

**EE 368**  
**Microprocessor Systems and Interfacing**  
**Laboratory**  
**Fall 2010 Edition**

**Department of Electrical  
and Computer Engineering  
University of South Alabama**

## **1. Engineering Ethics and Academic Honesty**

The Department of Electrical and Computer Engineering has the obligation to instill a strong sense of ethics in its students. Students are expected to accept the responsibility for honest behavior regarding all work submitted for a grade, and for assisting faculty with enforcement of rules pertaining to eliminate dishonest behavior. Each student is expected to support the standard of academic honesty by neither giving nor accepting assistance on tests, and by submitting only his/her own work for credit. Violations of the academic honesty will result in a disciplinary action that may result in dismissal from the school.

## **2. General Laboratory Policy**

All students that use the laboratory must cooperate to maintain the facility in its best possible form. We all need to encourage and enforce professional standards to insure that the laboratory remains a productive environment. All equipments and systems must be used in a careful and thoughtful manner to insure their integrity. Any equipment in need of repair should be reported immediately, so that it can be expeditiously brought back to a useable condition. Eating, drinking, and smoking are not permitted in the laboratory. All disks brought into the laboratory must be checked for computer viruses before executing any of the programs stored on the disks.

Each laboratory session is intended to provide a discovery and learning experience. It is essential that students come prepared to make the best use of the limited laboratory time. Every student must be thoroughly prepared for each laboratory project to insure that the laboratory session will be productive, efficient, and professional. The laboratory instructor may ask questions about the laboratory assignment to insure that each student is adequately prepared for the laboratory.

Laboratory attendance at the assigned laboratory period is required. Each group must submit their laboratory report on the due date at the beginning of the laboratory period. If a student misses a laboratory, his laboratory report will not be accepted.

## **3. Laboratory Safety**

Whenever electricity is used in an experiment, some danger exists. This should always be kept in mind. Most of the experiments in this laboratory will use only low voltages, which are not inherently dangerous. However, used equipments are powered by a high voltage. In addition, it is possible to incorrectly wire almost any experiment, which may lead to dangerous voltage levels. The careless use of electricity can hurt you and the equipments. Try to understand what you are asked to do. Consult your instructor if necessary. Never turn power on until you are confident that everything is safe.

To protect yourself, always treat electricity with respect. Don't handle hot lead wires. Don't leave wires dangling around in space. Try to use one hand at a time. This hopefully prevents the flow of electricity from one hand to the other through your body, which potentially causes the heart to stop. Keep in mind that when you touch an electrified wire, the electricity will always try to flow, and it is to your advantage to keep it from flowing through your body. Don't hold electrified wires, and make sure that no part of your body inadvertently comes into contact with a wire.

To protect equipment, make sure that all of the hookups between power supplies, oscilloscopes, multimeters, light emitting diodes (LEDs), bread boards, integrated circuits (ICs), etc., are as you want them to be. You should double check all wiring prior to turning power on. One way to destroy an IC is to reverse power leads. An abused IC may start smoking, or may become so hot to touch. While probing a circuit with a test lead, make

sure that you are making a good contact with the exact points you wish to test, and only those points.

When using a particular piece of test equipment, understand its limitations. Don't try to use it to test something it wasn't designed to test. Read the equipment manual whenever necessary.

In summary, the best way to keep yourself and the equipment from being damaged is to do everything slowly, cautiously, and carefully. In the event that there is an emergency situation, call campus security.

#### 4. Laboratory Report Format

Each group is required to write an informal laboratory report after successfully demonstrating a working laboratory project to the instructor according to the attached schedule. All laboratory reports must be written using word processing program, and all figures must be drawn using a CAD program or a drawing program on white 8.5" x 11" paper. The laboratory report must be neat, clean, orderly, homogenous, original, and with correct spelling and grammar.

Your laboratory report must include the following sections:

- **Title page:** The title page must include the course number, the experiment number and its title, your name(s) and group number, and due date.
- **Objectives:** The objectives section should include the purpose of the experiment.
- **Diagrams and Charts:** The procedure describes the steps taken to complete the experiment. It should also include all details of your experiment, including description of the program developed.
  - **Flow Chart or Logic Flow Diagram**
  - **Assembly-language File**
  - **List File**
  - **Screen Capture of Input / Output**
- **Comments on the Experiment:** Explain "how it went". What problems did you encounter? How did you solve them?
- **Results and Conclusions:** The results and conclusion section should include the results obtained from the experiment along with simulation results and any relevant discussion. It may include what you have learned from this experiment.

#### 5. General Grading Policy

Students must successfully complete all the laboratory assignments. A laboratory assignment is complete when it functions according to the given specifications, and a report is submitted (if required) for the corresponding laboratory assignment. No credit will be given for an incomplete laboratory assignment. Grading will be based on successful completion of laboratory assignment, originality of work, documentation and efficient programming, overall neatness and clarity of report (if required). Late laboratory reports will receive deductions. Plagiarism will not be tolerated, and may result in a disciplinary action that may result in dismissal from the school.

Partial credit is a matter of professional judgment and is not subject to negotiation.

