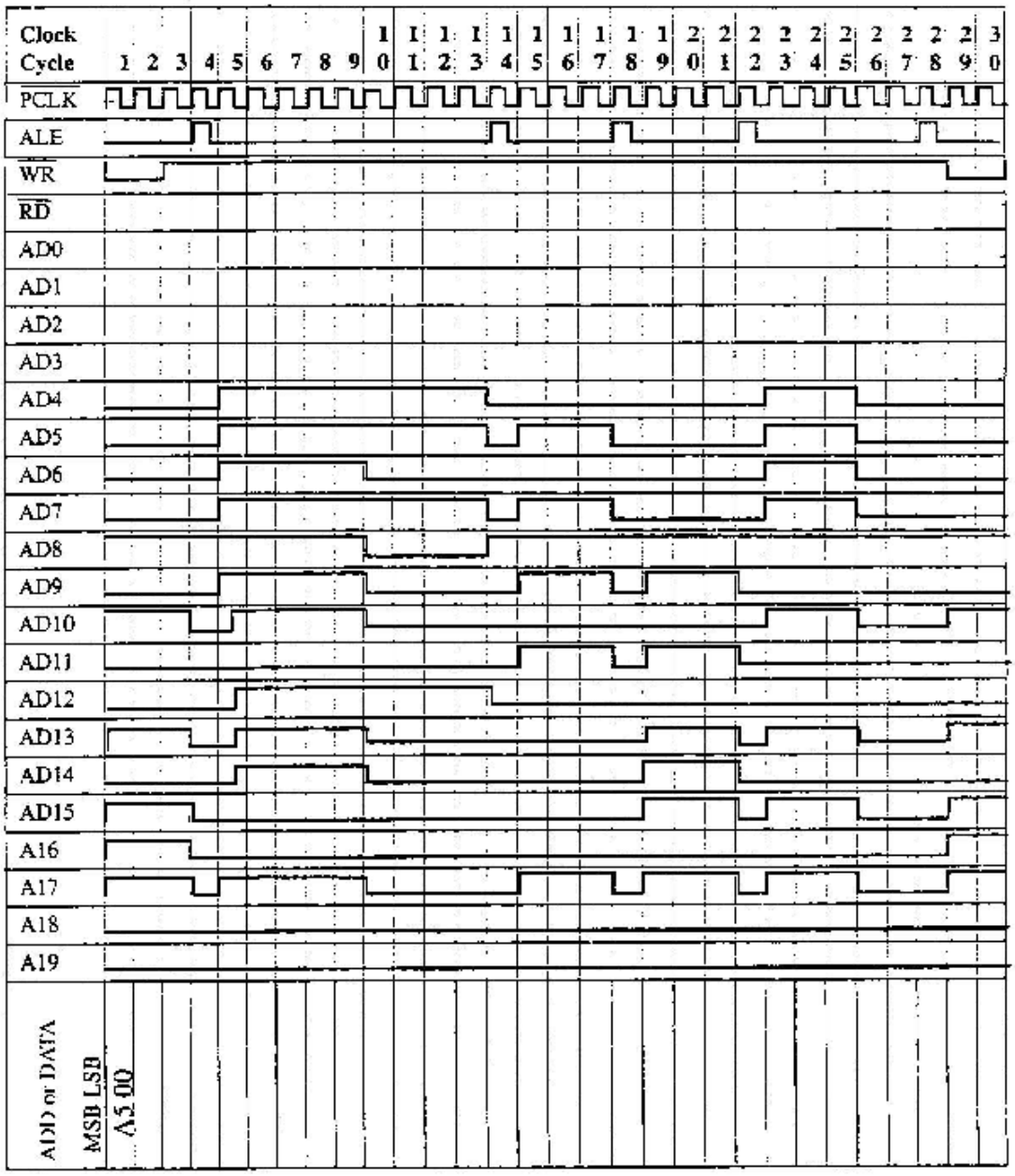
5. Explain why some of the address/data lines rise or fall in cycle 10?

6. Over which clock cycles is the MOV MEM, AL instruction executed (not fetched)?

7. Look at the program LAB2A.ASM and refer to the 8086/8088 User's Manual. How many clock cycles should this take? (Hint: Examine the relationship between the JMP instruction and the prefetch queue).

8. Why is the number of clock cycles observed different from the number calculated above?

TABLE 2. Timing Diagram



Program fragment from LAB2A.ASM

(Loop occurs at 00106, AL=A5H)

```
LOOP:   MOV     MEM, AL
JMP     LOOP
MEM     DB      00H, 0F0H, 77H
```

PART B: PARALLEL PORT PROGRAMMING AND THE D/A CONVERTER

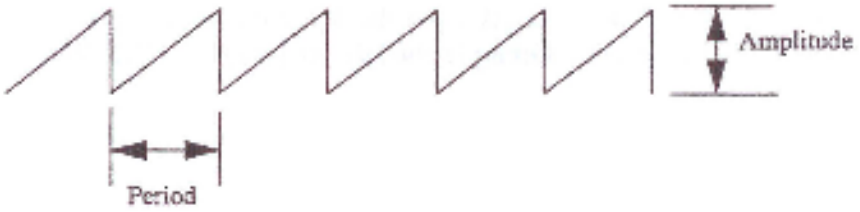
In this part of the lab you will program the 8255A parallel I/O port to generate analog voltages via the wire-wrapped DAC0800 D/A converter expansion on the SDK board. *You will have to write a program to generate a sawtooth waveform.*

The D/A converter obtains its data from the P2B port of the 8255A chip (See APPENDIX B). This data is then converted into the corresponding voltage at the analog output between -4V and +4V. You can monitor this voltage with the oscilloscope. To use the P2B port for this purpose, the 8255A must first be programmed for this function via the P2 Control Register.

1. Using the keypad monitor on the SDK board, program the 8255A port P2B for output by outputting the control byte B5H to the P2 Control Register which is set up as address FFFEH on the SDK board. This is done by using the OB (output byte) command. According to the Peripherals Handbook (See page 3-104), what does this program the 8255A to do? *Please show all eight bits with an explanation of what each bit does.*

2. Now output a data value to the D/A converter by outputting to P2B at address FFFAH on the SDK board. If you wish, you can see these values appear on pins 18 to 25 on the second (lower) 8255A chip. This is port P2B. Monitor the voltage appearing on the D/A output pins, using the ground provided.
3. What output data produces -4V?_____ 0V?_____ +4V?_____ +2V?_____
4. Write a program to generate a sawtooth waveform as shown below on the D/A output. *The program should allow the period of the cycle (i.e. the rising time of the slope) and the amplitude of the sawtooth to be specified and independently controlled.* This can be done by either reassembling and redownloading the code, or by altering the program via the keypad monitor. The D/A output is viewed by connecting the oscilloscope between the D/A output and the D/A ground (both are marked on the left hand side of the SDK board).

   This program must be demonstrated to the T.A. at the end of the lab. Remember to document it well!



LABORATORY #3

Interrupts

Description	Student # 1	Student # 2	T.A. Initials	Date
Student Name				
Student Number				
Prelab/ Attendance				
Lab #3 Completion				

HOME PREPARATION:

1. ***EXTREMELY IMPORTANT NOTE:*** Go to the lab in the week before your scheduled lab period and take copies of LAB3A.ASM and LAB3B.ASM. Print out these two programs while your are in the lab. There are office hours posted on the door of the lab. *You cannot do your home preparation without these programs.*

   Review: Appendix A, logic analyzer operation
   Appendix B, in particular the timer and PIC connections.

   Read: 8086/8088 Users's Manual, Chapter 4, Interrupt Structure section
   1990 Microprocessors Handbook, Interrupt Operations
   Peripherals Manual for Timer (8253)
   Peripheral Interrupt Controller (8259A)
   Keyboard/Display Controller (8279)
   Parallel Ports chip (8255A).

2. This lab has two parts. From the readings, understand the interrupt structure of the SDK-86; you will need this in both parts. For Part B, *write the assembler code before coming to the lab.*

3. Note that many blanks on these sheets can actually be filled in before you come to the lab. The programs you get from the lab have spaces for comments. You must fill in these blanks before handing in the lab. Bring in a 3.5" diskette.

LAB # 3: INSTRUCTIONS:

Fill in the remaining blanks on these question sheets (One per group). These sheets constitute your lab report. The T.A. will ask you questions about your work. You will have to demo your work to the T.A. *Be prepared to submit these lab sheets and your program listing at the end of the lab for marking.*

The SDK board is connected in such a way that a timer chip (8253) is connected to a programmable interrupt controller (PIC) (8259A), and the PIC is connected to the 8086 microprocessor. The timer produces waveforms to trigger the PIC. When the PIC receives interrupt requests, it ranks the requests according to their priorities and services them accordingly. The PIC will send will send a signal to the INTR pin of the 8086. Eventually the 8086 responds with an interrupt acknowledge signal (INTA). The PIC will then send the interrupt vector to the 8086 and the 8086 confirms with another INTA, then executes the appropriate interrupt service routine determined by the interrupt vector. Before the 8086 executes the interrupt service routine, it saves the flag, CS and IP registers. Once the interrupt service routine has been executed, the 8086 resumes its original task via the IRET instruction, which also restores the flags.

PART A: SDK-86 INTERRUPT TIMING DIAGRAMS

Check that the following jumpers are in place: W40 (2.45 MHz), W27 (zero wait states) Check that the following jumper is NOT in place: W36 (interrupt disable). In the 361/461 directory, you will find **LAB3A.ASM**. You have been expected to get a copy of it and study it the week before your lab.

