

University of Montana

## ScholarWorks at University of Montana

---

Graduate Student Theses, Dissertations, &  
Professional Papers

Graduate School

---

1992

### Spiral model approach to microprocessor laboratory system design

Tsu-i Chên  
*The University of Montana*

Follow this and additional works at: <https://scholarworks.umt.edu/etd>

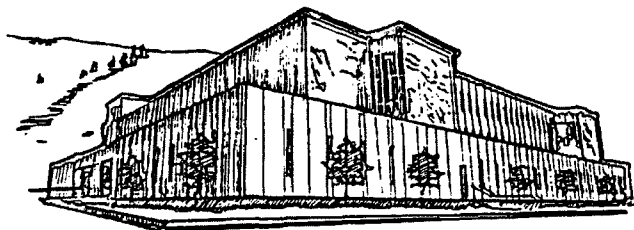
**Let us know how access to this document benefits you.**

---

#### Recommended Citation

Chên, Tsu-i, "Spiral model approach to microprocessor laboratory system design" (1992). *Graduate Student Theses, Dissertations, & Professional Papers*. 5509.  
<https://scholarworks.umt.edu/etd/5509>

This Thesis is brought to you for free and open access by the Graduate School at ScholarWorks at University of Montana. It has been accepted for inclusion in Graduate Student Theses, Dissertations, & Professional Papers by an authorized administrator of ScholarWorks at University of Montana. For more information, please contact [scholarworks@mso.umt.edu](mailto:scholarworks@mso.umt.edu).



Maureen and Mike  
MANSFIELD LIBRARY

---

Copying allowed as provided under provisions  
of the Fair Use Section of the U.S.  
COPYRIGHT LAW, 1976.

Any copying for commercial purposes  
or financial gain may be undertaken only  
with the author's written consent.

---

University of  
Montana



A Spiral Model Approach to  
Microprocessor Laboratory System Design

by

Zuyi Chen

B.A., Hangzhou University, 1983

M.A., University of Montana, 1989

Presented in Partial Fulfillment of the Requirements

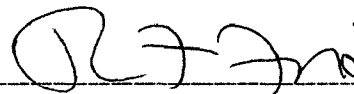
for the Degree of

Master of Science

University of Montana

1992

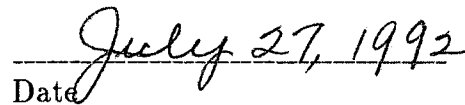
Approved by:



Chairman, Board of Examiners



Dean, Graduate School



Date

UMI Number: EP40973

All rights reserved

INFORMATION TO ALL USERS

The quality of this reproduction is dependent upon the quality of the copy submitted.

In the unlikely event that the author did not send a complete manuscript and there are missing pages, these will be noted. Also, if material had to be removed, a note will indicate the deletion.



UMI EP40973

Published by ProQuest LLC (2014). Copyright in the Dissertation held by the Author.

Microform Edition © ProQuest LLC.

All rights reserved. This work is protected against unauthorized copying under Title 17, United States Code



ProQuest LLC.  
789 East Eisenhower Parkway  
P.O. Box 1346  
Ann Arbor, MI 48106 - 1346

## TABLE OF CONTENTS

Table of Contents	i
List of Illustration	ii
1 Overview	1
2 The Spiral Model	2
2.1 Spiral Model Preview	2
2.2 Round One of Spiral Model	5
2.2.1 Requirements	5
2.2.2 Specifications	6
2.2.3 Development	8
2.2.4 Evaluation	10
3 Further Development	11
3.1 Development - Round Two	11
3.2 Evaluation - Round Two	13
3.3 Development - Round Three	13
3.4 Evaluation - Round Three	13
4 The Products	16
4.1 Lab Use Information	16
4.2 EVB Server Software	16
4.3 Lab Exercises and Instructor's Notes	17
4.3.1 Output Ports	18
4.3.2 Input Port	20
4.3.3 Timing Control	20
4.3.4 Hardware Setup	21
4.3.5 Software Interrupt	21
4.3.6 Timer and Output Compare Functions	22
4.3.7 Polled and Single Interrupts	22

4.3.8 Inter-process Synchronization	23
4.3.9 Practical Application	23
5 Conclusion	24
Acknowledgement	27
Reference	28
Appendix A: EVB server source files	
Appendix B: Lab Use Information Manual	
Appendix C: Lab Assignment Manual	
Appendix D: Lab Instructor's Manual	
Appendix E: Hardware Diagram Manual	
Appendix F: Shell Program Listings	

## List of Illustrations

Figure 1. Waterfall Model	3
Figure 2. Spiral Model	4
Figure 3. Spiral Model Round 1	7
Figure 4. Spiral Model Round 2	12
Figure 5. Spiral Model Round 3	14
Figure 6. Comparison between Assignments and Text Topics	25
Figure 7. Hardware Cost	25



# 1 Overview

This graduate project treats and analyzes a system design problem involving hardware, software, interfacing, and instructional elements, as a software engineering problem to be solved via the risk-driven spiral model described by Barry Boehm [1]. The goal of the project is to create a lab environment that provides a student or working engineer hands-on experience with microprocessors, computer architecture, simple device interfaces and assembly software programming. It is anticipated that this environment will be integrated with the revised University of Montana CS231-232 “Computer Architecture and Assembly Language” course sequence. The products of the project include a set of “lab use” information, lab exercises, instructor’s notes, hardware diagram manual, extra software to make lab procedures easier, and a summary of the cost of setting up such a lab.

The discussion in Chapter 2 and 3 of this report focuses mainly on the process used to complete this project, i.e., the system design activities. Chapter 4 describes the lab use information and lab exercises, and Chapter 5 summarizes the lab costs. The specific products resulting from the activities are also included as appendices.

## 2 The Spiral Model

Looking at the history of software life-cycle process models, two important models – waterfall development and spiral development – have been widely used to solve system design problems involving computer software. The waterfall model, developed over years since the 1950's, describes software system engineering as a fairly rigid sequence of stages, including system feasibility, software plans and requirements, product design, detailed design, code, integration, implementation, operations and maintenance. The resulting one way flow, as shown in Figure 1, looks like a waterfall. The spiral model has evolved from the waterfall model to describe a more flexible and realistic approach to software and system engineering. It is described in more detail below.

### 2.1 Spiral Model Preview

The spiral model was developed from the waterfall model by Boehm[1]. It is based on experience with use of the waterfall model in real project development. It can accommodate most of the proposed variations on the waterfall model, and treats them as special cases. As illustrated in Figure 2, the basic idea is that the quadrants represent general types of activities. The flow through these activities is non-linear; many activities are repeated several times, as indicated by the spiral, as a system is refined. The spiral model is risk-driven in nature. The more cycles and steps of the spiral model completed, the more cost, therefore, the more risks associated with the system being developed.

Figure 2 shows the details of applying the spiral model in the Microprocessor Lab System design. The cumulative cost for the steps accomplished to date is represented

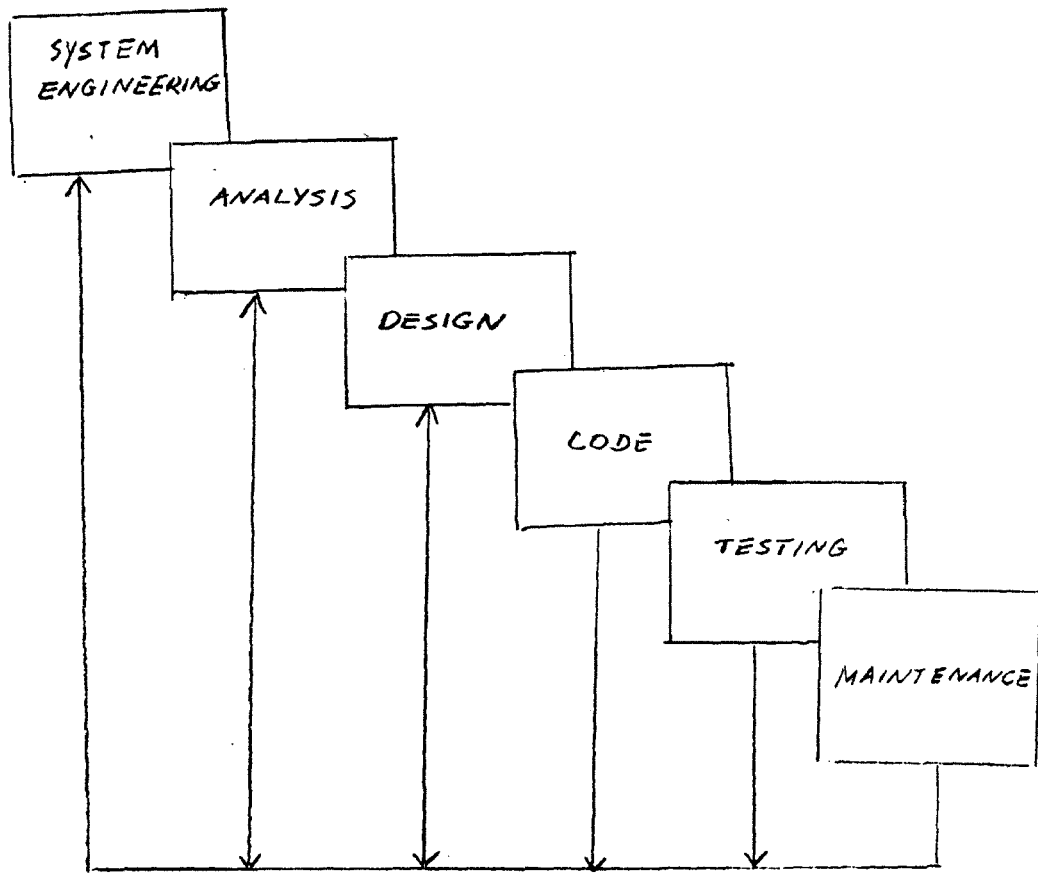


Figure 1: Waterfall Model

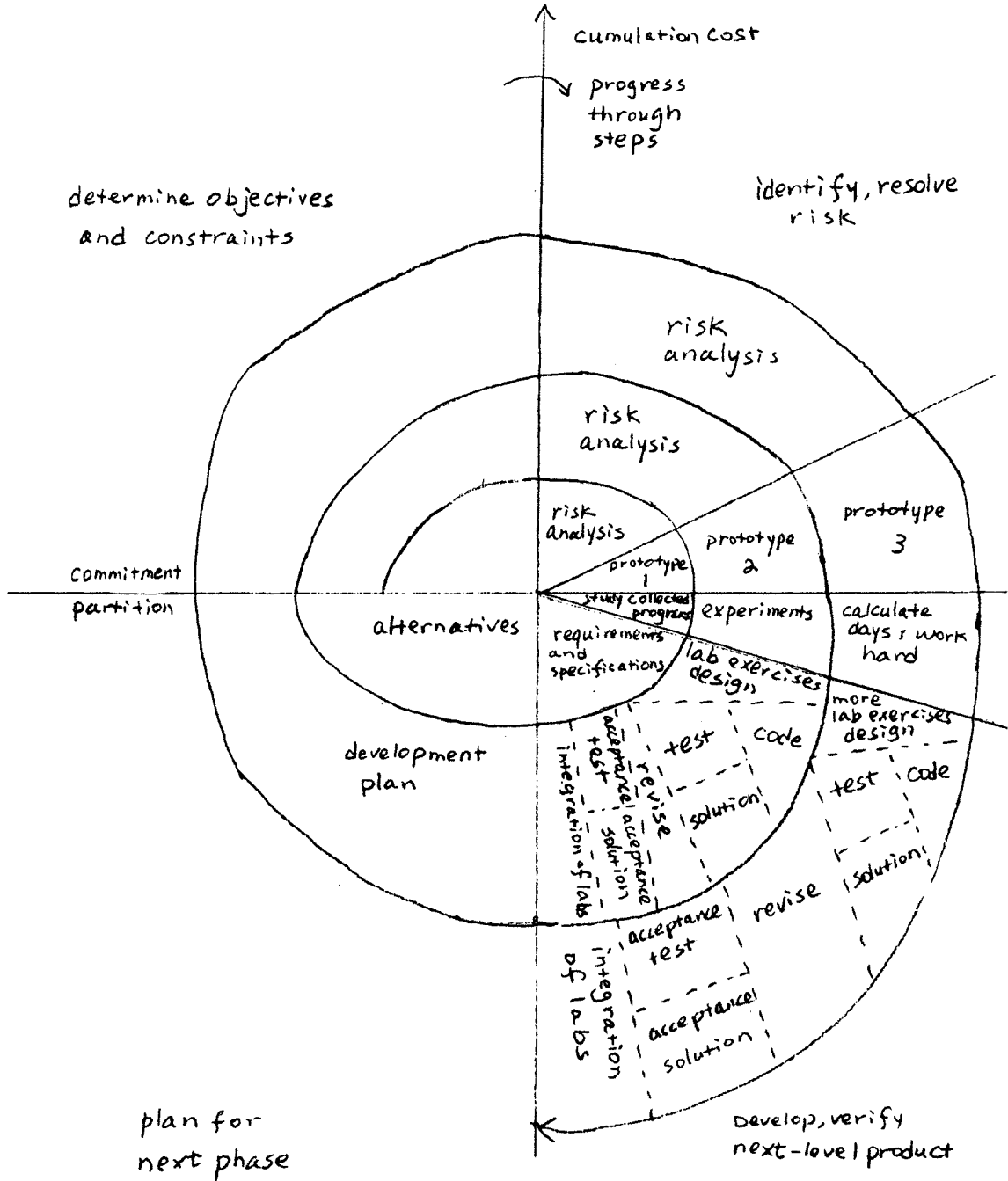


Figure 2: Spiral Model

by the radial dimension, and the progress made in completing each cycle of the spiral model is represented by the angular dimension in the figure. In the development of any system there can be many cycles, or **rounds**, that represent different levels of system elaborations. A typical round of the spiral model begins with the objectives of the system product being elaborated, or re-evaluated, including such aspects as functionality, performance and ability to suit the different requirements. The round continues by identifying the constraints of the system being developed, estimating and resolving the risks, developing and verifying the next-level products. The round ends in planning the next phases of development. As shown in Figure 2, the Microprocessor Lab System design has thus far involved three rounds of development, which are discussed in detail in the following sections.

## **2.2 Round One of Spiral Model**

Like any other design, the Microprocessor Lab System design starts with identification of basic goals, requirements, and constraints. Another important part of the first round is the accumulation of information related to the system being designed, or **target system**, ranging from definitions and terminology to hardware and software components. This round involved a single designer (me) working over a period of two months. The goals and requirements are discussed in the following subsections; related definitions and terminology, such as host, target, assembler vs. cross-assembler, upload, download, etc., are given in the Lab Use Manual in Appendix B.

### **2.2.1 Requirements**

This project involves designing a complete “lab experience” involving microprocessors, computer architecture, simple device interfaces, and assembly language pro-

gramming. The experience is to be based on lab exercises that use hardware and software in a microprocessor lab, operated with a small budget. The exercises are assumed to be integrated with an instructional program, such as the architecture and assembly language topics to be covered in the newly revised University of Montana CS231-232 semester sequence, or a comparable independent study course. The lab exercises should reinforce the “theory” covered in the instructional program. The labs require “hands-on” experience, in which a student typically constructs a **hardware circuit**, connects it to the microcomputer system, and executes software on the micro-system to produce tangible results on the circuit. Therefore, the labs emphasize the practice and details of how to get a primitive microcomputer system, or **embedded system**, to control external devices.

The mix of theory and practice is essential. In the advanced computer world, there is a big difference between the students who read theory only and those who can combine theory with practice. The former lack practical experience desired in the real world, and are far less competent in the computer world. Even a pure teaching job requires some practical experience in one field one or another. Although it is impossible to offer practice in every computer course, it is feasible to add practice in the computer architecture and assembly language class, and cheap enough for most schools and students to afford. Being able to see the system working, by watching its circuits in action, provides a tangible measure of success missing in other instructional programs. This kind of activity will surely help increase students’ interest in the related topics. All these above provide the basis for doing this project.

### **2.2.2 Specification**

If possible, the lab exercises are to be done on the Motorola M6800/68000 family

**Objectives** Design a sequence of 6 or 7 lab exercises to provide experience with microprocessors, computer architecture, simple device interfaces, and assembly language programming.

**Constraints** The labs to be designed by one person in a period of 2-3 months; minimum cost should be spent on the hardware for the labs.

**Risk** The products may not be useful, due to “cost” or failure to match instructional goals; time and energy may be wasted.

**Risk resolution** Collect and analyze a set of existing assembly programs on existing low-cost lab facilities.

**Risk resolution results** Most of the collected programs were not appropriate for the microprocessor labs, but analysis helped in identifying the basic lab constraints.

**Plan for next phase** Develop a new set of lab exercises and supporting lab information.

**Commitment** Implement next phase.

Figure 3: Spiral Model Round 1

of microcomputers, using one of several low-cost evaluation boards available from Motorola. The lab assignments are to cover both hardware and software aspects of microprocessor systems, as recommended by both ACM and IEEE instructional guidelines. A pure hardware focus may be appropriate for electronic engineering students, but isn't sufficient for those students with software engineering interests. On the other hand, a pure software focus would miss key architecture and interfacing details. Thus, it seems proper to combine topics of both hardware and software in the lab assignments.

### **2.2.3 Development**

The M68HC11EVB was selected for use as the microprocessor tool in Round 1, based on its low cost and appropriateness for education purposes. A kit with the M68HC11EVB board, manuals, and monitor software costs about \$70. The layout of the hardware, such as processor, memory and other chips, is simple and easy to understand. The software required to interact with the board is compatible with PCs, Macintoshes and Unix workstations. The monitor EPROM can be removed and carried around. The board is small enough that a student can carry it around in a back pack between home and labs. Finally, the board and related software are cheap enough that each student can buy his or her own.

The features of the EVB include the following.

1. Low cost tool containing an MC68HC11 microcomputer (M6800 instruction set)
2. On-line assembler/disassembler
3. Support for host computer downloading
4. On-board monitor with debugging support



5. MC68HC24 Port Replacement Unit (PRU) for MCU I/O support (i.e., for device interfacing)
6. MC6850 Asynchronous Communications Interface Adapter (ACIA) for host/target communication support
7. Special hardware registers and signal pins to support device interfacing and communication

To run a program on the EVB, a student can either use a terminal to enter and assemble code directly on the EVB, or use a host machine to create the program, cross-assemble it to create an S-record file, then download the S-records to the EVB. In either case the student can use the BUFFALO monitor program to execute the assembled program and monitor its execution. The project described here assumes that all lab assignments will use the host machine approach, and that the host will be a workstation or PC.

The starting point for the design of the lab exercises is the set of topics described in the course on microprocessor interfacing and communication by the IEEE Computer Society Curriculum Committee. In general, the goal for the lab exercises is to introduce students to modern microcomputer architecture, programming, and the interaction of computer software and hardware to realize control of simple external devices.

As stated in Figure 3, the primary risk in Round 1 of the spiral model is that whatever lab exercises are developed might turn out to be inappropriate for actual use. To minimize costs in Round 1, I started with an existing set of assembly programs and software for the 68HC11EVB, instead of starting from scratch.

The existing assembly programs and software came from various assignments and student projects from advanced courses, such as Embedded Systems and Parallel Processing. The plan was to modify these programs, to see if they could be used as lab assignments. Several days were spent on studying the results of the existing programs and modifying the programs to suit the new purposes.

### **2.2.4 Evaluation**

Roughly, the prototype lab Round 1 consists of a 68HC11EVB, the first set of lab exercises, and the extra hardware required to build the circuits used by the EVB. The EVB seemed adequate as a platform; the set of exercises and circuits needed more careful analysis. After the programs were modified and potential lab assignments defined, I started to match the labs with project requirements and specifications. Four of these labs were considered valuable – their ideas were retained for subsequent development. Most of the other labs were rejected because they didn't match the requirements and specifications, or they simply didn't serve as good lab exercises. For example, some labs were too hard or too long to fit as single units, focused on material outside of the topics of interest, or used circuits too complex or expensive for each student to duplicate. Therefore, this first system prototype had to be refined and extended to include new labs. There were two alternatives: to collect and adapt more assembly programs from other sources, or to design new labs from scratch. I chose the latter, and planned for the next round of design and implementation. This ended the first round of the spiral model, and started the second.

## 3 Further Development

### 3.1 Development - Round Two

The goal of the second phase of development was to create several new labs from scratch, particularly to focus on the concept of interrupt handling and its hardware and software details. In general, these labs must

1. provide students with hands-on experience in setting up input and output devices;
2. show students how microprocessor internal units and external chips relate to each other; and
3. show students how low-level computer software interacts with computer hardware to produce internal state changes and externally visible results (e.g., via lights, audio generators, character display, etc.).

Designing labs from scratch takes more time than designing labs based on the existing programs. For example, low-level programs can yield unexpected results with only subtle changes, so care must be taken to assure that the basic program used by each lab was reliable and predictable. After many hours of development and testing, a basic framework for key topics, such as interrupt handling, was developed, permitting the development of several interrelated labs. This major job having been done, I started to put together a second complete set of lab exercises, along with the solutions and supporting software. The result is System Prototype 2. Figure 4 summarizes Round 2 of development.

**Objectives** Start from scratch to design and implement 6 or 7 labs with emphasis on interrupt handling; design support software.

**Constraints** Reliability of interrupt handling on the EVB; portability of support software both to a UNIX workstation and to a PC; match between labs and course topics.

**Risk** The interrupt handling techniques on 68HC11 may not be appropriate for lab exercises for novices.

**Risk resolution** Read reference books; experience with interrupt techniques.

**Risk resolution results** Figured out how the interrupt technique works and a scheme to use it in several labs.

**Plan for next phase** Include interrupt handling in several labs; decide other topics for the labs; put together lab exercises, develop the solutions to the exercises.

**Commitment** Develop project prototype.

Figure 4: Spiral Model Round 2

## **3.2 Evaluation - Round Two**

Upon reflection the second system prototype was found to be too limited. Although all labs were pertinent to the study topics and could be assembled with reasonable cost, the set of the labs was too rigid. I asked myself the following questions: What if some labs turn out to be too easy or too difficult for students? What if the instructor of the course doesn't like a particular lab? What if a particular lab exercise doesn't match any of the topics covered in a particular course? Does the instructor have other choices? He is supposed to, right? Right! The development of alternatives and instruction flexibility triggered the third round of the spiral model, which is summarized in Figure 5 and described below.

## **3.3 Development - Round Three**

In addition to the base set of labs included in System 2, I realized more labs should be prepared to accommodate unexpected change. I decided to increase the number of lab exercises. The goal was to double the number of labs, and 17 lab exercises were eventually developed. The development problems and risks in this round were similar to these in Round 2; the assignments had to be appropriate, the solutions had to be accurate, and software had to be reliable. The result is described in Figure 5.

## **3.4 Evaluation - Round Three**

Upon completion of Round 3 of system design and implementation, I found that the products resulting from the activities of the system matched original goals quite well. The products include the exercises on key instructional elements, such as timing control, interrupt mechanisms, microprocessor internal circuits such as different I/O

**Objectives** Design twice as many labs as in Round 2 to provide alternatives.

**Constraints** Limited development time.

**Risk** The interrupt handling techniques on 68HC11 may not be appropriate for lab assignments.

**Risk resolution** Hard work, long hours.

**Risk resolution results** 17 labs designed without extending lab hardware/software requirements.

**Plan for next phase** Obtain feedback from actual use, then revise accordingly.

**Commitment** Revise the products; write documentation and the summary report on lab system design.

Figure 5: Spiral Model Round 3

ports, MCU timers, and others. In most of the lab assignments, students are required to understand how the “shell” of a given assembly language works, calculate things like instruction cycle times, and then modify or extend the given code. Some labs require students to set up specific circuits from examples or diagrams that are provided. The collection of lab exercises is not perfect, but I assume this system will be further evaluated and modified in subsequent phase of development. It is sufficiently well developed to allow “prototype testing”, in the form of actual use in an instructional context. The labs are expected to be used in CS231-232 in 1992/93, with feedback being used to direct further development.

## **4 Products**

The labs cover a variety of computer architecture topics, such as free-running timer, output compare functions, single and polled interrupt mechanisms, software interrupt mechanisms, real-time interrupts, various output ports, and timing control via instruction cycles. Extra software and general lab use information designed to complement all the labs are shown in Appendix A and B. The complete sequence of 17 labs is shown in Appendix C, along with an instructor's manual containing rationale and solution notes in Appendix D. Circuits used by the labs are given in Appendix E, and the program shells for the labs are listed in Appendix F. Each of these parts of the Microprocessor Lab System design is described briefly below.

### **4.1 Lab Use Information**

Lab use information and extra software (Appendices A and B) are provided to make it easier for students to understand lab procedures and to master the required topics. The information includes how to connect the EVB board to a host, how to use the BUFFALO monitor, how to download S-records from the host to the EVB, how to offload data from the EVB to the host, and how to use the EVB Server software package.

### **4.2 EVB Server Software**

It is assumed that course work will be hosted on PCs and Unix workstations. PC and workstation versions of an "EVB Server" package have been implemented for this purpose. EVB Server was derived from a version of such software that I implemented earlier as a project for CS495 Embedded Systems, Fall 1991/92. EVB Server is an



interfacing package that helps an EVB board to interact with a host machine, which can be either a PC or a Unix workstation. EVB Server is a menu driven system that combines several useful functions. Menu options allow the user to download S-records from the host to the EVB board, upload data (memory contents) from the EVB board to the host, and connect from the host to the board. Besides these, the user can also invoke the cross-assembler on the host to produce an S-record file, convert a file with hexadecimal contents to the decimal contents, edit a file, and display directory content. By combining these functions, EVB Server simplifies program development, during which program assembly, downloading and data offloading are performed again and again. EVB Server is written in C. Two slightly different versions have been written to account for differences between PCs and workstations. The major differences between the two versions are the communication port setup.

### **4.3 Lab Exercises and Instructor's Notes**

The labs are ordered in terms of topics and level of difficulty to match accompanying instruction. The lab exercises are described in detail in the lab manual (Appendix C), the answers to the exercises plus comments are given in the instructor's manual (Appendix D), and the circuits diagrams for the labs are provided in the hardware diagram manual (Appendix E). In the lab manual, each lab is described in a form that includes a problem title, a list of topics required and reinforced by the lab, the instructional purpose for the lab, and the lab problem specification. Typically the specification also includes the shell of an assembly language program to be used in the exercise. The instructor's manual includes similar information for each lab, along with a description of the background required by the lab and one or more programs and/or circuits that implement a correct solution.

Each lab exercise utilizes external input and/or output device(s), such as lights, buzzers, digit displays, character displays, etc. Each solution shows how the software should interact with the hardware to produce the control specified by the lab.

Students experience the following in the collection of labs.

1. They “wire-up” connections for input and output devices.
2. They set up timing and external signal control to implement real-time hardware control, via both interrupts and polling.
3. They observe an interface between the board and outside devices that produces both audio and visual results.
4. They witness concurrent yet synchronized execution of programs on two different boards.

In addition, students must define and implement their own control project independently, as the final exercise. The details of the assignments are based on specific 68HC11EVB details explained below.

### **4.3.1 Output Ports**

There are five parallel input and output ports in the EVB: Port A, Port B, Port C, Port D and Port E. Each bit in each port is connected to EVB header pins, making external connections very easy. Ports A through D can be used for general-purpose output. The 8-bit Port A is configured for general-purpose I/O or for timer or pulse accumulator functions. Bits 0 - 2 are used for Input Compare, therefore, cannot be used for output; bits 3 - 7 can be used for output compare in the timer architecture or for general-purpose output. When used for the latter, bits 3 - 6 are used directly, but bit 7 of the Port A data direction register must be written with 1.

The 8-bit Port B is a fixed-direction output port. It is used for general-purpose output and for simple strobe output.

Port C is a complex port, because it involves the bi-directional I/O. Pins 9 - 16 correspond to bits 0 - 7 of Port C. In order for Port C to be used for output, an 8-bit Port C data direction register must be first written with 1 on every bit.

Port D is a 6-bit bi-directional I/O port. Bits 0 - 5 of Port D correspond to the EVB header pins 20 - 25. Bit 0 serves as receive data pin, and always reads; and bit 1 serves as transmit data pin, and always writes. They are usually not used as outputs. Bits 2 - 5 are used either for general-purpose output or for the on-chip synchronous SPI (Serial Peripheral Interface) system. When used for the former, bits 2 - 5 of the corresponding Port D data direction register should be written with 1's.

To let students get more familiar with the output port topics covered in the course, Lab 1.0 covers output via each of Port A, B, C and D. The software drives data out of the Port A, B, C and D to bar graph LEDs to control the state of the LEDs. Each of the output pins of the ports are turned on in order of Port C, D, A, and B from the most significant to the least significant bit. A program shell for the software driver is provided to the student with details that must be filled in by the student to make the program executable.

### 4.3.2 Input Ports

The 8 pins of Port C can be used as general-purpose input, when the Port C data direction register is written with 0 to change the data direction for input. Input is more complex than output. In order to read in coming data on Port C, the parallel I/O control register needs to be alerted as to the arrival of the data. On the EVB, a control pin, STR-A, and a polling loop are usually set up to implement checking for incoming data. Lab 2.0 sends data output from Port B as input to Port C. Again, a program shell for the input driver is provided.

### 4.3.3 Timing Control

Microcomputers are often used to control real time. Some real world electrical devices are controlled by inputs by means of delay loop linked to real time intervals. The speed at which assembly instructions are executed is measured in terms of cycles. The EVB MCU is a 2-Mhz CPU, which means that the CPU executes 2,000,000 instruction cycles per second. It takes several cycles to execute each instruction, typically between 2 and 4, but ranging much higher for some complex instructions. Cycle information can be obtained from the 68HC11 manual, and is also typically printed on the assembly listing. A real-time execution interval is measured by summing the number of cycles in a section of code, and multiplying that sum by seconds per cycle on the MCU.

In software it is relatively easy to build code to delay N cycles, then to compute the real-time delay M by the technique described above. This approach can be used for simple timing control, such as to operate the lights in a traffic signal. In Labs 3.0 and 4.0, a “traffic light” is controlled for specified time intervals by software on the EVB board. Given a program shell, appropriate time delays must be created by

the student for each of the green, yellow and red lights, (i.e., green, yellow and red LEDs).

Lab 5.0 involves similar control of an audio device – a buzzer. This exercise demonstrates how the software can vary the “frequency” of output to a piezo buzzer to produce music tunes.

#### **4.3.4 Hardware Setup**

To have students gain hands-on experience, some hardware assignments are prepared. As part of Labs 6.0 and 7.0 students have a chance to set up wiring connections between LED bar graph and the EVB. Lab 8.0 involves displaying digits on a 7-segment LED. Students are given the electronic schematic for an external 7-segment display and a circuit in which the connections are scrambled deliberately; students are required to determine the correct connection by trial and error. Hopefully, this kind of activity will help students to understand how electricity is directed from the EVB to the external circuit. Lab 9.0 is an alternative assignment using the 7-segment LED.

#### **4.3.5 Software Interrupt**

Interrupt handling techniques are important ways to realize control. For example, if an exception is detected, the regular routine must be interrupted to give way to the interrupt routine, which handles the exception, then returns to the originally operating routine. Twenty types of vectored interrupts are described on M68HC11EVB User’s Manual [10], including a “Software interrupt”. Lab 10.0 is designed to show students how interrupt handling is implemented on the 68HC11, based on the software

interrupt.

### 4.3.6 Timer and Output Compare Functions

The EVB's MCU physical time is kept by a 16-bit free-running counter, which cannot be interrupted. This is the main element of the timer architecture of M68HC11, and is one of the most flexible parts of a single-chip microprocessor. The timer can produce a sine wave or other precisely timed pulses, which are used in touch-tone telephones, tape recorders and so on.

The output compare function is also an important element of the timer architecture of M68HC11. The output compare function is used to set an action to happen at specific time. The output compare register is compared to the free-running counter at every execution cycle. When the current count of the free-running counter matches the value held in the output compare register, an output is generated automatically. There are five output compare registers used as vectored interrupts.

Other elements of the timer architecture include timer control registers, timer interrupt masks, timer interrupt flag registers, timer output compare registers, etc. Lab 11.0 involves use of the timer and output compare register 5 and other registers described above.

### 4.3.7 Polled and Single Interrupts

Timing control can be realized by a single interrupt or polled interrupt. For a single interrupt, an interrupt service routine is set up so each time the interrupt occurs the service routine is called automatically. In a polled interrupt, however, there is no interrupt service routine set up. The way to find out if an interrupt

occurs is to set a polling loop to periodically check if the interrupt has occurred. To give students a variety of assembly interrupt experience, polled interrupt with output compare register 2 is used in Labs 12.0 and 14.0, and the single interrupt version with output compare register 5 is used in Labs 13.0 and 15.0.

### **4.3.8 INTER-PROCESS SYNCHRONIZATION**

Parallel processing and synchronization are very important techniques. Inter-processor communication is a form of synchronization used to allow one processor to send/receive data to/from another processor. Between a sender processor and a receiver processor, synchronization is required to assure that the sender will send data only when the receiver is ready to receive. One way to achieve this kind of synchronization is to calculate both processors' execution speed, then estimate how fast the receiver can receive data to determine how fast the sender can send data. Lab 16.0 is an exercise on synchronization that demonstrates how the execution of two EVB's can be coordinated.

### **4.3.9 Practical Application**

A big display screen posted by the road with current time or temperature displayed on it, or a small one on a vending machine that asks customers to insert money by displaying a string of characters moving from one end to another, are both controlled by similar techniques. Lab 17.0 shows students how software can drive an external multiple-character display device. The lab also gives students a chance to write procedures that produce characters to be displayed on a LCD-II display screen.

## 5 Conclusion

The Microprocessor Lab System, Version 3, resulting from the third round of development, is ready to be used in an instructional program. As a whole, the lab assignments relate closely to the topics that would be covered in a course such as “Computer Architecture and Assembly Language”, or a comparable independent study. A comparison of the collection of the labs with an example text book, “Microprocessor System Design” [2], is given in Figure 6. The hardware controlled by the software is interesting enough to draw students’ curiosity. The most fun moment in doing the exercises is to see the devices working correctly. Watching devices being turned on and off is attractive enough to lead students to experiment more on the related topics, thus helping them understand the topics better.

A careful assessment of possible course and topic coverage reveals that some topics in the typical text book are not covered in the current collection of labs. For example, Chapter 3 of the text discusses program design, which is not addressed in the labs. Testing of the collection of exercises in a specific course is required to indicate whether such omissions are major design flaws that need to be corrected by the addition or modification of lab exercises.

An assessment of the total cost for the hardware and external devices for the labs is shown in Figure 7. In addition to his or her own EVB board, or access to “shared” boards in a central lab, each student would need the hardware items listed in the figure. If students are required to buy the complete set, including EVB, the total cost for each student is estimated at \$146.66 (not including the cost of a host). On the other hand, the Computer Science Department could provide all these facilities, including a set of hosts and EVB’s “dedicated” to support the lab, but shared various students. For example, four host/EVB’s might serve a class of twenty-five students.



<u>Text Book Chapter</u>	<u>Assignments</u>
2	6, 7
2, 4	1, 3, 4, 5, 8, 9, 17
5, 9	2, 16
6, 8	11, 12, 13, 14, 15, 16

Figure 6: Comparison between Assignments and Text Topics

1 EVB board	\$69.00
14 transistors	$\$0.59 \times 14 = \$8.26$
14 1-k resistors	$\$0.08 \times 14 = \$1.12$
2 330-ohm resistors	$\$0.08 \times 2 = \$0.16$
1 potentiometer (variable resistor)	\$0.49
1 7-segment LED	\$1.79
1 MAN6610 (14-segment) LED	\$1.99
4 10-segment bar graph LED	$\$2.99 \times 4 = \$5.98$
1 red LED	\$0.40
1 green LED	\$0.40
1 yellow LED	\$0.40
1 piezo buzzer	\$1.75
1 LCD-II display (HD44100H/HD44780A00)	\$5.00
20 ft. of wire	\$3.49
30 ft. of thin wrap wire	\$1.43
1 straight through line	\$10.00
1 bread board	\$15.00
1 power supply (if it can't draw electricity from host machine)	\$20.00

Figure 7: Hardware Cost

With this approach, the estimated hardware cost for the department is \$586.64 (not including the hosts). All prices shown are for individual retail purchase price (e.g., from Radio Shack); if components can be purchased in quantity, most of the prices would be reduced dramatically.

The activities involved in this project – designing a collection of labs, creating the supporting information and solutions, and creating the total lab environment – are typical of those a teacher must experience in his teaching career. This has been a precious experience to me. The teaching profession has always attracted my interest. If some day I am lucky enough to have this profession as my career, the practical activity that I have experienced in doing this project will serve as a wonderful exercise for it.

## **Acknowledgement**

Special thanks go to Dr. Ray Ford of University of Montana, who supplied valuable advice and suggestions to the project. I would also like to extend my thanks to Sixing Gu of University of Montana, who supplied many suggestions.