**EE 368 - Microprocessor Systems and Interfacing Laboratory**  
**Experiment 1**  
**Motorola M68HC11EVB and AS11 Familiarization**

- **This experiment is included to give the student experience with the laboratory equipment and software tools. It will not be graded and a lab report is not necessary.**
- **Objective:** To learn to use the Motorola M68HC11EVB Evaluation Board, the AS11 Assembler on a PC, and the Hyper Terminal software package to enter and run M68HC11 assembly language programs.
- **Required Equipment:** The Motorola M68HC11EVB Evaluation Board with attached power supply and a 3.5" diskette
- **Procedure:**
  1. Review the M68HC11EVB Evaluation Board User's Manual, and familiarize yourself with the memory map and the monitor program (BUFFALO). Review the M68HC11, HCMOS Single-Chip Microcontroller data book. Refer to HC11: M68HC11 Reference Manual for more detailed information.
  2. Familiarize yourself with the editor and serial communications software package (Hyper terminal) available on the PC stations in the lab, as well as the HC11 assembler AS11.
  3. Enter and cross-assemble the included programs using the AS11 cross-assembler. It is strongly recommended that, rather than cutting and pasting the source code, you type it in by hand. This will let you develop familiarity with how to space and format assembly-language files, and will provide an opportunity to review the 68HC11 instruction set and how it is used.
  4. Download and run your programs on the MC68HC11EVB board and have your instructor check their operation.

- **Notes on Running Wookie:**

Wookie is a simple 68HC11 emulator, but it is different from the boards we will be using in several important ways. First, Wookie does not emulate a serial port, so you will have to "pretend" that Wookie is using it. Second, Wookie does not implement the built-in functions that you use in Buffalo to access the serial port.

So to use Wookie to run a program intended for Buffalo, you will need to do the following:

1. At the beginning of the Buffalo file are a series of "EQU" statements that mark places to call functions. For each of these statements, do an ORG to that location and put in an RTS statement. This places a subroutine return every place that the Buffalo code tries to call a subroutine, which returns execution back to your program.
2. Every time you call "inchar" the Buffalo program waits for input and returns a character in the A register. I emulated this by putting \$41 (ASCII for capital A) into register A and then doing the RTS. This emulates the user typing capital A and causes the program to behave normally.
3. When you launch Wookie, you will need to do the following.
  - a. Put Wookie into Rug Warrior Extended mode with a start address of 8000.
  - b. Use the AS11M format for the LST file.
4. Run Wookie single-step to watch what the program is doing in detail. You will see it call functions to send strings to the screen, for example, and then execute the functions to receive the characters.
5. Wookie comes with a different assembler than the one used in the lab. Do not use the Wookie assembler (asm11). Use the one for the lab (AS11).
6. You will probably find it helpful to copy the batch files and the AS11 assembler into the Wookie directory.

```

*****
* EE368 PRACTICE EXERCISE #1A *
* Program to get two bytes from terminal, multiply them, and *
* display resulting two bytes to terminal. *
*****

```

```

OUT1BYT EQU $FFB8 ADDRESSES MAY VARY BY BOARD
OUTCRLF EQU $FFC4
INCHAR EQU $FFCD
OUTSTRG EQU $FFC7
ORG $8800
RESHI FCB $00
RESLO FCB $00
MLTPLR FCB $00
MLTPND FCB $00
TEMP FCB $00
ORG $8000
MULT LDX #MSG
      JSR OUTSTRG
      JSR OUTCRLF OUTPUT CARRIAGE RETURN AND LINE FEED
      JSR GET1BYT GET MULTIPLIER BYTE AND STORE IT
      STAA MLTPLR
      LDX #MSG
      JSR OUTSTRG
      JSR OUTCRLF OUTPUT A CARRIAGE RETURN AND LINE FEED
      JSR GET1BYT GET MULTIPLICAND BYTE AND STORE IT
      STAA MLTPND
      JSR OUTCRLF OUTPUT A CARRIAGE RETURN AND LINE FEED
      LDAA MLTPLR LOAD A WITH MULTIPLIER
      LDAB MLTPND LOAD B WITH MULTIPLICAND
      MUL MULTILPY A X B => D
      STAA RESHI SAVE RESULT HIGH AND LOW
      STAB RESLO
      LDX #RESHI OUTPUT RESULTS HIGH AND LOW
      JSR OUT1BYT PUTPUT RESULTS HIGH AND LOW
      JSR OUT1BYT
      JSR OUTCRLF OUTPUT CARRIAGE RETUN AND LINE FEED
      BRA MULT LOOP BACK FOR ANOTHER PASS

```

```

***** SUBROUTINE GET1BYT *****
* Subroutine to get a byte from terminal *
*****

```

```

GET1BYT JSR GET1NIB GET FIRST NIBBLE IN A
        LSLA MOVE IT TO UPPER NIBBLE POSITION
        LSLA
        LSLA
        LSLA
        STAA TEMP SAVE IT IN A TEMPORARY MEMORY
        JSR GET1NIB GET SESOND NIBBLE IN A
        ORAA TEMP COMBINE IT WITH FIRST NIBBLE
        RTS

```

```

***** SUBROUTINE GET1NIB *****
* Subroutine to get a nibble from terminal *
*****

```

```

GET1NIB JSR INCHAR GET AN ASCII CHARACTER FROM TERMINAL
        SUBA #$30 REMOVE ASCII BIAS
        CMPA #$09
        BLS EXIT SKIP AROUND IF A 0-9 DIGIT
        ANDA #$DF MAKE IT CASE INSENSITIVE
        SUBA #$07
EXIT RTS RETURN
MSG FCC 'PLEASE ENTER A 2-DIGIT HEXADECIMAL NUMBER'
   FCB $04