The **LAB3A.ASM** program initializes the SDK-86 hardware to generate and allow interrupts. The main program is an infinite loop. The interrupt service routines do nothing except a return from interrupt. Some of the chips that have been initialized are on the Carleton extension of the SDK-86 board. On the actual board you have seen this extension as a column of chips at the left. At the end of the lab is a schematic diagram of the Carleton extension of the SDK-86 board. It contains two 32K byte RAM chips, an A/D converter, a D/A converter, a timer chip (8253) and an interrupt controller (8259A). The timer and interrupt controller are of concern in part A of this lab.

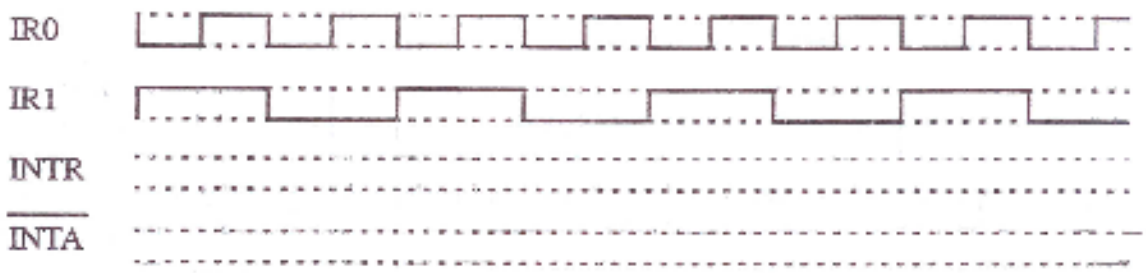
1. According to the program and peripherals manual, what has the timer chip (8253) been programmed to do?

   _____
   _____
   _____
   _____
   _____

2. What has the peripheral interrupt controller (8259A) been programmed to do?

   _____
   _____
   _____
   _____
   _____

3. Assemble/link and download LAB3A.ASM to the SDK-86 and run it from the serial monitor. (For the details of this procedure see Lab # 1.)
4. Using the logic analyzer, observe the signals INTR and $\overline{\text{INTA}}$ to complete the following timing diagram.Use the logic analyzer to trigger on IR1 and sketch INTR and $\overline{\text{INTA}}$. (See APPENDIX A for a listing of Logic Analyzer/CPU pins).
5. Complete the following timing diagram:



6. Describe what is happening:

PART B: PROGRAMMING THE TIMER, INTERRUPT CONTROLLER AND DISPLAY CHIP

You are expected to get a copy of LAB3B.ASM and study it in the week before your scheduled lab period. Here is a description of how it works.

The program initializes hardware for interrupts and then enters an infinite loop and waits for interrupts. The interrupt service routine for IR0 will output a value to the D/A converter to generate a sawtooth waveform. The interrupt service routine for IR1 will output a GARBAGE character to the rightmost digit of the SDK-86 LED display.

1. Assemble/link and download LAB3B.ASM to confirm that the program works. Use the oscilloscope to check that a sawtooth waveform is being produced by the D/A converter.

2. According to the program and peripherals manual, what has the timer chip (8253) been programmed to do?

3. What has the peripheral interrupt controller (8259A) been programmed to do? (Is this different from part A? _____)

4. What has the display chip (8279) been programmed to do? (Is this different from part A? _____)

5. Now that you have confirmed that LAB3B.ASM is generating interrupts as you would expect, you will have to change the program. *Your task is to add a third interrupt service routine called* ISR2 *that does* NOT *interfere with the existing* ISRs. ISR2 should be invoked every 0.5 seconds. To do this you will have to re-program the timer chip to generate a 0.5 second period signal on pin OUT2. Note that OUT2 is connected to IR2 on the PIC. You will also have to reprogram the PIC and the 8086. You are required to have ISR2 run at a higher priority than ISR1 and ISR0. ISR2 should display a digit in the leftmost LED display of the SDK-86 which continually counts down from 9 to 0. Refer to LAB1D.ASM for the binary to seven-segment digit translation code required by the 8279.

   Thus when your program is done, you should see a sawtooth waveform (from ISR0), a garbage character being updated every second in the rightmost LED (from ISR1) and a downward count every 0.5 seconds displayed in the leftmost digit (from ISR2).

6. In your new program, what have you programmed the timer chip (8253) to do?

7. What have you programmed the peripheral interrupt controller (8259A) to do?

8. What have you programmed the display chip (8279) to do?

9. In your program, what ISR has the lowest priority?_____

10. Write down the new 8253 control words that you would use to double the sawtooth frequency without changing the frequencies of the two SDK LED display updates. Note, you are *not required* to download a new program; just write down what you would do.

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. ***IMPORTANT:*** Go to the lab in the week before your lab session and get a copy of the files LAB4A.ASM and LAB4B.ASM that *you will have to modify.*
2. Read section 2-11 of the SDK User's Guide on the serial interface. Complete the programming for both parts A and B. The 8251A programming can be completed by looking in your text for the bit patterns.
3. 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 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 Receive Transmitter (USART) and is used for serial communications. On the SDK-86 board, it is used to communicate with the host IBM 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 IBM PC via the USART.

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

Note that the IBM PC also uses a USART for serial communication, and this chip is normally programmed by ASCII terminal emulation applications. The SDK program on the IBM 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 IBM PC and display them on the LED display of the SDK-86 board. Characters will be received from the IBM 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 **PC2SDK** 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 at 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 *requires a delay between successive writes or reads*. This is due to a manufacturing design oversight and a delay is required for proper chip operation. The **DELAY** subroutine should be called after each access to provide this delay.

2. 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 (64 x), 8 data bits, no parity, 1 stop bit.

								= _____H

Why does 64 x 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).

3. Run your program to make sure it works.

4. 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 causes repeating characters*. When this happens, an error flag in the 8279 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: PROGRAMMING THE 80X87 NUMERIC COPROCESSOR

In this part of the lab the SDK board will not be required. Here we will program the 80287 numeric coprocessor present in the IBM PCs to perform a few simple arithmetic operations. The program **LAB4B.ASM** in the 361/461 directory performs a few arithmetic operations on integers and reports the results. The code provided performs these operations first without the assistance of the coprocessor, then again using the 80287. The latter code is incomplete.

Your task is to complete the **NPX** procedure to perform similar operations as in the NONPX procedure, but using the coprocessor. You can treat the 80287 coprocessor identically to a 8087 coprocessor for this lab, (i.e. assume there is only an 8087). Although the MASM assembler supports coprocessor instruction, i.e. FLD, FMUL, etc., *you must assemble your additions by hand and insert the instruction bytes into the code using the DB directive.* An example of an integer load (FILD) command is given in the NPX subroutine.

1. List the bits of the instructions you are asked to code and complete the NPX subroutine.

Integer Division:

What is the final HEX Value?_____

Integer Add:

What is the final HEX Value?_____

Note that ESCAPE = 11011 (refer to page 7-9 in the 8086/8088 User's Manual). The coding for the 80x87 instructions may be found in the Microprocessor's Manual.

Note that the 8087 performs all of its arithmetic internally as 80-bit floating point operations. The operations take significantly less time than the identical floating point operations would take on an 8086 without the coprocessor (i.e. using an 8087 emulation software library). Also, while the coprocessor is calculating, the 8086 is free to do other work. This allows a degree of parallel processing using the 8086/87 (with careful programming!)

2. How many registers are there are on the 8087? _____

   Can they be treated in the same way as 8086 registers? If not, how are they used?

3. Why is a WAIT instruction required before executing 8087 instructions?

4. What 8087 signal is connected to the 8086 TEST signal? _____

5. Describe how the following instruction is (a) detected by the coprocessor (b) how the coprocessor accesses the data, and (c) what the instruction does. (See Chapter 7 of the 8086/8088 User's Manual)

   FILD DATA1

6. Compile and run the LAB4B.ASM program, following these steps on the IBM PC from the 361/461 directory:

   a) MASM LAB4B/r

   b) LINK LAB4B

   c) LAB4B

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

7. Is there a difference in the results between the NONPX and the NPX procedures of the LAB4B.ASM program? Can you explain this? (Hint, step through the calculation manually). What is the correct value of the calculation? (be sure to include fractional part of the correct answer).

_____

_____

_____

_____

If you have time, you may wish to single step through the program on the PC by using the DEBUG monitor. This is done using the following commands (read 'addr' as an 8086 address, for example. the variable DATA4 is at address DS: 0008).

Single Step Commands:

a) DEBUG LAB4B.EXE

b) Type '**u**' to unassemble from the current CS:IP location, 'T' for a single step (Steps over a single instruction, showing the current 8086 registers), '**g**' to run until the end of the program ('**g addr**' to run to a breakpoint at 'addr'), '**d addr1,addr2**' to display the memory locations and '**q**' to quit.

c) When the INT 21H instruction is reached you should either type 'g' or 'q' to stop, otherwise you will be stepping through the PC DOS.

From the debugger you can see the 8087 environment at the ENV location (DS:0050) after the FSAVE instruction has been executed. The layout of the environment is described in the8086/8088 User's Manual.

PART C: FLOATING POINT EMULATION

1. Compile and link the LAB4C.ASM program following the instruction provided in part B of this lab. This program provides a floating point multiplication routine that emulates part of the operation of the 8087 coprocessor. The routine multiplies 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).

2. 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 run it by typing LAB4C at the DOS prompt. Write the result in the format (± x.xx...) * 2^e, where x's are bits.

3. 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		

4. 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)

5. 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?

Student Names: _____ _____ _____

Student Numbers: _____ _____ _____

94. 61 LABORATORY #5

MC68000 Educational Computer Board

The objective of this laboratory is to provide a comprehensive introduction to systems based on the Motorola M68000 family of microcomputer products. This laboratory consists of four parts, each part requires simple programming using a small subset of the MC68000 microprocessor instructions. Programming information and system structure are described within each individual part. During the laboratory, you will gain experience on the external bus structure of the microprocessor and how it interfaces to the memory and peripheral devices. You will also have an opportunity to program two of the M68000 family peripheral chips (MC6850 and MC68230).