## REFERENCE

1. Boehm, Barry W., "A Spiral Model of Software Development and Enhancement", *IEEE Computer*, May 1988.
2. Clements, Alan, *Microprocessor System Design (68000 Hardware, Software, and Interfacing)*, 2nd ed., PWS-Kent Publishing Co., Boston, 1992.
3. Fan, Hong, Embedded System Project, CS495, Univ. of Montana, Fall 1991/92.
4. Ford, Ray, IPCsnd.asm/IPCrcv.asm Assignment, CS580, Univ. of Montana, Spring 1990/91.
5. Gu, Sixing, Embedded System Project, CS495, Univ. of Montana, Fall 1991/92.
6. *HCMOS Single-Chip Microcontroller*, Motorola, Inc., 1988.
7. *MC68HC11A8 Programming Reference Guide*, Motorola, Inc., 1990.
8. *M68HC11 Reference Manual*, Prentice Hall, Englewood Cliffs, New Jersey, 1989.
9. Lipovski, G. J., *Single- and Multiple-Chip Microcomputer Interfacing*, Prentice Hall, Inc. Englewood Cliffs, New Jersey, 1988.
10. *M68HC11EVB Evaluation Board User's Manual*, Motorola, Inc., 1986.

**Appendix A**

**Computer Architecture and Assembly Language**

**EVB Server Program Listing**

**ZUYI CHEN**

**July, 1992**

**Computer Science Department**

**University of Montana**

## TABLE OF CONTENTS

Table of Contents	i
EVBSERV.C	1
EVBWSERV.C	16

```

/*****
/*  EVBSERV.C
/*
/*  Zuyi Chen
/*  The Computer Science Department
/*  The University of Montana
/*  Missoula, Montana
/*
/*  The package EVBSERV is designed to provide the EVB/Buffalo
/*  users with convenience in interacting a PC host with the
/*  EVB board. It contains the following functions:
/*  setup(); ready(); receive(); prompt(); menu(); downld();
/*  offld(); edit(); show(); crossasm(); connect(); dir();
/*  convert().
/*
/*  October 20, 1991
/*  modified in July, 1992
/*
/*  Version 2.0
/*
/*****

```

```

/*****
/* You are welcome to copy and distribute unmodified source code
/* to other parties provided you include this notice, together
/* with the original file header, as a part of the file. You may
/* modify the source code for your own purpose, but any modified
/* code must carry the date of modification and indicated by whom
/* modified, with a general statement as to the purpose of the
/* modification.
/*****

```

```

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <dos.h>
#include <conio.h>
#include <bios.h>
#include <mem.h>

```

```

#define CONF 0xE3
#define COM1 0
#define BLOCK 2048
#define RS232 0x14
#define DATAR 0x100
#define DATAS 0x2000

```

```

#define B300 0x40
#define B1200 0x80
#define B2400 0xA0
#define B4800 0xC0
#define B9600 0xE0

```

```
#define NOPARITY 0x00
#define EVEN 0x18
#define ODD 0x08
```

```
#define WORD7 0x02
#define WORD8 0x03
```

```
#define STOP1 0x00
#define STOP2 0x40
```

```
#define PORT 0
/* #define BAUD 2400 */
#define WORD 8
#define PARITY 0
#define STOP 1
```

```
int BAUD;
int Delay = 6000;
int uart_rbr1 = 0x03f8;
char command[40];
char buf[50000];
char c, filename[20], edit_filename[30];
int i, j, k, n;
FILE *fp;
union REGS regs;
unsigned long staddr = 0xB0000000;
```

```

/*****
/*  setup()
/*
/*  Zuyi Chen
/*
/*  setup() will set up the communication port with
/*      port 0
/*      baud rate 9600
/*      word size 8
/*      parity check 0
/*      stop bit 1
/*  The port setup is easily modified by changing the define
/*  statements at the beginning of the source code.
/*
/*  October 20, 1991
/*
/*  Version 1.0
/*
*****/

```

```
setup(int port, int baud, int word, int parity, int stop)
{
unsigned char setup;

/* set up port */
setup = 0;
```



```
if(port != 0 && port != 1)
{
    printf("\nPort inappropriate\n");
    exit(1);
}

/* set up baud rate */
switch(baud){
    case 300:
        setup |= B300;
        break;
    case 1200:
        setup |= B1200;
        break;
    case 2400:
        setup |= B2400;
        break;
    case 4800:
        setup |= B4800;
        break;
    case 9600:
        setup |= B9600;
        break;
    default:
        printf("\nBaud rate inappropriate\n");
        exit(1);
}

/* set up word size */
if(word==7)
    setup |= WORD7;
else if(word==8)
    setup |= WORD8;
else
{
    printf("\nWORD bits inappropriate\n");
    exit(1);
}

/* set up parity check bit */
if(parity==0)
    setup |= NOPARITY;
else if(parity==1)
    setup |= EVEN;
else if(parity==2)
    setup |= ODD;
else
{
    printf("\nParity check inappropriate\n");
    exit(1);
}

/* set up stop bit */
if(stop==1)
```

```

    setup |= STOP1;
else if(stop==2)
    setup |= STOP2;
else
{
    printf("\nSTOP bit inappropriate\n");
    exit(1);
}

```

```

regs.h.ah = 0;
regs.x.dx = port;
regs.h.al = setup;
int86(RS232, &regs, &regs);
}

```

```

/*****/
/*  ready() */
/* */
/*  Zuyi Chen */
/* */
/*  ready() will check if a receive or send is ready. */
/* */
/*  October 20, 1991 */
/* */
/*  Version 1.0 */
/* */
/*****/

```

```

int ready(int statusbit){
    regs.h.ah=3;
    regs.x.dx=COM1;
    int86(RS232, &regs, &regs);
    return (regs.x.ax & statusbit);
}

```

```

/*****/
/*  receive() */
/* */
/*  Zuyi Chen */
/* */
/*  receive() allows the host to receive a character from the */
/*  EVB board. */
/* */
/*  October 20, 1991 */
/* */
/*  Version 1.0 */
/* */
/*****/

```

```

char receive(){
    regs.h.ah=2;
    regs.x.dx=COM1;
    int86(RS232, &regs, &regs);
}

```



```

PACKAGE\n\n\n");
p   r   i   n   t   f   (   "
*****\n");
printf(" * A.   Cross-assemble the assembly program
*\n");
printf(" * B.   Download S-record from host to EVB board
*\n");
printf(" * C.   Turn the host to a terminal for EVB board
*\n");
printf(" * D.   Offload data from EVB board to the host
*\n");
printf(" * E.           Edit a file using the existing editter
*\n");
printf(" * F.   Show the file content
*\n");
printf(" * G.   Convert hex data file to decimal file
*\n");
printf(" * H.   Display the working directory
*\n");
printf(" * Q.   Quit                               *\n");
p   r   i   n   t   f   (   "
*****\n");

printf("\n\n\n\n   Enter your Choice > ");

}

```

```

/*****/
/*  downld()                               */
/*                                          */
/*  Zuyi Chen                               */
/*                                          */
/*  downld() will download the S-record with the input record */
/*  name from the host to the EVB board.   It is important to */
/*  reset EVB board as indicated.          */
/*                                          */
/*  October 20, 1991                       */
/*                                          */
/*  Version 1.0                             */
/*                                          */
/*****/

```

```

downld(){
int i;
char cmd[40];

/* This while loop will exit upon the input 'N' or 'n' */
while(1){
printf("\n");
/* store 'type' to cmd */
strcpy(cmd, "type ");

/* accept the input */

```

```

printf("\nReset EVB board; and \n");
printf("Enter the name of the S-record to be downloaded > ");
gets(filename);
/* store filename and 'COM1' to cmd */
strcat(cmd, filename);
strcat(cmd, "> COM1");
strcpy(command, "load t\r");

prompt();
i=0;
/* send char to EVB board */
outport(uart_rbr1,command[i]);
for(k=0; k<Delay && !ready(DATAS); k++)
    ;
while(command[i] != '\r'){
/* send char to EVB board */
    outport(uart_rbr1,command[++i]);
    for(k=0; k<Delay && !ready(DATAS); k++)
        ;
}

/* call system utility 'system' */
system(cmd);
/* accept the input */
printf("\nDownload more S-record? (Y/N) > ");
if(toupper(getche()) != 'Y') break;
}
}

/*****
/*  offld()
/*
/*
/*  Zuyi Chen
/*
/*
/*  offld() will offload data in the range of the addresses
/*  specified by user from EVB board to the host. It is
/*  important to reset EVB board as indicated.
/*
/*
/*  October 20, 1991
/*
/*
/*  Version 1.0
/*
*****/

offld(){

/* This while loop will exit upon the input 'N' or 'n' */
while(1){
/* accept the input */
printf("\nGive a name for the output file > ");
gets(filename);

/* open file for write */

```

```

if((fp=fopen(filename, "w")) == NULL){
    perror(filename);
    exit(1);
}
/* accept the input */
printf("\nReset EVB board; then enter <md xxxx xxxx> command >");
gets(command);
n=strlen(command);
command[n]='\r';
printf("\nData receiving ...");
prompt();
i=0;

/* send char to EVB board */
outport(uart_rbr1,command[i]);
for(k=0; k<Delay && !ready(DATAS); k++)
;
while(command[i] != '\r'){
/* send char to EVB board */
    outport(uart_rbr1,command[++i]);
    for(k=0; k<Delay && !ready(DATAS); k++)
        ;
}

i=0;
/* check if receive is ready */
while(!ready(DATAR))
;
while(1){
/* recieve char from EVB board */
    c=receive();
    if(c!=0){
        buf[i]=c;          /* put char to buffer */
    }
    else{
/* recieve char from EVB board */
        c=receive();
        if(c=='>'){
            break;
        }
        else buf[i]=c;
    }
    i++;
}
j=0;
while(buf[j++] !='\n')
;
n=i-j;
j=1;
do
{
/* write buffer to output file block by block */
    if(n>=BLOCK) fwrite(&buf[j],1,BLOCK,fp);
    else fwrite(&buf[j],1,n,fp);
}

```

```

    j=j+BLOCK;
    n=n-BLOCK;
}
while(n>0);
fclose(fp);
/* accept the input */
printf("\nOffload more data? (Y/N) > ");
if(toupper(getche()) != 'Y') break;
}
}

```

```

/*****
/*  edit()
/*
/*  Zuyi Chen
/*
/*  edit() takes advantage of the existing editor installed in
/*  the host and allows the user to use the editor inside the
/*  package.
/*  October 20, 1991
/*      modified in July, 1992
/*
/*  Version 2.0
/*
*****/

```

```
edit(){
```

```

/* This while loop will exit upon the input 'N' or 'n' */
while(1){
    /* accept the input */
    printf("\nUse the existing editor from here: ");
    gets(edit_filename);
    /* store edit_filename to command string */
    strcpy(command, edit_filename);
    /* call system utility */
    system(command);
    /* accept the input */
    printf("\nEdit another file? (Y/N) > ");
    if(toupper(getche()) != 'Y') break;
}
}

```

```

/*****
/*  show()
/*
/*  Zuyi Chen
/*
/*  show() will display the file content at the input of the
/*  file name. It takes advantage of the command 'type' in
/*  the host.
/*  October 20, 1991
*****/

```

```

/*                                                                 */
/*  Version 1.0                                                    */
/*                                                                 */
/*****
show(){

/* This while loop will exit upon the input 'N' or 'n' */
while(1){
  /* store 'type ' to command string */
  strcpy(command, "type ");
  /* accept the input */
  printf("\nEnter the filename to be shown > ");
  gets(filename);
  /* store filename to command string */
  strcat(command, filename);
  /* call system utility */
  system(command);
  /* accept the input */
  printf("\n\nShow another file? (Y/N) > ");
  if(toupper(getche()) != 'Y') break;
}
}

/*****
/*  crossasm()                                                    */
/*                                                                 */
/*  Zuyi Chen                                                      */
/*                                                                 */
/*  crossasm() will assemble the assembly file specified by     */
/*  the input. It takes the advantage of the 'as11'              */
/*  executable installed in the host.                             */
/*                                                                 */
/*  October 20, 1991                                             */
/*                                                                 */
/*  Version 1.0                                                  */
/*                                                                 */
/*****

crossasm(){
char program[30];
char cmd[60];
char pgm[30];

/* This while loop will exit upon the input 'N' or 'n' */
while(1){
  /* accept the input */
  printf("\nEnter the assembly program name to be cross-assembled
> ");
  gets(program);
  /* check if the filename is more than 4 chars */
  if(strlen(program) <= 4) continue;

```



```

/* store 'as11 <program>.asm -l c > <program>.lst' to cmd string
*/
strcpy(cmd, "as11 ");
strcat(cmd, program);
/* strcat(cmd, " -l c > ");
for(i=0;i<(strlen(program)-4);i++)
    pgm[i] = program[i];
strcat(cmd, pgm);
strcat(cmd, ".lst");
*/
/* call system utility */
if(system(cmd) < 0) exit(1);
/* accept the input */
printf("Assemble another program? (Y/N) > ");
if(toupper(getche()) != 'Y') break;
}
}

```

```

/*****/
/* connect() */
/* */
/* Zuyi Chen */
/* */
/* connect() will turn the host to a dumb terminal for EVB */
/* board. It takes the advantage of kermit 3.0 installed */
/* in the host. */
/* October 20, 1991 */
/* */
/* Version 1.0 */
/* */
/*****/

```

```

connect(){

printf("\n\nType 'c' at prompt MS-Kermit> to connect EVB
board;\n");
printf("type 'ctrl-]c' to exit EVB board; and \n");
printf("type 'q' at 'MS-Kermit> to return to main manual.\n\n");
system("kermit");
}

```

```

/*****/
/* dir() */
/* */
/* Zuyi Chen */
/* */
/* dir() will display the current working directory. It takes */
/* advantage of the command 'dir' in the host. */
/* */
/* October 20, 1991 */
/* */
/* Version 1.0 */
/*****/

```

```

/*                                                                 */
/*****                                                             */

dir(){

/* This while loop will exit upon the input 'N' or 'n' */
while(1){
    system("dir");
    /* accept the input */
    printf("\nShow the directory again? (Y/N) > ");
    if(toupper(getche()) != 'Y') break;
}
}

/*****                                                             */
/* convert()                                                                 */
/*                                                                 */
/* Zuyi Chen                                                                 */
/*                                                                 */
/* convert() will convert a hex data file into a decimal data */
/* file. On a 16-bit PC it can only convert the hex number */
/* smaller than or equal to FFFF. */
/*                                                                 */
/* October 20, 1991 */
/*                                                                 */
/* Version 1.0 */
/*                                                                 */
/*****                                                             */

convert(){
FILE *ffp;
char infile[30], outfile[30];
char str[81];
unsigned long sum;
int current;

/* This while loop will exit upon the input 'N' or 'n' */
while(1){
    /* this while loop will loop back if the input file doesn't exist
       in the current directory */
    while(1){
        /* accept the input */
        printf("\nIt only converts numbers smaller than or equal to
        FFFF. ");
        printf("\nEnter the hex file name to be converted > ");
        gets(infile);
        /* open input file */
        if((fp=fopen(infile,"r")) == NULL){
            perror(infile);
            continue;
        }
        else break;
    }
}
}

```

```

/* accept the input */
printf("\nGive a name to the new output file > ");
gets(outfile);
/* open output file */
if((ffp=fopen(outfile, "w")) == NULL){
    perror(outfile);
    exit(1);
}

/* loop to read one line of string at a time and do conversion */
while(1){
    if((fgets(str, 81, fp)) == NULL) /* if at end of file */
        break;
    /* check if the chars are within 1-9 or A-F */
    if((str[0] >= 48 && str[0] <= 57) || (str[0] >= 65 && str[0] <=
70)){
        sum = 0;
        /* convert the first 4 chars of each line to decimal numbers
*/
        for(i=0;i<4;i++){
            if(str[i] >= 48 && str[i] <= 57){
                sum = sum * 16 + str[i] - 48;
            }
            else if(str[i] >= 65 && str[i] <= 70)
                switch(toupper(str[i])){
                    case 'A':
                        sum = sum * 16 + 10;
                        break;
                    case 'B':
                        sum = sum * 16 + 11;
                        break;
                    case 'C':
                        sum = sum * 16 + 12;
                        break;
                    case 'D':
                        sum = sum * 16 + 13;
                        break;
                    case 'E':
                        sum = sum * 16 + 14;
                        break;
                    case 'F':
                        sum = sum * 16 + 15;
                        break;
                    default:
                        break;
                }
        }
        fprintf(ffp, "%05ld", sum);

        /* convert the next 8 hex numbers, each with 6 digits,
to decimal numbers.
*/
        current = 4;
        for(j=0;j<8;j++){

```

```

sum = 0;
for(i=current;i<(current+6);i++){
    if(str[i] >= 48 && str[i] <= 57)
        sum = sum * 16 + str[i] - 48;
    else if(str[i] >= 65 && str[i] <= 70){
        switch(toupper(str[i])){
            case 'A':
                sum = sum * 16 + 10;
                break;
            case 'B':
                sum = sum * 16 + 11;
                break;
            case 'C':
                sum = sum * 16 + 12;
                break;
            case 'D':
                sum = sum * 16 + 13;
                break;
            case 'E':
                sum = sum * 16 + 14;
                break;
            case 'F':
                sum = sum * 16 + 15;
                break;
            default:
                break;
        }
    }
}
fprintf(ffp, " %05ld", sum);
current = i;
if(j==7)
    fprintf(ffp, "\n");
}
}
fclose(fp);
fclose(ffp);
printf("\nConvert another file? (Y/N) > ");
if(toupper(getche()) != 'Y') break;
}
}

```

/\*\*\*\*\*\* Main program\*\*\*\*\*\*/

```

main(){
char choice;

printf("Set baud rate (2400,9600, etc) > ");
scanf("%d", &BAUD);
fflush(stdin);
printf("%d\n", BAUD);

```

```
while(1){
  clrscr();
  menu();
  choice = getche();
  printf("\n");
  switch(toupper(choice)){
    case 'A':
      crossasm();
      break;
    case 'B':
      downld();
      break;
    case 'C':
      connect();
      break;
    case 'D':
      offld();
      break;
    case 'E':
      edit();
      break;
    case 'F':
      show();
      break;
    case 'G':
      convert();
      break;
    case 'H':
      dir();
      break;
    case 'Q':
      return;
    default:
      break;
  }
}
```

/\* clear the screen \*/  
/\* display the menu \*/  
/\* get input \*/  
/\* cross assemble a program \*/  
/\* download a S-record \*/  
/\* convert host to terminal for EVB \*/  
/\* offload data from EVB to host \*/  
/\* edit files \*/  
/\* display file content \*/  
/\* convert hex file to decimal file \*/  
/\* display the current directory \*/  
/\* exit the menu \*/

```

/*****
/*  EVBWSERV.C                                     */
/*                                             */
/*  Zuyi Chen                                     */
/*  The Computer Science Department             */
/*  The University of Montana                   */
/*  Missoula, Montana                           */
/*                                             */
/*  The package EVBSERV is designed to provide the EVB/Buffalo */
/*  users with convenience in interacting a workstation host */
/*  with the EVB board. It contains the following functions: */
/*  init(), menu(), downld(), txtfile(), binfile(), offld(), */
/*  edit(), show(), crossasm(), connect(), dir(), convert(), */
/*  myhtoi(), and myahtoi().                    */
/*                                             */
/*  Nov., 14, 1991                               */
/*  Version 1.0                                   */
/*                                             */
*****/

```

```

/*****
/*  You are welcome to copy and distribute unmodified source */
/*  code to other parties provided you include this notice,  */
/*  together with the original file header, as a part of the */
/*  file. You are welcome to modify the source code for your */
/*  own use.                                                  */
*****/

```

```

# include <stdio.h>
# include <sgtty.h>
# include <sys/file.h>
# include <sys/time.h>
# include <sys/ttydev.h>
# include <string.h>

```

```

# define BLOCK 4096

```

```

/* declarations                                     */
char command[80];
char c, h, filename[80];
char ch[5], s1[20], s2[20];
char buf[100000];
unsigned char s[64000];
unsigned char myahtoi(), myhtoi();
int ln, rbytes, wbytes, l;
int i, j, k, m, n, r1, r2, count;
FILE *fp;

```

```

struct sgtyb stbuf;
struct sgtyb savea;

```

```

void txtfile();
void binfile();
void init();

```



```

/*
/*****

```

```

void init() {
    if ((ln = open ("/dev/ttya", O_RDWR)) < 0) {
        printf ("\nUnable to open port ttya");
        exit(1);
    }
    stbuf.sg_ispeed = B9600; /*set ttya speed B9600 */
    stbuf.sg_ospeed = B9600;
    stbuf.sg_flags = O_RAW; /*set ttya port raw mode */

    ioctl (ln, TIOCGTP, &savea); /*save ttya port mode */
    ioctl (ln, TIOCSTP, &stbuf); /* set ttya port mode */
    /*no further opens are permitted */
    ioctl (ln, TIOCEXCL, (struct sgtyb *)NULL);

    printf ("\nPlease hit the RESET key on the EVB.\n");
    for (i = 1; i<=70; i++ ) {
        read(ln, &c, 1);
    }
    write(ln, "\r", 1);
    while (1) { /* get the BUFFALO prompt */
        read(ln, &c, 1);
        if (c=='>')
            break;
    }
}

```

```

/*****
/*  myatoi()
/*
/*  Zuyi Chen
/*
/*  This function is adopted from the same function written by
/*  Li Zheng.
/*
/*  Nov., 14, 1991
/*
/*****

```

```

unsigned char myatoi(byte)
char *byte;
{
    return(myhtoi(byte[0])*16 + myhtoi(byte[1]));
}

```

```

/*****
/*  myhtoi()
/*
/*  Zuyi Chen
/*

```



```

/*
/* This function is adopted from the same function written by */
/* Li Zheng. */
/*
/* Nov., 14, 1991 */
/*
/*****

```

```

unsigned char myhtoi(nibble)
char nibble;

```

```

{
    if (('0' <= nibble) && (nibble <= '9'))
        return (nibble - '0');
    else if (('A' <= nibble) && (nibble <= 'F'))
        return (nibble - 'A' + 10);
    else if (('a' <= nibble) && (nibble <= 'f'))
        return (nibble - 'a' + 10);
    else {
        perror("\nIllegal data.\n");
        exit(1);
    }
}

```

```

/*****
/* txtfile() */
/*
/* Zuyi Chen */
/*
/* This function is a modified version of the function written */
/* by Li Zheng. It sends command to EVB board; reads and */
/* writes the data from the EVB board to a printable format */
/* file in the workstation host. */
/*
/* Nov., 14, 1991 */
/*
/* Version 1.1 */
/*
/*****

```

```

void txtfile() {
    printf ("\nGive a name for the file to store data > ");
    gets(filename);
    printf ("\nEnter the EVB command <md xxxx xxxx> here > ");
    gets(command);
    n = strlen(command);
    command[n] = '\r';
    if ((fp=fopen(filename, "w")) == NULL) {
        printf ("\nCan not open %s", filename);
        exit(1);
    }
    init(); /*initialize the ttya port */
    printf("Data receiving ... \n");
    write(ln, command, n+1);
}

```

```

i = 0;
while (1) { /* read data one by one until reach the */
  read (ln, &c, 1); /* BUFFALO prompt > */
  if (c != 0) /* skip 0 value */
    buf[i] = c; /* put data into the buffer */
  else {
    read (ln, &c, 1);
    if ( c=='>' ) {
      break;
    }
    else
      buf[i] = c; /* put data into the buffer */
  }
  i++;
}

j = 0;
while (buf[j++] != '\n') /* get the actual number of bytes we */
  ; /* want to store */
n = i-j;
do {
  if (n >= BLOCK)
    fwrite(&buf[j], 1, BLOCK, fp);
  else
    fwrite(&buf[j], 1, n, fp); /* write to the file */
  j = j + BLOCK;
  n = n - BLOCK;
}
while( n > 0);

fclose(fp);
close (ln);
ioctl (ln, TIOCSETP, &savea); /* reset the ttya */
}

```

```

/*****
/*  binfile()
/*
/*  Zuyi Chen
/*
/*  This function is a modified version of the function written
/*  by Li Zheng. It sends the command to the EVB board; reads
/*  and writes the data from the EVB to a binary file in the
/*  workstation host.
/*
/*
/*  Nov., 14, 1991
/*
/*  Version 1.1
/*
*****/

```

```
void binfile() {
```

```

strcpy(command, "md ");
printf ("\nGive a name for the file to store binary data > ");
gets(filename);
printf ("\nStarting memory address in HEX > ");
gets (s1);
printf ("\nEnding memory address in HEX > ");
gets (s2);
strcat (command, s1);
l = strlen(command);
command[l] = ' ';
strcat (command, s2);
sscanf (s1, "%x", &r1); /* get the start address value */
sscanf (s2, "%x", &r2); /* get the end address value */
count = r2 - r1 + 1; /* get the actual number of bytes you
want */
l = strlen(command);
command[l] = '\r';
if ((fp=fopen(filename, "w")) == NULL) {
    printf("\nCan not open %s", filename);
    exit(1);
}
init(); /* initialize ttya port */
printf("\nData receiving ... \n");
write(ln, command, l+1);
i = 0;
while (1) { /*read data one by one until reach the */
    read (ln, &c, 1); /*BUFFALO prompt */
    if (c != 0) /* skip 0 value */
        buf[i] = c;
    else {
        read (ln, &c, 1);
        if ( c=='>' ) {
            break;
        }
        else
            buf[i] = c;
    }
    i++;
}

j = 0;
while (buf[j++] != '\n')
;
while (buf[j++] != '\n')
;
l = i-j;
m = 0;
while ( j<i ) {
    j+=5; /* skip address column */
    for(k=0; k<16; k++) {
        s[m++] = myatoi(&buf[j]); /* convert this data to binary
value*/
        j+=3; /* skip two characters and one space */
    }
}

```

```

    while(buf[j++]!='\n')
        ;
}
k = 0;
m = count;
do { /* write these binary numbers to a file */
    if (m >= BLOCK)
        fwrite(&s[k], 1, BLOCK, fp);
    else
        fwrite(&s[k], 1, m, fp);
    k = k + BLOCK;
    m = m - BLOCK;
}
while( m > 0);

fclose(fp);
close (ln);
ioctl (ln, TIOCSETP, &savea); /* reset ttya port */
}

```

```

/*****
/*  downld()
/*
/*  Zuyi Chen
/*
/*  downld() will download the S-record with the input record */
/*  name from the host to the EVB board.  It is important to*/
/*  reset EVB board as indicated.
/*
/*
/*  Nov., 14, 1991
/*
/*  Version 1.0
/*
*****/

```

```

downld(){
char *command1="load t\r";

/* This while loop will exit upon the input 'N' or 'n' */
while(1){
    printf("\n");
    strcpy(command, "dwnA ");
    printf ("\nEnter file name you want to download: ");
    gets(filename);
    strcat(command, filename);
    l = strlen(command1);
    init(); /* initialize the ttya port */
    printf("\nDownloading S-record ...\n");
    write(ln, command1,l); /* send load t command to the EVB */
    sleep(1);
    close (ln);
    ioctl (ln, TIOCSETP, &savea); /*reset ttya */
}

```

```

    /* call system utility 'system' */
    system(command); /*execute dwnA command */
    printf("\nDownload more S-record? (Y/N) > ");
    /* accept the input */
    gets(ch);
    if (toupper(ch[0])!='Y')
        break;
}
}

/*****
/*  offld()
/*
/*  Zuyi Chen
/*
/*  offld() will offload data in the range of the addresses
/*  specified by user from EVB board to the host. It is
/*  important to reset EVB board as indicated. It calls
/*  txtfile() or binfile() as specified by the users.
/*
/*
/*  Nov., 14, 1991
/*
/*  Version 1.0
/*
/*****

offld(){

/* This while loop will exit upon the input 'N' or 'n' */
while(1){
    /* accept the input */
    printf("\nSpecify the file type for the offloaded data.\n");
    printf("\nEnter 't' for text file, 'b' for binary file > ");
    gets(ch);
    if (toupper(ch[0]) == 'B')
        binfile(); /* save data in binary format file */
    else
        txtfile(); /* save data in printable format file */
    /* accept the input */
    printf("\nOffload more data? (Y/N) > ");
    gets(ch);
    if(toupper(ch[0]) != 'Y') break;
}
}

/*****
/*  edit()
/*
/*  Zuyi Chen
/*
/*  edit() takes advantage of vi editor installed in the host;*/
/*  and allows the users to use vi editor inside the package. */

```

```

/*                                                    */
/*  Nov., 14, 1991                                    */
/*                                                    */
/*  Version 1.0                                       */
/*                                                    */
/*****/

edit(){

/* This while loop will exit upon the input 'N' or 'n' */
printf("\nThis is the vi editer\n");
while(1){
    /* store 'vi' to command string */
    strcpy(command, "vi ");
    /* accept the input */
    printf("\nEnter the file name to be edited > ");
    gets(filename);
    /* store filename to command string */
    strcat(command, filename);
    /* call system utility */
    system(command);
    /* accept the input */
    printf("\nEdit another file? (Y/N) > ");
    gets(ch);
    if(toupper(ch[0]) != 'Y') break;
}
}

/*****/
/*  show()                                           */
/*                                                    */
/*  Zuyi Chen                                        */
/*                                                    */
/*  show() will display the file content at the input of the */
/*  file name. It takes advantage of the command 'cat' in the */
/*  host.                                           */
/*                                                    */
/*  Nov., 14, 1991                                    */
/*                                                    */
/*  Version 1.0                                       */
/*                                                    */
/*****/

show(){

/* This while loop will exit upon the input 'N' or 'n' */
while(1){
    /* store 'cat ' to command string */
    strcpy(command, "cat ");
    /* accept the input */
    printf("\nEnter the filename to be shown > ");
    gets(filename);
    /* store filename to command string */
    strcat(command, filename);

```

```

/* call system utility */
system(command);
/* accept the input */
printf("\n\nShow another file? (Y/N) > ");
gets(ch);
if(toupper(ch[0]) != 'Y') break;
}
}

/*****
/*   crossasm()
/*
/*   Zuyi Chen
/*
/*   crossasm() will assemble the assembly file specified by the */
/*   input. It takes the advantage of the 'asm11' executable */
/*   installed in the host.
/*
/*   October 20, 1991
/*
/*   Version 1.0
/*
*****/

crossasm(){
char program[30];
char cmd[60];

/* This while loop will exit upon the input 'N' or 'n' */
while(1){
/* accept the input */
printf("\nEnter the assembly program name to be cross-assembled
> ");
gets(program);
/* check if the filename is more than 4 chars */
if(strlen(program) <= 4) continue;

/* store 'asm11 <program>.asm' to cmd string */
strcpy(cmd, "asm11 ");
strcat(cmd, program);

/* call system utility */
if(system(cmd) < 0) exit(1);
/* accept the input */
printf("Assemble another program? (Y/N) > ");
gets(ch);
if(toupper(ch[0]) != 'Y') break;
}
}

/*****
/*   connect()
*****/

```





```

/*      Zuyi Chen                                     */
/*      */                                           */
/*      convert() will convert a hex data file into a decimal data */
/*      file. It can only convert the hex number smaller than or */
/*      equal to FFFF.                                     */
/*      */                                           */
/*      Nov., 14, 1991                                   */
/*      */                                           */
/*      Version 1.0                                       */
/*      */                                           */
/*****
convert(){
FILE *ffp;
char infile[30], outfile[30];
char str[81];
unsigned long sum;
int current;

/* This while loop will exit upon the input 'N' or 'n' */
while(1){
    /* this while loop will loop back if the input file doesn't exist
       in the current directory */
    while(1){
        /* accept the input */
        printf("\nIt only converts numbers smaller than or equal to
FFFF. ");
        printf("\nEnter the hex file name to be converted > ");
        gets(infile);
        /* open input file */
        if((fp=fopen(infile,"r")) == NULL){
            perror(infile);
            continue;
        }
        else break;
    }

    /* accept the input */
    printf("\nGive a name to the new output file > ");
    gets(outfile);
    /* open ouput file */
    if((ffp=fopen(outfile, "w")) == NULL){
        perror(outfile);
        exit(1);
    }

    /* loop to read one line of string at a time and do conversion */
    while(1){
        if((fgets(str, 81, fp)) == NULL)    /* if at end of file */
            break;
        /* check if the chars are within 1-9 or A-F */
        if((str[0] >= 48 && str[0] <= 57) || (str[0] >= 65 && str[0] <=
70)){
            sum = 0;

```

```

/* convert the first 4 chars of each line to decimal numbers
*/
for(i=0;i<4;i++){
if(str[i] >= 48 && str[i] <= 57){
    sum = sum * 16 + str[i] - 48;
}
else if(str[i] >= 65 && str[i] <= 70)
    switch(toupper(str[i])){
        case 'A':
            sum = sum * 16 + 10;
            break;
        case 'B':
            sum = sum * 16 + 11;
            break;
        case 'C':
            sum = sum * 16 + 12;
            break;
        case 'D':
            sum = sum * 16 + 13;
            break;
        case 'E':
            sum = sum * 16 + 14;
            break;
        case 'F':
            sum = sum * 16 + 15;
            break;
        default:
            break;
    }
}
fprintf(ffp, "%05ld", sum);

/* convert the next 8 hex numbers, each with 6 digits,
to decimal numbers.
*/
current = 4;
for(j=0;j<8;j++){
    sum = 0;
    for(i=current;i<(current+6);i++){
        if(str[i] >= 48 && str[i] <= 57)
            sum = sum * 16 + str[i] - 48;
        else if(str[i] >= 65 && str[i] <= 70){
            switch(toupper(str[i])){
                case 'A':
                    sum = sum * 16 + 10;
                    break;
                case 'B':
                    sum = sum * 16 + 11;
                    break;
                case 'C':
                    sum = sum * 16 + 12;
                    break;
                case 'D':
                    sum = sum * 16 + 13;

```



```
case 'F':
    show();           /* display file content */
    break;
case 'G':
    convert();       /* convert hex file to decimal file
*/
    break;
case 'H':
    dir();           /* display the current directory */
    break;
case 'Q':
    return;         /* exit the menu */
default:
    break;
}
}
}
```

**Appendix B**

**Computer Architecture and Assembly Language**

**LAB USE INFORMATION**

**ZUYI CHEN**

**July, 1992**

**Computer Science Department**

**University of Montana**

## TABLE OF CONTENTS

Table of Contents	i
1.0: Operating the M68HC11EVB	1
2.0: Cross-assembler and Host/EVB Downloading	3
3.0: Offloading Data from EVB to Workstation Host	5
4.0: Using the M68HC11 EVB Server	7

## 1.0: Operating the M68HC11EVB

(Written by Dr. Ray Ford of UM, and modified by Zuyi Chen of UM)

M68HC11EVB stands for M68HC11 Evaluation Board. It is a product of Motorola, Inc. The major components of the board include a MC68HC11 microcomputer unit (MCU), a MC68HC24 port replacement unit (PRU), terminal/host I/O ports, a debugging/monitor program called BUFFALO, which stands for Bit User Fast Friendly Aid to Logical Operations, and an optional 8-K RAM chip. The memory of the EVB ranges from \$0000 to \$FFFF. The user RAM is located between \$C000 and \$DFFF with the optional RAM from \$6000 to \$7FFF. The RAM part is not large since the EVB was designed for embedded system, which typically does a fixed job again and again in its life time, therefore requiring a small amount of RAM.

### A. EVB Hook-Up

This lab assumes that the EVB is connected for both power and communication to a dumb terminal (TTY). That is, there should be a serial line running from the TTY to the TTY on the EVB, and a power connector running from the TTY to the EVB's power inputs.

Check out these connections. If they are not correct, consult the EVB User's Manual and make the proper connections.

### B. TTY/EVB Power-Up

Switch the TTY on - since the EVB draws power from the TTY it too should power up. The TTY will (probably) display a simple prompt. Type "carriage return" (CR) to signal the EVB - the EVB should respond by displaying the BUFFALO Monitor (BUFFALO) header line. Type another CR - the EVB/BUFFALO should interpret this as a "help" command, and display a list of available commands.

Find the reset switch on the EVB and press it (these switches are flaky - you may have press it, then lift it up). This resets BUFFALO, and should cause the BUFFALO header to displayed.

### C. BUFFALO Commands

BUFFALO supports a wide range of interactive commands, including those describe briefly below (and more).

1. md: display the contents of specified memory cells
2. mm: change the contents of specified memory cells
3. rm: display and set the contents of registers
4. asm: enter, assemble, and load assembly instructions (one by one)
5. go: initiate the execution of a (assembly) program
6. br: define break points in the execution of a program through a specified number of instructions
7. load t: enable the downloading of a program in S-record form from the TTY port

Practice using these commands (all EXCEPT “load t”) by entering and executing the simple assembly language program shown below. Note that the program is shown in the form normally used as input of a cross-assembler; you may have to adapt this form for use with the rather rudimentary BUFFALO assembly capability. Be sure that you gain sufficient familiarity with the BUFFALO commands and outputs so that you are ready to monitor the execution of more complex and interesting programs.

\*\*\*\*\*

\* Pgm: simple.asm  
 \* Desc: load current time and save it to the memory \$D000-\$DFFF.  
 \* The program is supposed to start at memory address \$0000.  
 \* Note: \$ sign is not used when entering directly on the BUFFALO  
 \* Author: Zuyi Chen  
 \* Date: June, 1992

\*\*\*\*\*

```

ldx    #0000           ; initialize data
ldy    #D000           ; storage starting address
ldd    100E            ; get the current MCU time (pgm counter $0007)
std    0,Y             ; save time to the store
ldab   #2              ; load 2 to register B
aby    ; increase the memory address by 2 bytes
cpy    #DFFF           ; check if current address is $DFFF
blo    0007            ; back to get current time if address not $DFFF
nop    ; no operation
bra    0017            ; loop back to no operation instruction

```

## References

1. M68HC11-EVB User's Manual. A description of the 68HC11-EVB system, on-board facilities, and the BUFFALO Monitor.
2. M68HC11 Reference Manual. A complete description of the 68HC11 chip facilities, functional units, and assembly language.
3. M68HC11-A8 Programming Reference Guide. A “pocket” guide to 68HC11 chip facilities, functional units, and assembly language.