```

```

*****
* EE368 PRACTICE EXERCISE #1B *
* PROGRAM TO GENERATE RANDOM NUMBERS AND DISPLAY THEM *
*****
OUTLHLF EQU $FFB2
OUTRHLF EQU $FFB5
OUTCRLF EQU $FFC4
*
ORG $8800
TABLE RMB 5
*
ORG $8000          PROG STARTS IN EVB RAM AT $8000
START LDY #$0008   INITILAIZE A COUNTER FOR 8 TIMES
LOOP JSR RND       GO TO GENERATE A RANDOM NUMBER
      LDAA TABLE  GET THAT RANDOM NUMVER IN A
      JSR OUTLHLF   DISPLAY ITS LEFT NIBBLE
      LDAA TABLE  GET THAT RANDOM NUMVER IN A
      JSR OUTRHLF   DISPLAY ITS RIGHT NIBBLE D
      JSR OUTCRLF   DISPLAY A CARRIAGE RETURN AND LINE FEED
      DEY          DECREMENT LOOP COUNTER
      BNE LOOP     LOOP BACK IF NOT DONE YET
HALT  BRA HALT     ELSE, HALT (ENDLESS LOOP)
***** SUBROUTINE RND *****
* Subroutine to generate a random number in TABLE *
*****
RND LDAA TABLE+1
    CLC
    ADCA TABLE+2
    SEC
    ADCA TABLE+4
    STAA TABLE
    LDX #TABLE
MOVE LDAA 0,X
    STAA 1,X
    INX
    CPX #TABLE+5
    BNE MOVE
    RTS          RETURN TO CALLER

```

### Modified Lab 1A for running in Wookie:

```

OUT1BYT EQU $FFBB          ADDRESSES MAY VARY BY BOARD
OUTCRLF EQU $FFC4
INCHAR EQU $FFCD
OUTSTRG EQU $FFC7
* SET OF SUBROUTINE RETURNS TO EMULATE BUFFALO
* OUTPUTS ARE JUST RTS INSTRUCTIONS
* INCHAR PUTS CAPITAL A INTO REGISTER A
* REMOVE THESE BEFORE COMPILING WITH A BOARD RUNNING BUFFALO
    ORG $FFBB
    RTS
    ORG $FFC4
    RTS
    ORG $FFCD
    LDAA #$41
    RTS
    ORG $FFC7
    RTS
    ORG $8800
* END OF EMULATION SECTION TO BE REMOVED...
RESHI FCB $00          THE REST OF THE PROGRAM IS UNMODIFIED

```

## ASCII Table

Char	Dec	Oct	Hex	Char	Dec	Oct	Hex	Char	Dec	Oct	Hex	Char	Dec	Oct	Hex
(nul)	0	0000	0x00	(sp)	32	0040	0x20	@	64	0100	0x40	`	96	0140	0x60
(soh)	1	0001	0x01	!	33	0041	0x21	A	65	0101	0x41	a	97	0141	0x61
(stx)	2	0002	0x02	"	34	0042	0x22	B	66	0102	0x42	b	98	0142	0x62
(etx)	3	0003	0x03	#	35	0043	0x23	C	67	0103	0x43	c	99	0143	0x63
(eot)	4	0004	0x04	\$	36	0044	0x24	D	68	0104	0x44	d	100	0144	0x64
(enq)	5	0005	0x05	%	37	0045	0x25	E	69	0105	0x45	e	101	0145	0x65
(ack)	6	0006	0x06	&	38	0046	0x26	F	70	0106	0x46	f	102	0146	0x66
(bel)	7	0007	0x07	'	39	0047	0x27	G	71	0107	0x47	g	103	0147	0x67
(bs)	8	0010	0x08	(	40	0050	0x28	H	72	0110	0x48	h	104	0150	0x68
(ht)	9	0011	0x09	)	41	0051	0x29	I	73	0111	0x49	i	105	0151	0x69
(lf)	10	0012	0x0a	*	42	0052	0x2a	J	74	0112	0x4a	j	106	0152	0x6a
(vt)	11	0013	0x0b	+	43	0053	0x2b	K	75	0113	0x4b	k	107	0153	0x6b
(ff)	12	0014	0x0c	,	44	0054	0x2c	L	76	0114	0x4c	l	108	0154	0x6c
(cr)	13	0015	0x0d	-	45	0055	0x2d	M	77	0115	0x4d	m	109	0155	0x6d
(so)	14	0016	0x0e	.	46	0056	0x2e	N	78	0116	0x4e	n	110	0156	0x6e
(si)	15	0017	0x0f	/	47	0057	0x2f	O	79	0117	0x4f	o	111	0157	0x6f
(dle)	16	0020	0x10	0	48	0060	0x30	P	80	0120	0x50	p	112	0160	0x70
(dc1)	17	0021	0x11	1	49	0061	0x31	Q	81	0121	0x51	q	113	0161	0x71
(dc2)	18	0022	0x12	2	50	0062	0x32	R	82	0122	0x52	r	114	0162	0x72
(dc3)	19	0023	0x13	3	51	0063	0x33	S	83	0123	0x53	s	115	0163	0x73
(dc4)	20	0024	0x14	4	52	0064	0x34	T	84	0124	0x54	t	116	0164	0x74
(nak)	21	0025	0x15	5	53	0065	0x35	U	85	0125	0x55	u	117	0165	0x75
(syn)	22	0026	0x16	6	54	0066	0x36	V	86	0126	0x56	v	118	0166	0x76
(etb)	23	0027	0x17	7	55	0067	0x37	W	87	0127	0x57	w	119	0167	0x77
(can)	24	0030	0x18	8	56	0070	0x38	X	88	0130	0x58	x	120	0170	0x78
(em)	25	0031	0x19	9	57	0071	0x39	Y	89	0131	0x59	y	121	0171	0x79
(sub)	26	0032	0x1a	:	58	0072	0x3a	Z	90	0132	0x5a	z	122	0172	0x7a
(esc)	27	0033	0x1b	;	59	0073	0x3b	[	91	0133	0x5b	{	123	0173	0x7b
(fs)	28	0034	0x1c	<	60	0074	0x3c	\	92	0134	0x5c		124	0174	0x7c
(gs)	29	0035	0x1d	=	61	0075	0x3d	]	93	0135	0x5d	}	125	0175	0x7d
(rs)	30	0036	0x1e	>	62	0076	0x3e	^	94	0136	0x5e	~	126	0176	0x7e
(us)	31	0037	0x1f	?	63	0077	0x3f	_	95	0137	0x5f	(del)	127	0177	0x7f