Home Preparations:

- Study and understand all information which is provided within each part of the laboratory.
- Answer all home preparation questions on these sheets.
- Write all required programs for this laboratory. Ensure that they are well documented.
- All home preparations have to be completed before you are allowed to perform the experiment.

Laboratory Report

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. You will have to demo your work to the T.A.. Be prepared to submit these lab sheets and your programs at the end of the lab. Overdue lab reports will not be accepted.

Marking Scheme:	Home Preparations	30%
	Demos and Report	70%

Description of MEX68KECB/D2

The educational computer board (ECB), shown in Figure 1, is a small printed circuit card which includes an MC68000 16-bit microprocessor, memory, parallel input/output (I/O), and serial communications I/O. It only requires an RS232C connection to a terminal and power supplies to become a functional system.

The ECB has a resident firmware package that provides a self-contained programming and operating environment. The firmware, aptly named "TUTOR", provides the user with monitor/debug, assembly/disassembly, program entry, and I/O control functions.

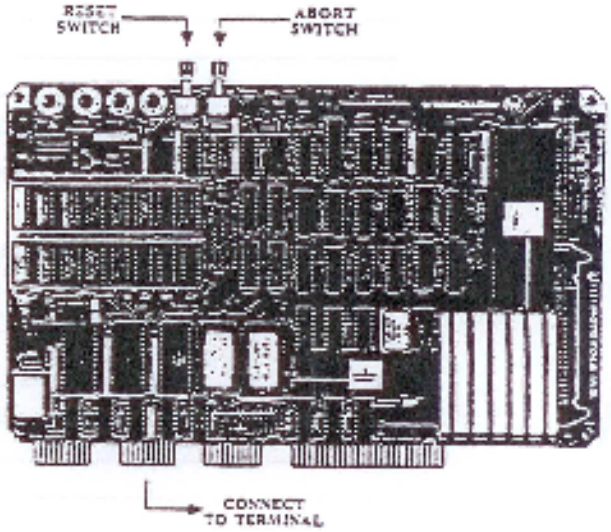


Figure 1: Educational Computer Board

A 4-MHz MC68000 MPU is used on the ECB (the functional block diagram is shown in Figure 2). All the memory and I/O devices communicate with the MPU via a common parallel bus. Some major functional areas of the board are described briefly in the following paragraphs.

System Memory

The system memory consists of 32K bytes of dynamic RAM and 16K bytes of ROM. The RAM is used for both scratchpad space of the TUTOR firmware and user programs. Approximately 2K bytes are reserved for the scratchpad; the remaining RAM ($900-$7FFF) is available to the user. The system firmware occupies the 16K bytes of read only memory. The memory map of the system is described in Figure 3.

Serial Communication Ports

Two asynchronous serial communication ports, designated Port 1 for the Terminal and Port 2 for the Host, are provided on the board. The MPU communicates with each of these ports through the MC6850 Asynchronous Communications Interface Adapter (ACIA) peripheral device. The terminal that provides user interface is connected to the ECB via Port 1, and Port 2 can be connected to a modem or directly to a host computer. Both serial ports can be jumpered for various data transmission rates (110-9600 baud).

Programmable Timer

Contained within the MC68230 Parallel Interface/Timer (PI/T) is a general purpose timer. The timer can be clocked by a 5-bit prescaler or directly. The clock source can be the 4-MHz system clock or an external clock. It can be used to generate periodic interrupts, a square wave, or a single interrupt after a programmed time period.

RESET Button

RESET is the black button located on the edge of the board. Depressing this button causes all processes to terminate, resets the MC68000 processor and MC68230 PI/T, and restarts the TUTOR firmware. Pressing the RESET button should be the appropriate action if all else fails.

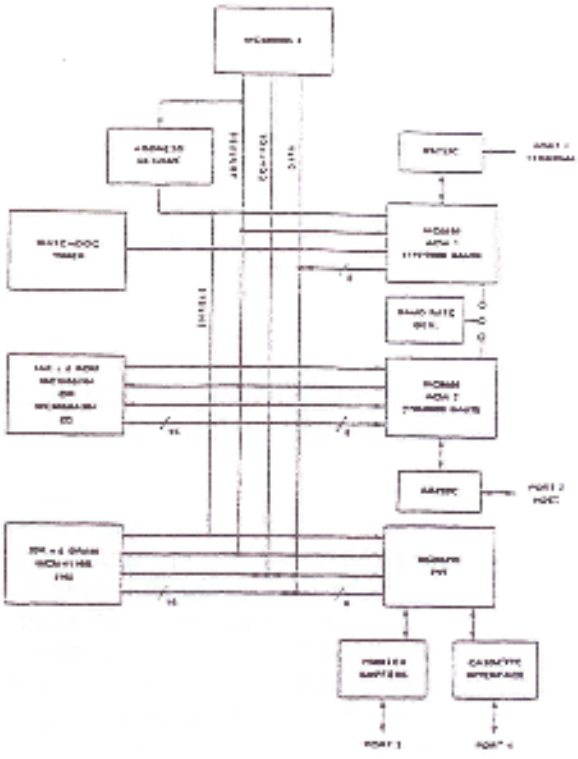


Figure 2: Functional Block Diagram

Function			Address
System Memory	Exception Vector Table	ROM/EPROM RAM	$000000-$000007 (1) $000008-$0003FF
	Tutor Scratchpad	RAM	$000400-$0008FF
User Memory		RAM	$000900-$007FFF
Tutor Firmware		ROM/EPROM	$008000-$00BFFF (1)
Not Used			$00C000-$00FFFF
I/O Devices		PI/T (Lower Byte only)	$010000-$01003F
		ACIA2 (Lower Byte) & ACIA1 (Upper Byte)	$010040-$010043
		Redundant mapping	$01FFFF
Not Used			$020000-$02FFFF
MS800 Page (SS)			$030000-$03FFFF
Not Used			$040000-$FFFFFF

NOTE: (1) Denotes read only

Figure 3: System Memory Map

ABORT Button

ABORT is the red button located next to the RESET button. The abort function causes an interrupt of the present processing (a level 7 interrupt on the MC68000) and gives control to the TUTOR firmware. This action differs from reset in that no processor register or memory contents are changed, the processor and peripherals are not reset, and TUTOR is not restarted. Also , in response to depressing the ABORT button, the contents of the MC68000 internal registers are displayed. The abort function is most appropriate when software is being debugged. The user can interrupt the processor without destroying the present state of the system.

System Setup Procedure

The system configuration used for this laboratory is described in Figure 4. The following procedure has to be performed step by step to ensure normal operation of the MC68000 system.

- Set the terminal baud rate by shorting the appropriate pins on header J10. The pins are jumpered together using a plastic jumper cap. The cap should be positioned correctly on header J10, as shown in Figure 5, to select 9600 bit/s.
- Connect the ECB to the PC using a custom-made cable. The cable has a 20-contact card edge connector on one end and a 25-contact "D" subminiature connector on the other end. The Port 1 (connector J3) of the ECB is connected to the card edge connector and the "D" connector is connected to the COM1 port of the PC.
- Connect the ECB to the power supply using the custom-made power connector.
- Power up the PC and the ECB.
- Input these commands to enter TUTOR environment:

```
cd \461
68kecb
```

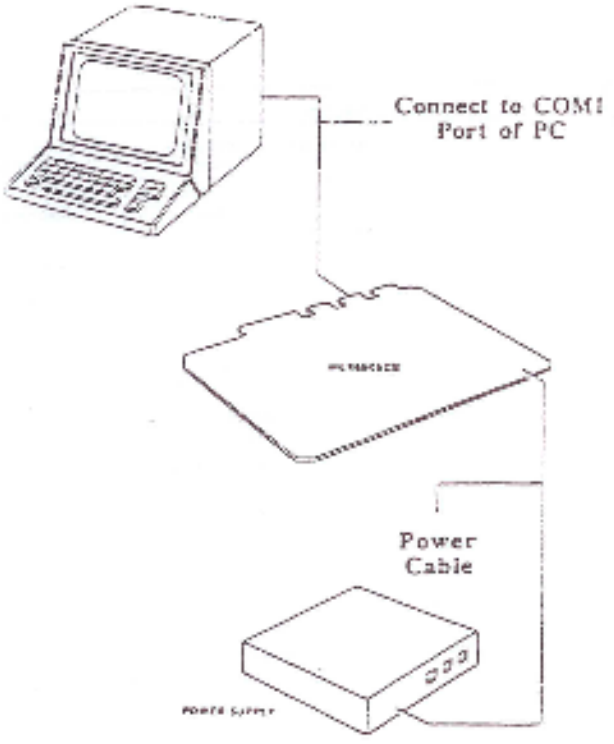


Figure 4: Lab Setup Configuration

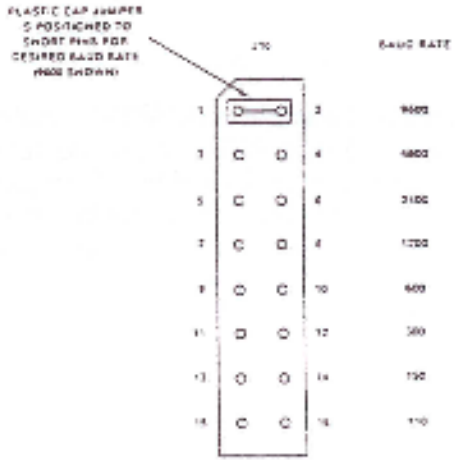


Figure 5: Terminal Baud Rate Select Jumper

Part A: Bus Signal and Waveforms

In this part of the lab, you are asked to monitor the bus signals of the MPU using an oscilloscope as it is executing a small program. You have to understand the relationship between the program and what appears on the bus.