## 2.0: Cross-assembler and Host/EVB Downloading

(Written by Dr. Ray Ford of UM, modified by Zuyi Chen of UM)

### Purpose

To describe key elements in the operation of the host/EVB development environment: the M68HC11 cross-assembler, kermit, terminal emulator, downloader, and the S-record load mode on the EVB.

### A. M68HC11 Cross-Assembly

A locally defined script, called "asm11" invokes the cross-assembler with a standard set of options. Assemble a program in a file named "pgm.asm" by simply executing the cross-assembler. The standard options produce a file containing a listing and symbol cross-reference table in "pgm.asm.lst". For error-free programs a second file "pgm.s19" is also created, which contains a pseudo-executable form of the program in a format called S-records. S-record details aren't important here - what is important is that they encode the executable AND the load information in an ASCII file, i.e., in a form suitable for downloading to the EVB via a simple terminal emulator such as kermit.

Documentation on the M68HC11 cross-assembler is available in the file "asm11.man". S-records are described in detail in the EVB User's Manual, Appendix A.

### B. Downloading An Assembled Program to the EVB

Once the S-record file "pgm.s19" has been produced, it must be downloaded to the M68HC11-EVB for final program loading and execution. This involves cabling the host and EVB together, putting the EVB into a state to accept incoming S-records, getting the host to send the S-records, then restoring the EVB to normal state.

1. **START-UP EVB:** Power up the EVB, with the EVB's "TTY" port connected to a dumb-terminal via a "straight through" RS-232 cable. Use the dumb-terminal to verify that the EVB is working properly.
2. **CABLE CONNECTION:** Disconnect the EVB/dumb-terminal cable at the EVB end. Connect a "straight-through" serial cable between the EVB's "TTY" port and the "TTYA" port on a workstation or "COM1" port on a PC.
3. **SOFTWARE CONNECTION:** Once the cable connects the host and EVB ports, initiate the software connection by executing "kermit" on the PC or workstation. At the kermit prompt simply type "c", then CR to establish connection with the EVB. With a few more CRs you should see the BUFFALO monitor prompt and be able to enter BUFFALO commands from the host. If the EVB fails to respond, press the EVB's reset key until it does respond. If nothing happens after a few resets, check the cable connection and restart.
4. **EVB LOAD/HOST DOWNLOAD:**
  - (a) When you are ready to download a program from the host, put EVB into "receive S-record program" state by entering the BUFFALO command "load t". The EVB now expects a sequence of S-records to be transmitted over a serial connection to its "TTY" port. As S-records are received BUFFALO decodes them and loads the appropriate executable version of the program into the EVB's memory.

- (b) Escape back to the host's kermit session by typing the escape sequence <CTRL-],c>. When you get the host's kermit prompt, simply use "quit" to quit the host's current kermit session.
- (c) (Back at the standard host command level) Next, transmit the S-record file by executing the command "dwnA pgm.s19" on the workstation or "type pgm.s19 > com1" on a PC. They "cat" the specified file to the host's port, thus sending it to the EVB.
- (d) Following execution of the downloading command the host should give another prompt, without any visible sign that the S-record transmission has taken place. Any other message suggests that the downloading has probably failed. If you have problems, check your file name and the connections, then try again.
- (e) Even after the end of the host's downloading the EVB remains in "load t" mode, expecting more incoming S-records. You get the EVB out of "load t" mode by resetting the EVB.

#### 5. EVB PROGRAM EXECUTION:

- (a) Once the program is downloaded to the EVB, you can initiate and monitor its execution using either the dumb-terminal or the host as the EVB interface. To use the host, execute another kermit command to initiate a new kermit session. Note that when you execute the "c" to connect to the EVB you will generally have to reset the EVB to get it out of "load t" mode. Following the reset the EVB should return to the BUFFALO monitor, and the host should display the BUFFALO prompt to indicate that it is connected as the EVB interface.
- (b) Verify the S-record transmission by using the BUFFALO memory display command (e.g., "md c000" should show the binary version of the program now loaded into the EVB memory).
- (c) If the program is downloaded correctly, it can be executed and monitored using the standard BUFFALO commands. If the program is not loaded correctly, re-try the downloading.

### 3.0: Offloading Data from EVB to Workstation Host

(Written by Dr. Ray Ford of UM, and modified by Zuyi Chen of UM)

#### Purpose

To describe approaches to offloading data from the EVB to a workstation for post-processing.

#### Pre-conditions

It is assumed that an EVB program has stored data in memory locations, and that a workstation host is physically connected to the EVB via a serial line.

#### A. Workstation Set-up

Initiate a script session on the workstation by executing the command "script data.file". "script" actually starts a new "shell" in which all characters sent to the workstation display (for this window) are also copied into the file "data.file", until the shell is explicitly terminated (eg., with a "CTRL-d"). Now, execute "kermA" to connect the workstation to the EVB, with the workstation emulating a dumb-terminal.

#### B. Data Capture

On the EVB, execute the BUFFALO command "md \$xxxx \$xxxx" (where \$xxxx are the starting and ending address between which the data will be captured). The contents of these EVB memory locations will be displayed, AND thus will be captured on the workstation in "data.file" by the script shell.

#### C. BUFFALO Flush

The data display/capture is buffered, so you MUST execute a few additional simple BUFFALO commands to add elements to the display to guarantee that the last output buffer is flushed (i.e., the buffer containing the last few lines of the \$xxxx-\$xxxx display). A command like "help" will usually be sufficient, but you should experiment with this yourself.

#### D. End of Capture

Once you are sure that all the desired data has been captured, escape back to the workstation and terminate the "kermA" session. Next, terminate the "script" session using "CTRL-d".

#### E. Post-processing

"data.file" includes the desired data, plus extraneous information at the start and end of the file associated with the scripting activity. It is essential that you edit the script file to remove the extraneous information. Also, note that this capture process has taken the hexadecimal display of EVB memory contents and encoded it in "data.file" as a particular list of ASCII characters, spaces, lines, etc. "Computational interpretation" of the data on the workstation must include reading the

ASCII file, interpreting the characters as the appropriate addresses, bytes, words, etc, and then translating the characters into a numerical form.

## 4.0: Using the M68HC11 EVB Server

### PURPOSE

To provide the users of the EVB with convenience in EVB operations, and to allow the users to do EVB programming without having to worry about the detailed procedures in downloading and offloading data.

Versions of EVB Server are available for both the PCs and Unix workstations. They offer identical functionality; they differ only in their communication handling.

The following is a list of the EVB Server menu and functions:

```
*****
* a. Cross-assemble the assembly program
* b. Download S-record from host to EVB board
* c. Turn host to a terminal for the EVB board
* d. Offload data from EVB board to host
* e. Edit a file using an existing editor
* f. Show the file content
* g. Convert hex data file to decimal data file
* h. Display the working directory
* i. Quit
*****
```

### Usage

The EVB Server menu will be displayed on the screen when the command <evbserv> or <evbwserv> is entered on a PC or workstation, respectively (assume the executable is already in the host). The following options are available for selection from the menu.

1. Function A serves as the cross-assembler. It takes the assembly program name as input; and outputs the S-record of the program, along with an assembly listing.
2. Function B provides the downloading service. It allows you to download the S-record from the host to the EVB board without having to go thru all the detailed procedures.
3. This option connects the host with the EVB. It turns the host to a dumb terminal for the EVB board.
4. This option provides the offloading service. It allows you to offload the data from the EVB board back to the host, and store the data in a file you designated in the working directory.
5. This function takes advantage of the existing editor on the host.
6. It allows you to look at the file contents in the working directory.
7. This option will take a file with hexadecimal content, and convert it into the decimal content.
8. Choosing this will allow you to see the working directory.

Note: It is important to reset the EVB board when indicated.

**Appendix C**

**Computer Architecture and Assembly Language**

**LAB MANUAL**

**ZUYI CHEN**

**July, 1992**

**Computer Science Department**

**University of Montana**

## TABLE OF CONTENTS

Table of Contents	i
Lab 1.0: Output Ports	1
Lab 2.0: Input Port	3
Lab 3.0: Traffic Signal at Port-C	4
Lab 4.0: Traffic Signal at Port-B	5
Lab 5.0: Music Tunes at Port-B	6
Lab 6.0: Traveling Light at Port-C	7
Lab 7.0: Traveling Light at Port-B	8
Lab 8.0: Modulo-9 Counter at Port-C	9
Lab 9.0: Modulo-9 Counter at Port-B	10
Lab 10.0: Software Interrupt Handling	11
Lab 11.0: Output Compare Function	12
Lab 12.0: Polling with OC2	13
Lab 13.0: OC5 Interrup	14
Lab 14.0: Timer Using Polling	15
Lab 15.0: Timer Using Interrupt Handling	16
Lab 16.o: Inter-Process Communication	17
Lab 17.0: Multi-Character Display	19

## Lab 1.0 Output Ports

**PROBLEM:** control output pins at Port-A, Port-B, Port-C and Port-D

**TOPIC:** pins for output at ports and timing control

**PURPOSE:** To introduce low-level device control achieved through the EVB port pins for output and timing control achieved through instruction cycles.

**ASSIGNMENT:** Study the program-shell *outports.asm* and hardware connection, noting specifically

1. that the EVB's memory-mapped pins for output at Port-A, Port-B Port-C and Port-D are used to control the 10-segment bar graph LEDs as external devices (for MCU I/O port connector pin assignments see P6-2 of M68HC11EVB Evaluation Board User's Manual);
2. that Port-A output pins are pin 3 - pin 7, and Port-D output pins are pin 2 - pin 5;
3. how the EVB output pins are connected to particular devices (ie, what program outputs control what segments of the bar graph LEDs);
4. how the program's outputs control the external devices (i.e., when a particular segment of the bar graph LEDs is turned on and off);
5. how subroutines *LITE\_A* and *CLEAR\_A* control the output pins of Port-A;
6. that you must complete portions of the program code to make the program perform the desired control function.

Part of the "bar graph display hardware" will be set up for you, and you are responsible for the rest. Each bar graph LED consists of 10 bars or segments. Each output pin should be connected to a distinct bar. The port-A output pins are already connected for you. You need to connect the output pins for Port-B, Port-C and Port-D.

To complete the program you must write some subroutines similar to *LITE\_A* and *CLEAR\_A*. These subroutines are:

```
LITE_B, CLEAR_B,  
LITE_C, CLEAR_C,  
LITE_D, CLEAR_D
```

You must cross-assemble the completed program, download it, test it, analyze the program results, and create a report on your analysis.

**BAR GRAPH CONTROL:** output pin connection of Port-C, Port-D, Port-A and Port-B in order:

1. turn on the most significant bit of Port-C;
2. add the next most significant bit without turning off the previous one;
3. repeat (2);



4. clear the port when all bits corresponding to output pins of a port are turned on; go to the next port, and do (1) - (4) in that port; after port-B is lit, go back to port-C and start over from (1).

## Lab 2.0 Input Port

**PROBLEM:** input pins at Port-C; output pins at Port-A and Port-B

**TOPIC:** pins for input and output at various ports and cycle analysis

**PURPOSE:** To introduce control of port pins for input and output, and free-running counter.

**ASSIGNMENT:** Study the program *inport.asm*, noting specifically

1. that setting up Port-C for input is realized by writing all 0's to the Port-C direction control register (DDRC);
2. how the memory-mapped free-running counter (TCNT) works.

This is a complete program. Cross-assemble it, download it, execute it several times. Offload two sets of data computed by the program in \$D000 - \$DFFF to the host machine, and save them in two different files. Analyze the data carefully, especially those related to cycles, and write a summary of your analysis results.

## Lab 3.0 Traffic Signal at Port-C

**PROBLEM:** TRAFFIC SIGNAL at Port-C

**TOPIC:** Port-C pins for output and timing control

**PURPOSE:** To introduce low-level device control achieved through the EVB Port-C pins, and timing control achieved by counting instruction cycles.

**ASSIGNMENT:** Study the program-shell *trafficC.asm* carefully, noting specifically

1. that the EVB's memory-mapped I/O Port-C pins are used to control the external devices of the traffic light control (for MCU I/O port connector pin assignments see P6-2 of M68HC11EVB Evaluation Board User's Manual);
2. how the program's Port-C outputs are connected to particular devices (i.e., what outputs control the red, yellow and green lights);
3. how the program's Port-C outputs control the external devices (i.e., when a particular light should be turned on and off);
4. that you must add timing information to make the program perform the desired control function.

The "traffic light hardware" will be set up for you, and you don't need to modify it to complete the assignment. The red, yellow and green LEDs on the breadboard are used as the red, yellow and green traffic lights. You need to understand how the EVB controls the hardware logically, not electronically.

To complete the program you must figure out the number of machine cycles that need to be delayed for each light signal. Simply compute the number of DELAY loop iterations, and replace the question marks in the program with those values, cross-assemble the program, download it, and test it. Write up an analysis that describes your timing computation and the way you mapped this to the loop iterations to give the real-time delay desired.

**TRAFFIC SIGNAL CONTROL:** the following numbers are ordered.

1. GREEN light ON for 10 seconds, then OFF
2. YELLOW light ON for 1 second, then OFF for 1 second
3. do (2) another two times
4. RED light ON for 10 seconds, then OFF
5. start from (1) again

## Lab 4.0 Traffic Signal at Port-B

**PROBLEM:** TRAFFIC SIGNAL at Port-B

**TOPIC:** Port-B pins for output and timing control

**PURPOSE:** To introduce low-level device control achieved through the EVB Port-B pins, and timing control achieved by counting instruction cycles.

**ASSIGNMENT:** Study the program-shell *trafficB.asm* carefully, noting specifically

1. that the EVB's memory-mapped I/O Port-B pins are used to control the external devices of the traffic light control (for MCU I/O port connector pin assignments see P6-2 of M68HC11EVB Evaluation Board User's Manual);
2. how the program's Port-B outputs are connected to particular devices (i.e., what outputs control the red, yellow and green lights);
3. how the program's Port-B outputs control the external devices (i.e., when a particular light should be turned on and off);
4. that you must add timing information to make the program perform the desired control function.

The "traffic light hardware" will be set up for you, and you don't need to modify it to complete the assignment. The red, yellow and green LEDs on the breadboard are used as the red, yellow and green traffic lights. You need to understand how the EVB controls the hardware logically, not electronically.

To complete the program you must figure out the number of machine cycles that need to be delayed for each light signal. Simply compute the number of DELAY loop iterations, and replace the question marks in the program with those values, cross-assemble the program, download it, and test it. Write up an analysis that describes your timing computation and the way you mapped this to the loop iterations to give the real-time delay desired.

**TRAFFIC SIGNAL CONTROL:** the following numbers are ordered.

1. GREEN light ON for 10 seconds, then OFF
2. YELLOW light ON for 1 second, then OFF for 1 second
3. do (2) another two times
4. RED light ON for 10 seconds, then OFF
5. start from (1) again

## Lab 5.0 Music Tunes at Port-B

**PROBLEM:** Music notes at Port-B

**TOPIC:** timing control

**PURPOSE:** To introduce low-level device control achieved thru the EVB Port-B pins, and timing control achieved by counting instruction cycles.

**ASSIGNMENT:** Study the program-shell *music.asm*, noting specifically

1. that the EVB's memory-mapped output Port-B pins are used to control the external devices of the music notes (for MCU I/O port connector pin assignments see P6-2 of M68HC11EVB Evaluation Board User's Manual);
2. how the program's Port-B outputs control the external devices (i.e., when a sound of particular frequency should be turned on and off);
3. that you must add timing information to make the program perform the desired control function.

The "music notes hardware" will be set up for you, and you don't need to modify it to complete the assignment. The program is supposed to generate music notes *mee*, *rai*, *do*, *tee*, *la*, *so*, *fa*, lower *mee*, and then to wrap around, each staying on for 1/2 second.

To complete the program you must figure out the number of machine cycles that need be delayed for each sound signal to stay on for 1/2 second. Simply compute the number of iterations RESONANT should loop for each frequency, and replace the question marks in the program with those values, cross-assemble it, download it, and test it. Write up your timing analysis.

**MUSIC NOTES CONTROL:** Real-time intervals between signals in the following order.

1. ring *mee* for 1/2 sec
2. ring *rai* for 1/2 sec
3. ring *do* for 1/2 sec
4. ring *tee* for 1/2 sec
5. ring *la* for 1/2 sec
6. ring *so* for 1/2 sec
7. ring *fa* for 1/2 sec
8. ring lower *mee* for 1/2 sec
9. go back to (1) and repeat

## Lab 6.0 Traveling Light at Port-C

**PROBLEM:** Traveling light on Port-C output pins

**TOPIC:** pins for output at Port-C and timing control

**PURPOSE:** To introduce low-level device control achieved thru the EVB port pins for output, and timing control achieved by counting instruction cycles.

**ASSIGNMENT:** Study the program-shell *travelC.asm*, noting specifically

1. that the EVB's memory-mapped pins for output at Port-C are used to control the 10-segment bar graph LEDs as external devices (for MCU I/O port connector pin assignments see P6-2 of M68HC11EVB Evaluation Board User's Manual);
2. how the EVB output pins are connected to particular devices (ie, what program outputs control what segment of the bar graph LED);
3. how the program's outputs control the external devices (i.e., when a particular segment of the bar graph LED is turned on and off);
4. that you must add timing information and one instruction to complete the program, and to make it perform the desired control function.

You must set up the "bar graph LED hardware" for yourselves. Each output pin should be connected to a distinct bar of the bar graph LED.

To complete the program you must add timing information, and add one instruction to shift a bit to the right with the previous bit turned off after the instruction "jsr DELAY" in the main program. Complete the program, cross-assemble it, download it, test it, and write up an analysis that explains how your program performs the desired function.

**BAR GRAPH CONTROL:** send a signal to the output pins of the Port-C one at a time with the following order:

- a. 1/2 second for pin 7 ON only '1000 0000'
- b. 1/2 second for pin 6 on only '0100 0000'
- c. 1/2 second for pin 5 on only '0010 0000'
- d. 1/2 second for pin 4 on only '0001 0000'
- e. 1/2 second for pin 3 on only '0000 1000'
- f. 1/2 second for pin 2 on only '0000 0100'
- g. 1/2 second for pin 1 on only '0000 0010'
- h. 1/2 second for pin 0 on only '0000 0001'
- i. go back to (a)

## Lab 7.0 Traveling Light at Port-B

**PROBLEM:** Traveling light on Port-B output pins

**TOPIC:** pins for output at Port-B and timing control

**PURPOSE:** To introduce low-level device control achieved thru the EVB port pins for output, and timing control achieved by counting instruction cycles.

**ASSIGNMENT:** Study the program-shell *travelB.asm*, noting specifically

1. that the EVB's memory-mapped pins for output at Port-B are used to control the 10-segment bar graph LEDs as external devices (for MCU I/O port connector pin assignments see P6-2 of M68HC11EVB Evaluation Board User's Manual);
2. how the EVB output pins are connected to particular devices (ie, what program outputs control what segment of the bar graph LED);
3. how the program's outputs control the external devices (i.e., when a particular segment of the bar graph LED is turned on and off);
4. that you must add timing information and one instruction to complete the program, and to make it perform the desired control function.

You must set up the "bar graph LED hardware" for yourselves. Each output pin should be connected to a distinct bar of the bar graph LED.

To complete the program you must add timing information, and add one instruction to shift a bit to the right with the previous bit turned off after the instruction "jsr DELAY" in the main program. Complete the program, cross-assemble it, download it, test it, and write up an analysis that explains how your program performs the desired function.

**BAR GRAPH CONTROL:** send a signal to the output pins of the Port-B one at a time with the following order:

- a. 1/2 second for pin 7 ON only '1000 0000'
- b. 1/2 second for pin 6 on only '0100 0000'
- c. 1/2 second for pin 5 on only '0010 0000'
- d. 1/2 second for pin 4 on only '0001 0000'
- e. 1/2 second for pin 3 on only '0000 1000'
- f. 1/2 second for pin 2 on only '0000 0100'
- g. 1/2 second for pin 1 on only '0000 0010'
- h. 1/2 second for pin 0 on only '0000 0001'
- i. go back to (a)

## Lab 8.0 Modulo-9 Counter at Port-C

**PROBLEM:** Modulo-9 counter using Port-C output pins

**TOPIC:** pins for output at Port-C and timing control; hardware

**PURPOSE:** To introduce low-level device control achieved thru the EVB port pins for output, timing control achieved by counting instruction cycles, and simple electronic hardware.

**ASSIGNMENT:** Study the program *moduloC.asm*, noting specifically

1. that the EVB's memory-mapped pins for output at Port-C are used to control a 7-segment LED as an external device (for MCU I/O port connector pin assignments see P6-2 of M68HC11EVB Evaluation Board User's Manual);
2. how the data at the end of the program are formed;
3. Each of the Port-C pins except pin 4 controls one of the 7 segments of the display by connecting to the outlet for that segment;
4. how the program's outputs control the external device (i.e., when a particular segment of 7-segment LED is turned on and off);

Partial hardware will be set up for you. The +5v pin and GND pin of the EVB are already connected to the 7-segment LED. You must figure out which output pin of Port-C should be connected to which outlet of the 7-segment display.

Connect the Port-C pins to the correct outlets of the 7-segment LED, cross-assemble the program, download it, test it, and write up a summary.

### **HARDWARE CONTROL:**

1. display 0 for 1/2 sec; clear 0 for 1/2 sec;
2. display 1 for 1/2 sec; clear 1 for 1/2 sec;
3. display 2 for 1/2 sec; clear 2 for 1/2 sec;
4. display 3 for 1/2 sec; clear 3 for 1/2 sec;
5. display 4 for 1/2 sec; clear 4 for 1/2 sec;
6. display 5 for 1/2 sec; clear 5 for 1/2 sec;
7. display 6 for 1/2 sec; clear 6 for 1/2 sec;
8. display 7 for 1/2 sec; clear 7 for 1/2 sec;
9. display 8 for 1/2 sec; clear 8 for 1/2 sec;
10. display 9 for 1/2 sec; clear 9 for 1/2 sec;
11. repeat from (1)



## Lab 9.0 Modulo-9 Counter at Port-B

**PROBLEM:** Modulo-9 counter using Port-B output pins

**TOPIC:** pins for output at Port-B and timing control; hardware

**PURPOSE:** To introduce low-level device control achieved thru the EVB port pins for output, timing control achieved thru instruction cycles, and simple electronic hardware.

**ASSIGNMENT:** Study the program *moduloB.asm*, noting specifically

1. that the EVB's memory-mapped pins for output at Port-B are used to control a 7-segment LED as an external device (for MCU I/O port connector pin assignments see P6-2 of M68HC11EVB Evaluation Board User's Manual);
2. how the data at the end of the program are formed;
3. Each of the Port-B pins except pin 4 controls one of the 7 segments of the display by connecting to the outlet for that segment;
4. how the program's outputs control the external device (i.e., when a particular segment of 7-segment LED is turned on and off);

Partial hardware will be set up for you. The +5v pin and GND pin of the EVB are already connected to the 7-segment LED. You must figure out which output pin of Port-B should be connected to which outlet of the 7-segment display.

Connect the Port-B pins to the correct outlets of the 7-segment LED, cross-assemble the program, download it, test it, and write up a summary.

### **HARDWARE CONTROL:**

1. display 0 for 1/2 sec; clear 0 for 1/2 sec;
2. display 1 for 1/2 sec; clear 1 for 1/2 sec;
3. display 2 for 1/2 sec; clear 2 for 1/2 sec;
4. display 3 for 1/2 sec; clear 3 for 1/2 sec;
5. display 4 for 1/2 sec; clear 4 for 1/2 sec;
6. display 5 for 1/2 sec; clear 5 for 1/2 sec;
7. display 6 for 1/2 sec; clear 6 for 1/2 sec;
8. display 7 for 1/2 sec; clear 7 for 1/2 sec;
9. display 8 for 1/2 sec; clear 8 for 1/2 sec;
10. display 9 for 1/2 sec; clear 9 for 1/2 sec;
11. repeat from (1)

## Lab 10.0 Software Interrupt Handling

**PROBLEM:** Modulo-9 counter using software interrupt technique

**TOPIC:** software interrupt

**PURPOSE:** To introduce software interrupt handling techniques, interrupt vector jump table, and low-level device control.

**ASSIGNMENT:** Study the program *swi.asm* carefully, noting specifically

1. PVSWI is the pseudo vector address for software interrupt
2. that the EVB's memory-mapped pins for output at Port-B are used to control a 7-segment LED as external devices (for MCU I/O port connector pin assignments see P6-2 of M68HC11EVB Evaluation Board User's Manual);
3. "fcb" means "form constant byte"; "rmb" means "reserve memory bytes";
4. how the data at the end of the program are formed;
5. each of the Port-B pins except pin 4 controls one of the 7 segments of the LED by connecting to the outlet for that segment;
6. how the program's outputs control the external devices (i.e., when a particular segment of the 7-segment LED is turned on and off);

The "7-segment LED hardware" will be set up for you. You don't have to modify the hardware in order to execute the program.

Analyze the program, especially how the control of the program flows in the LOOP loop. Cross-assemble the program, download it, test it, and write up a summary that describes how the software interrupt works, and how the control of the program flows.

### **HARDWARE CONTROL:**

1. display 0 for 1 sec;
2. display 1 for 1 sec;
3. display 2 for 1 sec;
4. display 3 for 1 sec;
5. display 4 for 1 sec;
6. display 5 for 1 sec;
7. display 6 for 1 sec;
8. display 7 for 1 sec;
9. display 8 for 1 sec;
10. display 9 for 1 sec;
11. repeat from (1)

## Lab 11.0 Output Compare Function

**PROBLEM:** output compare using interrupt technique

**TOPIC:** output compare register 5 (OC5) interrupt technique

**PURPOSE:** To introduce interrupt handling using output compare register 5 (OC5), interrupt vector jump table and free-running counter.

**ASSIGNMENT:** Study the program *oc5int.asm* carefully, noting specifically

1. REG\_ST (\$1000) is the starting address of the register block. With offset specified in the "equ" directive, the memory location for timer control register 1 (TCTL1), timer interrupt mask (TMSK1), timer interrupt flag 1 (TFLG1), timer output compare register 5 (TOC5), and free-running counter (TCNT) can be found;
2. PVOC5 is the pseudo vector address for OC5 interrupt;
3. that the program writes data to the memory starting at \$D000, and stops at \$DFFF, since \$E000 and up are monitor EPROM, and cannot be written to (see EVB Memory Map Diagram on P5-3 of M68HC11EVB Evaluation Board User's Manual);
4. that \$D000 cannot be changed to \$6000, since the program doesn't limit the upper boundary, and will otherwise overwrite the user program data from \$C000;
5. that OC5 controls pin 3 of Port-A;
6. what the functions of the TFLG1, TCNT, TMSK1 and TOC5 in the program are.

Cross-assemble the program, download it, execute it several times, offload data computed by the program at \$D000- \$DFFF to the host machine, and save it in different files. Carefully analyze the different data files, and the functions of the TFLG1, TMSK1, TCNT, and TOC5 in the program. Write up a summary of your analysis.

## Lab 12.0 Polling with OC2

**PROBLEM:** modulo-9 counter with output compare function and polling technique

**TOPIC:** output compare register 2 (OC2) and polling technique

**PURPOSE:** To introduce polling techniques using output compare register 2(OC2) and free-running counter.

**ASSIGNMENT:** Study the program-shell *timepoll.asm* carefully, noting specifically

1. that BASE (\$1000) is the base address of the register block. With offset specified in the “equ” directive, the memory location for timer interrupt flag 1 (TFLG1), timer output compare register 2 (TOC2), and free-running counter (TCNT) can be found.
2. how the polling technique in T\_LOOP works.
3. that you must add timing information to complete the program

The “modulo-9 hardware” will be set up for you. You don’t need to modify the hardware in order to complete the program. The function of the program is to display digits on the 7-segment LED. Digit 0 is initialized, and should be displayed on the 7-segment LED for one second before it is incremented. A tic sound accompanies each display. Each incremented digit should be displayed for one second. After digit 9 is displayed it is reset to 0, and then the same procedure is repeated.

Compute the number of iterations T\_LOOP should loop in order for each digit to be displayed for one second. Replace the question marks in “ldy #???” with the value you computed, cross-assemble the program, download it, and test it. Carefully analyze the POLLING part of the program, especially the instructions related to free-running counter (TCNT) and output compare register 2 (TOC2). Write up a summary of your analysis.

### HARDWARE CONTROL:

1. display 0 for 1 sec with a tic sound;
2. display 1 for 1 sec with a tic sound;
3. display 2 for 1 sec with a tic sound;
4. display 3 for 1 sec with a tic sound;
5. display 4 for 1 sec with a tic sound;
6. display 5 for 1 sec with a tic sound;
7. display 6 for 1 sec with a tic sound;
8. display 7 for 1 sec with a tic sound;
9. display 8 for 1 sec with a tic sound;
10. display 9 for 1 sec with a tic sound;
11. repeat from (1)

## Lab 13.0 OC5 Interrupt

**PROBLEM:** modulo-9 counter with output compare function and interrupt handling

**TOPIC:** output compare register 5 (OC5) interrupt technique

**PURPOSE:** To introduce interrupt handling using output compare register 5(OC5) and free-running counter.

**ASSIGNMENT:** Study the program-shell *timeint.asm* carefully, noting specifically

1. that BASE (\$1000) is the starting address of the register block. With offset specified in the "equ" directive, the memory location for timer interrupt mask (TMSK1), timer interrupt flag 1 (TFLG1), timer output compare register 5 (TOC5), and free-running counter (TCNT) can be found.
2. that PVOC5 is the pseudo vector address for OC5 interrupt
3. how interrupt service routine works
4. what the functions of the TFLG1, TCNT, TMSK1 and TOC5 are in the program.
5. that you must add timing information to complete the program

The "modulo-9 hardware" will be set up for you. You don't need to modify the hardware in order to complete the assignment. The function of the program is to display digits on the 7-segment LED. Digit 0 is initialized, and should be displayed on the 7-segment LED for one second before it is incremented. A tic sound accompanies each display. Each incremented digit should be displayed for one second. After digit 9 is displayed it is reset to 0, and then the same procedure is repeated.

Compute the number of iterations TLP should loop in order for each digit to be turned on for one second. Replace the question marks in "ldy #???" with the value you computed, cross-assemble the program, download it, and test it. Carefully analyze the interrupt service routine INTERRUPT of the program, especially the instructions related to free-running counter (TCNT) and output compare register 5 (TOC5). Write up a summary of your analysis.

### HARDWARE CONTROL:

1. display 0 for 1 sec with a tic sound;
2. display 1 for 1 sec with a tic sound;
3. display 2 for 1 sec with a tic sound;
4. display 3 for 1 sec with a tic sound;
5. display 4 for 1 sec with a tic sound;
6. display 5 for 1 sec with a tic sound;
7. display 6 for 1 sec with a tic sound;
8. display 7 for 1 sec with a tic sound;
9. display 8 for 1 sec with a tic sound;
10. display 9 for 1 sec with a tic sound;
11. repeat from (1)

## Lab 14.0 Timer Using Polling

**PROBLEM:** alarm system with output compare function and polling techniques

**TOPIC:** output compare register 2 (OC2) and polling technique

**PURPOSE:** To introduce polling techniques using output compare register 2(OC2) and free-running counter.

**ASSIGNMENT:** Study the program-shell *alarmpol.asm* carefully, noting specifically

1. that BASE (\$1000) is the base address of the register block. With offset specified in the “equ” directive, the memory location for timer interrupt flag 1 (TFLG1), timer output compare register 2 (TOC2), and free-running counter (TCNT) can be found.
2. how the polling technique in T\_LOOP works.
3. that you must add timing information to complete the program

The “alarm system hardware” will be set up for you. You don’t need to modify the hardware in order to complete the assignment. The program drives a 14-segment LED and a piezo buzzer. Digit 99 is initialized, and should be displayed on the MAN6610 (14-segment) LED for 1/4 second before it is decreased, and then the decreased number will be displayed for 1/4 second. It goes on until 0 is displayed. A tic sound accompanies each display. After the digit 0 is displayed it generates beeps until the EVB reset button is pressed.

Compute the number of iterations T\_LOOP should loop in order for each digit to be on for one second. Replace the question marks in the “ldy #???” with the value you computed, cross-assemble the program, download it, and test it. Carefully analyze the POLLING part of the program, especially the instructions related to free-running counter (TCNT) and output compare register (TOC2). Write up a summary of your analysis.

### HARDWARE CONTROL:

1. display 99 for 1/4 sec with a tic sound;
2. display 98 for 1/4 sec with a tic sound;
- ...
99. display 1 for 1/4 sec with a tic sound;
100. display 0 for 1/4 sec with a tic sound;
101. generate beeps until the RESET button is pressed

## Lab 15.0 Timer Using Interrupt Handling

**PROBLEM:** alarm system with output compare function and interrupt handling

**TOPIC:** output compare register 5 (OC5) interrupt handling technique

**PURPOSE:** To introduce interrupt handling using output compare register 5(OC5) and free-running counter.

**ASSIGNMENT:** Study the program-shell *alarmint.asm* carefully, noting specifically

1. that BASE (\$1000) is the starting address of the register block. With offset specified in the “equ” directive, the memory location for timer interrupt mask (TMSK1), timer interrupt flag 1 (TFLG1), timer output compare register 5 (TOC5), and free-running counter (TCNT) can be found.
2. that PVOC5 is the pseudo vector address for OC5 interrupt
3. how interrupt service routine works
4. what the functions of the TFLG1, TCNT, TMSK1 and TOC5 are in the program.
5. that you must add timing information to complete the program

The “alarm system hardware” will be set up for you. You don’t need to modify the hardware in order to complete the assignment. The program drives a 14-segment LED and a piezo buzzer. Digit 99 is initialized, and should be displayed on the MAN6610 (14-segment) LED for 1/4 second before it is decreased, and then the decreased number will be displayed for 1/4 second. It goes on until 0 is displayed. A tic sound is accompanied to each display. After the digit 0 is displayed it generates beeps until the EVB reset button is pressed.

Compute the number of iterations TLP should loop in order for each digit to be turned on for 1/4 second. Replace the question marks in “ldy #????” with the value you computed, cross-assemble the program, download it, and test it. Carefully analyze the interrupt service routine INTERRUPT of the program, especially the instructions related to free-running counter (TCNT) and output compare register 5 (TOC5). Write up a summary of your analysis.

### HARDWARE CONTROL:

1. display 99 for 1/4 sec with a tic sound;
2. display 98 for 1/4 sec with a tic sound;
- ...
99. display 1 for 1/4 sec with a tic sound;
100. display 0 for 1/4 sec with a tic sound;
101. generate beeps until the RESET button is pressed

## Lab 16.0 Inter-Process Communication

**PROBLEM:** inter-process communication at Port-B and Port-C

**TOPIC:** distributed programming; polling mechanism

**PURPOSE:** To introduce you to REAL low-level distributed programming mechanisms for inter-processor SEND and RECEIVER.

**ASSIGNMENT:** Study the program-shell *ipcsnd.asm* and *ipcrcv.asm*, noting specifically

1. that the sender and receiver must be synchronized at a very low level to assure that the receiver is ready to accept incoming data when the sender is ready to send it. This means the delay between two sends must be larger than the worst case “receiver ready” delay.
2. that in the subroutine EVENT between two sends, some delay value must be set on purpose to wait for the receiver to get ready to receive.
3. that you must add timing information to make the programs perform the desired function with the minimal cost.

To get the sender and receiver synchronized, you must

1. determine the maximal number of instruction cycles in the receiver program that may lapse between the time the receiver recognizes an arbitrary “send(n)” and the time it is ready to recognize the next “send(n+1)”;
2. compute delay cycles in the sender program for each of the 4 atomic sends in the subroutine EVENT, assuming the 2nd instruction in the subroutine IPC\_PAUSE is “ldx # $\$01$ ”;
3. compute the minimal iterations of D\_LOOP in the sender’s subroutine IPC\_PAUSE to guarantee an appropriate delay between each of the 4 atomic sends, and replace the question marks “???” with the value you computed.

Cross-assemble the sender and receiver programs, download sender to one EVB board and receiver to another, and execute them. Check if there are message sending reliability problems. If there are, recalculate delay value, and modify it if necessary. Execute the programs twice, offloading the stored data from both the sender and receiver following each execution to produce two sets of data for the process/processor mapping. Swap the assignment of sender and receiver processes (i.e., exchange the boards the sender and receiver were downloaded), execute them another two times, save the data to produce another two sets of data for the exchanged process/processor mapping. Note that the swap requires exchanging both the software on boards and the hardware connection of the boards. Edit the files containing the sets of offloaded data to prepare them for post-processing analysis. Run the program IPCproc with three arguments: send-data-file, receive-data-file, output-file. IPCproc implements a preliminary analysis of the sender/receiver execution, and puts the results in the output file. Note the difference between sender and receiver times shouldn’t be constant because of the uncertainty of the polling nature, but it should be bounded with a predictable average.

Write a written summary about your analysis. Point out any unexpected results, especially those supporting the hypotheses about the ideal behavior of the system or those disagreeing with expected behavior. Support your summary with data.