# EE 368 - Microprocessor Systems and Interfacing Laboratory

## Experiment 2

### Subroutines and the Indexed Addressing Mode

- **Objective:** To learn to write programs using subroutines in M68HC11 assembly language and to learn to use the indexed addressing mode.
- **Required Equipment:** The Motorola M68HC11EVB Evaluation Board with attached power supply and a 3.5" diskette

- **Procedure:**

*Please note that the hardware addresses cited below may be different for some boards.*

1. Write a subroutine that will take a "string" of ASCII character data from a location in memory and display it on the screen.
    - a. The 16-bit start address of the message is to be stored in memory locations \$9000 and \$9001, and the subroutine is to use indexed addressing to find the first and each subsequent ASCII character.
    - b. You should store a special value such as \$00 at the end of your character string to indicate "end of string" to your subroutine. (\$00 is called the NULL character, and programs written in C typically use NULL to mark the end of a string.)
    - c. Make your subroutine save and restore used registers. For example, if your subroutine uses register A, push register A onto the stack at the beginning of the routine and pull it from the stack at the end. Be careful to pull in the reverse order from which you pushed!
    - d. To output an ASCII character to the screen use a subroutine called OUTA, which is in the monitor ROM of the M68HC11EVB. OUTA takes the character in register A and sends it to the terminal. First, define OUTA to \$FFB8 using a "EQU" statement (as seen in the Lab 1 example). Second, load the character to be sent into Register A. Third, execute JSR OUTA to call the subroutine.
  2. Write a main program (starting at \$8000) that will use the subroutine to display three messages stored at different locations in memory. The three messages are :
    - message 1: "EE 368 – Microprocessor Systems and Interfacing Laboratory"
    - message 2: *Your Name*
    - message 3: *A greeting message of your choice*
    - a. Don't forget to include a carriage return (\$0D) and line feed (\$0A) in each of your messages so that the terminal display will look normal.
    - b. Refer to the table of ASCII characters if necessary.
  3. Enter and cross-assemble your program and subroutine using the AS11 cross-assembler.
  4. Download and run your program on the M68HC11EVB board and have your instructor check its operation.
  5. Capture a complete run session and attach it to you report
- **Report Contents:** Your report must follow the format found in the "Report Format" template.
  - **Report Preparation:** All material included should be presented in a neat and orderly fashion. Use of a word processor and drawing package is required.

# EE 368 - Microprocessor Systems and Interfacing Laboratory

## Experiment 3

### The Indexed Addressing Mode

- **Objective:** To learn to write programs in M68HC11 assembly language and to learn to use the indexed addressing mode.
- **Required Equipment:** The Motorola M68HC11EVB Evaluation Board with attached power supply and a 3.5" diskette

- **Procedure:**

*Please note that the hardware addresses cited below may be different for some boards.*

1. Write a program (starting at \$8000) that will take ten 8-bit **negative** numbers (\$B4, \$D2, \$A3, \$84, \$E0, \$91, \$C4, \$F6, \$B7, \$D8), will take the two's complement of each number, and will sort them in ascending order.
    - a. The numbers are to be stored in locations \$9000 to \$9009 and the sorted numbers are to be placed in locations \$9010 to \$9019.
    - b. You may use a "bubble" sorting algorithm, or any other. Your program should display the original ten numbers as well as the sorted ten numbers.
    - c. Recall that a bubble sorting algorithm starts at the bottom and goes up to the top, swapping the two numbers if they are in the wrong order. Once the sort goes all the way to the top, it starts over again at the next-to-last location. It is the least efficient sorting algorithm you can use, but the easiest to write.
    - d. Hint: You will probably find it convenient to use *indexed addressing*. To use it, set one of the index registers (X or Y) to the address you want to access. Then issue an access using indexed addressing. For example, you could set X to \$9000. Then you can do an access relative to X (neg \$00,x) and then you can increment X. This will let you march through the 10 numbers and take the 2's complement of each number.
  2. Enter and cross-assemble your program using the AS11 cross-assembler.
  3. Download and run your program on the M68HC11EVB board and have your instructor check its operation.
  4. Capture a complete run session and attach it to your report.
- **Report Contents:** Your report must follow the format found in the "Report Format" template.
  - **Report Preparation:** All material included should be presented in a neat and orderly fashion. Use of a word processor and drawing package is required.

# EE 368 - Microprocessor Systems and Interfacing Laboratory

## Experiment 4

### Interrupts and the Stack Pointer

- **Objective:** To learn to cause an IRQ interrupt to save the address of a message on the stack, and cause an XIRQ interrupt to display the message on the screen.
- **Required Equipment:** The Motorola M68HC11EVB Evaluation Board with attached power supply and a 3.5" diskette

- **Procedure:**

*Please note that the hardware addresses cited below may be different for some boards.*