Signal and Bus Operation Description (Brief)

Figure 6 describes the pin assignments of the microprocessor. The MC68000 has an asynchronous data bus and data transfers are controlled by the *address strobe, read/write, upper and lower data strobes*, and *data transfer acknowledge* signals.

Figure 6: Pin Assignment

Address Bus (A1 through A23)
This 23-bit, unidirectional, three-state bus is capable of addressing 16 Mbytes of data. This bus provides the address for bus operation during all cycles except interrupt acknowledge cycles. During interrupt acknowledge cycles, address lines A1, A2, and A3 provide the level number of the interrupt being acknowledged, and address lines A4 through A23 are driven to logic high.

Data Bus (D0 through D15)
This bidirectional, three-state bus is the general-purpose data path. It is 16 bits wide. The bus can transfer and accept data of either word or byte length. During an interrupt-acknowledge cycle, the external device supplies the vector number on data lines D0 through D7.

Address Strobe (AS)*
This three-state signal indicates that the information on the address bus is a valid address.

Read/Write (R/W)*
This three-state signals defines the data-bus transfer as a read or write cycle.

Upper and Lower Data Strobes (UDS,LDS*)*
These three-state signals and R/W* control the flow of data on the data bus. When both UDS* and LDS* are low, D0-D15 are valid data bits. When UDS* is low and LDS* is high, D8-D15 are valid data bits (data byte at **even** address). When UDS* is high and LDS* is low, D0-D7 are valid data bits (data byte at **odd** address). When the R/W* line is high, the processor reads from the data bus. When the R/W* is low, the processor drives the data bus.

Data Transfer Acknowledge (DTACK)*
This input signal indicates the completion of the data transfer. When the processor recognizes DTACK* during a read cycle, data is latched, and the bus cycle is terminated. When DTACK* is recognized during a write cycle, the bus cycle is terminated.

Figure 7 is the read-cycle timing diagram of the MC68000. Figure 8 shows the write-cycle timing of the bus signals. A good understand of each state of the read and write cycles is necessary to complete the preparation of this part of the lab.

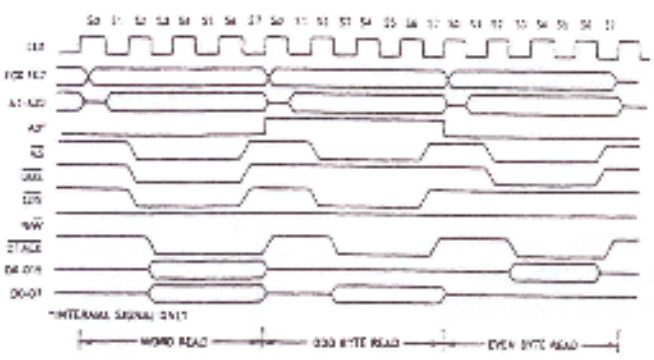


Figure 7: Read Cycle Timing Diagram

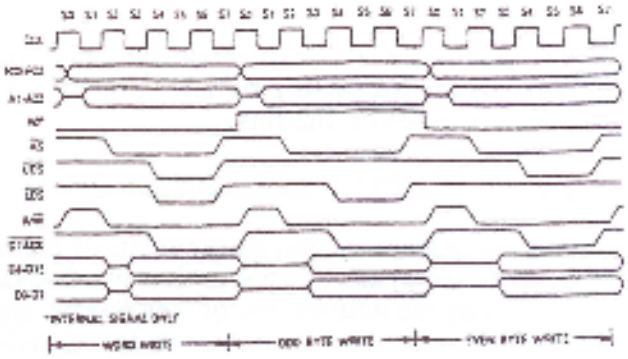


Figure 8: Write Cycle Timing Diagram

State 0 The read cycle starts in state 0 (S0). The processor places valid function codes on FC0-FC2 and drives R/W high to identify a read cycle.
State 1 Entering state 1 (S1), the processor drives a valid address on the address bus.
State 2 On the rising edge of state 2 (S2), the processor asserts AS, UDS, and LDS.
State 3 During state 3 (S3), no bus signals are altered.
State 4 During state 4 (S4), the processor waits for a cycle termination signal (DTACK is one of the cycle termination signals). If no termination signal is asserted before the falling edge at the end of S4, the processor inserts waits states (full clock cycles) until DTACK is asserted.
State 5 During state 5 (S5), no bus signals are altered.
State 6 During state 6 (S6), data from the device is driven onto the data bus.
State 7 On the falling edge of the clock entering state 7 (S7), the processor latches data from the addressed device and negates AS, UDS, and LDS. At the rising edge of S7, the processor places the address bus in the high-impedance state. The device negates DTACK at this time.

State 0 The write cycle starts in state 0 (S0). The processor places valid function codes on FC0-FC2 and drives R/W high (if a preceding write cycle has left R/W low).
State 1 Entering state 1 (S1), the processor drives a valid address on the address bus.
State 2 On the rising edge of state 2 (S2), the processor asserts AS and drives R/W low.
State 3 During state 3 (S3), the data bus is driven out of the high-impedance state as the data to be written is placed on the bus.
State 4 At the rising edge of state 4 (S4), the processor asserts UDS, LDS. The processor waits for a cycle termination signal (DTACK is one of the cycle termination signals). If no termination signal is asserted before the falling edge at the end of S4, the processor inserts waits states (full clock cycles) until DTACK is asserted.
State 5 During state 5 (S5), no bus signals are altered.
State 6 During state 6 (S6), no bus signals are altered.
State 7 On the falling edge of the clock entering state 7 (S7), the processor negates AS, UDS, and LDS. As the clock rises at the end of S7, the processor places the address and data buses in the high-impedance state, and drives R/W high. The device negates DTACK at this time.

Subset of TUTOR Commands

.reg [<expression>]	Individual register display/change.
.SR	display the value of status register.
.PC 1000	set program counter to $1000.
GO [<address>]	Begin execution in real time.
GO	begin execution at address in PC.
GO address	set PC=address and begin execution at that address.
GT <breakpoint address>	Go until breakpoint.
GT address	set temporary breakpoint=address and begin execution at address in PC.

MM <address>[;<options>] Memory Modify. This command is also used to invoke the on board assembler/disassembler.

Description of MM command

MM <address> After entering this command, one byte of data is displayed at one time starting at the specified address location followed by the "?" prompt. This command has several submodes of operation that allow modification and verification of data. The subcommands are in the format:

[<data>](cr) Update location and sequence forward.
[<data>]^(cr) Update location and sequence backward.
[<data>]=(cr) Update location and reopen same location.
[<data>].(cr) Update location and terminate.

MM <address>;DI The DI option invokes the assembler/disassembler function. The address entered should be the starting address for an instruction (opcode) word. The instruction will then be displayed in disassembled form.

The displayed instruction is follows by a question mark "?" that indicates a new source line may be entered. If a new line is entered, the instruction is immediately assembled, stored, and displayed. To enter a new line, the following format is used:

? <space><operation field><space><operand field>(cr)

Upon entry of the carriage return, the new instruction will overwrite the old instruction and enter the new one. To exit the command, a period "." is entered immediately after the question mark, followed by a carriage return.

If an error is found in the new instruction, the new line is redisplayed with and "X" immediately under the field suspected of causing a problem in the assembler. The "X" is followed by a question mark to allow re-entry of the corrected source line.

eg.

```
? MOVE.B #$F5,D1
?.
```

The above commands input a MOVE instruction and then exit from the assembler/disassembler. This MC68000 instruction stores an immediate hex byte ($F5) into data register D1.

Part A Instruction and Preparation

Lab Instruction

1) Enter the following MC68000 instructions using the MM command starting at address $1000 and execute the code.

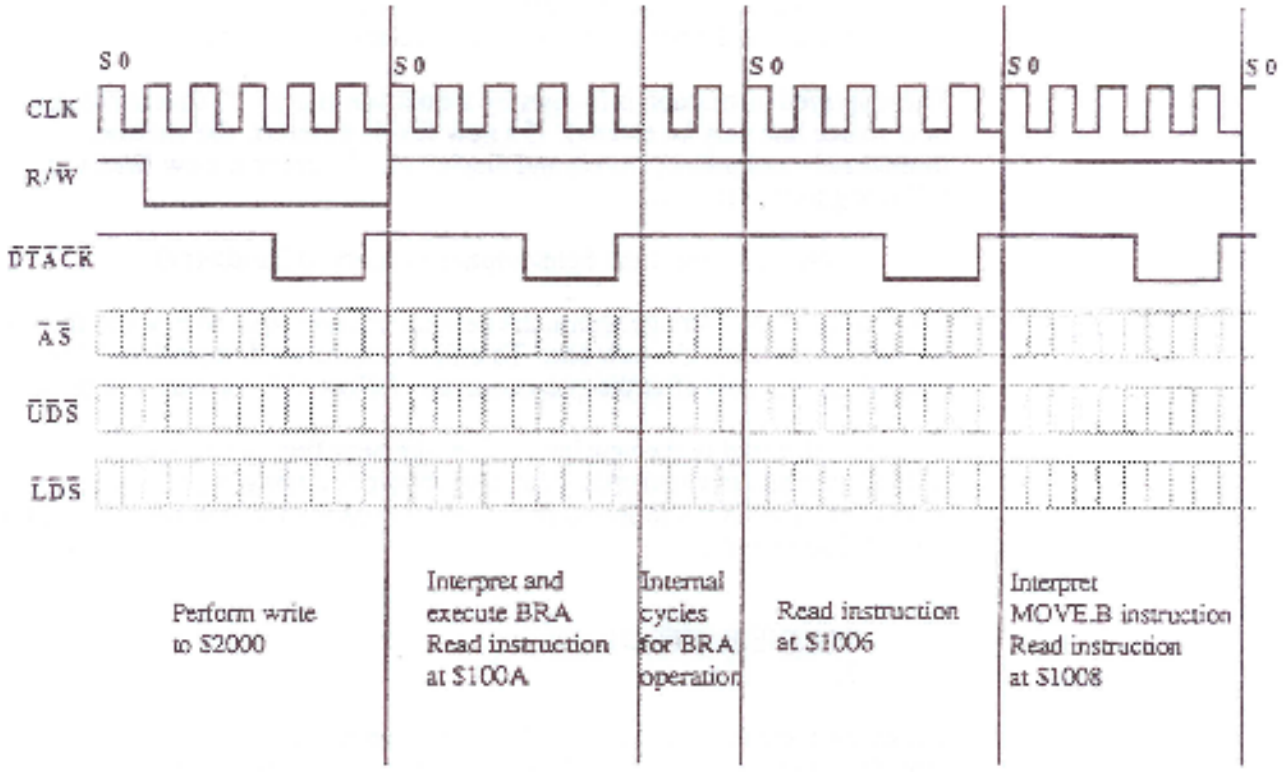
Address	Instruction
$1000	MOVE.L #$2000,A1
$1006	MOVE.B D1,(A1)
$1008	BRA $1006
$100A	- -

2) Connect an oscilloscope to monitor the R/W* on channel 1 and set triggering on the falling edge of the R/W* signal. Use channel 2 to observe the other bus signals.