*IPCsnd.asm/IPCrcv.asm* relies on the following M68HC11EVB facilities:



1. each write to Port-B drives an 8-bit value out of the EVB's Port-B pins, and drives a '1' out of the EVB's STR-B pin;
2. an incoming '1' on an EVB's STR-A pin causes the EVB to "latch" a data value from its Port-CL pins into its Port-CL internal register;
3. an successful latch on a Port-CL value is signalled to the receiver by a status bit in EVB's PIOC register.

Therefore, sender's Port-B needs to be connected to receiver's Port-C, and sender's STR-B to receiver's STR-A. The connection of their specific data bits are listed below.

Port-B (SENDER) data bit	Port-C(RECEIVER) data bit
0	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7

STR-A: header pin 4

STR-B: header pin 6

## Lab 17.0 Multi-Character Display

**PROBLEM:** LCD-II display at Port-B and Port-C

**TOPIC:** low-level device control

**PURPOSE:** To introduce device control of multi-character display hardware.

**ASSIGNMENT:** Study the program-shell *string.asm*, noting specifically

1. that the EVB's memory-mapped output Port-B and Port-C pins are used to control the external devices of the LCD-II display (for MCU I/O port connector pin assignments see P6-2 of M68HC11EVB Evaluation Board User's Manual);
2. that you must complete some sub-programs to make the program perform the desired control function.

The LCD-II is supposed to display the message "HOW NOW, BROWN COW?" coming out from the right end of the screen. When all characters are displayed, the message is cleared, and the same procedure is repeated again until the EVB RESET button is pressed.

The "LCD-II hardware" will be set up for you, and you don't need to modify it to complete the assignment. To complete the program you must write the subroutines marked ??? in the program similar to the subroutines B and C, which use ASCII to display the characters 'B' and 'C', respectively. These subroutines are:

H, N, O, R, W, COMMA, QUESTION\_MARK

Cross-assemble the completed program, download it, test it, and write up a summary of your analysis.

**Appendix D**

**Computer Architecture and Assembly Language**

**INSTRUCTOR'S MANUAL**

**ZUYI CHEN**

**July, 1992**

**Computer Science Department**

**University of Montana**

## TABLE OF CONTENTS

Table of Contents	i
1.0: Output Ports	1
2.0: Input Port	3
3.0: Traffic Signal at Port-C	5
4.0: Traffic Signal at Port-B	7
5.0: Music Tunes at Port-B	9
6.0: Traveling Light at Port-C	10
7.0: Traveling Light at Port-B	12
8.0: Modulo-9 Counter at Port-C	14
9.0: Modulo-9 Counter at Port-B	15
10.0: Software Interrupt Handling	16
11.0: Output Compare Function	18
12.0: Polling with OC2	20
13.0: OC5 Interrup	22
14.0: Timer Using Polling	24
15.0: Timer Using Interrupt Handling	26
16.0: Inter-Process Communication	28
17.0: Multi-Character Display	30

## Lab 1.0 Output Ports

**PROBLEM:** output pins at Port-A, Port-B, Port-C and Port-D

**TOPIC:** pins for output at ports and timing control

**PURPOSE:** To introduce low-level device control achieved thru the EVB port pins for output, and timing achieved by counting instruction cycles.

**ASSIGNMENT:** see Lab 1.0

### BACKGROUND REQUIRED:

1. Memory-mapped Port-A, Port-B, Port-C and Port-D pins
2. Memory-mapped Port-A, Port-C and Port-D direction control register
3. When data '1' is driven out of a port pin, its voltage is high; when data '0' is driven out of a port pin, its voltage is low
4. To light a bar graph LED, the positive side of the display should be connected to EVB port output pins, and the other side connected to EVB's GND pin

**HARDWARE DIAGRAM:** see Diagram 1.0

**SOLUTION:** All pins of Port-B and Port-C can be used for output. In Port-A, however, pin 1 - pin 3 are input only, and pin 7 can be used for output if the corresponding bit in Port-A direction control register (DDRA) is set. In Port-D, only pin 2 - pin 5 are for output. Pin 1 always reads, and pin 2 always writes, so its voltage is always high.

For the hardware setup, connect each of the output pins of the ports to a distinct bar of the bar graph LEDs on the positive side, usually, the side with serial number and other codes; connect the other side to the GND pin of the EVB. The implemented subprograms follow.

```
*****
* Subpgm LITE_B
* turn on the bar graph display one bit at a time connected to Port-B
*****
LITE_B ldab #8
ldaa #$80
MORE_B staa PORTB
ldy #INNER
jsr DELAY
asra
decb
bne MORE_B
rts
*****
* Subpgm CLEAR_B
* turn off the bar graph display connected to Port-B
```

```

*****
CLEAR_B
ldaa #$00
staa PORTB
rts
*****
* Subpgm LITE_C
* turn on the bar graph display one bit at a time connected to
* Port-C
*****
LITE_C ldab #8
ldaa #$80
MORE_C staa PORTC
ldy #INNER
jsr DELAY
asra
decb
bne MORE_C
rts
*****
* Subpgm CLEAR_C
* turn off the bar graph display connected to Port-C
*****
CLEAR_C
ldaa #$00
staa PORTC
rts
*****
* Subpgm LITE_D
* turn on the bar graph display one bit at a time connected to
* Port-D
*****
LITE_D ldab #4
ldaa #$E0
MORE_D staa PORTD
ldy #INNER
jsr DELAY
asra
decb
bne MORE_D
rts
*****
* Subpgm CLEAR_D
* turn off the bar graph display connected to Port-D
*****
CLEAR_D
ldaa #$00
staa PORTD
rts

```

## Lab 2.0 Input Port

**PROBLEM:** input pins at Port-C; output pins at Port-A and Port-B

**TOPIC:** pins for input and output at various ports and cycle analysis

**PURPOSE:** To introduce port pins for input and output, and to free-running counter.

**ASSIGNMENT:** see Lab 2.0

### BACKGROUND REQUIRED:

1. Port-C direction control register
2. Memory-mapped free-running counter

**HARDWARE DIAGRAM:** see Diagram 2.0

**SOLUTION:** "ldx TCNT" captures the current time. Let Delta be the difference between the current time stamp and the one in the next iteration of LOOP in the program. Theoretically it takes  $54 = 84$  cycles between two time stamps. Here is how the value is obtained. The current time stamp starts after the instruction "ldx TCNT" is executed in the subprogram STORE\_OUTPUT.

```
STORE_OUTPUT
[ 5 ] staa 0,Y
[ 5 ] staa 1,Y
[ 5 ] ldx TCNT
[ 6 ] stx 2,Y
[ 5 ] rts
```

The instructions after this in the subprogram takes  $6 + 5 = 11$  cycles, and let the 11 cycles be A. The control returns to the instruction after the "jsr STORE\_OUTPUT" in the following.

```
LOOP
[ 4 ] ldab DATA
[ 4 ] stab PORTB
[ 6 ] jsr GET_INPUT
[ 6 ] jsr STORE_OUTPUT
[ 2 ] incb
[ 4 ] stab DATA
[ 2 ] ldab #$04
[ 4 ] aby
[ 5 ] cpy #END+4
[ 3 ] bne LOOP
```

Let cycles be  $B = 2 + 4 + 2 + 4 + 5 + 3 + 4 + 4 + 6 + 6 = 40$  cycles from this point to the point "jsr STORE\_OUTPUT" is executed. Now expand the subprogram GET\_INPUT.

```
GET_INPUT
```

```
[ 4 ] ldaa PIOC  
[ 2 ] bita #$80  
[ 3 ] beq GET_INPUT  
[ 4 ] ldaa PORTCL  
[ 5 ] rts
```

The subprogram takes  $C = 4 + 2 + 3 + 4 + 5 = 18$  cycles. Finally expand the subprogram STORE\_OUTPUT to the point "ldx TCNT" is executed. The instructions there take  $D = 5 + 5 + 5 = 15$  cycles. The control has traveled from the previous time stamp to the current time stamp. The total cycles  $\Delta = A + B + C + D = 11 + 40 + 18 + 15 = 84$ .

Now check the data computed by the program in \$D000 - \$DFFF. Between two consecutive time stamps the difference is  $\$54 = 85$  cycles. Therefore the theoretical data matches the real output. Note the free-running counter is a 16-bit counter, it is a modulo-\$FFFF and wraps around after \$FFFF.



## Lab 3.0 Traffic Signal at Port-C

**PROBLEM:** TRAFFIC SIGNAL at Port-C

**TOPIC:** Port-C pins for output and timing control

**PURPOSE:** To introduce low-level device control achieved through the EVB Port-C pins, and timing achieved by counting instruction cycles.

**ASSIGNMENT:** see Lab 3.0

### BACKGROUND REQUIRED:

1. Memory-mapped I/O Port-C pins
2. Memory-mapped Port-C direction control register
3. When data '1' is driven out of a header pin, its voltage is high; when data '0' is driven out of a header pin, its voltage is low
4. The EVB's MCU internal E clock is 2 MHz = 2,000,000 machine cycles
5. Each instruction's machine cycle can be obtained by cross-assembling the program with switches "-l c" For example: "as11 pgm.asm -l c > pgm.lst" on a PC The cycles are enclosed in square brackets [ ]

**HARDWARE DIAGRAM:** see Diagram 3.0

**SOLUTION:** Theoretically, the number of iterations of DELAY loop is computed to be 68 to the nearest integer for the delay of 10 seconds, and 7 for the delay of 1 second. Here is how the values are obtained. The GREEN light starts on after the instruction "staa PORTC" is executed in subroutine GREEN, and should stay on until the "staa PORTC" is executed in subroutine YELLOW. Between these two points, arbitrarily including the second "staa PORTC", there are the following instructions following the control flow:

```
[5] rts
[4] ldy #68
[6] jsr DELAY
[2] ldab #3
BLINK_LOOP
[6] jsr YELLOW
```

Let these cycles be C, there are  $C = 23$  machine cycles in the above. Now expand the subroutine DELAY in the following:

```
DELAY
[3] ldx #DCOUNT
[3] DLOOP dex
[3] bne DLOOP
```

```

[4] dey
[3] bne DELAY
[5] rts

```

DCOUNT = \$C000 = 49152. The DLOOP will repeat  $N = 6 * 49152 = 294912$  times. The rest of DELAY (except rts, since rts are not repeated in the DELAY loop) take 10 machine cycles. Adding to N yields  $N = 294922$ , which will be repeated the number of times equal to the operand in "ldy #????". "rts" in DELAY takes 5 cycles, and is added to C above, making  $C = 23 + 5 = 28$ . Now expand the subroutine YELLOW to the point of "staa PORTC":

```

[2] ldaa #2
[4] staa PORTC

```

Adding these 6 cycles to C above yields  $C = 34$ . In order to let the light stay on for 10 seconds = 20,000,000 cycles, the following equation should hold.

$$20,000,000 = 294922 * N + C, \text{ where } C \text{ is } 34, \text{ and } N \text{ is the iterations of DELAY loop.}$$

N is computed to be 68, rounding to the nearest integer from 67.81, for GREEN light ON and RED light ON. Likewise, for 1 second delay for YELLOW light N is computed to be 7, rounding from 6.78. There are a few cycle deviations for C between different light signals, but they can be ignored here.

The program runs correctly with the above values inserted in the appropriate places.

## Lab 4.0 Traffic Signal at Port-B

**PROBLEM:** TRAFFIC SIGNAL at Port-B

**TOPIC:** Port-B pins for output and timing control

**PURPOSE:** To introduce low-level device control achieved through the EVB Port-B pins, and timing achieved by counting instruction cycles.

**ASSIGNMENT:** see Lab 4.0

### BACKGROUND REQUIRED:

1. Memory-mapped I/O Port-B pins
2. When data '1' is driven out of a header pin, its voltage is high; when data '0' is driven out of a header pin, its voltage is low
3. The EVB's MCU internal E clock is 2 MHz = 2,000,000 machine cycles
4. Each instruction's machine cycle can be obtained by cross-assembling the program with switches "-l c" For example: "as11 pgm.asm -l c > pgm.lst" on a PC The cycles are enclosed in square brackets [ ]

**HARDWARE DIAGRAM:** see Diagram 4.0

**SOLUTION:** Theoretically, the number of iterations of DELAY loop is computed to be 68 to the nearest integer for the delay of 10 seconds, and 7 for the delay of 1 second. Here is how the values are obtained. The GREEN light starts on after the instruction "staa PORTB" is executed in subroutine GREEN, and should stay on until the "staa PORTB" is executed in subroutine YELLOW. Between these two points, arbitrarily including the second "staa PORTB", there are the following instructions following the control flow:

```
[5] rts
[4] ldy #68
[6] jsr DELAY
[2] ldab #3
BLINK_LOOP
[6] jsr YELLOW
```

Let these cycles be C, there are C = 23 machine cycles in the above. Now expand the subroutine DELAY in the following:

```
DELAY
[3] ldx #DCOUNT
[3] DLOOP dex
[3] bne DLOOP
[4] dey
```

```
[3] bne DELAY
[5] rts
```

DCOUNT = \$C000 = 49152. The DLOOP will repeat  $N = 6 * 49152 = 294912$  times. The rest of DELAY (except rts, since rts are not repeated in the DELAY loop) take 10 machine cycles. Adding to N yields  $N = 294922$ , which will be repeated the number of times equal to the operand in "ldy #????". "rts" in DELAY takes 5 cycles, and is added to C above, making  $C = 23 + 5 = 28$ . Now expand the subroutine YELLOW to the point of "staa PORTB":

```
[2] ldaa #2
[4] staa PORTB
```

Adding these 6 cycles to C above yields  $C = 34$ . In order to let the light stay on for 10 seconds = 20,000,000 cycles, the following equation should hold.

$20,000,000 = 294922 * N + C$ , where C is 34, and N is the iterations of DELAY loop.

N is computed to be 68, rounding to the nearest integer from 67.81, for GREEN light ON and RED light ON. Likewise, for 1 second delay for YELLOW light N is computed to be 7, rounding from 6.78. There are a few cycle deviations for C between different light signals, but they can be ignored here.

The program runs correctly with the above values inserted in the appropriate places.

## Lab 5.0 Music Tunes at Port-B

**PROBLEM:** Music notes at Port-B

**TOPIC:** timing control

**PURPOSE:** To introduce low-level device control achieved thru the EVB Port-B pins, and timing control achieved by counting instruction cycles.

**ASSIGNMENT:** see Lab 5.0

### BACKGROUND REQUIRED:

1. Memory-mapped output Port-B pins
2. When data '1' is driven out of a Port-B pin, its voltage is high; when data '0' is driven out of a Port-B pin, its voltage is low
3. The EVB's MCU internal E clock is 2 MHz
4. Each instruction's machine cycle can be obtained by cross-assembling the program with switches "-l c" For example: "as11 pgm.asm -l c > pgm.lst" on a PC The cycles are enclosed in square brackets [ ]

**HARDWARE DIAGRAM:** see Diagram 5.0

**SOLUTION:** There are  $12 * \text{FREQ} + 32$  cycles in RESONANT subroutine. All except the 5 cycles for rts will be repeated the number of times equal to the value of index register Y on the subprogram entry. To make each note stay for half sec, it should take the nearest cycles to 1,000,000 for the 2 MHz EVB MCU. The part for each music note takes 18 cycles plus 5 cycles for rts of RESONANT subroutine, totalling 23 cycles. Hence the equation  $Y*(12*\text{FREQ}+27)+23 = 1,000,000$ , where FREQ varies for each music note, and Y is the integer value of the index register Y on RESONANT entry for each music tune. Note the music notes are not standard.

For music tune "la"  $N*(12*392+27) + 23 = 1000000$ . N is computed to be 211.

For music tune "so"  $N*(12*440+27) + 23 = 1000000$ . N is computed to be 188.

For music tune "fa"  $N*(12*494+27) + 23 = 1000000$ . N is computed to be 168.

For lower music tune "mee"  $N*(12*523+27) + 23 = 1000000$ . N is computed to be 159.

## Lab 6.0 Traveling Light at Port-C

**PROBLEM:** Traveling light on Port-C output pins

**TOPIC:** pins for output at Port-C and timing control

**PURPOSE:** To introduce low-level device control achieved thru the EVB port pins for output, and timing achieved by counting instruction cycles.

**ASSIGNMENT:** see Lab 6.0

### BACKGROUND REQUIRED:

1. Memory-mapped Port-C
2. Memory-mapped Port-C direction control register
3. To light a bar graph LED, the positive side of the LED should be connected to EVB port output pins, and the other side connected to EVB's GND pin
4. Assembly instruction logical shift right "lsr"

**HARDWARE DIAGRAM:** see Diagram 6.0

**SOLUTION:** All pins of Port-C can be used for output if the Port-C direction control register has been written with \$FF. The instruction to be added after "jsr DELAY" in the main program is "lsrb".

The number of iterations to the nearest integer for DELAY loop is 42. Here is how it is obtained. In the subprogram DELAY

```

DELAY
[ 3 ] ldx #DLOOP_COUNT
[ 3 ] DLOOP dex
[ 3 ] bne DLOOP
[ 4 ] dey
[ 3 ] bne DELAY
[ 5 ] rts

```

$(3+(3+3)*4000+4+3)$  cycles will be repeated the number of times equal to DELAY\_COUNT. Let the result be A,  $A = (3+(3+3)*4000+4+3)*DELAY\_COUNT = 24010 * DELAY\_COUNT$ .

The miscellaneous part includes 5 cycles for "rts" DELAY and for the instructions in the main program.

```

[ 2 ] PIN_POS ldaa #8
[ 2 ] ldab #$80
[ 4 ] LOOP stab PORTC
[ 4 ] ldy #DELAY_COUNT

```

```

[ 6 ] jsr DELAY
[ 2 ] lsr b
[ 2 ] dec a
[ 3 ] bne LOOP
[ 3 ] bra PIN_POS

```

They take  $2 + 2 + 4 + 4 + 6 + 2 + 2 + 3 + 3$  cycles for one iteration of PIN\_POS loop, and  $4 + 4 + 6 + 2 + 2 + 3$  cycles for one iteration of LOOP loop. The difference between the two is trivial, and can be ignored. So let miscellaneous part be  $M$ ,  $M = 5 + 2 + 2 + 4 + 4 + 6 + 2 + 2 + 3 + 3 = 33$ .

To let each light stay on for half a second, the following equation holds.

$$1,000,000 = A + M = 24010 * \text{DELAY\_COUNT} + 33$$

DELAY\_COUNT is computed to be 42, rounding from 41.65.

For the “hardware setup”, connect each of the output pins of the ports to a distinct bar of the bar graph LED on the positive side, usually, the side with serial number; connect the other side to the GND pin of the EVB.

## Lab 7.0 Traveling Light at Port-B

**PROBLEM:** Traveling light on Port-B output pins

**TOPIC:** pins for output at Port-B and timing control

**PURPOSE:** To introduce low-level device control achieved thru the EVB port pins for output, and timing achieved by counting instruction cycles.

**ASSIGNMENT:** see Lab 7.0

### BACKGROUND REQUIRED:

1. Memory-mapped Port-B
2. To light a bar graph LED, the positive side of the LED should be connected to EVB port output pins, and the other side connected to EVB's GND pin
3. Assembly instruction logical shift right "lsr"

**HARDWARE DIAGRAM:** see Diagram 7.0

**SOLUTION:** The instruction to be added after "jsr DELAY" in the main program is "lsrb".

The number of iterations to the nearest integer for DELAY loop is 42. Here is how it is obtained. In the subprogram DELAY

```

DELAY [ 3 ] ldx #DLOOP.COUNT
      [ 3 ] DLOOP dex
      [ 3 ] bne DLOOP
      [ 4 ] dey
      [ 3 ] bne DELAY
      [ 5 ] rts

```

$(3+(3+3)*4000+4+3)$  cycles will be repeated the number of times equal to DELAY.COUNT. Let the result be A,  $A = (3+(3+3)*4000+4+3)*DELAY.COUNT = 24010 * DELAY.COUNT$ .

The miscellaneous part includes 5 cycles for "rts" DELAY and for the instructions in the main program.

```

[ 2 ] PIN_POS ldaa #8
[ 2 ] ldab #$80
[ 4 ] LOOP stab PORTB
[ 4 ] ldy #DELAY.COUNT
[ 6 ] jsr DELAY
[ 2 ] lsrb
[ 2 ] deca
[ 3 ] bne LOOP
[ 3 ] bra PIN_POS

```



They take  $2 + 2 + 4 + 4 + 6 + 2 + 2 + 3 + 3$  cycles for one iteration of PIN\_POS loop, and  $4 + 4 + 6 + 2 + 2 + 3$  cycles for one iteration of LOOP loop. The difference between the two is trivial, and can be ignored. So let miscellaneous part be M,  $M = 5 + 2 + 2 + 4 + 4 + 6 + 2 + 2 + 3 + 3 = 33$ .

To let each light stay on for half a second, the following equation holds.

$$1,000,000 = A + M = 24010 * \text{DELAY\_COUNT} + 33$$

DELAY\_COUNT is computed to be 42, rounding from 41.65.

For the hardware setup, connect each of the output pins of the ports to a distinct bar of the bar graph LED on the positive side, usually, the side with serial number; connect the other side to the GND pin of the EVB.

## Lab 8.0 Modulo-9 Counter at Port-C

**PROBLEM:** Modulo-9 counter using Port-C output pins

**TOPIC:** pins for output at Port-C and timing control; hardware

**PURPOSE:** To introduce low-level device control achieved thru the EVB port pins for output, timing achieved by counting instruction cycles, and to simple electronic hardware.

**ASSIGNMENT:** see Lab 8.0

**BACKGROUND REQUIRED:**

FCB - form constant byte

**HARDWARE DIAGRAM:** see Diagram 8.0

**SOLUTION:** The orders of Port-C header pins are scrambled. Students need to figure out which pin controls which segment by try and fail.

\*\*\*\*\*  
 \* DATA: Table of digit/Port-C pin mapping  
 \*\*\*\*\*

```

*
*  -b0-----|-----|---[+5v]-
*  -b1-----| |-----|---b6---
*  -[+5v]-| |-----|
*  |-----| |-----|---b7---
*  |-----| |-----|---b5---
*  -----| .|-----|---[+5v]-
*  -b2-----|-----|---b3---
*

```

\* Note: b0, b1, b2 .. indicates pin 0, pin 1, pin 2 .. of PORT-C.  
 \* Bit 4 of port-C is not used; [+5v] indicates connecting +5v pin  
 to one of the 3 outlets annotated by [+5v]



## Lab 10.0 Software Interrupt Handling

**PROBLEM:** Modulo-9 counter using software interrupt handling

**TOPIC:** software interrupt handling

**PURPOSE:** To introduce software interrupt handling technique and interrupt vector jump table, to low-level device control achieved thru the EVB port pins for output.

**ASSIGNMENT:** see Lab 10.0

### BACKGROUND REQUIRED:

1. Interrupt vector jump table
2. Software interrupt technique
3. Interrupt service routine

**HARDWARE DIAGRAM:** see Diagram 10.0

**SOLUTION:** The following four lines

```
ldaa #$7E ; extended op code of jump instruction
staa PVSWI ; pseudo vector for SWI
ldx #INTERRUPT ; put address of Interrupt Routine
stx PVSWI+1 ; after the address of jmp
```

Set the instruction for jump to interrupt service routine. Next look at the main program and the interrupt service routine:

```
LOOP swi
bra LOOP
INTERRUPT
ldy #83
jsr DELAY
jsr CLEAR
ldab CUR_DIGIT
jsr OUT_DIGIT
incb
cmpb #10
bne SKIP
ldab #0
SKIP stab CUR_DIGIT
rti
```

When "swi" is executed, the software interrupt is enabled, and the interrupt service routine INTERRUPT is executed. In the routine, it sets delay time for about 1 second before a digit is sent to be displayed on a 7-segment LED. It then increments the digit by 1, displays that digit. It resets

the digit to 0 after 9 is displayed. There is a real time interrupt (rti) at the end of the routine; and the control of the main program loops back to LOOP by "bra LOOP". Next time "swi" is executed, it repeats the same procedure described above.

## Lab 11.0 Output Compare Function

**PROBLEM:** output compare using interrupt technique

**TOPIC:** output compare register 5 (OC5) interrupt technique

**PURPOSE:** To introduce interrupt technique using output compare register 5(OC5), interrupt vector jump table, and free-running counter.

**ASSIGNMENT:** see Lab 11.0

### BACKGROUND REQUIRED:

1. Timer control register 1 (TCTL1),
2. Timer interrupt mask (TMSK1),
3. Timer interrupt flag 1 (TFLG1),
4. Timer output compare register 5 (TOC5), and
5. Free-running counter (TCNT)
6. Interrupt vector jump table
7. Difference of RAM and ROM
8. EVB memory map diagram
9. Relation between OC5 and pin 3 of Port-A

**NOTE:** \$D000 cannot be changed to \$6000 in the program, since the program doesn't limit the upper boundary, and will otherwise overwrite the user program data from \$C000.

**HARDWARE DIAGRAM:** see Diagram 11.0

**SOLUTION:** The Port-A pin control block includes logic for timer function beside for general-purpose I/O. Pin PA6-PA3 can be used as output-compare pins. PA3 is used as output-compare 5 (OC5) pin. So whenever the OC5 bit is set in timer control register (TCTL1) pin PA3 is set.

For the MCU, physical time is kept by the count of the 16-bit free-running counter, which can not be interrupted. Output compare functions are used to set an action to happen at a specific time. The output compare register is compared to the free-running counter at every bus-cycle. When the current count of the free-running counter matches the value held in the output compare register, an output is generated automatically.

In the program initialization, jump to interrupt service routine is realized by the instructions

```
ldaa #$7E
staa PIOC5
ldx #INT5
```

```
stx PVOC5+1
```

OL5 bit is set in timer control register 1 TCTL1 register to make OC5 pin toggle on successful compare. OC5F bit in timer interrupt flag register 1 (TFLG1) is cleared to make sure interrupt hasn't happened yet; OC5I bit in timer interrupt mask register 1 (Tmsk1) is set to 1 so that the OC5 interrupt is enabled. "cli" enables all interrupts by clearing interrupt mask bit in the register CCR. Output compare register 5 (TOC5) value is incremented by \$0100 = 256 cycles each time interrupt service routine INT5 is called. When TOC5 value matches the current count of the free-running counter (TCNT), the OC5F bit in timer interrupt flag 1 (TFLG1) is automatically set to 1, and an output of 1 is generated at PA3/OC5 pin. Before leaving the interrupt service routine OC5F bit must be cleared in the TFLG1 in case there are other interrupt requests, otherwise it will result in a system lockup where the service routine is executed continuously to the exclusion of all other. There are two common ways to clear a flag in TFLG1. In this case:

1. ldaa #\$08  
   staa TFLG1
2. bclr TFLG1 \$7F

Since the free-running counter is not interrupted when instructions are executed in the BUFFALO monitor; and since the control flowing from the time stamp of "ldd TCNT,X" in the current service routine to the one in the next service routine call (65 cycles total including the instruction "bra \*") takes fewer than \$0100 = 256 cycles, the difference between the two time stamps should be always \$0100 = 256 cycles. Checking the data offloaded from the \$D000-\$DFFF, it is found to be true.

## Lab 12.0 Polling with OC2

**PROBLEM:** modulo-9 counter with output compare function and polling technique

**TOPIC:** output compare register 2 (OC2) and polling technique

**PURPOSE:** To introduce polling technique using output compare register 2(OC2) and free-running counter.

**ASSIGNMENT:** see Lab 12.0

### BACKGROUND REQUIRED:

1. Timer interrupt flag 1 (TFLG1),
2. Timer output compare register 2 (TOC2), and
3. Free-running counter (TCNT)
4. Polling technique

**HARDWARE DIAGRAM:** see Diagram 12.0

**NOTE:** Bit 4 of Port-C is connected to a piezo buzzer, and is used to generate a tic sound.

**SOLUTION:** The number of iterations for T\_LOOP is 497 in decimal number. Here is how it is obtained. In the T\_LOOP below

```
DIG_LOOP
[6] jsr CLEAR
[4] ldab CUR_DIGIT
[6] jsr OUT_DIGIT
[2] incb
[2] cmpb #10
[3] bne SKIP
[2] ldab #0
[4] SKIP stab CUR_DIGIT
```

```
*****
```

```
* POLLING the free-running counter
```

```
*****
```

```
[4] ldy #497
T_LOOP
[5] ldd TCNT,X
[4] addd #4000
[5] std TOC2,X
[7] brclr TFLG1,X $40 *
[7] bclr TFLG1,X $BF
[4] dey
[3] bne T_LOOP
```



## [3] bra DIG\_LOOP

“ldd TCNT,X” takes 5 cycles. Since 4000 cycles are added to TOC2, “brclr TFLG1,X \$40 \*” finds OC2F bit set about 4000 cycles later. “brclr” instruction takes 7 cycles. The instruction has been executed many times before OC2F is set, but it took fewer than 4000 cycles. Since the larger number of cycles, in this case 4000, should be calculated, the cycles for that instruction are ignored. Note the delay cycles for each T\_LOOP should not be constant, because of the uncertainty of the polling nature, but the difference is only a few cycles, and can be ignored here. The rest of T\_LOOP take  $7+4+3 = 14$  cycles. Let the total cycles for the T\_LOOP be A,  $A = 5+4000+7+14 = 4026$ . For T\_LOOP to iterate N times, let the total number of cycles for N iterations of T\_LOOP be B,  $B = N * A = N * 4026$ .

The delay, between the time one digit is turned on and the time the next digit is turned on, includes instruction cycles in the rest of the DIG\_LOOP. Let them be C,  $C = 6+4+6+2+2+3+4+4+3 = 34$ . 2 cycles for “ldab #0” is skipped because in most of iterations it is not executed. Next expand subprogram CLEAR and OUT\_DIGIT, since they are called in DIG\_LOOP. CLEAR takes  $D = 2+4+5 = 11$  cycles, and OUT\_DIGIT takes  $E = 4+3+3+4+5+4+5 = 28$  cycles. In order for each digit stay on for 1 second, the following equation must hold.

$$2,000,000 = B + C + D + E = N * 4026 + 34 + 11 + 28 = N * 4026 + 73$$

N is computed to be 497, rounding from 496.75

In the program initialization, OC2F bit in timer interrupt flag register 1 (TFLG1) is cleared to make sure output compare hasn't happened yet; In the T\_LOOP, 4000 cycles are added to the current count of the free-running counter (TCNT), and then stored to TOC2. The program then keeps polling to check if the output compare comes by executing the instruction “brclr TFLG1,X \$40 \*”, which means branching to itself if OC2F bit is not set in TFLG1. When the value held in TOC2 matches the current count of TCNT finally, the control exits the branching, clears the OC2F bit, and goes on. Since it is a 16-bit TCNT, and the maximal cycles are 65535, the output compare has to be repeated a number of times for a one-second (2000000 cycles for 2MHz MCU) delay.

## Lab 13.0 OC5 Interrupt

**PROBLEM:** modulo-9 counter with output compare function and interrupt handling

**TOPIC:** output compare register 5 (OC5) interrupt technique

**PURPOSE:** To introduce interrupt technique using output compare register 5 (OC5) and free-running counter.

**ASSIGNMENT:** see Lab 13.0

### BACKGROUND REQUIRED:

1. Timer interrupt mask (TMSK1),
2. Timer interrupt flag 1 (TFLG1),
3. Timer output compare register 5 (TOC5), and
4. Free-running counter (TCNT)
5. Interrupt vector jump table
6. Interrupt service routine

**HARDWARE DIAGRAM:** see Diagram 13.0

**NOTE:** Bit 4 of Port-C is connected to a piezo buzzer, and is used to generate a tic sound.

**SOLUTION:** For the MCU, physical time is kept by the count of the 16-bit free-running counter, which can not be interrupted. Output compare functions are used to set an action to happen at a specific time. The output compare register is compared to the free-running counter at every bus-cycle. When the current count of the free-running counter matches the value held in the output compare register, an interrupt occurs.

In the program initialization, jump to interrupt service routine is realized by the following instructions.

```
ldaa #$7E
staa PVOC5
ldx #INTERRUPT
stx PVOC5+1
```

OC5F bit in timer interrupt flag register 1 (TFLG1) is cleared to make sure interrupt hasn't happened yet; OC5I bit in timer interrupt mask register 1 (Tmsk1) is set to 1 so that the OC5 interrupt is enabled. "cli" clears the interrupt mask bit in CCR, and enables all interrupts.

Each time the interrupt service routine INTERRUPT is called the value \$A000 is added to the current count of free-running counter (TCNT), and the result is stored to the output compare register 5 (TOC5). When TOC5 value matches the current count of the TCNT, the OC5F bit in

timer interrupt flag 1 (TFLG1) is automatically set to 1, and the interrupt occurs. Before leaving the interrupt service routine OC5F bit must be cleared in the TFLG1, in case there are other interrupt requests, otherwise it will result in a system lockup where the service routine is executed continuously to the exclusion of all other. There are two common ways to clear a flag in TFLG1. In this case:

1. `ldaa #$08`  
`staa TFLG1`
2. `bclr TFLG1 $7F`

Since it is a 16-bit TCNT, and the maximal cycles are 65535, the output compare has to be repeated a number of times for a one-second (2000000 cycles for 2MHz MCU) delay.

The number of iterations TLP should loop is 49. Here is how it is obtained. In the interrupt service routine, the current count of the TCNT is added to the value \$A000, and the result is stored to TOC5. There should be an interrupt after \$A000 = 40960 cycles from the point the current time stamp is captured. After the interrupt, the number of iterations decrement by 1, and the interrupt service routine is called again. After N iterations or N\*40960 cycles, the number of iterations decrement to 0. That is the major part of the delay for a digit staying on for one second. The minor part of the delay consists of the cycles for instructions in the DIG\_LOOP, which are  $4+7+4+3+6+4+6+2+2+3+4+3 = 48$  cycles. 2 cycles for "ldab #0" are omitted, since most of time the instruction is not executed. The subprograms CLEAR and OUT\_DIGIT also need to be expanded, since they are called in DIG\_LOOP, and they take  $2+4+5 = 11$  and  $4+3+3+4+5+4+5 = 28$  cycles, respectively, so the equation for one-second delay is

$$2000000 = N * 40960 + 48 + 11 + 28 = N * 40960 + 87$$

N is computed to be 49, rounding from 48.8

## Lab 14.0 Timer Using Polling

**PROBLEM:** alarm system with output compare function and polling technique

**TOPIC:** output compare register 2 (OC2) and polling technique

**PURPOSE:** To introduce polling technique using output compare register 2(OC2) and free-running counter.

**ASSIGNMENT:** see Lab 14.0

### BACKGROUND REQUIRED:

1. Timer interrupt flag 1 (TFLG1),
2. Timer output compare register 2 (TOC2), and
3. Free-running counter (TCNT)
4. Polling technique

**HARDWARE DIAGRAM:** see Diagram 14.0

**NOTE:** Bit 4 of Port-C and that Port-B are connected to a piezo buzzer, and are used to generate a tic sound.

**SOLUTION:** The number of iterations for T\_LOOP is 124 in decimal number. Here is how it is obtained. In the T\_LOOP below

```

DIG_LOOP
[ 6 ] jsr CLEAR
[ 4 ] ldab B_DIGIT
[ 6 ] jsr GET_DIGIT
[ 4 ] staa PORTB,X
[ 2 ] decb
[ 2 ] cmpb #-1
[ 3 ] bne SKIP1
[ 2 ] ldab #9
[ 4 ] SKIP1 stab B_DIGIT
[ 4 ] ldab C_DIGIT
[ 6 ] jsr GET_DIGIT
[ 4 ] staa PORTC,X
[ 4 ] ldaa B_DIGIT
[ 2 ] cmpa #9
[ 3 ] bne SKIP2
[ 2 ] decb
[ 2 ] cmpb #-1
[ 3 ] bne SKIP2
[ 6 ] jsr ALARM
[ 4 ] SKIP2 stab C_DIGIT

```

```

[ 4 ] ldy #124
[ 5 ] T_LOOP ldd TCNT,X
[ 4 ] addd #4000
[ 5 ] std TOC2,X
[ 7 ] brclr TFLG1,X $40 *
[ 7 ] bclr TFLG1,X $BF
[ 4 ] dey
[ 3 ] bne T_LOOP
[ 3 ] bra DIG_LOOP

```

“ldd TCNT,X” takes 5 cycles. Since 4000 cycles are added to TOC2, “brclr TFLG1,X \$40 \*” finds OC2F bit set about 4000 cycles later. “brclr” instruction takes 7 cycles. The instruction has been executed many times before OC2F is set, but it took fewer than 4000 cycles. Since the larger number of cycles, in this case 4000, should be calculated, the cycles for that instruction are ignored. Note the delay cycles for each T\_LOOP should not be constant, because of the uncertainty of the polling nature, but the difference is only a few cycles, and can be ignored here. The rest of T\_LOOP take  $7+4+3 = 14$  cycles. Let the total cycles for the T\_LOOP be A,  $A = 5+4000+7+14 = 4026$ . For T\_LOOP to iterate N times, let the total number of cycles for N iterations of T\_LOOP be B,  $B = N * A = N * 4026$ .