1. Declare 2 bytes using the DC.B statement. This will reserve two bytes (16 bits) in memory, give it a globally visible name, and set it to zero. This is the same as declaring an integer variable in C.
  2. Write an IRQ interrupt routine that will save the starting address of a message (stored in memory) in the globally visible variable and return to the main program. The message is "EE 368: Microprocessor Systems and Interfacing Laboratory IRQ Routine Installed".
  3. Write an XIRQ interrupt routine that will recover the starting address of the message from variable, display the message on the screen, and return to the main program. To display the message on the screen, use indexed addressing to find the first and each subsequent ASCII character, and subroutine OUTA (at address \$FFB8) to output each character on the screen. Don't forget that you should store a special value such as \$00 at the end of your characters string to indicate "end of string" to your display routine. The XIRQ interrupt should detect the case when the variable is still zero, meaning that the IRQ has not been asserted yet, and take no action.
  4. Write a main program (starting at \$8000) that will initialize the interrupt vector jump table entries for IRQ (addresses \$00EE to \$00F0) and for XIRQ (addresses \$00F1 to \$00F3), enable both IRQ and XIRQ in the CCR, and enter a continuous loop. The interrupt vector jump table is a list of JMP instructions. To handle an interrupt correctly, go to the jump table entry location and create a JMP instruction to your interrupt handler. For example, if your IRQ handler is at \$8010, the jump table entry at \$00EE should be JMP \$8010. For more information about setting up the interrupt vectors, refer to section 3.3 in the M68HC11EVB Evaluation Board User's Manual.
  5. Enter and cross-assemble your program and interrupt handling routines using the AS11 cross-assembler.
  6. Download and run your program on the M68HC11EVB board and use the digital pulser to cause an IRQ. Then, use the digital pulser again but to cause an XIRQ. The message "EE 368: Microprocessor Systems and Interfacing Laboratory IRQ Routine Installed" should now appear on the screen. Have your instructor check its operation.
  7. Capture a complete run session and attach it to you report
- **Report Contents:** Your report must follow the format found in the "Report Format" template.
  - **Report Preparation:** All material included should be presented in a neat and orderly fashion. Use of a word processor and drawing package is required.
  - **Notes on Running Wookie:** The location of the interrupt handler is different in Wookie than in Buffalo. When running Wookie, you don't need to put a branch in a jump table. Instead, put the label for your interrupt handler at the native 68HC11 IRQ vector location. The IRQ label goes in \$FFF2 and the XIRQ label goes in \$FFF4. For example, to set up a routine named IRQHAND for IRQ, put in ORG \$FFF2 followed by FDB IRQHAND.

**EE 368 - Microprocessor Systems and Interfacing Laboratory**  
**Experiment 5**  
**Interrupts and the use of the Parallel I/O Ports**  
**for Reading and Driving Mechanical and Display Devices**

- **Objective:** To learn to use the parallel input/output ports of the M68HC11 to read mechanical input devices (switches) when an IRQ interrupt occurs and to display the switch values in LEDs when an XIRQ interrupt occurs.
- **Required Equipment:** The Motorola M68HC11EVB Evaluation Board with attached power supply and a 3.5" diskette. A digital pulser and a switch/LED box and necessary cable.

- **Procedure:**

*Please note that the hardware addresses cited below may be different for some boards.*

1. Declare a 1-byte global variable to store the state of the switches that will be wired to port C.
  2. Write an IRQ interrupt routine that will read the state of the switches from port C (address \$1003), save that byte in a global variable and return to the main program. Make sure that port C is configured as input by writing \$00 to its data direction register (address \$1007).
  3. Write an XIRQ interrupt routine that will recover the value entered through the switches from the global variable, write that byte to the LEDs connected to port B (address \$1004), and return to the main program.
  4. Write a main program (starting at \$0100) that will initialize the interrupt vector jump table entries for IRQ (addresses \$00EE to \$00F0) and for XIRQ (addresses \$00F1 to \$00F3), initialize the global variable to the switch-state in port C, enable both IRQ and XIRQ in the CCR, and enter a continuous loop. For more information about setting up the interrupt vectors, refer to section 3.3 in the M68HC11EVB Evaluation Board User's Manual.
  5. Enter and cross-assemble your program and interrupt handling routines using the AS11 cross-assembler.
  6. Connect the switch/LED box to the M68HC11EVB board using the proper cable. Download and run your program on the board. Set an arbitrary value on the switches. Use the digital pulser to cause an IRQ. Then, use the digital pulser to cause an XIRQ. The LEDs should now have the same value as the switches. Repeat this step for different switch values. Have your instructor check its operation.
- **Report Contents:** Your report must follow the format found in the "Report Format" template.
  - **Report Preparation:** All material included should be presented in a neat and orderly fashion. Use of a word processor and drawing package is required.