Home Preparation

1) Use the provided information (read and write cycle operations) to predict the waveform of AS*, UDS*, and LDS*. Draw the waveform in the provided space. Verify your prediction during the lab session by performing instructions 1 and 2. **Instruction fetch is a Word Read Cycle.**

```
$1000    MOVE.L  #$2000,A1
$1006    MOVE.B  D1,(A1)
$1008    BRA     $1006
$100A    -       -
```



Part B: Programming the ACIA (MC6850)

In this part of the lab, you are asked to program the MC6850 to interact with the keyboard and the monitor of the PC.

Description of Small Subset of MC68000 Instructions

The MC68000 executes instructions in one of two modes - user mode or supervisor mode. The user mode provides the execution environment for the majority of application programs. The supervisor mode, which allows some additional instructions and privileges, is used by the operating system and other system software.

The user programmer's model of the MC68000 consists of eight 32-bit data registers (D0-D7), seven 32-bit address registers (A0-A6), one 32-bit user stack pointer (A7), one 32-bit program counter (PC), and one 8-bit status register (SR).

The status register in the supervisor programmer's model is 16-bit long. This status register contains the interrupt mask and the condition codes: overflow (V), zero (Z), negative (N), carry (C), and extend (X). Additional status bits indicate that the processor is in trace (T) mode and/or in the supervisor (S) state. The interrupt mask can be used to mask out unwanted interrupts at a particular priority level. When a higher level interrupt is masked, all interrupts which has lower priority than that are masked too.

The MC68000 is a memory map I/O microprocessor. The processor communicates with its peripheral devices by writing to or reading from the memory location where the device is located. A good understanding on the MC68000 memory map of the ECB is needed before you can communicate and program the ACIA or any other peripheral device.

In this lab, use only the following list of instructions for all your MC68000 programming. (Word = 2 bytes, Long Word = 2 words = 4 bytes. $F4 is hex constant F4 which is 244 in decimal. The "$" is used to inform the assembler that the constant is a hex number.)

MOVE.L #nnn,An	Move immediate data (long word) to address register An
MOVE.B #nnn,(An)	Move immediate data (byte) to memory location specified by the content of address register An
MOVE.B (An),Dn	Move the content (one byte) at the memory location specified by An into the data register Dn.
BTST #n,Dn	Test a bit in the data register Dn and sets the Z condition code appropriately.
BEQ.S address	Short branch to "address" if Z=1
BRA.S address	Short branch to "address" always
RTE	Return from interrupt

ACIA Programmer's model on the ECB

Figure 9 describes the ACIA Address Map of the terminal port. Figure 10 describes the programming model of the ACIA.

Address (S)	ACIA Registers (Terminal Port)
010040	Control Register (Write Only) Status Register (Read Only)
010042	Transmit Data Register (Write Only) Receive Data Register (Read Only)

Figure 9: ACIA Address Map

Transmit Data Register (S010042)
Writing data into the register causes the Transmit Data Register Empty bit in the status Register to go low. Data can then be transmitted. If the transmitter is idling and no character is being transmitted, then the

transfer will take place within 1-bit time of the trailing edge of the Write command. If a character is being transmitted, the new data character will commence as soon as the previous character is completed. The transfer of data causes the Transmit Data Register Empty (TDRE) bit to indicate empty.

Receive Data Register ($010042)
Data is automatically transfered to the empty Receive Data Register from the receiver deserializer (a shift register) upon receiving a complete character. This event causes the Receive Data Register Full bit (RDRF) in the status buffer to go high (full). Data may then be read through the bus by addressing the ACIA and selecting the Receive Data Register ($010040) when the ACIA is enabled. The non-destructive read cycle causes the RDRF bit to be cleared although the data is retained in the Receive Data Register. The status is maintained by RDRF as to whether or not the data is current. When the Receive Data Register is full, the automatic transfer of data from the Receiver Shift Register to the Data Register is inhibited and the Data Register contents remain valid with its current status stored in the Status Register.

7	6	5	4	3	2	1	0
CR7	CR6	CR5	CR4	CR3	CR2	CR1	CR0

Control Register

7	6	5	4	3	2	1	0
IRQ	PE	OVRN	FE	CTS	DCD	TDRE	RDRF

Status Register

CR1	CR0	Function
0	0	÷ 1
0	1	÷ 16
1	0	÷ 64
1	1	Master Reset

CR7	Function
0	Receive Interrupt Enabled
1	Receive Interrupt Disabled

CR4	CR3	CR2	Function
0	0	0	7 Bits + Even Parity + 2 Stop Bits
0	0	1	7 Bits + Odd Parity + 2 Stop Bits
0	1	0	7 Bits + Even Parity + 1 Stop Bits
0	1	1	7 Bits + Odd Parity + 1 Stop Bits
1	0	0	8 Bits + 2 Stop Bits
1	0	1	8 Bits + 1 Stop Bit
1	1	0	8 Bits + Even Parity + 1 Stop Bit
1	1	1	8 Bits + Odd Parity + 1 Stop Bit

CR6	CR5	Function
0	0	RTS = low, Transmitting Interrupt Disabled
0	1	RTS = low, Transmitting Interrupt Enabled
1	0	RTS = high, Transmittion Interrupt Disabled
1	1	RTS = low, Transmits a Break level on the Transmit Data Output. Transmitting Interrupt Disabled

Figure 10: ACIA Programming Model

Control Register
The ACIA Control Register consists of eight bits of write-only buffer. This register controls the function of the receiver, transmitter, interrupt enables, and the Request-to-Send peripheral/modem control output.

Counter Divide Select Bits (CR0 and CR1)
The Counter Divide Select Bits determine the divide ratios utilized in both the transmitter and receiver sections of the ACIA. These counter select bits provide for the clock divide ratios as describe in the Figure.

Word Select Bits (CR2,CR3 and CR4)
The Word Select bits are used to select word length, parity and the number of stop bits. Word Length, Parity Select, and Stop Bit changes are not buffered and therefore become effective immediately. The encoding format is as shown in the Figure.

Transmitter Control Bits (CR5 and CR6)
Two Transmitter Control bits provide for the control of the interrupt from the Transmit Data Register Empty condition, the Request-to-Send output, and the transmission of a Break level (space). The encoding format is shown in the Figure.

Receive Interrupt Enable Bit (CR7)
The following interrupts will be enabled by a high level in bit position 7 of the Control Register: Receive Data Register Full, Overrun, or a low to high transition on the Data Carrier Detect signal line.

Status Register
Information on the status of the ACIA is available to the MPU by reading the ACIA status register. Information stored in this register indicates the status of the Transmit Data Register, the Receive Data Register and error logic, and the peripheral/modem status inputs of the ACIA.

Receive Data Register Full (RDRF), Bit 0
Receive Data Register Full, when set, indicates that there is a character waiting to be read. RDRF is cleared after an MPU read of the Receive Data Register. This status bit has to be checked before reading the Receive Data Register.

Transmit Data Register Empty (TDRE), Bit 1
Transmit Data Register Empty bit being set high indicates that the Transmit Data Register contents have been transferred and that new data may be entered. This status bit has to be checked before writing to the Transmit Data Register.

Part B Instruction and Preparation

Home Preparation

1) What is the value, in hex, of the control byte which sets the ACIA to clock divide ratio to ÷ 16, data encoding format to 8 data bits and 1 stop bit, RTS to low , and transmit and receive interrupts disabled? _____

2) Use the answer to instruction 1 and write a program which sends an asterisk (ASCII $2A) to the PC monitor whenever there is a keypress on the keyboard.

Lab Instruction

1) Input your program into the system using the described TUTOR commands, debug, and demo it to the T.A.

Part C: Square Wave Generation Using MC68230 (PI/T)

In this part of the lab, you are asked to program the MC68230 to generate a periodic square wave.