The delay, between the time the current value is turned on for display and the time next value is turned on, includes instruction cycles in the rest of the DIG\_LOOP. Let them be C,  $C = 6+4+6+4+2+2+3+4+4+6+4+4+2+3+4 = 58$ . The cycles of

```

ldab #0
...
decb
cmpb #-1
bne SKIP2
jsr ALARM

```

are not calculated, since in most of iterations they are not executed. Next expand subprogram CLEAR and GET\_DIGIT, since they are called in the DIG\_LOOP. CLEAR takes  $D = 2+4+4+5 = 15$  cycles, and GET\_DIGIT takes  $E = 4+4+5+5 = 18$  cycles. In order for each number to be displayed on the MAN6610 LED for 1/4 second, the following equation must be true.

$$500,000 = B + C + D + 2E = N * 4026 + 58 + 15 + 2*18 = N * 4026 + 109$$

In the equation E is calculated twice because GET\_DIGIT is called twice in DIG\_LOOP. N is computed to be 124, rounding from 124.16

In the program initialization, OC2F bit in timer interrupt flag register 1 (TFLG1) is cleared to make sure output compare hasn't happened yet; In the T\_LOOP, the current count of the free-running counter (TCNT) is added to 4000 cycles, and the result stored to TOC2. The program then keeps polling to check if the output compare comes by executing the instruction “brclr TFLG1,X \$40 \*”, which means branching to itself if OC2F bit is not set in TFLG1. When the value held in TOC2 matches the current count of TCNT finally, the control exits the branching, clears the OC2F bit, and goes on. Since it is a 16-bit TCNT, and the maximal cycles are 65535, the output compare has to be repeated a number of times for a 1/4 second (500,000 cycles for 2MHz MCU) delay.

## Lab 15.0 Timer Using Interrupt Handling

**PROBLEM:** alarm system with output compare function and interrupt technique

**TOPIC:** output compare register 5 (OC5) interrupt technique

**PURPOSE:** To introduce interrupt handling technique using output compare register 5(OC5) and free-running counter.

**ASSIGNMENT:** see Lab 15.0

### BACKGROUND REQUIRED:

1. Timer interrupt mask (TMSK1),
2. Timer interrupt flag 1 (TFLG1),
3. Timer output compare register 5 (TOC5), and
4. Free-running counter (TCNT)
5. Interrupt vector jump table
6. Interrupt service routine

**HARDWARE DIAGRAM:** see Diagram 15.0

**NOTE:** Bit 4 of Port-C and that Port-B are connected to a piezo buzzer, and are used to generate a tic sound.

**SOLUTION:** For the MCU, physical time is kept by the count of the 16-bit free-running counter, which can not be interrupted. Output compare functions are used to set an action to happen at a specific time. The output compare register is compared to the free-running counter at every bus-cycle. When the current count of the free-running counter matches the value held in the output compare register, an interrupt comes.

In the program initialization, jump to interrupt service routine is realized by the following instructions.

```
ldaa #$7E
staa PVOC5
ldx #INTERRUPT
stx PVOC5+1
```

OC5F bit in timer interrupt flag register 1 (TFLG1) is cleared to make sure interrupt hasn't happened yet; OC5I bit in timer interrupt mask register 1 (TMSK1) is set to 1 so that the OC5 interrupt is enabled. "cli" clears the interrupt mask bit in CCR, and enables all interrupts. Each time the interrupt service routine INTERRUPT is called the value \$A000 is added to the current count of free-running counter (TCNT), and the result is stored to the output compare register 5 (TOC5). When TOC5 value matches the current count of the TCNT, the OC5F bit in timer interrupt flag

1 (TFLG1) is automatically set to 1, and an interrupt occurs. Before leaving the interrupt service routine OC5F bit must be cleared in the TFLG1, in case there are other interrupt requests, otherwise it will result in a system lockup where the service routine is executed continuously to the exclusion of all other. There are two common ways to clear a flag in TFLG1. In this case:

1. `ldaa #$08 staa TFLG1`
2. `bclr TFLG1 $7F`

Since it is a 16-bit TCNT, and the maximal cycles are 65535, the output compare has to be repeated a number of times for a 1/4-second (500000 cycles for 2MHz MCU) delay.

The number of iterations TLP should loop is 12. Here is how it is obtained. In the interrupt service routine, the current count of the TCNT is added to the value \$A000, and the result is stored to TOC5. There should be an interrupt after \$A000 = 40960 cycles from the point the current time stamp is captured. After the interrupt, the number of iterations decrement by 1, and the interrupt service routine is called again. After N iterations or N\*40960 cycles, the number of iterations decrement to 0. That is the major part of the delay for a number staying on for 1/4 second. Let it be A,  $A = N * 40960$ . The minor part of the delay consists of the cycles for instructions in the DIG\_LOOP, which are assigned to B,  $B = 4+7+4+3+6+4+6+4+2+2+3+4+4+6+4+4+2+3+4+3 = 79$  cycles. The cycles for

```
ldab #0
...
decb
cmpb #-1
bne SKIP2
jsr ALARM
```

are not calculated, since in most of iterations they are not executed. Next expand subroutines CLEAR and GET\_DIGIT, since they are called in the DIG\_LOOP. CLEAR takes  $C = 2+4+4+5 = 15$  cycles, and GET\_DIGIT takes  $D = 4+4+5+5 = 18$  cycles. In order for each number to be displayed on the MAN6610 LED for 1/4 second, the following equation must be true.

$$500,000 = A + B + C + 2D = N * 40960 + 79 + 15 + 2*18 = N * 4026 + 130$$

In the equation D is calculated twice because GET\_DIGIT is called twice in DIG\_LOOP. N is computed to be 12, rounding from 12.2.

## Lab 16.0 Inter-process Communication

**PROBLEM:** inter-process communication at Port-B and Port-C

**TOPIC:** distributed programming; polling mechanism

**PURPOSE:** To introduce REAL low-level distributed programming mechanisms for inter-processor SEND and RECEIVER.

**ASSIGNMENT:** see Lab 16.0

### BACKGROUND REQUIRED:

1. Parallel I/O control register (PIOC)
2. Free-running counter (TCNT)
3. Port-C Latched Data register (Port-CL)
4. Difference of RAM and ROM
5. EVB memory map diagram

**HARDWARE DIAGRAM:** see Diagram 16.0

### SOLUTION:

1. The MAXIMAL number of instruction cycles in the receiver program that may lapse between the time the receiver recognizes an arbitrary “send(n)” and the time it is ready to recognize the next “send(n+1)” is 43. This number is calculated after the time the GET\_INPT subroutine finds the data arrival (beq GET\_INPT) to the time the software is ready for new data arrival. The cycles spent in GET\_INPT include those for the following instructions.

```
[4] ldaa PORTCL
[5] ldx TCNT
[5] rts
```

which amount to 14 cycles.

The worst case for the receiver to get ready to recognize new data is after the 4th “jsr GET\_INPT”. In order to get to the next “jsr GET\_INPT”, it has to go thru

```
[5] staa 3,Y
[6] stx 6,Y
[4] aby
[5] cpy #MEM_END
[3] blo MORE_INPT
[6] jsr MORE_INPT
```



which amount to 29 cycles. Therefore the MAXIMAL number of instructions cycles that may lapse is  $14+29 = 43$ .

2. The delay between the first and second send is 34, because the subroutine IPC\_PAUSE contains 24 cycles, plus  $4+6 = 10$  cycles for the instructions

```
[4] staa PORTB
[6] jsr IPC_PAUSE
```

between the 1st and 2nd send. So the total is  $24+10 = 34$  cycles. Likewise, the delay between the 2nd and 3rd send is  $24+6+3+4 = 37$  cycles, and that between the 3rd and 4th is  $24+6+4 = 34$  cycles. From the 4th send back to the 1st one it takes  $24+3+6+5+6+6+6+5+6+2 + 4+5+3+3+6+5+4 = 99$  cycles.

3. Theoretically, in order for the receiver to be ready to receive, D\_LOOP in IPC\_PAUSE of the sender program must iterate more than once. Since the MAXIMAL number of instruction cycles is 43 in (1), and the MINIMAL number of cycles in (2) is 34, we must add delay of  $43-34 = 9$  cycles or more between 2 sends in the sender program in order for receiver to get ready to receive. Since each loop of D\_LOOP takes 6 cycles, and one loop of it is already calculated in (2), we need to add 2 more loops ( $2*6 = 12 > 9$ ). Therefore the iterations of the D\_LOOP in IPC\_PAUSE is 3.

The experiments, however, show that as few as 1 D\_LOOP can be used without damaging any incoming data, as can be verified from the postprocessing output. DEL\_TS is 170 when 1 D\_LOOP is used, and is 194 when 2 D\_LOOP is used. The difference is 24 ( $194-170 = 24$ ) because each D\_LOOP takes 6 cycles, and the IPC\_PAUSE is called 4 times in EVENT, therefore DEL\_TS(EVENT completion time - EVENT initiation time) varies by 24 when the D\_LOOP varies by 1. Note the difference between sender and receiver times is not constant, since the polling loop is uncertain, but it is bounded. The standard range is 9.

## Lab 17.0 Multi-character Display

**PROBLEM:** LCD-II display at Port-B and Port-C

**TOPIC:** low-level device control

**PURPOSE:** To introduce low-level device control achieved thru the EVB Port-B and Port-C pins, and use of multi-character display hardware.

**ASSIGNMENT:** see Lab 17.0

### BACKGROUND REQUIRED:

Understand ASCII code

**HARDWARE DIAGRAM:** see Diagram 17.0

**NOTE:** Turn top part of the variable resister to get the best vision of the display. Once it is set, don't touch it, as it can be easily disturbed.

### SOLUTION:

```

*****
* Subpgm: H
* Desc: display character 'H'
*****
H
ldaa #$48
jsr OUTCHAR
rts
*****
* Subpgm: N
* Desc: display character 'N'
*****
N
ldaa #$4E
jsr OUTCHAR
rts
*****
* Subpgm: O
* Desc: display character 'O'
*****
O
ldaa #$4F
jsr OUTCHAR
rts
*****
* Subpgm: R
* Desc: display character 'R'

```

```

*****
R
ldaa #$52
jsr OUTCHAR
rts
*****
* Subpgm: W
* Desc: display character 'W'
*****
W
ldaa #$57
jsr OUTCHAR
rts
*****
* Subpgm: COMMA
* Desc: display character ','
*****
COMMA
ldaa #$2C
jsr OUTCHAR
rts
*****
* Subpgm: QUESTION_MARK
* Desc: display character '?'
*****
QUESTION_MARK
ldaa #$3F
jsr OUTCHAR
rts

```

# **Appendix E**

**Computer Architecture and Assembly Language**

## **Diagram Manual**

**ZUYI CHEN**

**July, 1992**

**Computer Science Department**

**University of Montana**

## TABLE OF CONTENTS

Table of Contents	i
Diagram 1.0: outports.asm device	1
Diagram 2.0: inpport.asm hardware	2
Diagram 3.0: trafficC.asm device	3
Diagram 4.0: trafficB.asm device	4
Diagram 5.0: music.asm device	5
Diagram 6.0: travelc.asm device	6
Diagram 7.0: travelb.asm device	7
Diagram 8.0: ModuloC.asm device	8
Diagram 9.0: ModuloB.asm device	9
Diagram 10.0: swi.asm device	10
Diagram 11.0: oc5int.asm hardware	11
Diagram 12.0: timepoll.asm device	12
Diagram 13.0: timeint.asm device	13
Diagram 14.0: alarmpol.asm device	14
Diagram 15.0: alarmint.asm device	15
Diagram 16.o: IPCsnd.asm/IPCrvc.asm hardware	16
Diagram 17.0: string.asm device	17

DIAGRAM MANUAL 1.0  
OUTPORTS.ASM SETUP

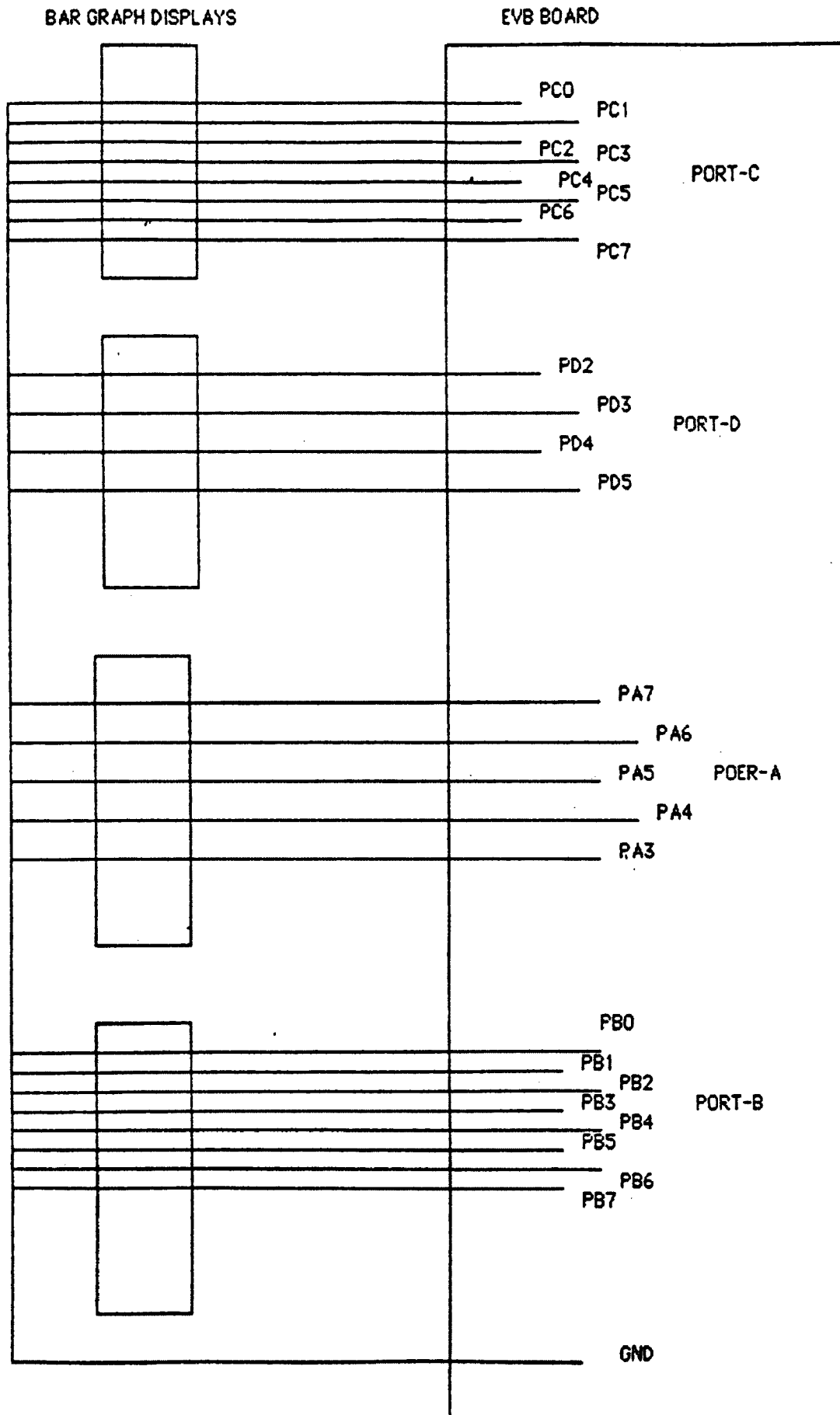
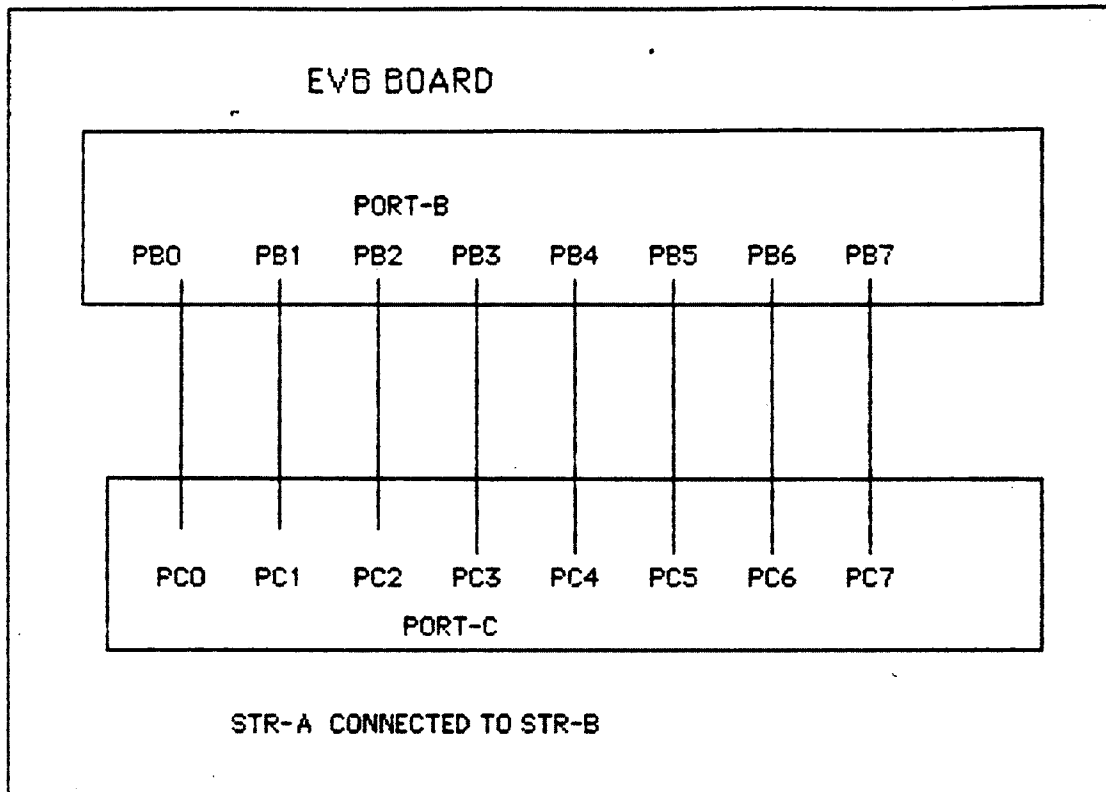


DIAGRAM MANUAL 2.0  
INPORT.ASM SETUP



# DIAGRAM MANUAL 3.0 TRAFFIC.ASM SETUP

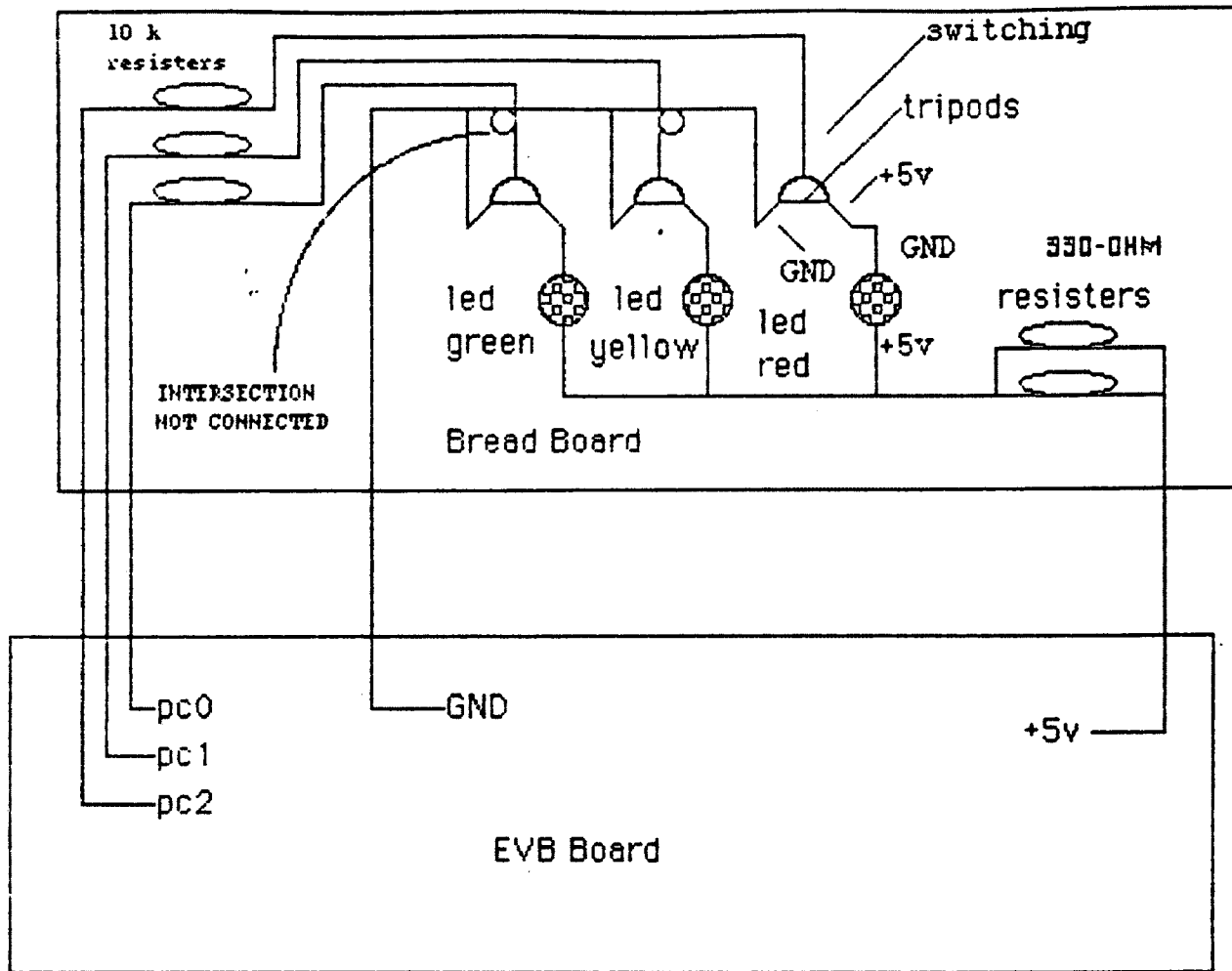




DIAGRAM MANUAL 4.0  
TRAFFICB.ASM SETUP

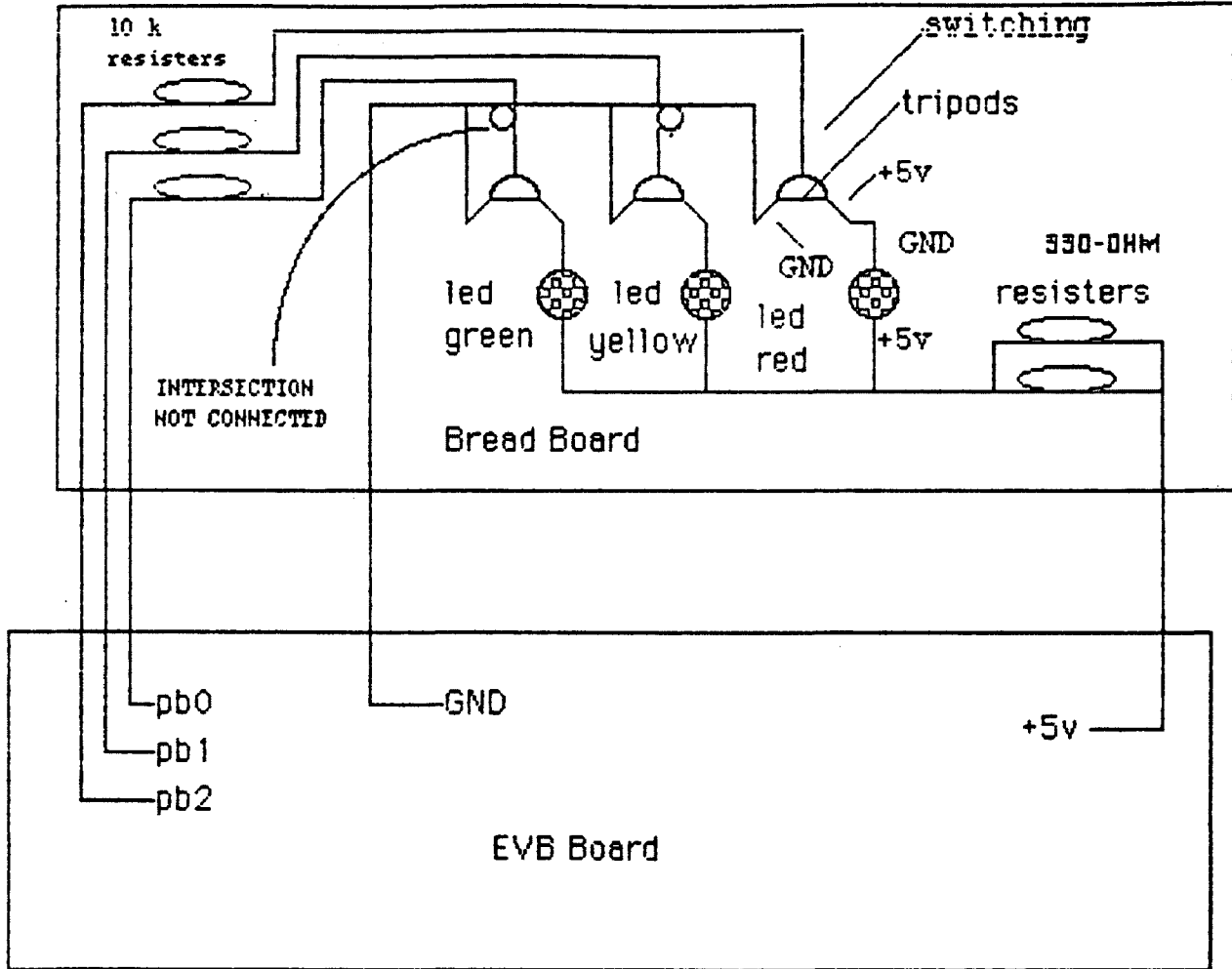
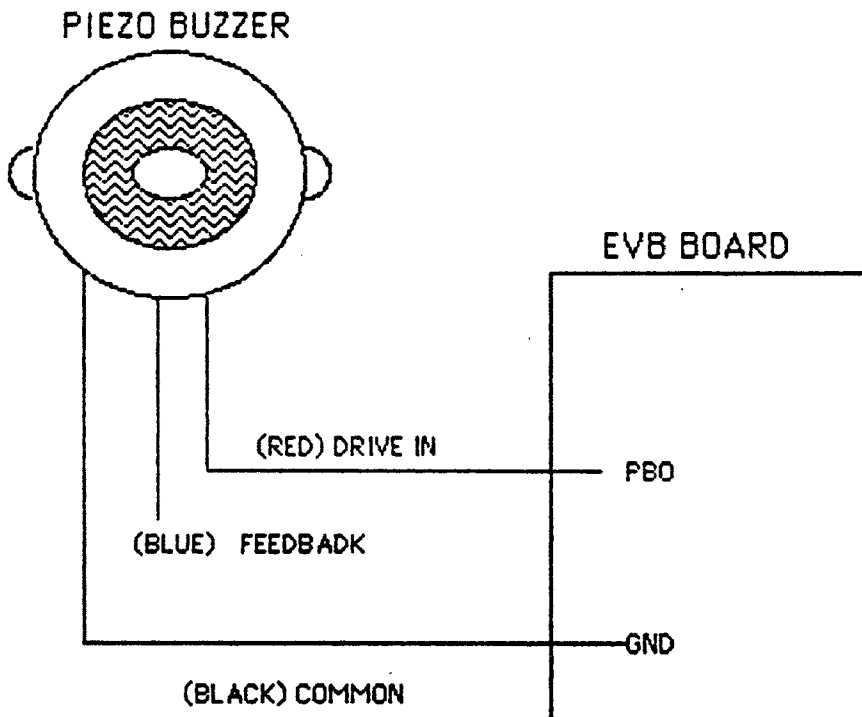


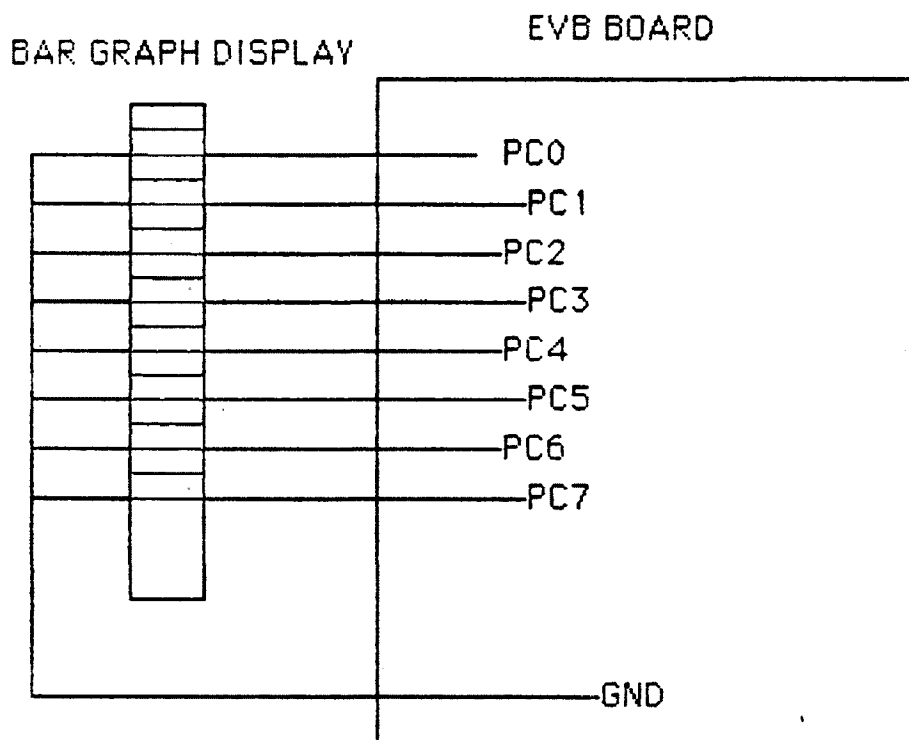
DIAGRAM MANUAL 5.0

HARDWARE SETUP FOR MUSIC.ASM



## DIAGRAM MANUAL 6.0

HARD SETUP FOR TRAVELC.ASM  
CONNECT PORT-C PINS AND A BAR GRAPH DISPLAY



## DIAGRAM MANUAL 7.0

HARD SETUP FOR TRAVELB.ASM  
CONNECT PORT-B PINS AND A BAR GRAPH DISPLAY

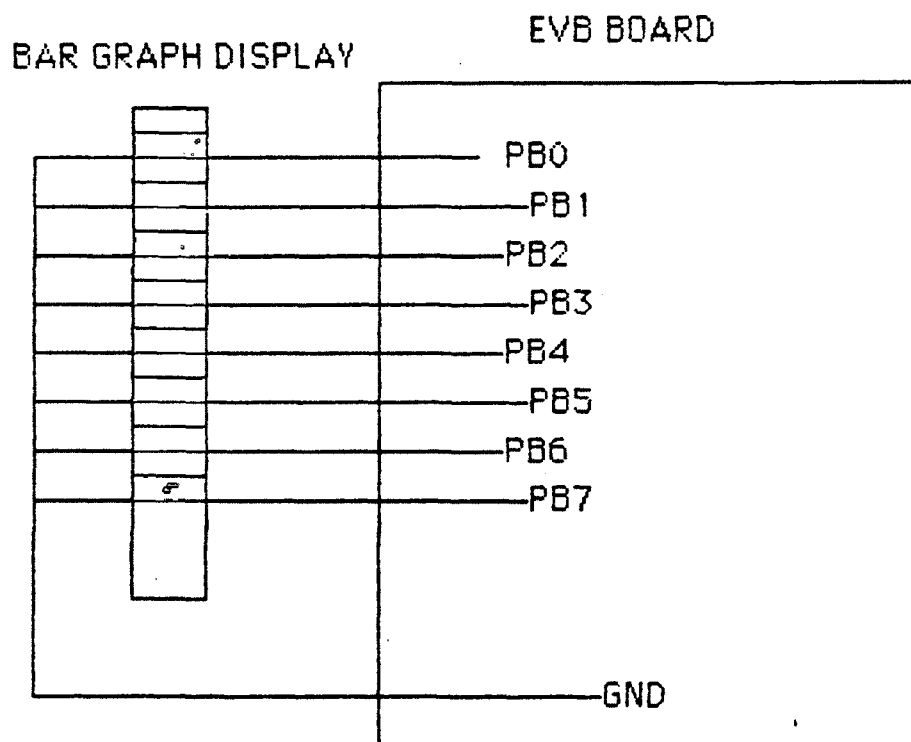


DIAGRAM MANUAL 8.0

MODULOC.ASM SETUP

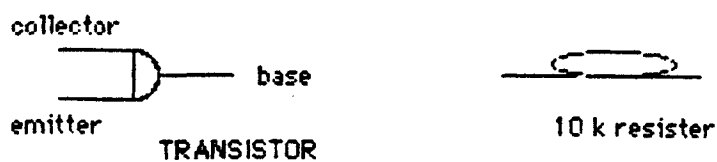
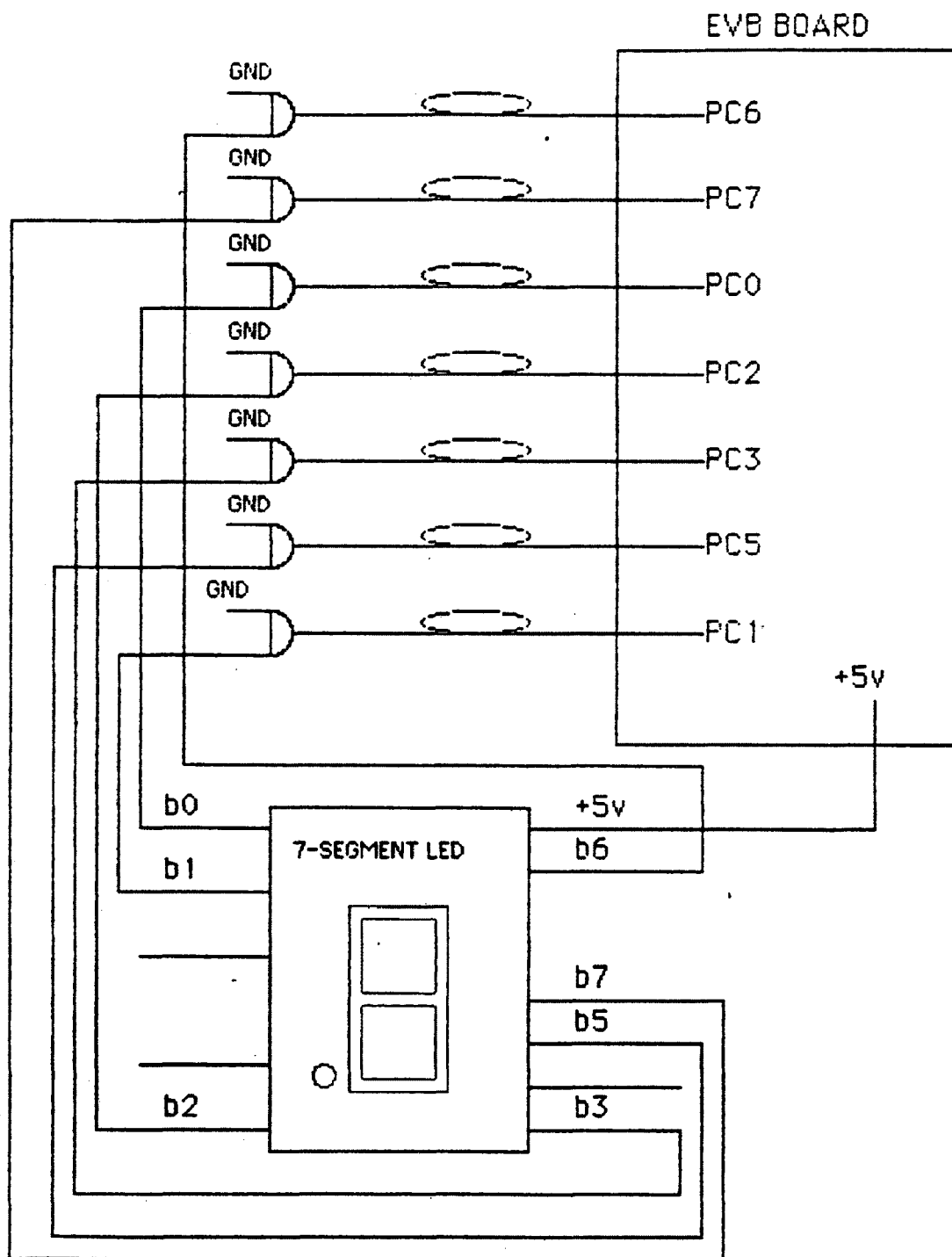