# EE 368 - Microprocessor Systems and Interfacing Laboratory

## Experiment 6

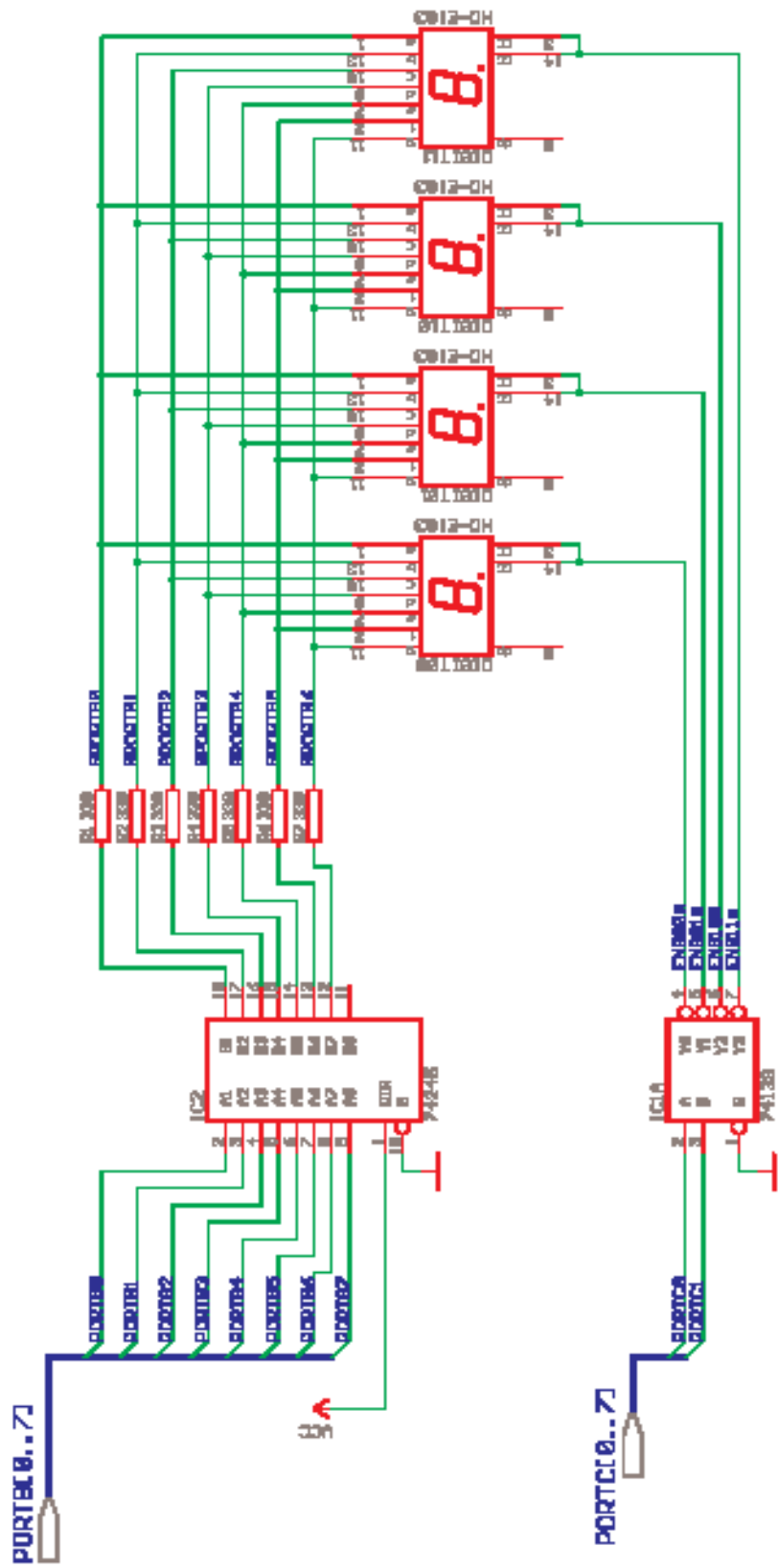
### Use of Parallel I/O Ports and Timing Loops to Drive Seven-Segment Display Devices

- **Objective:** To learn to use the parallel input/output ports of the M68HC11 to drive a display consisting of seven-segment LEDs.
- **Required Equipment:** The Motorola M68HC11EVB Evaluation Board with attached power supply and a 3.5" diskette. Four seven-segment display chips, one 74HCT139 (dual 2-4 decoder), a protoboard, and the necessary cable.

- **Procedure:**

*Please note that the hardware addresses cited below may be different for some boards.*

1. Write a subroutine that will take in a hex digit ranging from \$0 to \$F, inclusively, lookup a table to get the corresponding seven-segment code, and pass back that code.
    - a. To develop the lookup table, use the schematic below to see which output bit drives which seven-segment display segment, and then develop a table of 16 values. Each value should be a map of which display segments are to be lit up for the corresponding character. For example, the entry for '5' should be the segments that make up the '5' on the display.
  2. Write a main program (starting at \$0100) that will take an arbitrary four-digit hexadecimal number stored in memory (locations \$0000 and \$0001) in packed form, and unpack it to four hex digits. Then, for each digit, the program will call the subroutine from (1.) above to get the corresponding seven-segment code, display it by writing its code to a seven-segment display chip for an appropriate period of time.
  3. The program will continuously repeat displaying the four digits. If the digit display time is too short, the LED segments will not have time to respond and the display will be dim or dark, and if the display time is too long, the display will flicker. Write software timing loops which can be used to adjust the display time. The seven-segment chips will be driven by port B (address \$1004), while the decoder will be driven by the lower two bits of port C (address \$1003). The decoder determines which display is lit up. Make sure that port C is configured as output by writing \$FF to its data direction register (address \$1007). The number to display is \$A05B.
  4. Connect the four seven-segment display chips and the 74HCT139 decoder on the protoboard as indicated on the attached sheet, and test the circuit's operation.
  5. Enter and cross-assemble your program and interrupt handling routines using the AS11 cross-assembler.
  6. Connect the protoboard to the M68HC11EVB board using the proper cable. Download and run your program on the board. The seven-segment chips should display the above number. Have your instructor check its operation.
- **Report Contents:** Your report must follow the format found in the "Report Format" template.
  - **Report Preparation:** All material included should be presented in a neat and orderly fashion. Use of a word processor and drawing package is required.



ECE Dept., USA

TITLE: EE368L6B

Document Number:

REV:  
1.0

Date: 2/26/2007 08:49:42a

Sheet: 1/1

# EE 368 - Microprocessor Systems and Interfacing Laboratory

## Experiment 7

### Use of I/O Ports to Display Messages on LCD Module

- **Objective:** To learn to use the input/output ports of the M68HC11 to display simple messages on LCD.
- **Required Equipment:** The Motorola M68HC11EVB Evaluation Board with attached power supply. One HD 44780-based LCD module, and the necessary cable.