Programer's Model of PI/T

Address ($)	PI/T Registers
010021	Timer Control Register (TCR)
010023	Timer Interrupt Vector Register (TIVR)
010027	Counter Preload Register High (CPRH)
010029	Counter Preload Register Middle (CPRM)
01002B	Counter Preload Register Low (CPRL)
01002F	Count Register High (CNTRH)
010031	Count Register Middle (CNTRM)
010033	Count Register Low (CNTRL)
010035	Timer Status Register (TSR)

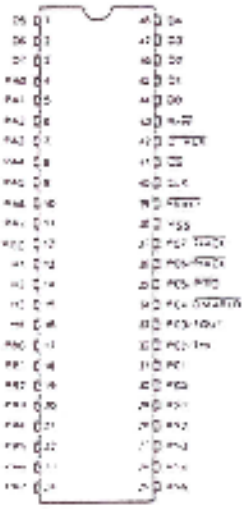


Figure 11: PI/T Address Map and Pin Assignment

Figure 11 describes the address map and the pin assignment of MC68230. The PI/T timer contains a 24-bit synchronous down counter that is loaded from three 8-bit counter preload registers. The

24-bit counter may be clocked by the output of a 5-bit (divide-by-32) prescaler or by an external timer input (TIN). If the prescaler is used, it may be clocked by the system clock (CLK pin) or by the TIN external input. The Counter signals the occurrence of an event primarily through zero detection. (A zero is when the counter of the 24-bit timer is equal to zero.) This sets the zero detect status (ZDS) bit in the timer status register. It may be checked by the processor or may be used to generate a timer interrupt. The ZDS bit can be reset by writing a one to the timer status register in that bit position independent of timer operation.

The timer is fully configured and controlled by programming the 8-bit timer control register. It controls: 1) the choice between the port operation and the timer operation of three timer pins, 2) whether the counter is loaded from the counter preload register or rolls over when zero detect is reached, 3) the clock input, 4) whether the prescaler is used, and 5) whether the timer is enabled. The following paragraphs describe the programer's model of the PI/T timer.

4.9 TIMER CONTROL REGISTER (TCR)

Timer Control Register (TCR)

7	6	5	4	3	2	1	0
TOUT/TIACK Control			Z. D. Control	-	Clock Control		Timer Enable

The timer control register (TCR) determines all operations of the timer. Bits 7-5 configure the PC3/TOUT and PC7/$\overline{\text{TIACK}}$ pins for port C, square wave, vectored interrupt, or autovectored interrupt operation; bit 4 specifies whether the counter receives data from the counter preload register or continues counting when zero detect is reached; bit 3 is unused and is read as zero; bits 2 and 1 configure the path from the CLK and TIN pins to the counter controller; and bit 0 enables the timer. This register is readable and writable at all times. All bits are cleared to zero when the $\overline{\text{RESET}}$ pin is asserted.

TCR 7 6 5 — TOUT/TIACK Control

- 0 0 X — The dual-function pins PC3/TOUT and PC7/$\overline{\text{TIACK}}$ carry the port C function.
- 0 1 X — The dual-function pin PC3/TOUT carries the TOUT function. In the run state it is used as a square-wave output and is toggled on zero detect. The TOUT pin is high while in the halt state. The dual-function pin PC7/$\overline{\text{TIACK}}$ carries the PC7 function.
- 1 0 0 — The dual-function pin PC3/TOUT carries the TOUT function. In the run or halt state it is used as a timer interrupt request output. The timer interrupt is disabled, thus, the pin is always three stated. The dual-function pin PC7/$\overline{\text{TIACK}}$ carries the $\overline{\text{TIACK}}$ function; however, since interrupt request is negated, the PI/T produces no response (i.e., no data or DTACK) to an asserted $\overline{\text{TIACK}}$. Refer to 5.1.3 Timer Interrupt Acknowledge Cycles for details.
- 1 0 1 — The dual-function pin PC3/TOUT carries the TOUT function and is used as a timer interrupt request output. The timer interrupt is enabled; thus, the pin is low when the timer ZDS status bit is one. The dual-function pin PC7/$\overline{\text{TIACK}}$ carries the $\overline{\text{TIACK}}$ function and is used as a timer interrupt acknowledge input. Refer to the 5.1.3 Timer Interrupt Acknowledge Cycles for details. This combination supports vectored timer interrupts.
- 1 1 0 — The dual-function pin PC3/TOUT carries the TOUT function. In the run or halt state it is used as a timer interrupt request output. The timer interrupt is disabled; thus, the pin is always three-stated. The dual-function pin PC7/$\overline{\text{TIACK}}$ carries the PC7 function.
- 1 1 1 — The dual-function pin PC3/TOUT carries the TOUT function and is used as a timer interrupt request output. The timer interrupt is enabled; thus, the pin is low when the timer ZDS status bit is one. The dual-function pin PC7/$\overline{\text{TIACK}}$ carries the PC7 function and autovectored interrupts are supported.

TCR 4 — Zero Detect Control

- 0 — The counter is loaded from the counter preload register on the first clock to the 24-bit counter after zero detect, then resumes counting.
- 1 — The counter rolls over on zero detect, then continues counting.

TCR 3 — Unused and is always read as zero.

TCR
2 1 Clock Control

0 0 The PC2/TIN input pin carries the port C function, and the CLK pin and prescaler are used. The prescaler is decremented on the falling transition of the CLK pin; the 24-bit counter is decremented, rolls over, or is loaded from the counter preload registers when the prescaler rolls over from $00 to $1F. The timer enable bit determines whether the timer is in the run or halt state.

0 1 The PC2/TIN pin serves as a timer input, and the CLK pin and prescaler are used. The prescaler is decremented on the falling transition of the CLK pin; the 24-bit counter is decremented, rolls over, or is loaded from the counter preload registers when the prescaler rolls over from $00 to $1F. The timer is in the run state when the timer enable bit is one and the TIN pin is high; otherwise, the timer is in the halt state.

1 0 The PC2/TIN pin serves as a timer input and the prescaler is used. The prescaler is decremented following the rising transition of the TIN pin after being synchronized with the internal clock. The 24-bit counter is decremented, rolls over, or is loaded from the counter preload registers when the prescaler rolls over from $00 to $1F. The timer enable bit determines whether the timer is in the run or halt state.

1 1 The PC2/TIN pin serves as a timer input and the prescaler is not used. The 24-bit counter is decremented, rolls over, or is loaded from the counter preload registers following the rising edge of the TIN pin after being synchronized with the internal clock. The timer enable bit determines whether the timer is in the run or halt state.

TCR
0 Timer Enable

0 Disabled

1 Enabled

4.10 TIMER INTERRUPT VECTOR REGISTER (TIVR)

The timer interrupt vector register contains the 8-bit vector supplied when the timer interrupt acknowledge pin $\overline{\text{TIACK}}$ is asserted. The register is readable and writable at all times, and the same value is always obtained from a normal read cycle or a timer interrupt acknowledge bus cycle ($\overline{\text{TIACK}}$). When the $\overline{\text{RESET}}$ pin is asserted the value of $0F is loaded into the register. Refer to 5.1.3 Timer Interrupt Acknowledge Cycles for more details.

4.11 COUNTER PRELOAD REGISTER H, M, L (CPRH-L)

Counter Preload Register H, M, L (CPRH-L)

7	6	5	4	3	2	1	0	
Bit 23	Bit 22	Bit 21	Bit 20	Bit 19	Bit 18	Bit 17	Bit 16	CPRH
Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8	CPRM
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	CPRL

The counter preload registers are a group of three 8-bit registers used for storing data to be transferred to the counter. Each of the registers is individually addressable.

These registers are readable and writable at all times. A read cycle proceeds independently of any transfer to the counter, which may be occurring simultaneously. To insure proper operation of the PI/T timer, a value of $000000 may not be stored in the counter preload registers for use with the counter. The $\overline{\text{RESET}}$ pin does not affect the contents of these registers.

4.12 COUNT REGISTER H, M, L (CNTRH-L)

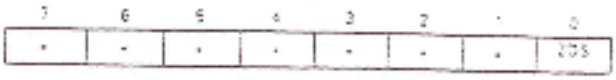
Count Register H, M, L (CNTRH-L)

7	6	5	4	3	2	1	0	
Bit 23	Bit 22	Bit 21	Bit 20	Bit 19	Bit 18	Bit 17	Bit 16	CNTRH
Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8	CNTRM
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	CNTRL

The count registers are a group of three 8-bit addresses at which the counter can be read. The contents of the counter are not latched during a read bus cycle; thus, the data read at these addresses is not guaranteed if the timer is in the run state. Write operations to these addresses result in a normal bus cycle but the data is ignored.

4.13 TIMER STATUS REGISTER (TSR)

Timer Status Register (TSR)



The timer status register contains one bit from which the zero detect status can be determined. The ZDS status bit (bit 0) is an edge-sensitive flip-flop that is set to one when the 24-bit counter decrements from 000001 to 000000. The ZDS status bit is cleared to zero following the direct reset operation or when the timer is halted. Note that when the $\overline{\text{RESET}}$ pin is asserted the timer is disabled, and thus enters the halt state.

This register is always readable without consequence. A write access performs a direct reset operation if bit 0 in the written data is one. Following that, the ZDS bit is zero.

This register is constructed with a reset dominant S-R flip-flop so that all clearing conditions prevail over the possible zero detect condition.

Part C Instruction and Preparation

Home Preparation

1) Use "00" as the clock control (the system clock and divide-by-32 prescaler are used) value. The system clock is 4 MHz. What are the count values, in hex, for the counter preload registers in order to generate a 1.6 ms periodic square wave at TOUT (pin 33)? _____

2) What value, in hex, has to be loaded into the Timer Control Register (TCR) to generate the square wave? _____

3) Write a program to instruct the PI/T to generate a 1.6 ms periodic square wave.

Lab Instruction

1) Input your program into the system using the described TUTOR commands, debug, and show it to the T.A.. Use an oscilloscope to observe the waveform output of TOUT.

2) If you have problem running your program, push the reset button before executing the program sequence. This will cause the system to reload the SR to mask out all maskable interrupts and reset the PI/T.

Part D: Interrupt Programming

The MC68000 recognizes seven interrupt priority levels. The interrupt priority levels are numbered from one to seven with level seven having the highest priority (the equivalent of a non-maskable interrupt). The MC68000 status register contains a three-bit mask which indicates the current processor priority level. Figure 12 shows the status register of the MC68000. Interrupts are inhibited for priority levels less than or equal to the current processor priority. An interrupt request is made to the processor by encoding the interrupt request level on the interrupt request lines (IP0-IP2) with a zero indicating no interrupt requests.