DIAGRAM MANUAL 9.0

MODULOB.ASM SETUP

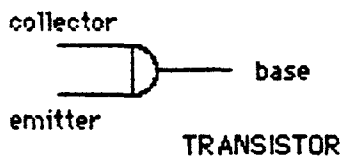
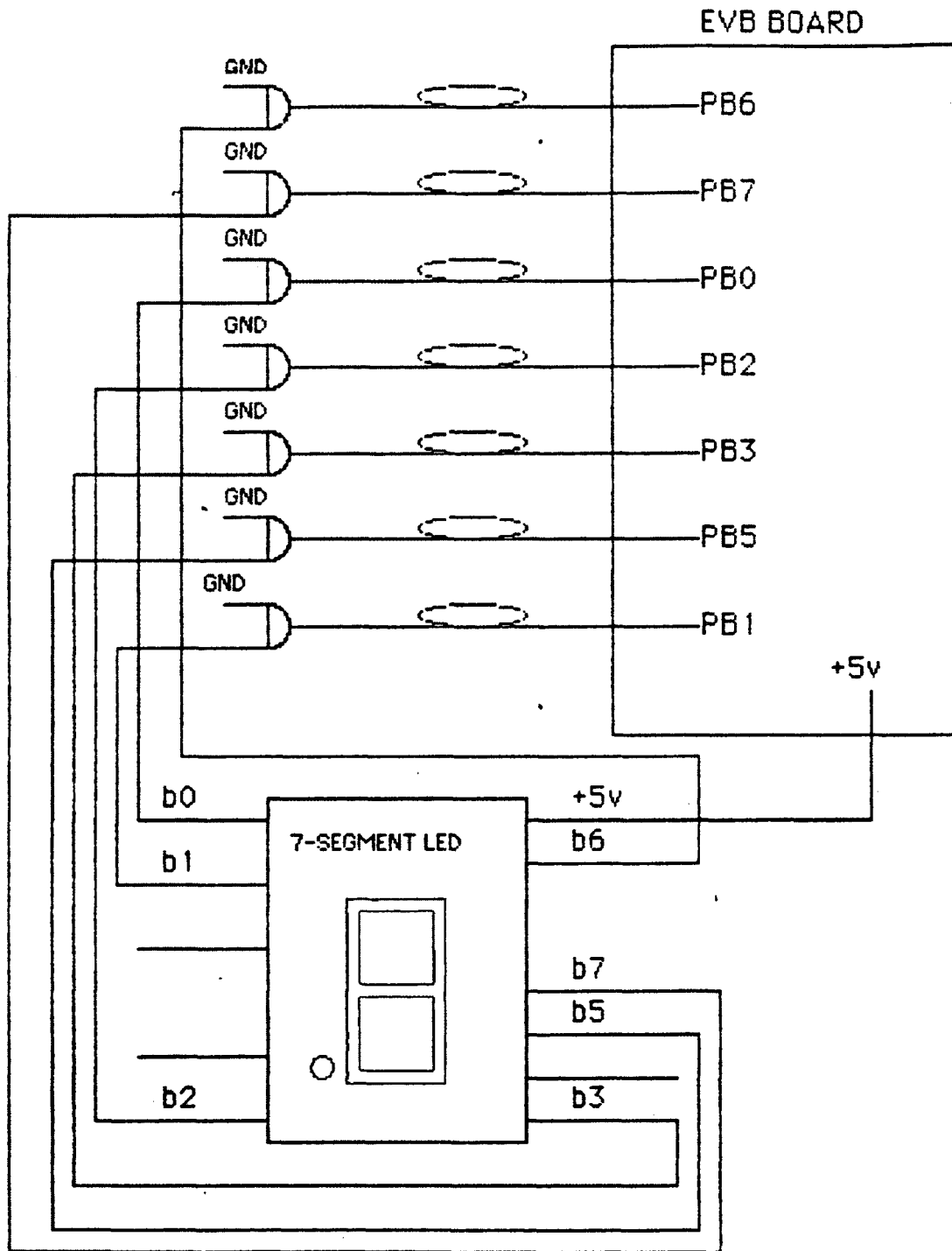
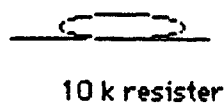
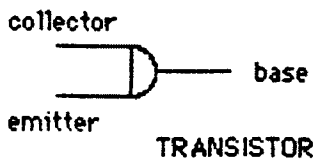
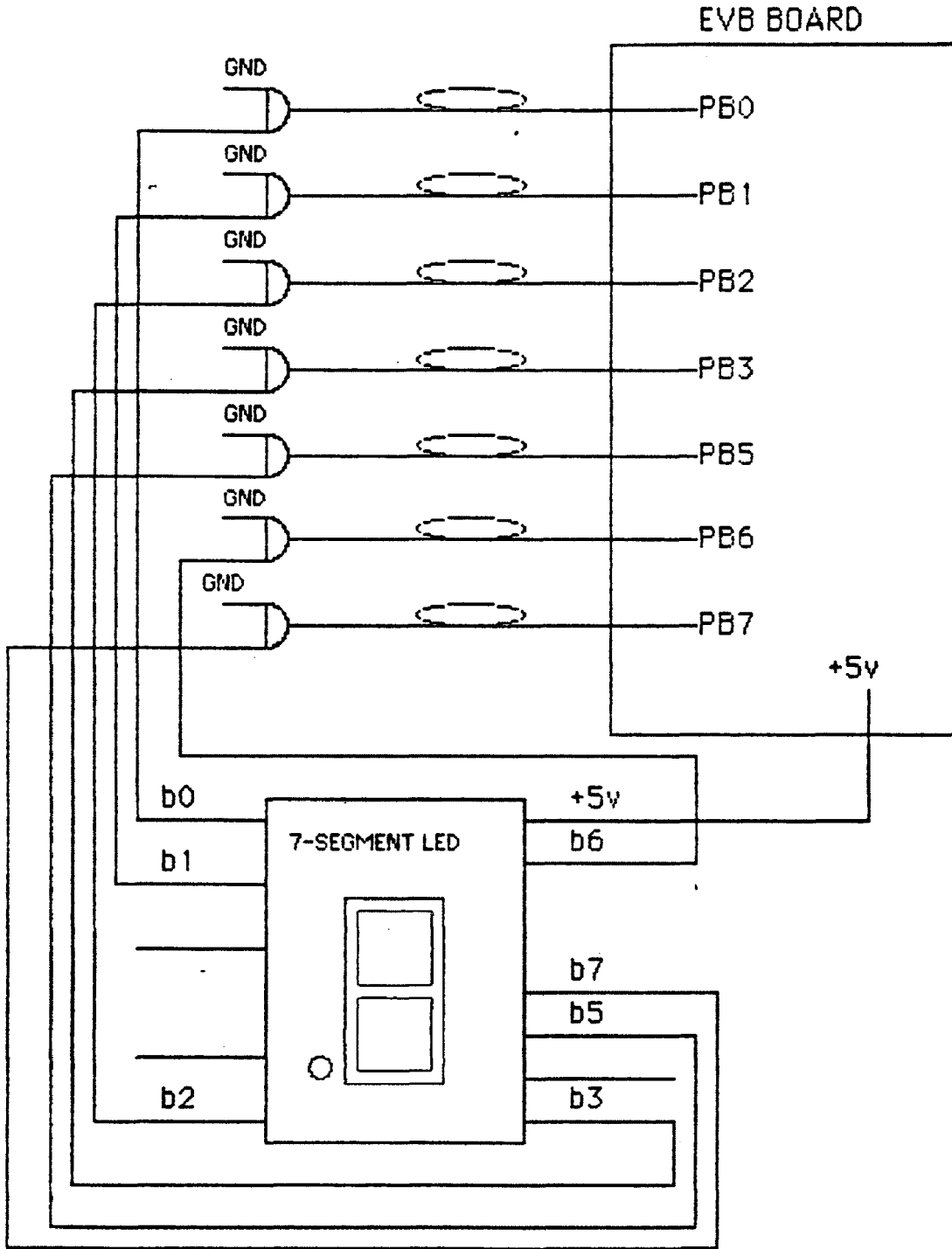


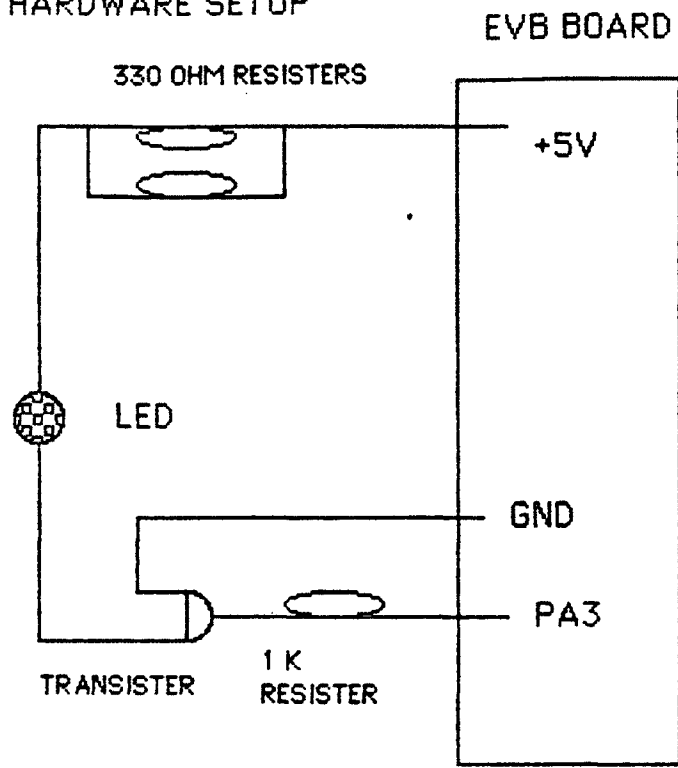
DIAGRAM MANUAL 10.0

SWI.ASM SETUP



## DIAGRAM MANUAL 11.0

## OCSINT.ASM HARDWARE SETUP





## DIAGRAM MANUAL 12.0

## TIMEPOLL.ASM SETUP

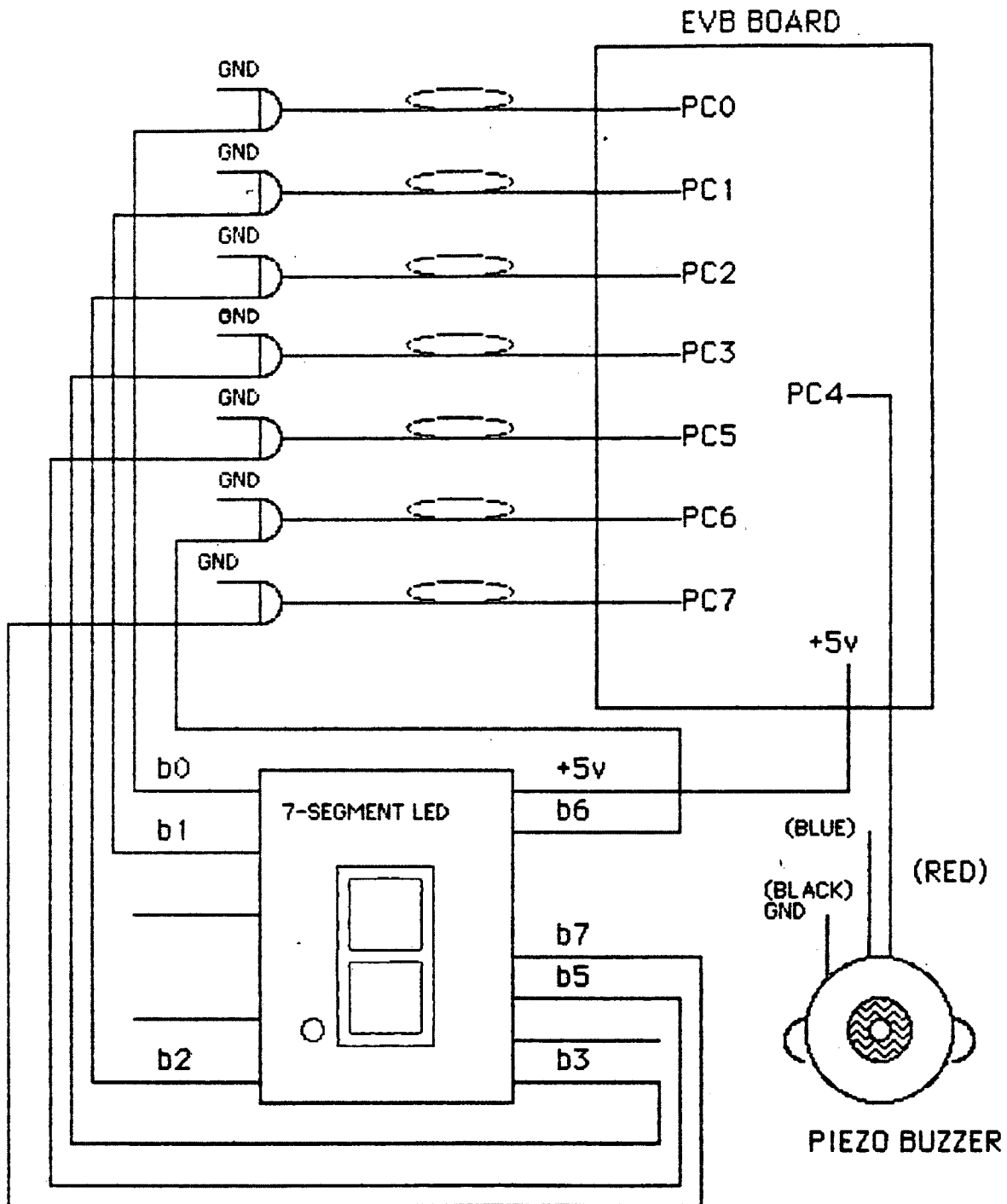


DIAGRAM MANUAL 13.0

TIMEINT.ASM SETUP

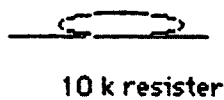
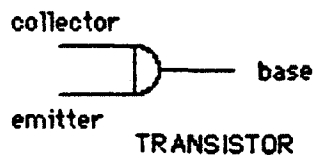
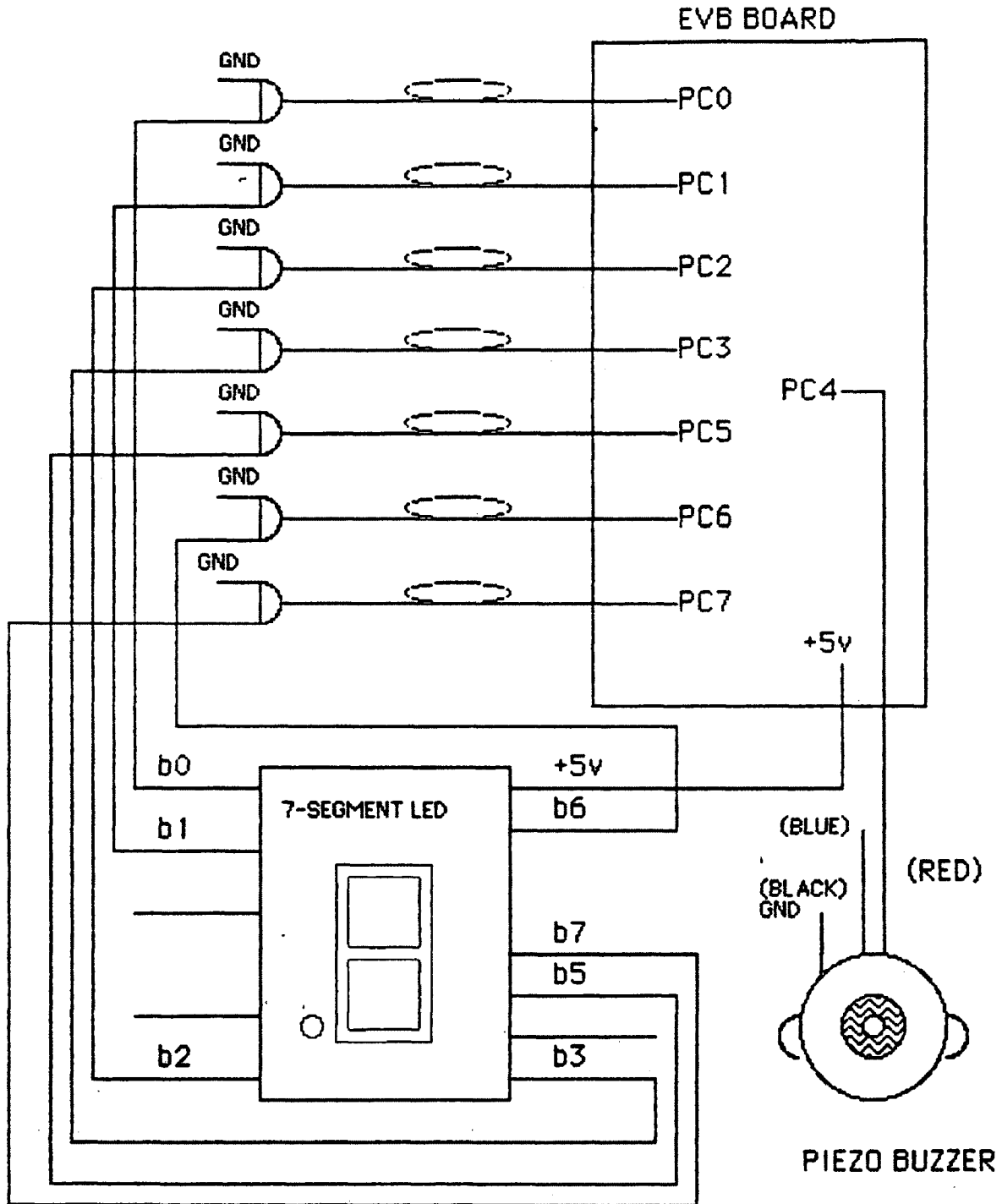


DIAGRAM MANUAL 14.0  
ALARMPOL.ASM SETUP

EVB BOARD

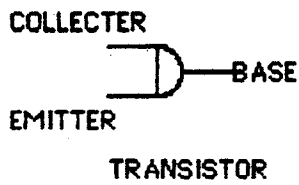
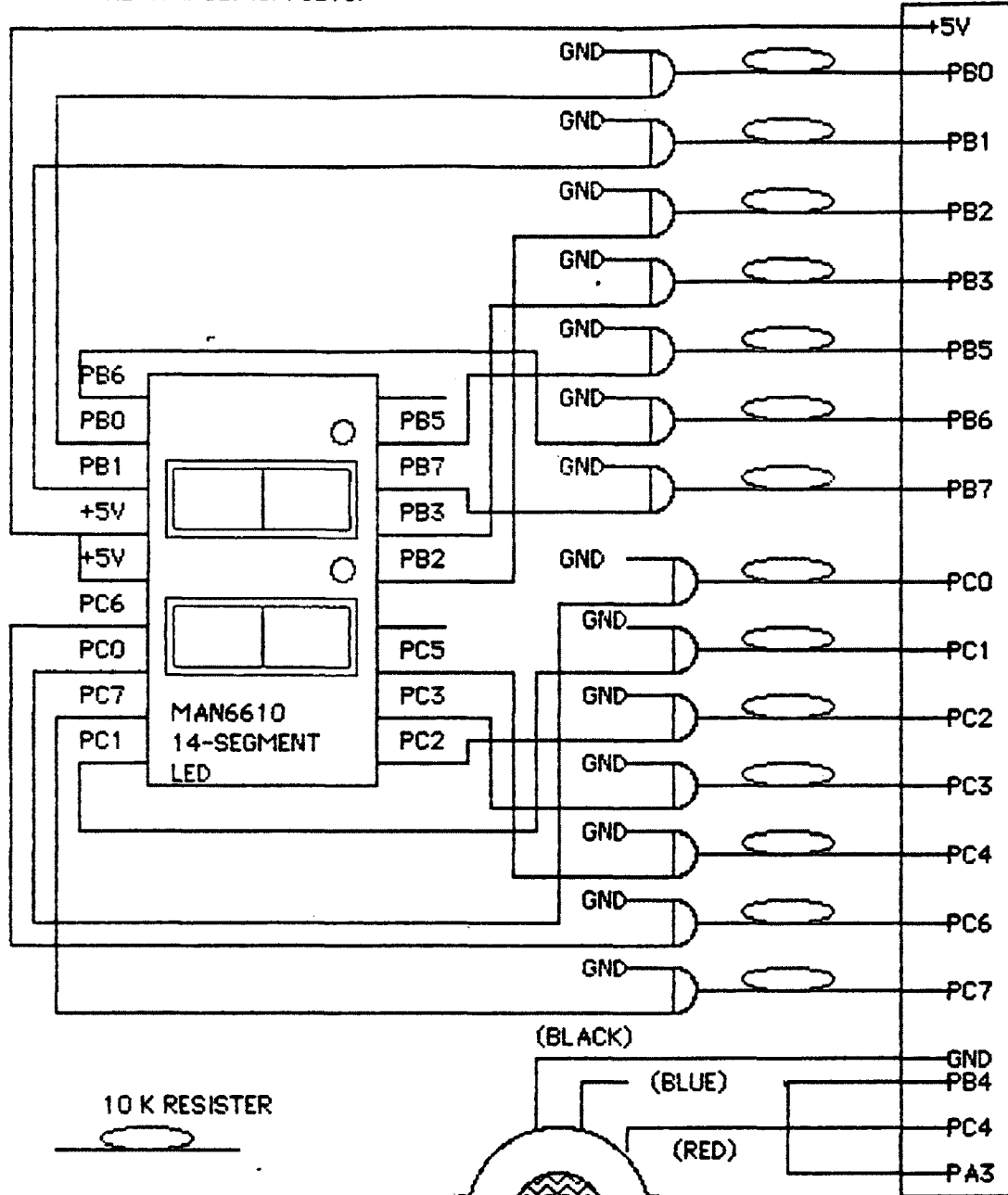
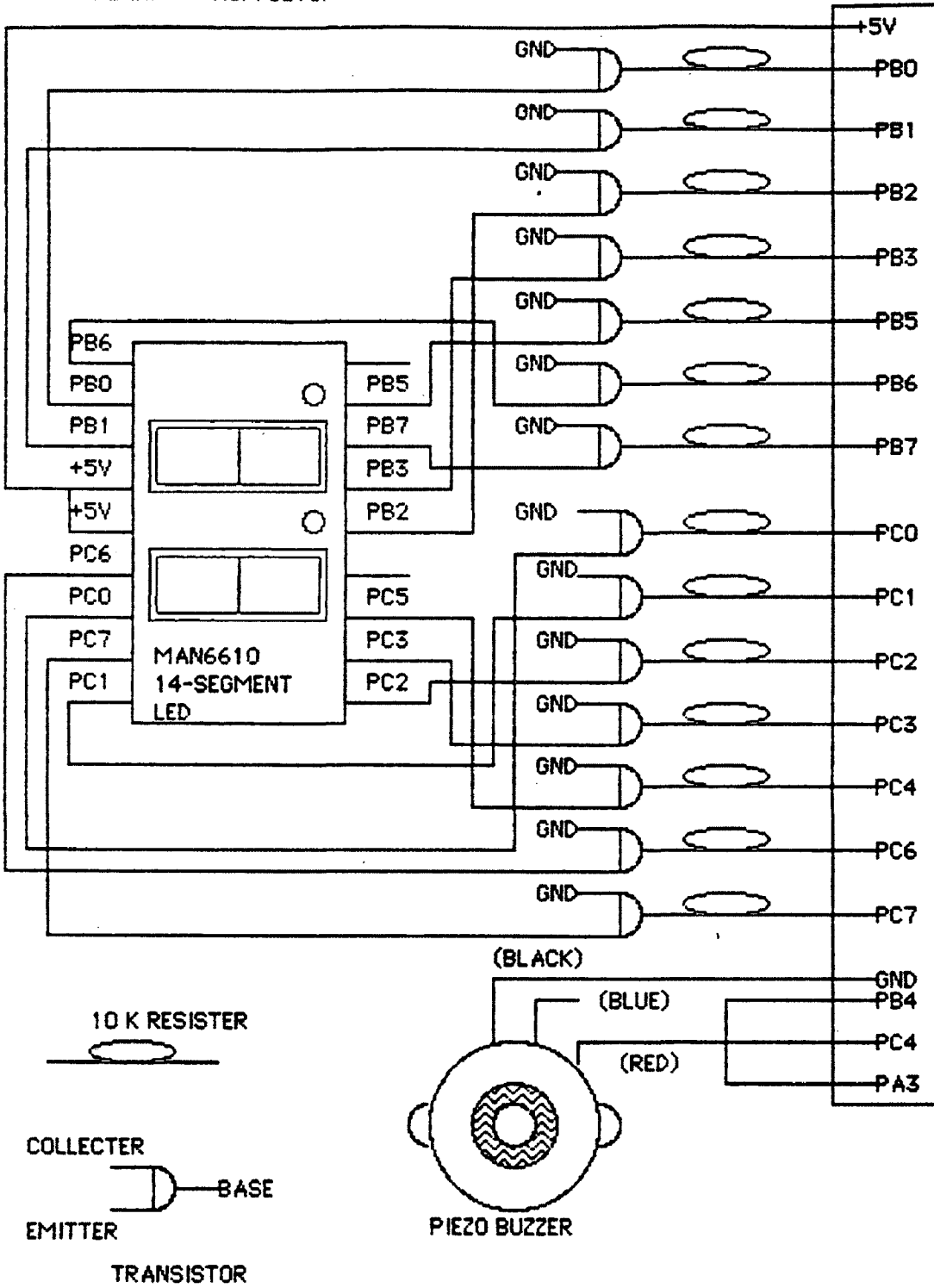


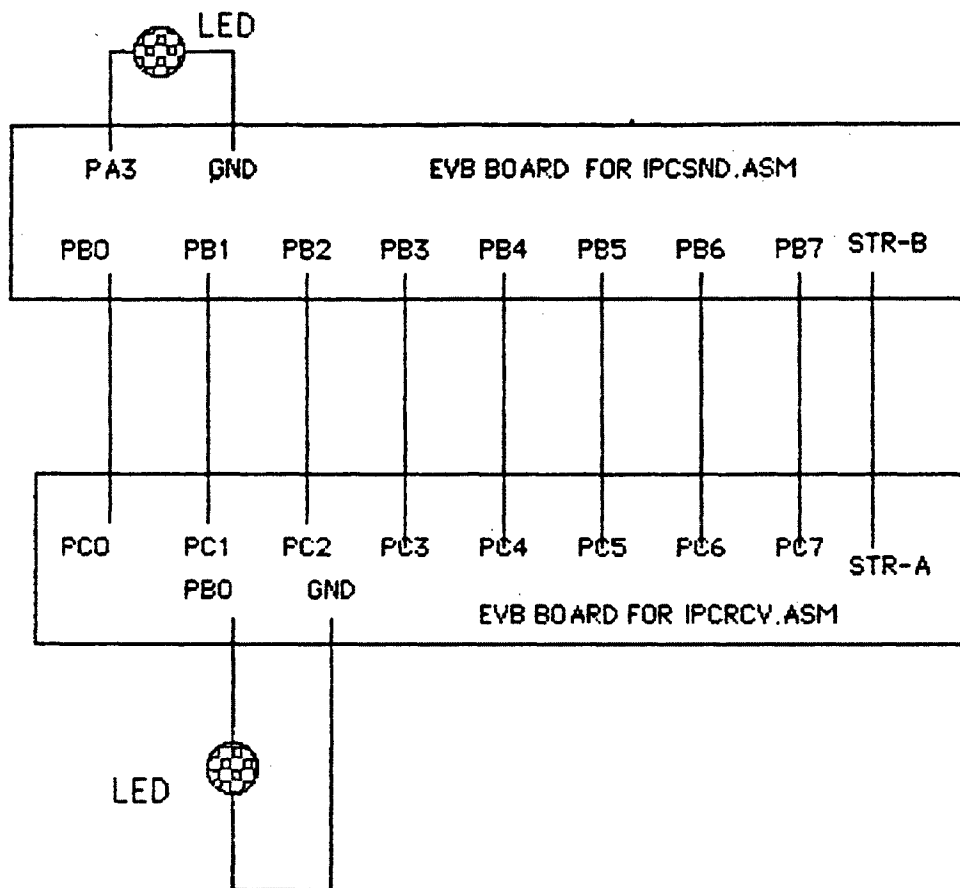
DIAGRAM MANUAL 15.0  
ALARMINT.ASM SETUP

EVB BOARD



## DIAGRAM MANUAL 16.0

## HARDWARE SETUP FOR IPCSND.ASM/IPCRCV.ASM

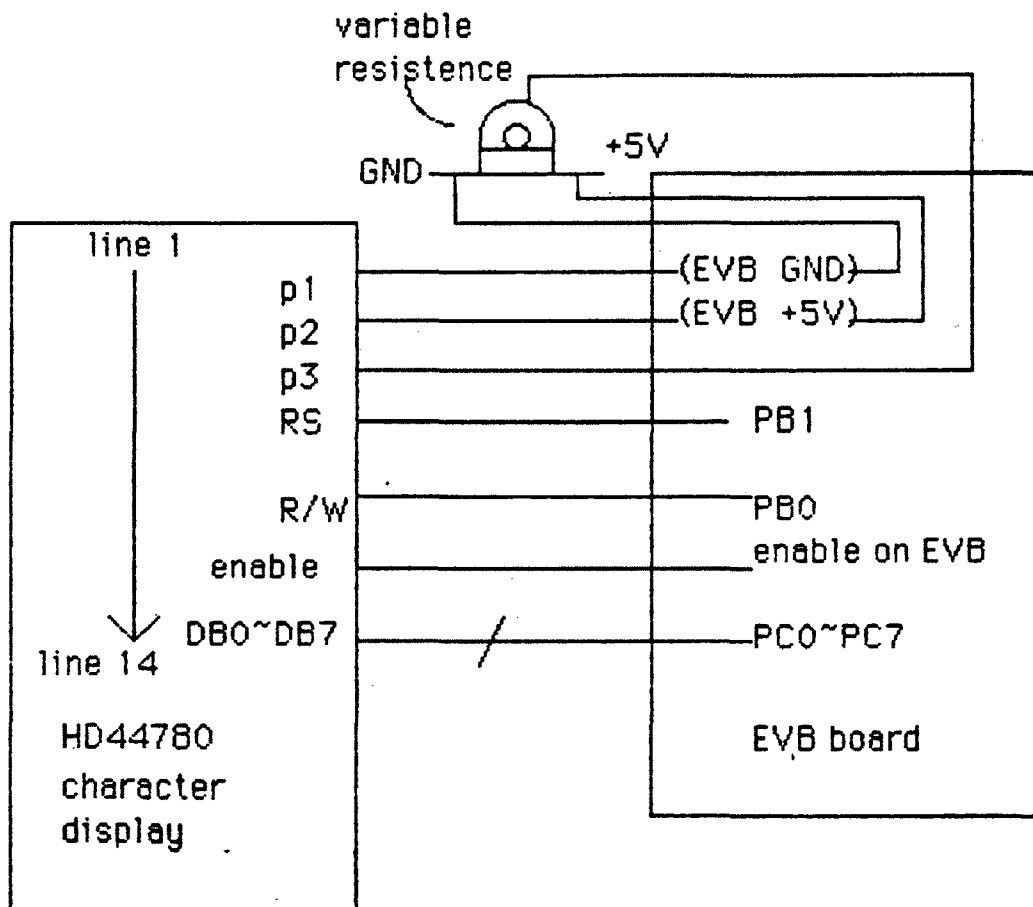


## DIAGRAM MANUAL 17.0

## HARDWARE SETUP FOR STRING.ASM

LCD-II display

- hardware:
1. MC68HC11EVB
  2. LCD-II display controller
  3. variable resistance (10)



# Appendix F

Computer Architecture and Assembly Language

Lab Program Shells

ZUYI CHEN

July, 1992

Computer Science Department

University of Montana

## TABLE OF CONTENTS

Table of Contents	i
Shell 1.0: alarmint.asm	1
Shell 2.0: alarmpol.asm	5
Shell 3.0: input.asm	9
Shell 4.0: IPCsnd.asm	11
Shell 5.0: IPCrcv.asm	14
Shell 6.0: moduloB.asm	17
Shell 7.0: moduloC.asm	19
Shell 8.0: music.asm	21
Shell 9.0: oc5int.asm	23
Shell 10.0: outports.asm	25
Shell 11.0: string.asm	29
Shell 12.0: swi.asm	35
Shell 13.0: timeint.asm	38
Shell 14.0: timepoll.asm	41
Shell 15.0: trafficB.asm	44
Shell 16.0: trafficC.asm	47
Shell 17.0: travelB.asm	50
Shell 18.0: travelC.asm	52



```

*****
*   Pgm: alarmint.asm
*
*   Desc: use interrupt mechanism to mimic a count-down alarm.
*         It reads the free-running counter, adds it to a delay time
*         and stores the result to Output Compare Register 5 (TOC5).
*         The Output Compare Flag will be set when TOC5 value
*         equal to the value of free-running counter. Since the
*         cycle range of the free-running counter is about 32 ms
*         for the 2MHz CPU, a number of iterations for OC5 is
*         performed. The program initialize count to 99, decrements
*         the digit to be displayed per 1/4 second, and sends
*         digit to a MAN6610 LED from port-C and port-B. Meanwhile,
*         it generates tic sound for each count thru port-B pin 4.
*         When it counts down to "00", it blinks "00", and generate
*         beeps in two different frequencies. It repeats the beeps
*         until reset button is pressed.
*
*   Author: ZUYI CHEN (University of Montana)
*
*   Date: June, 1992
*****
PVOC5    equ     $00D3    ; pseudo vector address of OC5
BASE     equ     $1000    ; base address of register block
PORTA    equ     $00      ; (offset from base address) port-A
PORTB    equ     $04      ; (offset from base address) port-B
PORTC    equ     $03      ; (offset from base address) port-C
DDRC     equ     $07      ; (offset) port-C control reg
TCNT     equ     $0E      ; (offset) free-running counter
TMSK1    equ     $22      ; (offset) timer interrupt mask
TFLG1    equ     $23      ; (offset) timer flag 1
TOC5     equ     $1E      ; (offset) output compare register 5

        org     $C000    ; pgm start on 68HC11 - EVB
        jsr     INIT     ; initialize the interrupt

DIG_LOOP
        ldy     #????    ; number of iterations mapping 1/4
*                               ; sec
TLP      brclr   TFLG1,X $08 * ; wait until the interrupt comes
        dey     ; decrement OC5 interrupt iteration
        bne     TLP
        jsr     CLEAR    ; Clear the display

        ldab   B_DIGIT   ; get CUR_DIGIT
        jsr    GET_DIGIT ; display the right-hand digit
        staa   PORTB,X
        decb   ; decrement the digit
        cmpb  #-1        ; if value smaller than 0 then reset
        bne   SKIP1     ; to 9
        ldab  #9
SKIP1    stab   B_DIGIT   ; store to memory
        ldab  C_DIGIT   ; get CUR_DIGIT
        jsr   GET_DIGIT ; display the left-hand digit

```

```

    staa PORTC,X          ; (multiple of 10)
    ldaa B_DIGIT
    cmpa #9               ; check if right-hand digit has been
    bne     SKIP2         ; counted down to 0
    decb                    ; decrement the digit
    cmpb #-1             ; if value smaller than 0 then ring
    bne     SKIP2         ; the alarm
    jsr     ALARM         ; ring the alarm

SKIP2   stab C_DIGIT     ; store to memory
        bra     DIG_LOOP

```

```

*****
* Subpgm: INIT
* Desc: initialize digit to be displayed and OC5 interrupt
*****
INIT

```

```

        ldaa #9
        staa C_DIGIT     ; initialize the digit to be
displayed staa B_DIGIT   ; to "99"

        ldaa #$7E       ; get extended op code for jump
        staa PVOC5      ; and store to pseudo vector OC5
        ldx  #INTERRUPT ; get address of Interrupt Routine
        stx  PVOC5+1    ; and store after jump in vector

table   ldx  #BASE      ; get base address of the
*       ; register block

        ldaa #$FF
        staa DDRC,X     ; set port-C for output only
        ldaa #$08       ; get OC5F bit
        staa TFLG1,X    ; set OC5F bit of timer flag
        staa TMSK1,X    ; enable OC5F interrupt
        cli              ; enable interrupts
        rts

```

```

*****
* ISR: INTERRUPT
* Desc: get the current free-running counter; add 4000 cycles,
*       and store to OC5. Interrupt comes when the value in OC5
*       equals to the free-running counter
*****

```

```

INTERRUPT
        ldd  TCNT,X     ; get free-running counter
        add  #$A000     ; add $A000 cycles
        std  TOC5,X    ; store to Output Compare register 5
        bclr TFLG1,X $F7 ; clear the OC5F bit for next use
        rti

```

```

*****
* Subpgm: GET_DIGIT
* Desc: Display the number on the MAN6610 display

```

```

*****
GET_DIGIT
    ldy      #DIGITO      ; load the address of DIGITO
    aby      ; add value of reg B to index reg X
    ldaa 0,Y      ; load digit/pin map
    rts

```

```

*****
* Subpgm CLEAR
* Desc: clear the display
*****

```

```

CLEAR
    ldaa  #00      ; Clear the display
    staa  PORTC,X
    staa  PORTB,X
    rts

```

```

*****
* There are 14 * FREQ + 39 cycles in RESONANT subroutine.
* All except the 5 cycles for rts will be repeated the number
* of times equal to the value of index register Y on the
* subprogram entry. To make each beep stay for half sec, it
* should take the nearest cycles to 1,000,000 for the 2MHz
* EVB CPU.
*****

```

```

ALARM
    ldaa #118      ; 118*(RESONANT cycles)+misc
    ldab #$7F      ; are nearest to 1/2 sec
    stab PORTB,X   ; blink "00" on display
    stab PORTC,X   ; blink "00" on display
    ldy  #605      ; frequency for music note 'dou'
    sty  FREQ      ; store to the memory
    jsr  RESONANT  ; produce the sound

    jsr  CLEAR     ; blink "00" on display
    ldaa #166      ; 166*(RESONANT cycles)+misc
    ldab #$7F      ; are close to 1/2 sec
    ldy  #430      ; frequency for music note 'sou'
    sty  FREQ      ; store to the memory
    jsr  RESONANT  ; produce the sound

    bra  ALARM

```

```

*****
* Subpgm: RESONANT
* Desc: produce the music note specified by the FREQ on the entry
*****

```

```

RESONANT
    stab PORTA,X   ; turn on the sound
    ldy  FREQ      ; get the frequency

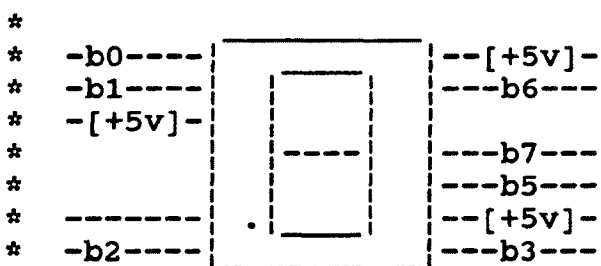
LOOP_ON
    dey           ; decrement frequency
    bne  LOOP_ON  ; back to LOOP_ON if frequency not 0
    pshb        ; push reg B to stack

```

```

        ldab #0
        stab PORTA,X          ; turn off the sound
        pulb                  ; pop reg B off stack
        ldy  FREQ             ; get the frequency
LOOP_OFF
        dey                   ; decrease frequency
        bne  LOOP_OFF        ; back to LOOP_OFF if frequency not
*                               ; 0
        deca                  ; decrement Y value
        bne  RESONANT        ; back to RESONANT if Y not 0
        rts
    
```

\*\*\*\*\*  
 \* DATA: Table of digit/Port-C/Port-B pin mapping  
 \*\*\*\*\*



\* Note: bit 4 of port-C and that of PORT-B are connected to a  
 \* piezo buzzer, and are used to generate a tic sound;  
 \* [+5v] indicates connecting +5v pin to one of the 3 outlets  
 \* specified by [+5v]

-----  
 \* Port-C/Port-B bits 7 6 5 4 3 2 1 0  
 -----

DIGIT0	fcB	\$7F	; 0 1 1 1 1 1 1 1
DIGIT1	fcB	\$70	; 0 1 1 1 0 0 0 0
DIGIT2	fcB	\$DD	; 1 1 0 1 1 1 0 1
DIGIT3	fcB	\$F9	; 1 1 1 1 1 0 0 1
DIGIT4	fcB	\$F2	; 1 1 1 1 0 0 1 0
DIGIT5	fcB	\$BB	; 1 0 1 1 1 0 1 1
DIGIT6	fcB	\$BF	; 1 0 1 1 1 1 1 1
DIGIT7	fcB	\$71	; 0 1 1 1 0 0 0 1
DIGIT8	fcB	\$FF	; 1 1 1 1 1 1 1 1
DIGIT9	fcB	\$F3	; 1 1 1 1 0 0 1 1

C\_DIGIT rmb 2 ; reserve 2 bytes memory

B\_DIGIT rmb 2

FREQ rmb 2

\*\*\*\*\*  
 \* End of Pgm  
 \*\*\*\*\*

\*\*\*\*\*

\* Pgm: alarmpol.asm

\*

\* Desc: use polling mechanism to mimic a count-down alarm.  
 \* It reads the free-running counter, add it to a delay time  
 \* and stores the result to Output Compare Register 2 (TOC2).  
 \* The Output Compare Flag will be set when TOC2 value  
 \* equal to the value of free-running counter. Since the  
 \* cycle range of the free-running counter is about 32 ms  
 \* for the 2MHz CPU, a number of iterations for OC2 is  
 \* performed. The program initialize count to 99, decrements  
 \* the digit to be displayed per 1/4 second, and sends  
 \* digit to a MAN6610 LED from port-C and port-B. Meanwhile,  
 \* it generates tic sound for each count thru port-B pin 4.  
 \* When it counts down to "00", it blinks "00", and generate  
 \* beeps in two different frequencies. It repeats the beeps  
 \* until reset button is pressed.