- **Procedure:**

*Please note that the hardware addresses cited below may be different for some boards.*

1. Write four subroutines. (See the next page for important details.)
  - a. The first subroutine "WRINST" will write an instruction to the LCD.
  - b. The second subroutine "WRDATA" will write data to the LCD. Both subroutines will need to set the LCD through port C and transfer instruction/data to the LCD through port B.
  - c. The third subroutine "DELAY" will create a 5 ms delay time for the characters to be displayed on LCD.
  - d. The last subroutine "DISPMSG" will be used to display a message on the LCD by sending one byte at a time until all characters in the message are sent, thus it needs to call subroutine "WRDATA".
  - e. All subroutines except "DELAY" need to call subroutine "DELAY" (may need to call it more than once) for delay requirement. Please refer to "How to control a HD44780-based Character-LCD" that is available on the EE368 class "Help" folder under the title "LCD\_Command\_Set". You may also refer to the manufacturer's datasheet in the folder as well.
2. Write a main program (starting at \$0100) that will, first, need to make sure that port C is configured as output by writing \$FF to its data direction register (address \$1007). Then, it will get the address of the initialization sequence bytes (\$30, \$30, \$30, \$38, \$0C, \$01, \$06) and send them to initialize and clear the LCD. The main program will send the start address of DDRAM in the LCD to the start of the 1st line by calling subroutine "WRINST", and, then, pass the address of the first message to display on the LCD by calling subroutine "DISPMSG". To display the second message on the 2nd LCD line, follow the same procedure. (Again, see the next page for important details.)
3. Connect the LCD module to evaluation board using the proper cable.
4. Enter and cross-assemble your program and subroutine using the AS11 cross-assembler.
5. Download and run your program on the M68HC11EVB board and have your instructor check its operation and LCD display.

- **Report Contents:** Your report must follow the format found in the "Report Format" template.
- **Report Preparation:** All material included should be presented in a neat and orderly fashion. Use of a word processor and drawing package is required.

## Using the LCD Module

The LCD module works over an 8-bit interface. You will be sending it a string of commands to initialize it, then a command to set a certain memory address, and then a string of data (which are the characters to be displayed.)

First, the initialization command sequence for a 2-line display and an 8-bit interface is the following:  
\$30, \$30, \$30, \$38, \$0C, \$01, \$06

Command-> Meaning

- \$30 ----> Function set: 8-bit mode
- \$30 ----> Function set: 8-bit mode
- \$30 ----> Function set: 8-bit mode
- \$38 ----> Function set: 8-bit mode, 2 lines, 5x7 dots
- \$0C ----> Display on, Cursor off, Blink cursor off
- \$01 ----> Clear display
- \$06 ----> Increment cursor position, No display shift

Note that you *must delay 5ms between commands.*

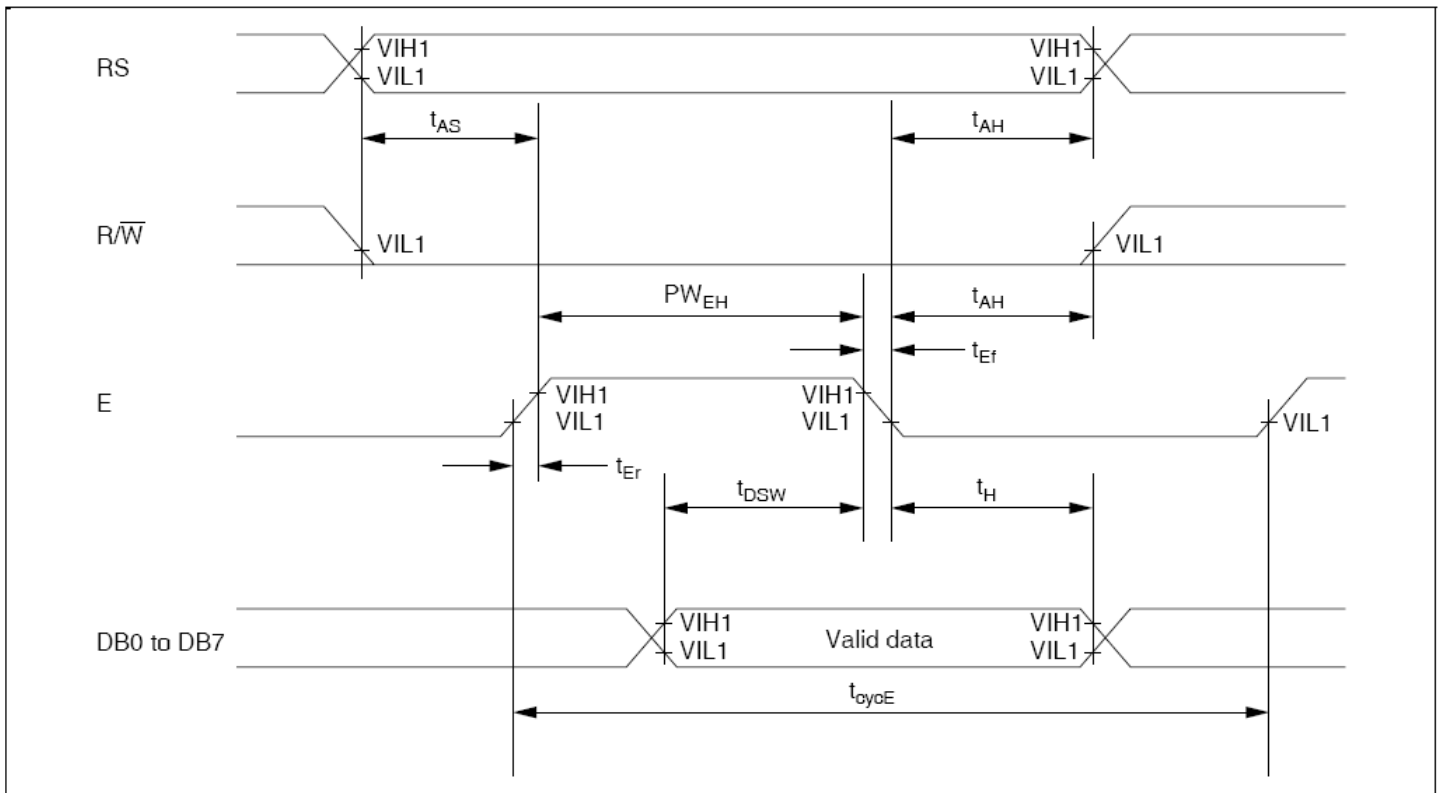
Second, the characters to be displayed are stored in a Display Data RAM (which they call “DDRAM”) inside the LCD module itself. So in order to display characters, you have to write them into the LCD module’s DDRAM. You do this by first setting the address of DDRAM you want to start writing to, and then sending the characters one-by-one. The first line begins at display data RAM (DDRAM) address \$00, and the second line begins at DDRAM address \$40.