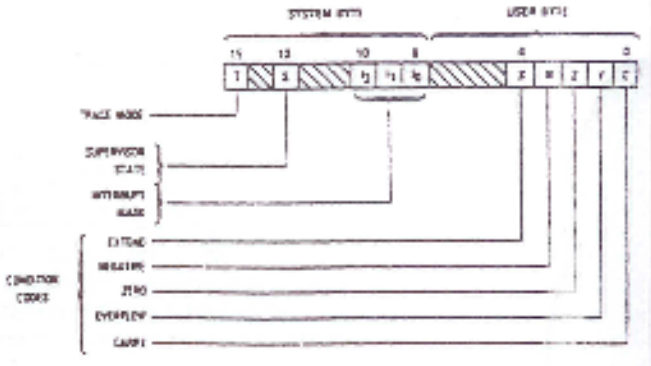
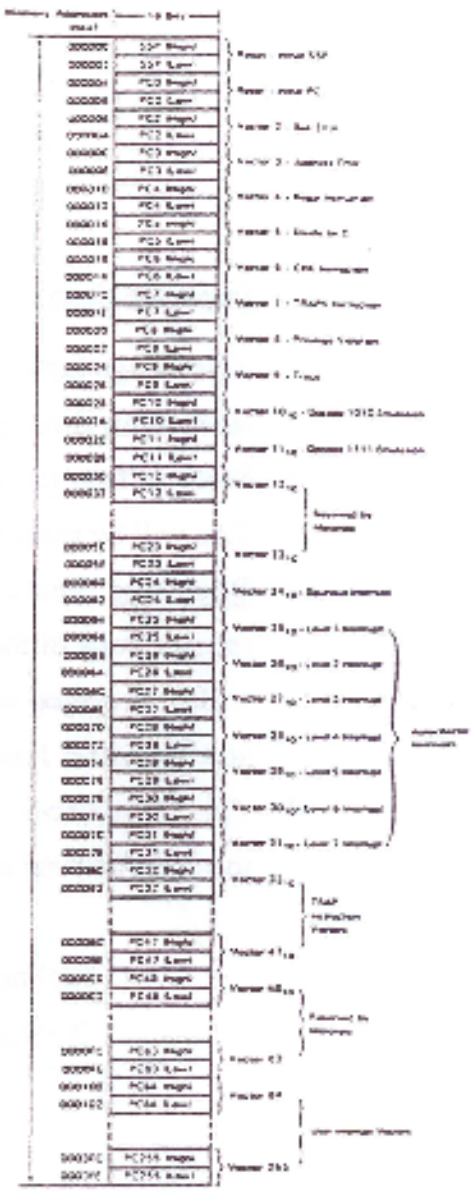


Figure 12: MC68000 Status Register

The interrupt priority level of the PI/T timer is a level 2 non-autovector interrupt. As a result the PI/T has to provide an 8-bit vector number to the MPU during each interrupt. This interrupt vector number has to be programmed into the PI/T timer. The table on the right hand side describes the interrupt vector table of the MC68000.



Periodic Interrupt Programming and Operation

The processor loads the counter preload registers (CPR), the timer interrupt vector register (TIVR), and the timer control register (TCR), and then enables the timer. When the 24-bit counter passes from $000001 to $000000, the ZDS status bit is set and the TOUT (interrupt request) pin is asserted. At the next clock to the 24-bit counter, it is again loaded with the contents of the CPRs and thereafter decrements. The processor must direct clear the status bit to negate the interrupt request. **The ZDS bit can be reset by writing a one to the timer status register in that bit position independent of timer operation.**

Part D Instruction and Preparation

Home Preparation

1) What value, in hex, has to be program into the TCR to generate a periodic interrupt ? _____

2) With "00" as the clock control value, what values, in hex, have to be loaded into the preload count registers to generate an interrupt every one second ? _____
_____

3) If interrupt vector number 64 is used, what memory location has to contain the starting address of the interrupt service routine ?
_____

4) Write a program, interrupt driven using vector number 64, to send a beep (ASCII $07) to the PC every second. The program must include the initializations of the ACIA and the PI/T timer, and a simple interrupt service routine. The service routine sends a beep character to the PC for each interrupt. Make sure you reset the ZDS at the entry of the service routine.

Lab Instructions

1) Input your program into the system. Use ".SR 2100" to unmask the timer interrupt and use the MM command to store the starting address of the service routine into the vector table before running your program. You have to debug your program and demo it to the T.A..

2) If you have any problem, push the reset button to reset the board. Keep in mind that every time the system is reset, the SR and the vector table are re-initialized. Make sure you have reprogram the SR and the vector table before executing your program again.

Carleton University
Systems and Computer Engineering, 94.361*
Laboratory #5: Microprocessor Interfacing

1. Instructions

1. This design lab is to be completed in groups, one submitted report per group, **no more than three students per group**.
2. Students may (and are encouraged to) begin the lab work as soon as possible. The TAs will be available to assist students during the normal lab periods.
3. **Due Date**: June 27th 2000. To be submitted during class. Students are advised to retain a copy of their final report for their records and study purposes. Late reports will be accepted with a 15% penalty up until June 30th 2000. No reports will be accepted after June 30th, 2000.
4. This lab replaces Lab 5 in the original lab manual and carries the same weight towards the final grade. The marking scheme is as follows:

Technical Correctness	– 15
Organization and Presentation	– 5
Total	20

5. It is the students' responsibility to work effectively in the lab teams, complete the designs, organize the report, and present the solution in a logical and neat submission that another engineer could understand. Not every wire connecting any pin to another pin must be shown. Collect common connections into buses to reduce drawing complexity, but clearly indicate the connections proposed by your group. Reports need not be typed, but must be neatly presented.
6. Do not attempt to include all interfaces on the same drawing; complete schematic diagrams for each task as outlined on the following pages.

2. Background

A small embedded 8086-based microprocessor system is needed for a supervisory control system that requires 128 switches and 128 LEDs. The general architecture is shown in Figure 1. The purpose of this lab is to design the microprocessor, memory, and I/O subsystems (to the left of the dashed line). The system is based on an 8086 with 8284A clock generator, 256Kbyte of SRAM, 128Kbyte of EPROM, peripheral I/O consisting of an 8255 PPI, 8253 Timer, 8279 keypad/display controller and an 8251A serial communications interface. The system clock is to be 5MHz, however one wait state is required for access to the EPROM.

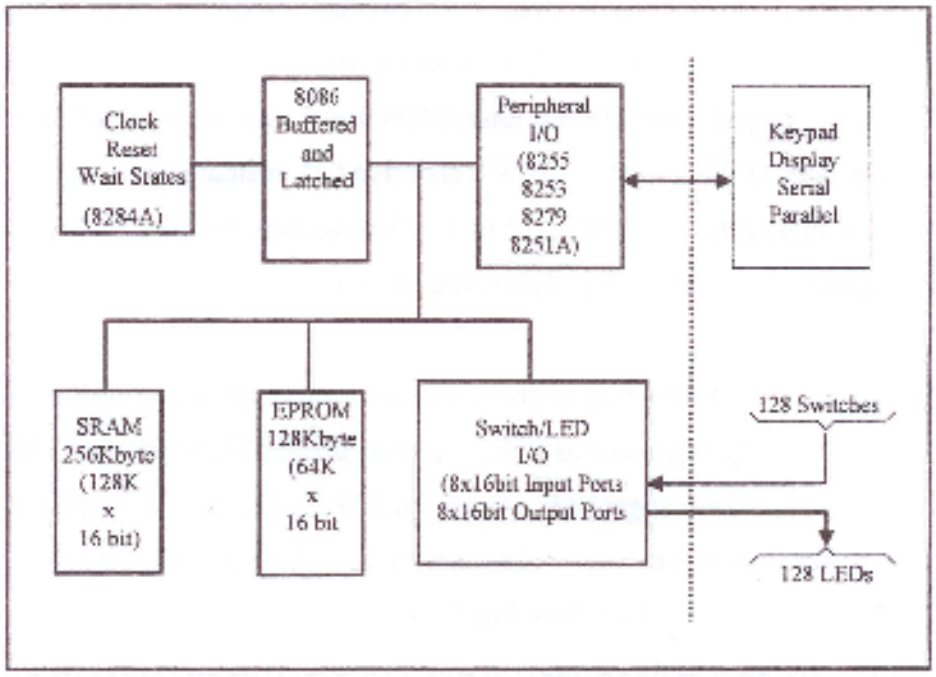


Figure 1: 8086 embedded system for supervisory control.

3. Procedure

Complete the following design tasks:

Task 1: Buffer and latch the 8086 microprocessor using standard components ('244, '245, '373) as covered in the lecture. Connect an 8284A clock generator to the 8086 for 5MHz operation and reset capability. Design a wait state generator offering 0-to-7 wait states and connect this to the 8284A. **Describe the operation** of the wait state generator and how it will be set and activated to insert one wait state whenever the microprocessor reads from EPROM.

Task 2: Design address decoding circuitry and data and control bus connections for 256Kbytes of SRAM organized as 128K by 16 bits using 62256 SRAM chips (32K by 8 bit each) **starting at** address 00000H. Each 62256 has 8 data lines, and three active low control lines labeled WE*, OE*, and CS* (note that (.)* indicates an active low signal). Any standard logic gates or decoders (2-to-4 or 3-to-8) may be used in your design. The complete address bus must be decoded in your design.