\*

\* Author: ZUYI CHEN (University of Montana)

\*

\* Date: June, 1992

\*\*\*\*\*

```

BASE      equ      $1000      ; base address of register block
PORTA     equ      $00        ; (offset from base address) port-A
PORTB     equ      $04        ; (offset from base address) port-B
PORTC     equ      $03        ; (offset from base address) port-C
DDRC      equ      $07        ; (offset) port-C control reg
TCNT      equ      $0E        ; (offset) free-running counter
TFLG1     equ      $23        ; (offset) timer flag 1
TOC2      equ      $18        ; (offset) output compare register 2
          org      $C000      ; pgm start on 68HC11 - EVB
          ldx      #BASE      ; get base address of the register
          ; block
          ldaa     #$FF
          staa     DDRC,X     ; set port-C for output only
          ldaa     #9
          staa     C_DIGIT    ; initialize the digit to be
          ; displayed
          staa     B_DIGIT    ; to "99"
          ldaa     #$8        ; get OC2F bit
          staa     TFLG1,X    ; set OC2F bit of timer flag

DIG_LOOP
          jsr     CLEAR      ; Clear the display
          ldab    B_DIGIT    ; get CUR_DIGIT
          jsr     GET_DIGIT   ; display the right-hand digit
          staa    PORTB,X
          decb    ; decrement the digit
          cmpb   #-1         ; if value smaller than 0 then reset
          bne    SKIP1       ; to 9
          ldab   #9
SKIP1     stab   B_DIGIT     ; store to memory

```

```

        ldab C_DIGIT          ; get CUR_DIGIT
        jsr  GET_DIGIT        ; display the left-hand digit
        staa PORTC,X          ; (multiple of 10)
        ldaa B_DIGIT
        cmpa #9                ; check if right-hand digit has been
        bne  SKIP2            ; counted down to 0
        decb                    ; decrement the digit
        cmpb #-1               ; if value smaller than 0 then ring
        bne  SKIP2            ; the alarm
        jsr  ALARM            ; ring the alarm

SKIP2   stab C_DIGIT          ; store to memory

```

```

*****
* POLLING free-running counter
*****
        ldy      #????        ; number of T_LOOP iteration
*
*                               ; mapping 1/4 sec
T_LOOP  ldd      TCNT,X        ; get free-running counter
        addd    #4000         ; add 4000 cycles
        std     TOC2,X        ; store to Output Compare Register 2
        brclr   TFLG1,X $40 * ; wait for output compare
        bclr   TFLG1,X $BF    ; clear the OC2F bit for next
*
*                               ; use
        dey     ; decrement count of OC2
*
*                               ; iteration
        bne    T_LOOP
        bra    DIG_LOOP      ; start all over

```

```

*****
* Subpgm: GET_DIGIT
* Desc: Display the number on the MAN6610 display
*****

```

```

GET_DIGIT
        ldy  #DIGIT0          ; load the address of DIGIT0
        aby                    ; add value of reg B to index
*
*                               ; reg X
        ldaa 0,Y              ; load digit/pin map
        rts

```

```

*****
* Subpgm CLEAR
* Desc: clear the display
*****

```

```

CLEAR
        ldaa  #0             ; Clear the display
        staa PORTC,X
        staa PORTB,X
        rts

```

```

*****
* There are 14 * FREQ + 39 cycles in RESONANT subroutine.
* All except the 5 cycles for rts will be repeated the number
* of times equal to the value of index register Y on the

```

\* subprogram entry. To make each beep stay for half sec, it  
 \* should take the nearest cycles to 1,000,000 for the 2MHz  
 \* EVB CPU.

\*\*\*\*\*

ALARM

```

    ldaa #118          ; 118*(RESONANT cycles)+misc
    ldab #$7F         ; are nearest to 1/2 sec
    stab PORTB,X      ; blink "00" on display
    stab PORTC,X      ; blink "00" on display
    ldy #605          ; frequency for music note 'dou'
    sty  FREQ         ; store to the memory
    jsr  RESONANT     ; produce the sound

    jsr  CLEAR        ; blink "00" on display
    ldaa #166         ; 166*(RESONANT cycles)+misc
    ldab #$7F         ; are close to 1/2 sec
    ldy #430          ; frequency for music note 'sou'
    sty  FREQ         ; store to the memory
    jsr  RESONANT     ; produce the sound

    bra  ALARM
  
```

\*\*\*\*\*

\* Subpgm: RESONANT

\* Desc: produce the music note specified by the FREQ on the entry

\*\*\*\*\*

RESONANT

```

    stab PORTA,X      ; turn on the sound
    ldy  FREQ         ; get the frequency

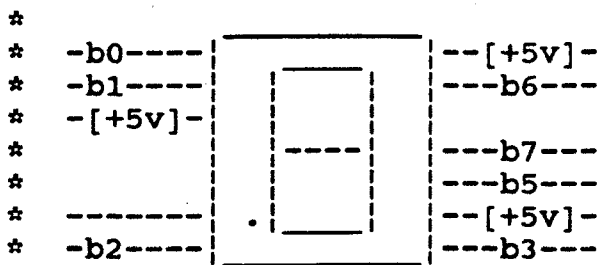
LOOP_ON
    dey              ; decrement frequency
    bne  LOOP_ON     ; back to LOOP_ON if frequency not 0
    pshb             ; push reg B to stack
    ldab #0
    stab PORTA,X     ; turn off the sound
    pulb             ; pop reg B off stack
    ldy  FREQ         ; get the frequency

LOOP_OFF
    dey              ; decrease frequency
    bne  LOOP_OFF     ; back to LOOP_OFF if frequency not
*                  ; 0
    deca             ; decrement Y value
    bne  RESONANT    ; back to RESONANT if Y not 0
    rts
  
```

\*\*\*\*\*

\* DATA: Table of digit/Port-C/Port-B pin mapping

\*\*\*\*\*



\* Note: bit 4 of PORT-C and that of PORT-B are connected  
\* to a piezo buzzer, and are used to generate a tic sound,  
\* [+5v] indicates connecting +5v pin to one of the 3  
\* outlets specified by [+5v]

-----  
\* Port-C/Port-B bits 7 6 5 4 3 2 1 0  
-----

```

DIGIT0    fcb      $7F   ; 0 1 1 1 1 1 1 1
DIGIT1    fcb      $70   ; 0 1 1 1 0 0 0 0
DIGIT2    fcb      $DD   ; 1 1 0 1 1 1 0 1
DIGIT3    fcb      $F9   ; 1 1 1 1 1 0 0 1
DIGIT4    fcb      $F2   ; 1 1 1 1 0 0 1 0
DIGIT5    fcb      $BB   ; 1 0 1 1 1 0 1 1
DIGIT6    fcb      $BF   ; 1 0 1 1 1 1 1 1
DIGIT7    fcb      $71   ; 0 1 1 1 0 0 0 1
DIGIT8    fcb      $FF   ; 1 1 1 1 1 1 1 1
DIGIT9    fcb      $F3   ; 1 1 1 1 0 0 1 1

```

```

C_DIGIT   rmb  2    ; 2 bytes variable

```

```

B_DIGIT   rmb  2

```

```

FREQ      rmb  2

```

\*\*\*\*\*

\* End of Pgm

\*\*\*\*\*



```

*****
*   Pgm: input.asm
*
*   Desc: This program outputs data from port-B, and inputs
*         the data from port-C. Assume port-B and port-C pins
*         are connected. No external devices are set for these
*         ports, otherwise the devices draw voltage from outputs
*         and the voltage of the latter may not be high enough to
*         be input to port-C. When the data input is finished,
*         port-A pin 3 is set. A LED is connected to that pin to
*         check if the pin is set. The inputs are stored in memory
*         location $D000-$DFFF.
*
*   Author: ZUYI CHEN (University of Montana)
*
*   Date: June, 1992
*****

```

```

PIOC      equ      $1002      ; Port-B/Port-C I/O control reg
PORTA     equ      $1000      ; memory location fo port-A
PORTB     equ      $1004      ; memory location of Port-B
PORTCL    equ      $1005      ; location of Port-C latch
*         ; register
DDRC      equ      $1007      ; Port-C direction control
*         ; register
TCNT      equ      $100E      ; free-running counter
START     equ      $D000      ; local memory $D000 - $DFFF
END       equ      $DFFF      ; upper 4K of the user RAM

          org      $C000      ; pgm start on 68HC11 - EVB

```

```

*****
* MAIN
*****

```

```

        ldaa #$00          ; initial Port-C for input
        staa DDRC         ; all 8 bits
        staa PORTB        ; clear port-B
        staa DATA        ; store in memory
        ldy  #START       ; get starting storage

```

```

LOOP

```

```

        ldab DATA        ; output a number to port-B
        stab PORTB        ; outputs thru port-B
        jsr  GET_INPUT     ; inputs thru port-C
        jsr  STORE_OUTPUT  ; store data and current time
*         ; to memory
        incb              ; increment data
        stab DATA        ; store to memory
        ldab #$04         ; increment memory address
        aby               ; by 1
        cpy  #END+4       ; if not passed over ending storage

```

```

    bne  LOOP
DONE
    ldaa #$08                ; light port-A pin 3
    staa PORTA
    bra  DONE

GET_INPUT
    ldaa PIOC                ; load data arrival flag
    bita #$80                ; bit 7 "on" => Z=0, "off" =>
*
                                ; Z=1
    beq  GET_INPUT           ; busy wait if Z=1 (no input)
    ldaa PORTCL              ; capture inputs
    rts

STORE_OUTPUT
    staa 0,Y                 ; store data to memory
    staa 1,Y                 ; store data to memory
    ldx  TCNT                 ; get current time stamp
    stx  2,Y                 ; store to memory
    rts

DATA    rmb      2
*****
*  End of Pgm
*****
```

\*\*\*\*\*

\* Pgm IPCsnd: 3/7/91 Version

\*  
 \* Experimental version of IPC Sender, with event  
 \* descriptor transmission implemented as the explicit  
 \* subpgm EVENT. EVENT uses dump-and-run protocol  
 \* (Port-B/STR-B to Port-C/STR-A), and creates an event  
 \* (sender) time stamp and an event\_dump\_complete (sender)  
 \* time stamp. EVENT also guarantees a "safe" time  
 \* interval between individual packet sends for the  
 \* event descriptor.

\* Author: Ray Ford (University of Montana)

\* Modified by Zuyi Chen of UM in June, 1992

\* \* changed \$6000 - \$7FFF to \$D000-\$DFFF since the optional  
 \* 8k memory does not come with the board;  
 \* \* changed PORT-D to PORT-A, and removed DDRD;  
 \* \* removed SIGNAL subroutine.

\* Date: March 1991

\* A. 4 byte Event Descriptor:

\* 0,1: 16-bit time stamps TS\_EI, captured at EVENT entry  
 \* 2,3: 16-bit data computed by dummy main pgm

\* B. Dump-And-Run Protocol

\* 1. The event descriptor is driven out of Port-B  
 \* in four 1-byte packets, in order:  
 \* <TS\_EI(HOB),TS\_EI(LOB),Data1(HOB),Data1(LOB)>  
 \* where HOB: high order byte and LOB: low order byte  
 \* 2. Port-B is set to operate in simple strobe mode where  
 \* a write to Port-B automatically pulses the STR-B pin.  
 \* Sender/STR-B is assumed to be connected to Receiver/STR-A,  
 \* so that the Sender/STR-B pulse signals a PORT-CL input  
 \* capture by the Receiver. Simple strobe mode for Port-B is  
 \* indicated by "0" in bit 4 of PIOC, at \$1002. [Note: this  
 \* is the default mode of operation for Port-B.]

\* C. Local data storage, 8-byte packets:

\* 0,1: 16-bit time stamps TS\_EI  
 \* 2,3: 16-bit data, in IND-X at EVENT entry  
 \* 4,5: 16-bit data, copy of above (dummy)  
 \* 6,7: 16-bit time stamp TS\_EC ("event complete")  
 \* A "done" signal is written out on Port-A when local  
 \* memory is full.

\* D. Implementation Notes

\* Main Pgm  
 \* IND\_X: data IND\_Y: memory index  
 \* ACC\_A,ACC\_B,ACC\_D: time stamps and misc

\*\*\*\*\*

PORTB equ \$1004 ; Port-B (data output)  
 PORTA equ \$1000 ; Port-A ("done" signal)  
 TCNT equ \$100E ; Free Running Timer

```

MEM_ST    equ    $D000                ; local memory $D000..$DFFF
MEM_END   equ    $DFFF

                org    $C000          ; pgm start on 68HC11 - EVB

TEMP      fdb    $0000                ; temporary storage

*****
* MAIN PROGRAM: IPCsnd
*****
INIT
        ldx    # $0000                ; initialize Data
        ldy    # MEM_ST               ; initialize memory index

CYCLE
        inx                    ; "compute" Data
        jsr    EVENT               ; generate "event"

        std    0,Y                ; save TS_EI
        stx    2,Y                ; save Data
        stx    4,Y                ; save Data (dummy extra copy)
        ldd    TCNT                ; capture TS_EC
        std    6,Y                ; save TS_EC

        ldab   # $08                ; load memory increment
        aby                    ; increment memory index
        cpy    # MEM_END            ; if IND_Y < $7FFF then C=1
        blo    CYCLE               ; branch if C=1

DONE
        ldaa  # $08
        staa  PORTA                ; generate "done" signal
        bra   DONE

*****
* Subpgm EVENT
* At entry: IND_X=data  IND_Y=memory index
* At exit:  IND_X=data  IND_Y=memory index  ACC_D=TS_EI
* (a) capture TS_EI
* (b) send data and TS_EI
*****
EVENT
        ldd    TCNT                ; capture TS_EI in ACC_D
        staa  PORTB                ; send TS_EI(HOB)
        jsr    IPC_PAUSE           ; wait safe time interval
        stab  PORTB                ; send TS_EI(LOB)
        jsr    IPC_PAUSE           ; wait safe time interval
        xgdx                    ; SWAP: ACC_D=data, IND_X=TS_EI
        staa  PORTB                ; send Data(HOB)
        jsr    IPC_PAUSE           ; wait safe time interval
        stab  PORTB                ; send Data(LOB)
        xgdx                    ; swap back: ACC_D=TS_EI,
*                                ; IND_X=data
        jsr    IPC_PAUSE           ; wait safe time interval

```

rts

\*\*\*\*\*

\* Subpgm IPC\_PAUSE

\*\*\*\*\*

IPC\_PAUSE

stx TEMP ; save IND\_X value

ldx #\$\$\$?? ; approx X cycle delay

D\_LOOP

dex ; busy wait loop

bne D\_LOOP

ldx TEMP ; restore value of IND\_X

rts

\*\*\*\*\*

\* End of Pgm

\*\*\*\*\*

\*\*\*\*\*

\* Pgm IPCrcv

\*

\* Experimental version of IPC Receiver, with dump-and-run  
\* protocol using Port-B/STR-B to Port-C/STR-A, with both  
\* sender and receiver time stamps, and with POLLED message  
\* receipt

\*

\* Author: Ray Ford (University of Montana)

\*

\* Modified by Zuyi Chen of UM in June, 1992

\* Changed \$6000 - \$7FFF to \$D000-\$DFFF since the optional  
\* 8k memory does not come with the board.

\*

\* Date: March 1991

\*

\* Data/Dump-And-Run Protocol -- Sender:

\* (a) data is driven out of Port-B, in a 4-byte packet

\* 1,2: <SenderTS(HOB),SenderTS(LOB)

\* 3,4: Data1(HOB),Data1(HOB)

\* where HOB: high order byte and LOB: low order byte

\* (b) Port-B is set to operate in simple strobe mode

\* so that a write to Port-B automatically pulses STR-B

\* pin. This assumes that Sender/STR-B is connected to

\* Receiver/STR-A, so that the pulse signals an input

\* capture by the Receiver. Simple strobe mode for Port-B is

\* indicated by "0" in bit 4 of PIOC, at \$1002. [Note: this

\* is the default mode of operation for Port-B.]

\*

\*\*\*\*\*

\*

Data/Dump-And-Run Protocol -- Receiver

\* (a) Sender's STR-B is connected to receiver's STR-A, so that  
\* the incoming "send" signal on STR-A triggers a "receive"  
\* signal, i.e., a "latch incoming data" in PORT-CL (note: not  
\* on Port-C).

\* (b) Two local time stamps are captured to record the receipt  
\* of the event descriptor on the monitor processor

\* (i) "IO\_Initiated" (IOInitTS) is captured when the signal  
\* indicating the start of a new message is received

\* (ii) "IO\_Complete" (IOCompTS) is captured when all 6 bytes  
\* of the incoming message have been received

\* (c) data is stored locally in an 8-byte packet

\* 1,2: <SenderTS(HOB),SenderTS(LOB)

\* 3,4: Data1(HOB),Data1(HOB)

\* 5,6: IOInitTS(HOB),IOInitTS(LOB)

\* 7,8: IOCompTS(HOB),IOCompTS(LOB) >

\*

\*\*\*\*\*

\* This Implementation uses

\*

\* (a) POLLING to detect incoming messages

\* (b) Normal Reg Use:

\* ACC\_A: incoming data (from PORT-CL)

```

* ACC_B: memory index increment value (8)
* IND_Y: memory index
* IND_X: local time stamp capture
* (c) output "done" signal via Port-B when local
* memory is full

```

```
*****
```

```

PIOC      equ  $1002      ; Port-B/Port-C I/O control reg
PORTB     equ  $1004      ; location of Port-B
PORTC     equ  $1003      ; location of Port-C
PORTCL    equ  $1005      ; location of Port-C
DDRC      equ  $1007      ; Port-C control register
TCNT      equ  $100E      ; Free Running Timer
MEM_ST    equ  $D000      ; $D000..$DFFF local
MEM_END   equ  $DFFF      ; data storage

```

```
org $C000 ; pgm start on 68HC11 - EVB
```

```
*****
```

```
* MAIN PROGRAM: IPCrcv
```

```
*****
```

```
INIT_MEM
```

```

ldaa #$00 ; initial Port-C for input,
staa DDRC ; all 8 bits
ldab #$08 ; load memory index increment value
ldy #MEM_ST ; load address of data storage area

```

```
MORE_INPT
```

```

jsr GET_INPT ; ACC-A: SenderTS(HOB), IND-X: IOInitTS
staa 0,Y ; store SenderTS(HOB)
stx 4,Y ; store IOInitTS

```

```

jsr GET_INPT ; ACC-A: SenderTS(LOB), IND-X: misc TS
staa 1,Y ; store SenderTS(LOB) -- discard misc TS

```

```

jsr GET_INPT ; ACC-A: Data1(HOB), IND-X: misc TS
staa 2,Y ; store Data1(HOB) -- discard misc TS

```

```

jsr GET_INPT ; ACC-A: Data1(LOB), IND-X: IOCompTS
staa 3,Y ; store Data1(LOB)
stx 6,Y ; store IOCompTS

```

```

aby ; increment memory index by 8
cpy #MEM_END ; watch for full memory
blo MORE_INPT

```

```
DONE
```

```

ldaa #$FF ; load "done" signal
stab PORTB ; write "done" signal to Port-B
bra DONE

```

```
*****
```

```
* Subpgm GET_INPT:
```

```
* (a) data arrival indicated by value of bit 7 (STAF flag) in
```

```
* PIOC
```

```
* (b) at return:
```

```
* ACC-A is captured input value
* IND-X is input arrival time stamp
* PIOC is cleared (automatically, by test and PORTCL load)
```

```
*****
```

```
GET_INPT
```

```
    ldaa PIOC          ; load data arrival flag
    bita #$80          ; bit 7 "on" => Z=0, "off" => Z=1
    beq GET_INPT       ; busy wait if Z=1 (no input)
    ldaa PORTCL        ; capture data
    ldx TCNT           ; input => capture local time stamp
    rts
```

```
*****
```

```
* End of Pgm
```

```
*****
```



\*\*\*\*\*

\* Pgm: moduloB.asm

\*

\* Desc: mimic a modulo-9 counter by driving signals out of  
 \* port-B pins to a 7-segment display. It displays  
 \* number 0 and keeps it on for 1/2 sec; then it clears  
 \* the display for 1/2 sec. It increments the number by 1,  
 \* displays it, clears it until 9 is displayed and cleared.  
 \* It then resets the number to 0, repeats whole procedure.

\* Author: ZUYI CHEN (University of Montana)

\*

\* Date: June, 1992

\*\*\*\*\*

```

PORTB equ    $1004      ; memory location of "port B"
COUNT equ    42        ; number DELAY iterations about 1/2
*                               ; sec
DCOUNT equ    4000      ; number of DLOOP iterations

        org    $C000    ; pgm start on 68HC11 - EVB

```

\*\*\*\*\*

\* MAIN

\*\*\*\*\*

```

        jsr    CLEAR    ; Clear 7-segment display

RESET   ldab #0          ; initialize the 7-segment display

LOOP    jsr    OUT_DIGIT ; display the number
        ldy   #COUNT   ; set number of the DELAY iterations
        jsr   DELAY     ; wait
        incb                ; increment the number to be
*                               ; displayed
        jsr   CLEAR     ; Clear the display
        cmpb #9          ; compare current number with 9
        bgt  RESET      ; back to RESET if number
*                               ; larger than 9
        bra  LOOP

```

\*\*\*\*\*

\* Subpgm: OUT\_DIGIT

\* Desc: Display the number on the 7-segment display

\*\*\*\*\*

```

OUT_DIGIT
        ldx  #DIGITO    ; load the address of DIGITO
        abx                ; add value of reg B to index
*                               ; reg X
        ldaa 0,x        ; load digit/pin map
        staa PORTB      ; write to PORT B
        rts

```

```
*****
* Subpgm DELAY
* Desc: "busy wait" for the number of DELAY loop iterations
* specified by the value of index register Y at subprogram
entry.
```

```
*****
DELAY
      ldx  #DCOUNT      ; set DLOOP iterations
DLOOP dex          ; decrement index register X value
      bne  DLOOP       ; back to DLOOP if X value not 0
      dey          ; decrement index register Y count
      bne  DELAY       ; back to DELAY if Y count not 0
      rts              ; return to calling routine
```

```
*****
```

```
* Subpgm CLEAR
* Desc: clear the 7-segment display
```

```
*****
```

```
CLEAR
      ldaa  #$00        ; Clear 7-segment display by sending
      staa  PORTB      ; '0000 0000' to port-B
      ldy   #COUNT    ; delay cycles
      jsr  DELAY
      rts
```

```
*****
```

```
* DATA: Table of Port-B pin assignments
```

```
*****
```

```
*-----
*                               Port-B bits 7 6 5 4 3 2 1 0
*-----
DIGIT0  fcb      $6F          ; 0 1 1 * 1 1 1 1
DIGIT1  fcb      $60          ; 0 1 1 * 0 0 0 0
DIGIT2  fcb      $CD          ; 1 1 0 * 1 1 0 1
DIGIT3  fcb      $E9          ; 1 1 1 * 1 0 0 1
DIGIT4  fcb      $E2          ; 1 1 1 * 0 0 1 0
DIGIT5  fcb      $AB          ; 1 0 1 * 1 0 1 1
DIGIT6  fcb      $AF          ; 1 0 1 * 1 1 1 1
DIGIT7  fcb      $61          ; 0 1 1 * 0 0 0 1
DIGIT8  fcb      $EF          ; 1 1 1 * 1 1 1 1
DIGIT9  fcb      $E3          ; 1 1 1 * 0 0 1 1
```

```
*****
```

```
* End of Pgm
```

```
*****
```

```
*****
* Pgm: moduloC.asm
*
* Desc: mimic a modulo-9 counter by driving signals out of
*       port-C pins to a 7-segment display. It displays
*       number 0, and keeps it on for 1/2 sec; then it clears
*       the display for 1/2 sec. It increments the number by 1,
*       displays it, clears it. After 9 is displayed, and cleared,
*       it resets the number to 0, and repeats the whole
*       procedure.
```

```
* Author: ZUYI CHEN (University of Montana)
```

```
* Date: June, 1992
```

```
*****
```

```
PORTC equ    $1003      ; memory location of "port C"
DDRC  equ    $1007      ; port C I/O control
COUNT equ    42        ; number DELAY iterations
*                               ; about 1/2 sec
DCOUNT equ    4000      ; number of DLOOP iterations

org    $C000           ; pgm start on 68HC11 - EVB

ldaa   #$FF           ; set port-C to output only by
staa   DDRC           ; sending '1111 1111' to port-C
```

```
*****
```

```
* MAIN
```

```
*****
```

```
        jsr    CLEAR          ; Clear 7-segment display

RESET   ldab   #0             ; initialize the 7-segment display
LOOP    jsr    OUT_DIGIT      ; display the number
        ldy   #COUNT        ; set number of the DELAY iterations
        jsr   DELAY          ; wait
        incb                ; increment the number to be
*                               ; displayed
        jsr   CLEAR          ; Clear the display
        cmpb  #9             ; compare current number with 9
        bgt  RESET          ; back to RESET if number
*                               ; larger than 9
        bra   LOOP
```

```
*****
```

```
* Subpgm: OUT_DIGIT
```

```
* Desc: Display the number on the 7-segment display
```

```
*****
```

```
OUT_DIGIT   ldx   #DIGITO      ; load the address of DIGITO
```

```

abx                ; add value of reg B to index reg X
ldaa 0,x           ; load digit/pin map
staa PORTC        ; write to PORT C
rts

```

```
*****
```

```
* Subpgm DELAY
```

```
* Desc: "busy wait" for the number of DELAY loop iterations
*       specified by the value of index register Y at subprogram
*       entry.
```

```
*****
```

```
DELAY
```

```

DLOOP      ldx  #DCOUNT    ; set DLOOP iterations
           dex                ; decrement index register X value
           bne  DLOOP      ; back to DLOOP if X value not 0
           dey                ; decrement index register Y count
           bne  DELAY      ; back to DELAY if Y count not 0
           rts                ; return to calling routine

```

```
*****
```

```
* Subpgm CLEAR
```

```
* Desc: clear the 7-segment display
```

```
*****
```

```
CLEAR
```

```

ldaa  #00          ; Clear 7-segment display by sending
staa  PORTC        ; '0000 0000' to port-C
ldy   #COUNT      ; delay cycles
jsr   DELAY
rts

```

```
*****
```

```
* DATA: Table of Port-C pin assignments
```

```
*****
```

```

*-----
*                               Port-C bits 7 6 5 4 3 2 1 0
*-----
DIGIT0   fcb      $6F           ; 0 1 1 * 1 1 1 1
DIGIT1   fcb      $60           ; 0 1 1 * 0 0 0 0
DIGIT2   fcb      $CD           ; 1 1 0 * 1 1 0 1
DIGIT3   fcb      $E9           ; 1 1 1 * 1 0 0 1
DIGIT4   fcb      $E2           ; 1 1 1 * 0 0 1 0
DIGIT5   fcb      $AB           ; 1 0 1 * 1 0 1 1
DIGIT6   fcb      $AF           ; 1 0 1 * 1 1 1 1
DIGIT7   fcb      $61           ; 0 1 1 * 0 0 0 1
DIGIT8   fcb      $EF           ; 1 1 1 * 1 1 1 1
DIGIT9   fcb      $E3           ; 1 1 1 * 0 0 1 1

```

```
*****
```

```
* End of Pgm
```

```
*****
```

```

*****
* Pgm: music.asm
*
* Desc: mimics music notes by driving signals out of port-B pins
*       to a piezo buzzer in following order:
*       1. ring music note 'mee' for 1/2 sec
*       2. ring music note 'rai' for 1/2 sec
*       3. ring music note 'do' for 1/2 sec
*       4. ring music note 'tee' for 1/2 sec
*       5. ring music note 'la' for 1/2 sec
*       6. ring music note 'so' for 1/2 sec
*       7. ring music note 'fa' for 1/2 sec
*       8. ring lower music note 'mee' for 1/2 sec
*       9. go back to (1) and repeat
*
* Author: ZUYI CHEN (University of Montana)
*
* Date: June, 1992
*****

```

```

PORTB equ $1004 ; memory location of "port B"
INNER equ 21
org $C000 ; pgm start on 68HC11 - EVB

```

```

*****
* Note: the music note is not standard.
*****

```

```

ldab #01 ; are nearest to 1/2 sec
START
ldy #315
ldx #262 ; frequency for music note 'mee'
stx FREQ ; store to the memory
jsr RESONANT ; produce the sound

ldy #281 ; 150*(RESONANT cycles)+25
ldx #294 ; frequency for music note 'rai'
stx FREQ ; store to the memory
jsr RESONANT ; produce the sound

ldy #251 ; 164*(RESONANT cycles)+25
ldx #330 ; frequency for music note 'do'
stx FREQ ; store to the memory
jsr RESONANT ; produce the sound

ldy #237 ; 173*(RESONANT cycles)+25
ldx #349 ; frequency for music note 'tee'
stx FREQ ; store to the memory
jsr RESONANT ; produce the sound

ldy #???? ; number of RESONANT iterations mapping 1/2 sec
ldx #392 ; frequency for music note 'la'
stx FREQ ; store to the memory
jsr RESONANT ; produce the sound

ldy #???? ; number of RESONANT iterations mapping 1/2 sec
ldx #440 ; frequency for music note 'so'
stx FREQ ; store to the memory
jsr RESONANT ; produce the sound

ldy #???? ; number of RESONANT iterations mapping 1/2 sec
ldx #494 ; frequency for music note 'fa'
stx FREQ ; store to the memory
jsr RESONANT ; produce the sound

ldy #???? ; number of RESONANT iterations mapping 1/2 sec
ldx #523 ; frequency for lower music note 'mee'

```

```
stx  FREQ      ; store to the memory
jsr  RESONANT  ; produce the sound
```

F22

```
bra  START
```

```
*****
* Subpgm: RESONANT
* Desc: produce the music note specified by the FREQ on the entry
*****
```

```
RESONANT
```

```
    stab      PORTB      ; turn on the sound
    ldx       FREQ       ; get the frequency
LOOP_ON
    dex              ; decrement frequency
    bne       LOOP_ON   ; back to LOOP_ON if frequency not 0
    ldaa     #0
    staa     PORTB      ; turn off the sound
    ldx       FREQ       ; get the frequency
LOOP_OFF
    dex              ; decrease frequency
    bne       LOOP_OFF  ; back to LOOP_OFF if frequency not 0
    dey              ; decrement Y value
    bne       RESONANT  ; back to RESONANT if Y not 0
    rts
```

```
FREQ rmb      2
```

```
*****
```

```
* End of Pgm
```

```
*****
```

```

*****
* Pgm: oc5int.asm
*
* Desc: use Output Compare 5 (OC5) interrupt to generate a square
*       wave at the PA3 output pin. The program writes data to
*       memory starting at the address $D000, and stops at $DFFF.
*       $E000 and up are monitor EPROM, and can be written
*       by user programs.
*       Note: REG_ST ($1000) is the starting address of the register
*             block. With offset specified in the "equ" directive,
*             the memory location for Timer Control Register 1
*             (TCTL1), Timer Interrupt Mask (TMSK1), Timer Interrupt
*             Flag 1 (TFLG1), Timer Output Compare Register 5 (TOC5),
*             and Free-running Counter (TCNT) can be found.
*
* Author: ZUYI CHEN (University of Montana)
*
* Date: June, 1992
*****

```

```

PVOC5    equ    $00D3    ; pseudo vector address for OC5
REG_ST   equ    $1000    ; starting address of register
*                               ; block
TCTL1    equ    $20      ; timer control register 1
TMSK1    equ    $22      ; timer interrupt mask register
TFLG1    equ    $23      ; timer interrupt flag
TOC5     equ    $1E      ; Output Compare register 5
TCNT     equ    $0E
MEM      equ    $D000

        org    $C000    ; pgm start on 68HC11 - EVB

```

```

*****
* MAIN
*****

```

```

        ldaa   #$7E      ; extended op code of jump instruction
        staa   PVOC5     ; pseudo vector for OC5
        ldx    #INT5     ; put address of Interrupt Routine
        stx    PVOC5+1   ; after address of jmp

        ldy    #MEM      ; get memory start to fill data
        sty    STORE

        ldx    #REG_ST   ; get register block start address
        ldaa   #$01      ; set OL5 bit on in TCTL1
        staa   TCTL1,X
        ldaa   #$08
        staa   TFLG1,X   ; clear OC5F bit if it is set
        staa   TMSK1,X   ; enable OC5 interrupt
        cli    ; enable interrupts
        bra   *          ; interrupt driven from here

```

```
*****
* ISR: INT5
* Desc: OC5 interrupt service routine to generate a square wave.
*      called at each OC5 interrupt
*****
```

```
INT5
```

```
    ldy  STORE                ; get address to fill data
    ldd  TCNT,X               ; get current time stamp
    std  0,Y                  ; record the time in memory
    ldab #02                  ; increment address by 2 bytes
    aby
    sty  STORE                ; save the address
    ldd  #0100                ; cycle value
    addd TOC5,X               ; add to last compare value
    std  TOC5,X               ; update OC5
    bclr TFLG1,X $F7          ; clear OC5F
    rti                       ; return from interrupt
*                               ; service routine
```

```
STORE    rmb    2
```

```
*****
* End of Pgm
*****
```



```

*****
* Pgm: outports.asm
*
* Desc: The program checks the output pin connections by turning
*       on the bits of port-C port-D, port-A and port-B in order.
*       It first turns on the most significant bit, then add the
*       next bit without turning off the previous one. When all
*       the bits of a port are turned on, it clears the port, goes
*       to the next port, and does the same in that port. After
*       port-B is lit, it goes back to port-C and starts all over.
* Note: the output pins for port-A are pin 3-7 and those for
*       port-D are pin 2-5
*
* Author: ZUYI CHEN (University of Montana)
*         (based on che_output.asm by Sixing Gu of UM)
*
* Date: June, 1992
*****

PORTC    equ     $1003    ; memory location of "port C"
DDRC     equ     $1007    ; memory location of port C control
*                               ; reg
PORTA    equ     $1000    ; memory location of "port A"
DDRA     equ     $1026    ; memory location of port A pin 7
*                               ; direction
PORTB    equ     $1004    ; memory location of "port B"
PORTD    equ     $1008    ; memory location of "port D"
DDRD     equ     $1009    ; memory location of port D control
*                               ; reg
INNER    equ     3        ; set count of DELAYs for inner ports
INTER    equ     7        ; set count of DELAYs for inter ports
DCOUNT   equ     $C000    ; set count of DLOOPS in DELAY

        org     $C000    ; pgm start on 68HC11 - EVB

*****
* MAIN
*****

        ldaa   #$FF      ; Set port A and C to output by sending
        staa   DDRC      ; '1111 1111' to Port-C control register

        ldaa   #$3C      ; Set Port-D pin 2-5 to output by sending
        staa   DDRD      ; '0011 1100' to Port-D control register

        ldaa   #$80      ; Set Port-D pin 2-5 to output by sending
        staa   DDRA      ; '1000 0000' to Port-A control register

        jsr    CLEAR_ALL ; clear all ports

LOOP    jsr    LITE_C     ; turn port-C ON, others OFF
        ldy   #INTER     ; set delay cycles
        jsr   DELAY      ; check port-C

```

```

jsr    CLEAR_C    ; clear port-C
jsr    LITE_D     ; turn port-D ON, others OFF
ldy    #INTER     ; set delay cycles
jsr    DELAY      ; check port-D

jsr    CLEAR_D    ; clear port-D

jsr    LITE_A     ; turn PORT-A ON, others OFF
ldy    #INTER     ; set delay cycles
jsr    DELAY      ; check port-A
jsr    CLEAR_A    ; clear port-A

jsr    LITE_B     ; turn PORT-B ON, others OFF
ldy    #INTER     ; set delay cycles
jsr    DELAY      ; check port-B

jsr    CLEAR_B    ; clear port-B

bra    LOOP

```

\*\*\*\*\*

```

* Subpgm DELAY
* Desc: "busy wait" for the number of DELAY loop iterations
*       specified by the value of index register Y at subprogram
*       entry.

```

\*\*\*\*\*

```

DELAY
DLOOP    ldx      #DCOUNT      ; hardcoded unit cycle delay
          dex      ; decrement index register X
          *        ; value
          bne     DLOOP      ; back to DLOOP if X value not
          *        ; 0
          dey     ; decrement index register Y
          *        ; count
          bne     DELAY      ; back to DELAY if Y count not
          *        ; 0
          rts      ; return to calling routine

```

\*\*\*\*\*

```

* Subpgm CLEAR_ALL
* turn off all the bar graph displays

```

\*\*\*\*\*

```

CLEAR_ALL
ldaa    #$00      ; Clear all signals by sending
staa    PORTC     ; '0000 0000' to Port-C, port-D
staa    PORTD     ; port-A and port-B
staa    PORTA
staa    PORTB
rts

```

```
*****
* Subpgm LITE_A
* turn on the bar graph display one bit at a time connected to
* PORT-A
*****
```

```
LITE_A   ldab #5           ; set number of bits for the output
         ldaa #$80        ; turn on port-A one bit at a time
MORE_A   staa PORTA
         ldy #INNER       ; set count of DELAY
iterations
         jsr DELAY        ; pause
         asra             ; extend 1 to the bit on the right
         decb
         bne MORE_A
         rts
```

```
*****
* Subpgm CLEAR_A
* turn off the bar graph display connected to PORT-A
*****
```

```
CLEAR_A   ldaa #$00        ; clear port-A by
         staa PORTA      ; sending '0000 0000' to port-A
         rts
```

```
*****
* Subpgm LITE_B
* turn on the bar graph display one bit at a time connected to
* PORT-B
*****
????
```

```
*****
* Subpgm CLEAR_B
* turn off the bar graph display connected to PORT-B
*****
????
```

```
*****
* Subpgm LITE_C
* turn on the bar graph display one bit at a time connected to
* PORT-C
*****
????
```

```
*****
* Subpgm CLEAR_C
* turn off the bar graph display connected to PORT-C
*****
????
```

```
*****
* Subpgm LITE_D
* turn on the bar graph display one bit at a time connected to
```

```
* PORT-D  
*****  
????
```

```
*****  
* Subpgm CLEAR D  
* turn off the bar graph display connected to PORT-D  
*****  
????
```

```
*****  
* End of Pgm  
*****
```

```

*****
* Pgm:  string.asm
*
* Desc:  displays a string of characters on the LCD-II (HD44780).
*        The characters show from the right end of the screen; travel
*        to the left side; and disappear after the last character
*        shows up. The procedure will then repeat. The string is
*        "HOW NOW, BROWN COW?"
*
* Author: ZUYI CHEN (University of Montana)
*         (based on mtxdri.asm by Hong Fan of UM)
*
* Date:  June, 1992
*****

```

```

PORTC    equ    $1003        ; memory location of "port C"
PORTB    equ    $1004        ; memory location of "port B"
DDCR     equ    $1007        ; memory location of port C control
*                                     ; reg
PAUSE    equ    2
DCOUNT   equ    $E000        ; number of DLOOP iterations in DELAY

        org    $C000        ; pgm start on 68HC11 - EVB

```

```

*****
* MAIN
*****

```

```

        ldaa  #$FF          ; Set Port-C to output, by sending
        staa  DDCR          ; '1111 1111' to Port-C control reg
        jsr   INIT          ; initialization to HD44780

```

```

LOOP    jsr   ENTRY_MODE    ; set entry mode
        jsr   MV_CURSOR     ; move cursor to the right of
*                                     ; screen
        jsr   DISP_SHIFT    ; set mode to shift left when
*                                     ; display

```

```

*****
* Display the string "HOW NOW, BROWN COW?"
*****

```

```

        jsr   H              ; display H
        jsr   O              ; display O
        jsr   W              ; display W
        jsr   DISP_BLANK    ; display a blank
        jsr   N              ; display N
        jsr   O              ; display O
        jsr   W              ; display W
        jsr   COMMA         ; display ','
        jsr   DISP_BLANK    ; display a blank
        jsr   B              ; display B
        jsr   R              ; display R
        jsr   O              ; display O
        jsr   W              ; display W
        jsr   N              ; display N

```

```

jsr    DISP_BLANK    ; display a blank
jsr    C              ; display C
jsr    O              ; display O
jsr    W              ; display W
jsr    QUESTION_MARK ; display '?'
jsr    DISP_BLANK    ; display a blank

```

```

ldaa   #PAUSE
jsr    DELAY          ; wait a while
jsr    CLEAR          ; clear the screen
bra    LOOP           ; back and repeat

```

\*\*\*\*\*

\* Subpgm DELAY

\* Desc: "busy waits" for the number of DELAY loop iterations  
 \* specified by the value of register A at subpgm entry.

\*\*\*\*\*

DELAY

```

DLOOP  ldx    #DCOUNT    ; get number DLOOP iterations
        dex    ; decrement DLOOP count
        bne    DLOOP
        deca   ; decrement DELAY count
        bne    DELAY
        rts

```

\*\*\*\*\*

\* Subpgm: INIT

\* Desc: Initialization of the HD44780. This subpgm does the  
 \* following:

- \* 1. turn the display off
- \* 2. set the interface data length and number of  
 \* display lines
- \* 3. turn the display on

\*\*\*\*\*

INIT

```

ldab   #$00          ; set control mode
stab   PORTB

ldaa   #$01          ; clear display
staa   PORTC
ldaa   #PAUSE        ; delay
jsr    DELAY

jsr    FUNSET        ; function set to 8-bits and 1-line
jsr    DISPLAY_ON   ; turn on display
rts

```

\*\*\*\*\*

\* Subpgm: FUNSET

\* Desc: This subpgm does the following

- \* sets to 8-bit operation and selects 1-line display lines  
 \* and character fonts. ( Number of display lines and  
 \* character fonts cannot be changed hereafter. )

\*\*\*\*\*

## FUNSET

```

ldaa    #$30          ; function set
staa    PORTC
ldaa    #PAUSE       ; delay
jsr     DELAY
rts

```

```

*****
* Subpgm: turn display on
* Desc: This subpgm will turn the display on
*****

```

## DISPLAY\_ON

```

ldaa    #$0E          ; turn the display on
staa    PORTC
ldaa    #PAUSE       ; delay
jsr     DELAY
rts

```

```

*****
* Subpgm: Entry mode setting of the HD44780
* Desc: This subpgm sets the entry mode to increment the address
*       by one and to shift the display to the left at the time
*       of write to the DD/DG RAM
*****

```

## ENTRY\_MODE

```

ldab    #$00          ; set the to control mode
stab    PORTB
ldaa    #$06          ; set entry mode described above
staa    PORTC
ldaa    #PAUSE       ; delay
jsr     DELAY
rts

```

```

*****
* Subpgm: Clear display
* Desc: This subpgm clears the screen
*****

```

## CLEAR

```

mode    ldab    #$00          ; instruction for the control
        stab    PORTB
        ldaa    #$01          ; instruction for clear screen
        staa    PORTC
        ldaa    #PAUSE       ; delay
        jsr     DELAY
        rts

```

```

*****
* Subpgm: DISP_BLANK
* Desc: This subpgm displays a blank by shifting the display to
*       the left by one position
*****

```

## DISP\_BLANK

```

ldab    #$02          ; data write mode
stab    PORTB

```

```

        ldaa    #$14          ; shift display to left
        staa    PORTC
        ldab    #$01          ; set busy flag for internal
*                               ; operation
        stab    PORTB
        ldaa    #PAUSE        ; delay
        jsr    DELAY
        rts

```

```

*****
* Subpgm: MV_CURSOR
* Desc: moves the cursor the right of screen so that it display
*       from there
*****

```

MV\_CURSOR

```

        ldx     #$21          ; set the loop index
CLOOP   ldab    #$02          ; set to the data write mode
        stab    PORTB
        ldaa    #$14          ; instruction for the shift right
        staa    PORTC
        ldab    #$01          ; set busy flag for internal
operation
        stab    PORTB
        ldaa    #PAUSE        ; delay
        staa    DELAY
        dex
        bne    CLOOP          ; CLOOP
        rts

```

```

*****
* Subpgm: Set the mode to shift when display
* Desc: changes the mode the shift cursor when display
*****

```

DISP\_SHIFT

```

        ldab    #$00          ; set the mode to control mode
        stab    PORTB
        ldaa    #$07          ; instruction for mode describe
above
        staa    PORTC
        ldaa    #PAUSE        ; delay
        jsr    DELAY
        rts

```

```

*****
* Subpgm: OUTCHAR
* Desc: displays a character
*****

```

OUTCHAR

```

mode    ldab    #$02          ; set the mode to write data
        stab    PORTB
        staa    PORTC
        ldab    #$01          ; set busy flag for internal
*                               ; operation

```



```

stab    PORTB
ldaa    #PAUSE
jsr     DELAY      ; delay
rts

```

```
*****
```

```
* Subpgm: B
* Desc: display character 'B'

```

```
*****
```

```
B
      ldaa    #$42      ; character 'B'
      jsr     OUTCHAR
      rts

```

```
*****
```

```
* Subpgm: C
* Desc: display character 'C'

```

```
*****
```

```
C
      ldaa    #$43      ; character 'C'
      jsr     OUTCHAR
      rts

```

```
*****
```

```
* Subpgm: H
* Desc: display character 'H'

```

```
*****
```

```
????
```

```
*****
```

```
* Subpgm: N
* Desc: display character 'N'

```

```
*****
```

```
????
```

```
*****
```

```
* Subpgm: O
* Desc: display character 'O'

```

```
*****
```

```
????
```

```
*****
```

```
* Subpgm: R
* Desc: display character 'R'

```

```
*****
```

```
????
```

```
*****
```

```
* Subpgm: W
* Desc: display character 'W'

```

```
*****
```

```
????
```

\*\*\*\*\*  
\* Subpgm: COMMA  
\* Desc: display character ','  
\*\*\*\*\*  
???

\*\*\*\*\*  
\* Subpgm: QUESTION\_MARK  
\* Desc: display character '?'  
\*\*\*\*\*  
???

\*\*\*\*\*  
\* End of Pgm  
\*\*\*\*\*

```

*****
* Pgm: swi.asm
*
* Desc: use soft ware interrupt to mimic a modulo-9 counter. In
*       the interrupt service routine it increments the digit
*       to be displayed each second, and sends digit to a
*       7-segment LED from port-B. Software interrupt (SWI)
*       from interrupt vector jump table is used here.
*
* Author: ZUYI CHEN (University of Montana)
*
* Date: June, 1992
*****

PORTB      equ          $1004          ; memory location of "port B"
DCOUNT     equ          4000
PVSWI      equ          $00F4          ; pseudo vector address of SWI

          org          $C000          ; pgm starts on 68HC11 - EVB

*****
* MAIN
*****
          ldaa #7E          ; extended op code of jmp instruction
          staa PVSWI        ; pseudo vector for SWI
          ldx #INTERRUPT    ; put address of Interrupt Routine
          stx PVSWI+1      ; after the address of jmp
          jsr CLEAR        ; Clear the 7-segment display
          clr CUR_DIGIT    ; initialize CUR_DIGIT to 0

LOOP
          swi              ; software interrupt
          bra LOOP         ; interrupt driven from here

*****
* ISR: INTERRUPT
* Desc: increments the number to be displayed on the 7-segment
*       display. called at each interrupt
*****
INTERRUPT
          ldy #83          ; set number of DELAY
iterations
          jsr DELAY        ; wait
          jsr CLEAR        ; Clear the display
          ldab CUR_DIGIT   ; get CUR_DIGIT
          jsr OUT_DIGIT    ; display it on 7-segment
*
          incb             ; increment the digit
          cmpb #10         ; if digit larger than 9 then reset
          bne SKIP        ; to 0
          ldab #0
SKIP      stab CUR_DIGIT
          rti

```

\*\*\*\*\*

\* Subpgm: OUT\_DIGIT

\* Desc: Display the number on the 7-segment display

\*\*\*\*\*

OUT\_DIGIT

```

ldx      #DIGIT0      ; load the address of DIGIT0
abx      ; add value of reg B to index
*        ; reg X
ldaa     0,x          ; load digit/pin map
staa     PORTB        ; write to PORT B
rts

```

\*\*\*\*\*

\* Subpgm CLEAR

\* Desc: clear the 7-segment display

\*\*\*\*\*

CLEAR

```

ldaa     #$00        ; Clear 7-segment display
staa     PORTB
rts

```

\*\*\*\*\*

\* Subpgm DELAY

\* Desc: "busy wait" for the number of DELAY loop iterations

\* specified by the value of index register Y at

subprogram entry.

\*\*\*\*\*

DELAY

```

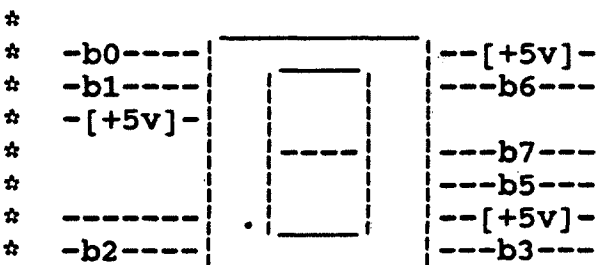
DLOOP    ldx      #DCOUNT      ; set DLOOP iterations
          dex      ; decrement index register X value
          bne     DLOOP        ; back to DLOOP if X value not 0
          dey     ; decrement index register Y count
          bne     DELAY        ; back to DELAY if Y count not 0
          rts      ; return to calling routine

```

\*\*\*\*\*

\* DATA: Table of digit/Port-B pin mapping

\*\*\*\*\*



\* Note: bit 4 of port-B is not used; [+5v] indicates connecting  
\* +5v pin to one of the 3 outlets annotated by [+5v]

-----  
\* Port-B bits 7 6 5 4 3 2 1 0  
\* -----

```

DIGIT0   fcb      $6F  ; 0 1 1 * 1 1 1 1
DIGIT1   fcb      $60  ; 0 1 1 * 0 0 0 0

```

DIGIT2	fc	\$CD	;	1	1	0	*	1	1	0	1
DIGIT3	fc	\$E9	;	1	1	1	*	1	0	0	1
DIGIT4	fc	\$E2	;	1	1	1	*	0	0	1	0
DIGIT5	fc	\$AB	;	1	0	1	*	1	0	1	1
DIGIT6	fc	\$AF	;	1	0	1	*	1	1	1	1
DIGIT7	fc	\$61	;	0	1	1	*	0	0	0	1
DIGIT8	fc	\$EF	;	1	1	1	*	1	1	1	1
DIGIT9	fc	\$E3	;	1	1	1	*	0	0	1	1

CUR\_DIGIT rmb 2

\*\*\*\*\*  
\* End of Pgm  
\*\*\*\*\*

```

*****
* Pgm: timeint.asm
*
* Desc: use interrupt mechanism to mimic a modulo-9 counter
*        incrementing digit each sec with a tic sound.
*        It reads the free-running counter, adds it to a delay time,
*        and stores the result to Output Compare Register 5 (TOC5).
*        The Output Compare Flag will be set when TOC5 value
*        equal to the value of free-running counter, and TOC5 is
*        interrupted. Since the cycle range of the free-running
*        counter is about 32 ms for the 2MHz CPU, a number of
*        iterations for OC5 interrupt is performed. The program
*        sends the digit to be displayed to a 7-segment display
*        from port-C and the tic sound to a piezo buzzer out of
*        pin 4 of port-C.
*
* Author: ZUYI CHEN (University of Montana)
*
* Date: June, 1992
*****
PVOC5    equ    $00D3    ; pseudo vector address of OC5
BASE     equ    $1000    ; base address of register block
PORTC    equ    $03     ; (offset from base address) port-C
DDRC     equ    $07     ; (offset) port-C control reg
TCNT     equ    $0E     ; (offset) free-running counter
TMSK1    equ    $22     ; (offset) timer interrupt mask
TFLG1    equ    $23     ; (offset) timer flag 1
TOC5     equ    $1E     ; (offset) output compare register 5

        org    $C000    ; pgm start on 68HC11 - EVB
        jsr    INIT     ; initialize the interrupt

*****
* MAIN
*****
DIG_LOOP
        ldy    #????    ; number of TLP iterations
*
*        ; mapping 1 sec
TLP     brclr  TFLG1,X $08 * ; wait until the interrupt comes
        dey    ; decrement OC5 interrupt
*
*        ; iteration
        bne    TLP
        jsr    CLEAR    ; Clear the display
        ldab  CUR_DIGIT ; get CUR_DIGIT
        jsr    OUT_DIGIT ; display it on 7-segment
*
*        ; display
        incb  ; increment the digit
        cmpb  #10      ; if digit larger than 9 then
*
*        ; reset
        bne    SKIP    ; to 0
        ldab  #0
SKIP    stab   CUR_DIGIT ; store to memory
        bra   DIG_LOOP

```

\*\*\*\*\*

\* Subpgm: INIT

\* Desc: initialize digit to be displayed and OC5 interrupt

\*\*\*\*\*

INIT

```

      clr          CUR_DIGIT      ; initialize the digit to be
*                                     ; displayed
      ldaa         #$7E           ; get extended op code for jump
      staa        PVOC5          ; and store to pseudo vector OC5
      ldx         #INTERRUPT      ; get address of Interrupt
*                                     ; Routine
      stx         PVOC5+1        ; and store after jump in vector
*                                     ; table
      ldx         #BASE          ; get base address of the
*                                     ; register block
      ldaa        #$FF           ;
      staa        DDRC,X         ; set port-C for output only
      ldaa        #$08           ; get OC5F bit
      staa        TFLG1,X        ; set OC5F bit of timer flag
      staa        TMSK1,X        ; enable OC5F interrupt
      cli         ; enable interrupts
      rts

```

\*\*\*\*\*

\* ISR: INTERRUPT

\* Desc: get the current free-running counter; add 4000 cycles,

\* and store to OC5. Interrupt comes when the value in OC5

\* equals to the free-running counter

\*\*\*\*\*

INTERRUPT

```

      ldd         TCNT,X          ; get free-running counter
      add         #$A000         ; add $A000 cycles
      std         TOC5,X         ; store to Output Compare
*                                     ; Register 2
      bclr        TFLG1,X $F7    ; clear the OC2F bit for next
*                                     ; use
      rti

```

\*\*\*\*\*

\* Subpgm: OUT\_DIGIT

\* Desc: Display the number on the 7-segment display

\*\*\*\*\*

OUT\_DIGIT

```

      pshx         ; push register X contents to stack
      ldx         #DIGITO      ; load the address of DIGITO
      abx         ; add value of reg B to index reg X
      ldaa        0,X         ; load digit/pin map
      pulx        ; pop register X contents off stack
      staa        PORTC,X     ; write to PORT C
      rts

```

\*\*\*\*\*

\* Subpgm CLEAR

\* Desc: clear the 7-segment display





```

*****
* Pgm: timepoll.asm
*
* Desc: use polling mechanism to mimic a modulo-9 counter.
* It reads the free-running counter, adds it to a delay time,
* and stores the result to Output Compare Register 2 (TOC2).
* The Output Compare Flag will be set when TOC2 value
* equal to the value of free-running counter. Since the
* cycle range of the free-running counter is about 32 ms
* for the 2MHz MCU, a number of iterations for OC2 is
* performed. The program increments the digit to be displayed
* each second, and sends digit to a 7-segment LED from port-C.
*
* Author: ZUYI CHEN (University of Montana)
*
* Date: June, 1992
*****

```

```

BASE      equ      $1000      ; base address of register block
PORTC     equ      $03        ; (offset from base address) port-C
DDRC      equ      $07        ; (offset) port-C control reg
TCNT      equ      $0E        ; (offset) free-running counter
TFLG1     equ      $23        ; (offset) timer flag 1
TOC2      equ      $18        ; (offset) output compare register 2

          org      $C000      ; pgm start on 68HC11 - EVB

```

```

*****
* MAIN
*****

```

```

          ldx      #BASE      ; get base address of the register
*          ; block
          ldaa     #$FF
          staa     DDRC,X     ; set port-C for output only
          clr      CUR_DIGIT  ; initialize the digit to be
*          ; displayed

          ldaa     #$40       ; get OC2F bit
          staa     TFLG1,X    ; set OC2F bit of timer flag

DIG_LOOP
          jsr     CLEAR      ; Clear the display
          ldab    CUR_DIGIT  ; get CUR_DIGIT
          jsr     OUT_DIGIT  ; display it on 7-segment display
          incb   ; increment the digit
          cmpb   #10        ; if digit larger than 9 then reset
          bne   SKIP        ; to 0
          ldab   #0
SKIP     stab    CUR_DIGIT  ; store to memory

```

```

*****
* POLLING the free-running counter
*****
        ldy      #????      ; number of T_LOOP iterations mapping
*
*                               ; 1 sec
T_LOOP
        ldd      TCNT,X      ; get free-running counter
        add     #4000        ; add 4000 cycles
        std     TOC2,X      ; store to Output Compare Register 2
        brclr   TFLG1,X $40 * ; wait for output compare
        bclr    TFLG1,X $BF   ; clear the OC2F bit for next
*
*                               ; use
        dey     ; decrement count of OC2
*
*                               ; iteration
        bne     T_LOOP
        bra     DIG_LOOP     ; start all over

```

```

*****
* Subpgm: OUT_DIGIT
* Desc: Display the number on the 7-segment display
*****
OUT_DIGIT
        pshx     ; push register X contents to stack
        ldx     #DIGITO    ; load the address of DIGITO
        abx     ; add value of reg B to index reg X
        ldaa   0,X        ; load digit/pin map
        pulx     ; pop register X contents off stack
        staa   PORTC,X    ; write to PORT C
        rts

```

```

*****
* Subpgm CLEAR
* Desc: clear the 7-segment display
*****
CLEAR
        ldaa   #$00      ; Clear 7-segment display
        staa   PORTC,X
        rts

```

```

*****
* DATA: Table of digit/Port-C pin mapping
*****
*
*
*  -b0-----|      |-----[+5v]-
*  -b1-----|      |-----b6---
*  -[+5v]-   |      |
*            |      |-----b7---
*            |      |-----b5---
*            |      |-----[+5v]-
*  -b2-----|      |-----b3---
*
*

```

```

* Note: bit 4 of port-C is connected to a piezo buzzer, and is
*       used to generate tic sound;
*       [+5v] indicates connecting +5v pin to one of

```

\* the 3 outlets specified by [+5v]

\*-----

\* Port-C bits 7 6 5 4 3 2 1 0

\*-----

DIGIT0	fc	\$7F	;	0	1	1	1	1	1	1	1
DIGIT1	fc	\$70	;	0	1	1	1	0	0	0	0
DIGIT2	fc	\$DD	;	1	1	0	1	1	1	0	1
DIGIT3	fc	\$F9	;	1	1	1	1	1	0	0	1
DIGIT4	fc	\$F2	;	1	1	1	1	0	0	1	0
DIGIT5	fc	\$BB	;	1	0	1	1	1	0	1	1
DIGIT6	fc	\$BF	;	1	0	1	1	1	1	1	1
DIGIT7	fc	\$71	;	0	1	1	1	0	0	0	1
DIGIT8	fc	\$FF	;	1	1	1	1	1	1	1	1
DIGIT9	fc	\$F3	;	1	1	1	1	0	0	1	1

CUR\_DIGIT rmb 2

\*\*\*\*\*

\* End of Pgm

\*\*\*\*\*

```

*****
* Pgm trafficB.asm
*
* Desc: Implement a simple LED display driver for 68HC11 using
*       output PORT-B. The LED display mimics a traffic signal
*       with following order:
*       a. 10-sec GREEN light on only
*       b. 1-sec YELLOW light on only, then 1-sec lights off
*       c. do (b) another 2 times
*       d. 10-sec RED light on only
*       e. go back to (a)
*
* Author: ZUYI CHEN (University of Montana)
* (based on lightsC.asm by Dr. Ray Ford of UM)
*
* Date: June, 1992
*****

PORTB      equ          $1004          ; memory location of "port B"
DCOUNT     equ          $C000          ; iterations of DLOOP loop in
DELAY

                org          $C000          ; pgm start on 68HC11 - EVB

*****
* MAIN
*****
        jsr          CLEAR_LED          ; clear to start

LOOP
        jsr          GREEN              ; turn on green light only
        ldy          #????              ; set delay value 10 sec for green
light
        jsr          DELAY              ; wait -- green

*****
        ldab #3                          ; set value to blink yellow light 3
                                        ; times

BLINK_LOOP
        jsr          YELLOW             ; turn on yellow light only
        ldy          #????              ; set delay value 1 sec for yellow
light
        jsr          DELAY              ; wait -- yellow
        jsr          CLEAR_LED          ; clear yellow
        ldy          #????              ; set delay value 1 sec for "clear"
        jsr          DELAY              ; wait -- "clear"
        decb                          ; decrement the value in reg B
        bne BLINK_LOOP                  ; back to BLINK_LOOP if not 0 in reg
*                                        ; B

        jsr          RED                ; turn on red light only
        ldy          #????              ; set delay value 10 sec for red
light
        jsr          DELAY              ; wait -- red

```

```
bra      LOOP      ; go back to LOOP
```

```
*****
```

```
* Subpgm CLEAR_LED
```

```
* Desc: Send bits '0000 0000' to "port B"
```

```
*****
```

```
CLEAR_LED
```

```
ldaa #00          ; load 0 to register A
staa PORTB       ; send '0000 0000' to PORT B
rts              ; return to calling routine
```

```
*****
```

```
* Subpgm DELAY
```

```
* Desc: "busy wait" for the number of DELAY loop iterations
```

```
* specified by the value of index register Y at subroutine
```

```
* entry. Note the number DLOOP iterations is hard-coded
```

```
*****
```

```
DELAY
```

```
ldx      #DCOUNT      ; load the value of DELAY COUNT
DLOOP    dex          ; decrement index register X value
          bne         DLOOP      ; back to DLOOP if X value not 0
          dey         ; decrement index register Y count
          bne         DELAY      ; back to DELAY if Y count not 0
          rts          ; return to calling routine
```

```
*****
```

```
* Subpgm GREEN
```

```
* Desc: turn on the GREEN LED, assumed to be connected
```

```
* to "port B" pin 0
```

```
*****
```

```
GREEN
```

```
ldaa #01          ; GREEN mapped to pin 0
staa PORTB       ; send '0000 0001' to PORT B
rts
```

```
*****
```

```
* Subpgm YELLOW
```

```
* Desc: turn on the YELLOW LED, assumed to be connected
```

```
* to "port B" pin 1
```

```
*****
```

```
YELLOW
```

```
ldaa #02          ; YELLOW mapped to pin 1
staa PORTB       ; send '0000 0010' to PORT B
rts
```

```
*****
```

```
* Subpgm RED
```

```
* Desc: Light the RED LED, assumed to be connected
```

```
* to "port B" pin 2
```

```
*****
```

```
RED
```

```
ldaa #04          ; RED mapped to pin 2
staa PORTB       ; send '0000 0100' to PORT B
```

rts

```
*****  
* End of Pgm  
*****
```

\*\*\*\*\*

\* Pgm trafficC.asm

\*

\* Desc: Implement a simple LED display driver for 68HC11 using  
\* output PORT-C. The LED display mimics a traffic signal  
\* with following order:

\* a. 10-sec GREEN light on only  
\* b. 1-sec YELLOW light on only, then 1-sec lights off  
\* c. do (b) another 2 times  
\* d. 10-sec RED light on only  
\* e. go back to (a)

\*

\* Author: ZUYI CHEN (University of Montana)  
\* (based on lightsC.asm by Dr. Ray Ford of UM)

\*

\* Date: June, 1992

\*\*\*\*\*

```
PORTC      equ      $1003      ; memory location of "port C"
DDRC       equ      $1007      ; PORT C control register
DCOUNT     equ      $C000      ; iterations of DLOOP loop in
DELAY
```

```
org        $C000      ; pgm start on 68HC11 - EVB
```

\*\*\*\*\*

\* MAIN

\*\*\*\*\*

```
ldaa #$FF      ; load register A with '1111 1111'
staa DDRC      ; initialize PORT C for output only
jsr CLEAR_LED  ; clear to start
```

LOOP

```
jsr GREEN      ; turn on green light only
ldy #????      ; delay value 10 sec for green LED
jsr DELAY      ; wait -- green
```

\*\*\*\*\*

```
ldab #3        ; value to blink yellow light 3 times
BLINK_LOOP
```

```
jsr YELLOW     ; turn on yellow light only
ldy #????      ; delay value 1 sec for yellow LED
jsr DELAY      ; wait -- yellow
jsr CLEAR_LED  ; clear yellow
ldy #????      ; set delay value 1 sec for "clear"
jsr DELAY      ; wait -- "clear"
decb           ; decrement the value in reg B
bne BLINK_LOOP ; back to BLINK_LOOP if reg B not 0
jsr RED        ; turn on red light only
ldy #????      ; delay value 10 sec for red LED
jsr DELAY      ; wait -- red
```

```
bra LOOP      ; go back to LOOP
```

\*\*\*\*\*

\* Subpgm CLEAR\_LED

\* Desc: Send bits '0000 0000' to "port C"

\*\*\*\*\*

CLEAR\_LED

```
    ldaa #00          ; load 0 to register A
    staa PORTC       ; send '0000 0000' to PORT C
    rts              ; return to calling routine
```

\*\*\*\*\*

\*

\* Subpgm DELAY

\* Desc: "busy wait" for the number of DELAY loop iterations

\* specified by the value of index register Y at subprogram

\* entry. Note the number DLOOP iterations is hard-coded

\*\*\*\*\*

DELAY

```
    ldx          #DCOUNT      ; load the value of DELAY COUNT
```

DLOOP

```
    dex          ; decrement index register X value
    bne          DLOOP       ; back to DLOOP if X value not 0
    dey         ; decrement index register Y count
    bne          DELAY      ; back to DELAY if Y count not 0
    rts         ; return to calling routine
```

\*\*\*\*\*

\* Subpgm GREEN

\* Desc: turn on the GREEN LED, assumed to be connected

\* to "port C" pin 0

\*\*\*\*\*

GREEN

```
    ldaa #01          ; GREEN mapped to pin 0
    staa PORTC       ; send '0000 0001' to PORT C
    rts
```

\*\*\*\*\*

\* Subpgm YELLOW

\* Desc: turn on the YELLOW LED, assumed to be connected

\* to "port C" pin 1

\*\*\*\*\*

YELLOW

```
    ldaa #02          ; YELLOW mapped to pin 1
    staa PORTC       ; send '0000 0010' to PORT C
    rts
```

\*\*\*\*\*

\* Subpgm RED

\* Desc: Light the RED LED, assumed to be connected

\* to "port C" pin 2

\*\*\*\*\*

RED

```
    ldaa #04          ; RED mapped to pin 2
    staa PORTC       ; send '0000 0100' to PORT C
    rts
```



\*\*\*\*\*  
\* End of Pgm  
\*\*\*\*\*

```

*****
* PGM: travelb.asm
*
* DESC: Program sends a signal to the output pins of the PORT B
*       one at a time with 1/2 second delay in the following order:
*       a. pin 7 on only '1000 0000'
*       b. pin 6 on only '0100 0000'
*       c. pin 5 on only '0010 0000'
*       d. pin 4 on only '0001 0000'
*       e. pin 3 on only '0000 1000'
*       f. pin 2 on only '0000 0100'
*       g. pin 1 on only '0000 0010'
*       h. pin 0 on only '0000 0001'
*       i. go back to (a)
*       port B: general purpose output lines
*               pins 42-35 correspond to bits 0-7
*
* Author: ZUYI CHEN (University of Montana)
* Date:   June, 1992
*****

DELAY_COUNT    equ        ???? ; mapping to the nearest to 1/2
*               ; second
DLOOP_COUNT    equ        4000 ; set value for the DLOOP in DELAY
PORTB          equ        $1004 ; memory location for port B

                org        $c000

*****
* MAIN
*****

PIN_POS        ldaa #8          ; set value for the # of pins of PORT B
                ldab #$80       ; load '1000 0000' to register B
LOOP          stab PORTB       ; send '1000 0000' to PORT B
                ldy #DELAY_COUNT ; load value of specified by
DELAY_COUNT    jsr            DELAY ; wait
                ????           ; shift right by 1 bit with
*               ; previous bit turned off
                deca           ; decrement register A value
                bne            LOOP ; go to LOOP if pin 0 hasn't been
*               ; reached
                bra            PIN_POS

*****
* Subpgm DELAY
* Desc: "busy wait" for the number of DELAY loop iterations
*       specified by the value of index register Y at subprogram
entry.
*****
DELAY
DLOOP         ldx #DLOOP_COUNT ; hardcoded unit cycle delay
                dex           ; decrement index register X value

```

```
    bne DLOOP          ; back to DLOOP if X value not 0
    dey               ; decrement index register Y count
    bne DELAY         ; back to DELAY if Y count not 0
    rts               ; return to calling routine
```

```
*****
* END OF PROGRAM
*****
```

```

*****
* PGM: travelc.asm
*
* DESC: Program sends a signal to the output pins of the PORT C
*       one at a time with 1/2 second delay in the following order:
*       a. pin 7 on only '1000 0000'
*       b. pin 6 on only '0100 0000'
*       c. pin 5 on only '0010 0000'
*       d. pin 4 on only '0001 0000'
*       e. pin 3 on only '0000 1000'
*       f. pin 2 on only '0000 0100'
*       g. pin 1 on only '0000 0010'
*       h. pin 0 on only '0000 0001'
*       i. go back to (a)
*       port C: general purpose i/o lines
*               pins 9-16 correspond to bits 0-7
*
* Author: ZUYI CHEN (University of Montana)
* Date:   June, 1992
*****

```

```

DELAY_COUNT    equ    ????    ; mapping to the nearest to 1/2
second
DLOOP_COUNT    equ    4000    ; set value for the DLOOP in DELAY
DDRC           equ    $1007   ; port C I/O control register
PORTC          equ    $1003   ; memory location for port C

```

```

org    $c000

```

```

ldaa    #$FF    ; send '1111 1111' to DDRC to
staa    DDRC    ; set PORTC output only

```

```

*****
* MAIN
*****

```

```

PIN_POS    ldaa    #8        ; set value for the # of pins of PORT C
           ldab    #$80      ; load '1000 0000' to register B
LOOP       stab    PORTC     ; send '1000 0000' to PORT C
           ldy    #DELAY_COUNT ; load value of specified by
*           ; DELAY_COUNT
           jsr    DELAY      ; wait
           ????           ; shift right by 1 bit with
*           ; previous bit turned off
           deca           ; decrement register A value
           bne    LOOP      ; go to LOOP if pin 0 hasn't been
*           ; reached
           bra    PIN_POS

```

```
*****
* Subpgm DELAY
* Desc: "busy wait" for the number of DELAY loop iterations
*      specified by the value of index register Y at subprogram
*
*      entry.
*****
DELAY
DLOOP      ldx    #DLOOP_COUNT    ; hardcoded unit cycle delay
           dex    ; decrement index register X value
           bne   DLOOP           ; back to DLOOP if X value not 0
           dey    ; decrement index register Y count
           bne   DELAY          ; back to DELAY if Y count not 0
           rts    ; return to calling routine

*****
* END OF PROGRAM
*****
```