Task 3: Design address decoding circuitry and data and control bus connections for 128Kbytes of EPROM organized as 64K by 16 bits using 27128 EPROM chips (16K by 8 bit each) such that the **highest byte address** of EPROM resides at FFFFFH. Each 27128 has 8 data lines and two active low control lines labeled OE* and CE*. Any standard logic gates or decoders (2-to-4 or 3-to-8) may be used in your design. The complete address bus must be decoded in your design. Whenever EPROM is accessed, one wait state must be inserted in the read cycle to accommodate for the slower response of these devices.

NOTE: All addresses from the top of SRAM to the bottom of EPROM are to be left for future expansion.

Task 4: Design address decoding circuitry and data control bus connections for the peripheral I/O system using I/O-mapped-I/O at the addressed given in Table 1. Additional I/O may be required at a later date, so your design must only allow the specified devices to be activated at the addresses given. Any standard logic gates or decoders (2-to-4 or 3-to-8) may be used in your design. The complete I/O address bus must be decoded in your design.

Table 1: Register addresses for peripheral I/O.

Peripheral	Register	Address (H)
8255	Port A	FFC0
	Port B	FFC2
	Port C	FFC4
	Command Reg	FFC6
8253	Counter 0	FFD1
	Counter 1	FFD3
	Counter 2	FFD5
	Control Word Reg	FFD7
8279	Data Reg	FFE0
	Command Reg	FFE2
8251A	Data Reg	FFF0
	Command Reg	FFF2

Task 5: Design address decoding circuitry and data control bus connections for the switch/LED I/O system using I/O-mapped-I/O. Each switch has two positions (1=on or 0=off) and maintains its position when switched (detented toggle switches). Each LED is turned on by providing a logic high level to that LED, and is turned off by providing a logic low level to that LED.

Switches and corresponding LEDs are arranged in groups of 8 forming one byte of data that is read from (input port) a group of switches or written to (output port) a group of LEDs. Each 8-bit input/output port pair is to have the same address. For example, switches 0-7 are connected to an 8-bit input port which has the same port address as the output port which is connected to LEDs 0-7. The same is true for switches 8-15 and LEDs 8-15, etc.

I/O-mapped-I/O is to be used with addresses running sequentially from CBF0H through CBFFH. That is, the port for switches/LEDs 0-7 is CBF0H, 8-15 is CBF1H, 16-23 is CBF2H, etc., and 120-127 is CBFF. Switches/LEDs can be read/written byte-wise (8-bits at a time) or word-wise (16 bits at a time). Byte- or word-wise access can occur at even or odd addresses depending on the group of switches/LEDs to be accessed. Any standard logic gates or decoders (2-to-4 or 3-to-8) may be used in your design. The complete I/O address bus must be decoded in your design.

4. Lab Report

The report must be neatly prepared (not necessarily typed) and logically presented. Reports that are simply too messy to decipher will be returned with a mark of zero. **Include a title page that clearly identifies the name and student number of each member of the lab group. The maximum number of students per group is 3.** Give a short introduction and then document your designs under the Task headings given above. Include a discussion of your design under each Task and a concluding section.

APPENDIX A

Operation of the Logic Analyzer

1.0 STATUS KEY

The status key brings up the system status. It is a passive screen (i.e. you canot change the settings in this screen) The settings will look something like the following:

SOURCE:	Internal	WIDTH: 16 bits
PERIOD:	0040 nsec	DATA QUAL: off
* DELAY:	0 clocks	
* SEQUENCE:	A	
* TRIGGER	* Format POD: LOGIC 16	

A: xxxxxxxx xxxxxxxx
B: xxxxxxxx xxxxxxxx
C: xxxxxxxx xxxxxxxx
D: xxxxxxxx xxxxxxxx

Note: One of the bits on Channel A should be set for the selected trigger line where the *leftmost* bit is MSB and the *rightmost* bit is the LSB

2.0 KEY DESCRIPTION AND SELECTION SETTING PROCEDURE

a. FORMAT Key

POD Mode: Select Logic8 or Logic16. This will set the rest of the fields on the screen.

b. CLOCK Key

SOURCE: Choose 0 for INTERNAL

INTERNAL: Choose ?? nsec for clock period (TA's will be able to help you here). Note: ignore external settings.

c. CONFIG Key

RECORD WIDTH: Select 8 or 16 bits. The rest of the fields in this screen are irrelevant.

d. TRIGGER Key

With a 16 bit (Logic 16) sequence, only the first 16 positions on A will be used. The MSB will be on the *leftmost* bit and the LSB will be on the *rightmost* bit. Also, the selected sequence will be displayed at the bottom of this screen. Note: A maximum of 12 channels can be viewed at once on the display.

e. **DELAY Key**

The delay key allows the user to set a window of 2000 events (when logic 16 or 16 bits is selected). Therefore, depending on the your requirements, the window can be set to capture events after (delay = 0), before (delay = 1999) or in the middle of the trigger occurence.

f. **SEQUENCE Key**

TRIGGER SEQUENCE: Select 0 for trigger sequence A.

Remember to recheck the sysem status after setting things up by pressing the STATUS *Key.*

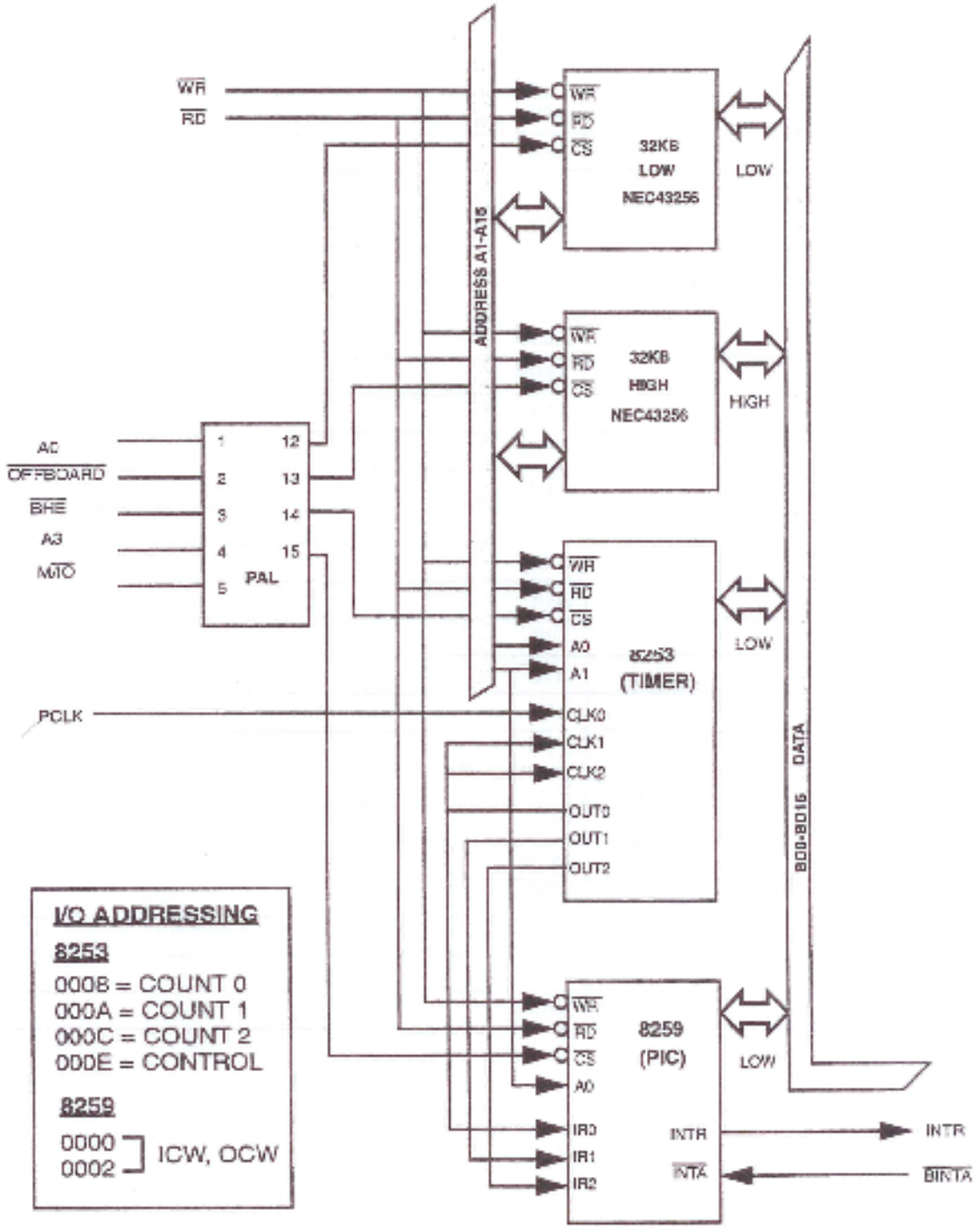
TABLE 1. Logic Analyzer Pinout

Logic Analyzer Signal	CPU Signal (SDK Board)	CPU Signal (68000 Board)
0	$\overline{\text{PCLK}}$	CLK
1	ALE	R/$\overline{\text{W}}$
2	$\overline{\text{WR}}$	$\overline{\text{DTACK}}$
3	$\overline{\text{RD}}$	$\overline{\text{LDS}}$
4	AD0	$\overline{\text{UDS}}$
5	AD1	$\overline{\text{AS}}$
6	AD2	PC3/TOUT
7	AD3	
8	AD4	
9	AD5	
10	AD6	
11	AD7	
12	AD8	
13	AD9	
14	AD10	
15	AD11	
16	AD12	
17	AD13	
18	AD14	
19	AD15	
20	A16	
21	A17	
22	A18	
23	A19	
24	IR0	
25	IR1	
26	IR2	
27	INTR	
28	$\overline{\text{INTA}}$	
29	N/C	
30	N/C	
31	N/C	

APPENDIX B:

SDK System Extensions

1.0 8253 Timer and 8259 Peripheral Interrupt Controller



2.0 8255A Chip and DAC/ADC Expansion