So you will need to do the following:

- \$80 ----> Set DDRAM address to \$00 (This sets the start address and tells it to be ready to receive data.)
- Set RS bit to 1 (tells the LCD that you are sending data, not instructions)
- Send first line message one ASCII character at a time
- Set RS bit to 0 (tells the LCD that you are sending instructions)
- \$C0 ----> Set DDRAM address to \$40
- Set RS bit to 1
- Send second line message one ASCII character at a time
- Set RS bit to 0

As above, delay 5 ms between commands.

To send an instruction or data, you will need to understand the signal timing and how the LCD signals map to 68HC11 signals. First, the LCD timing...



**Figure 7.1: Timing Diagram from LCD Datasheet**

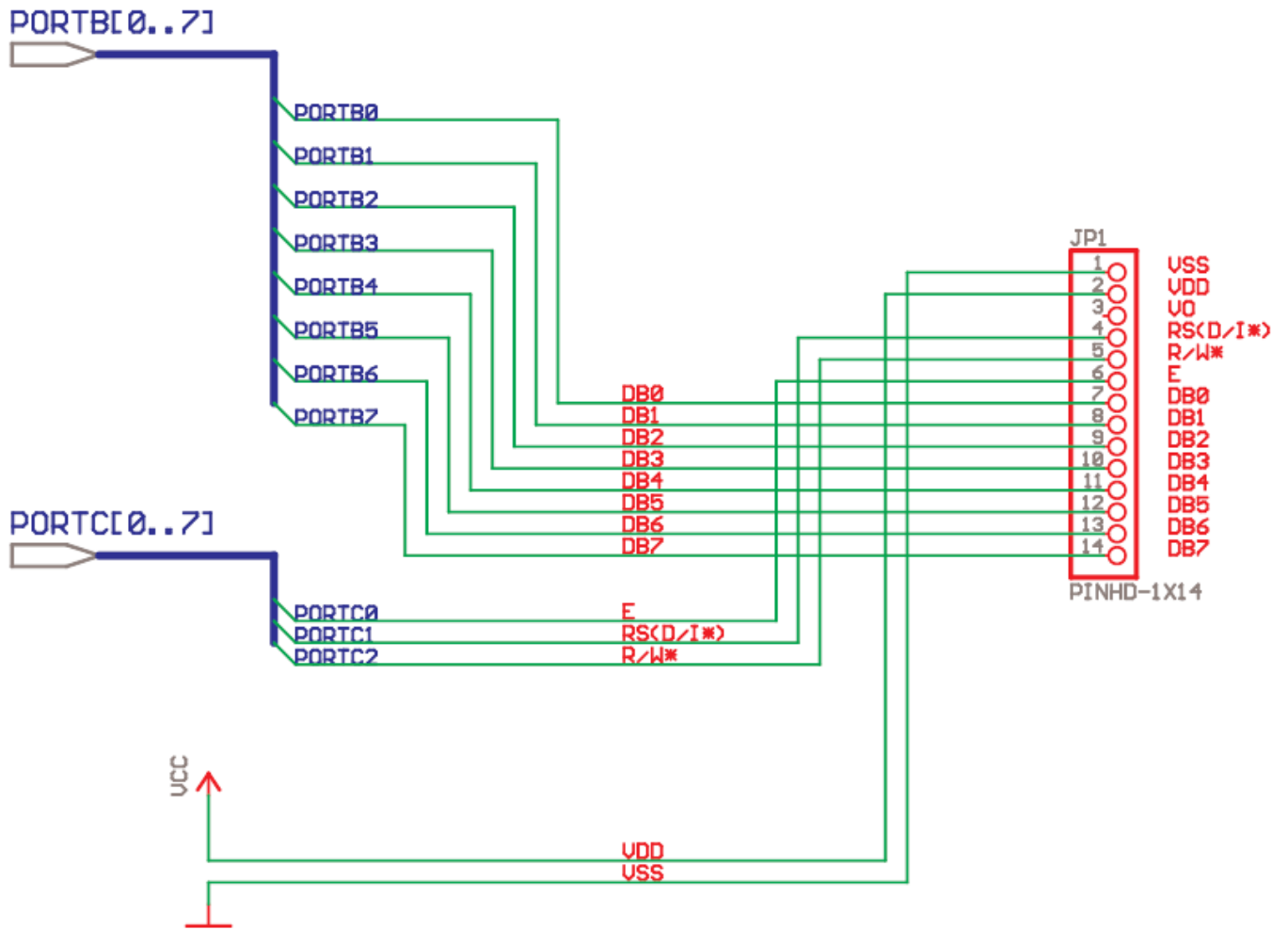
To send an instruction:

- Set RS bit to 0
- Set R/W\* bit to 0
- Put 8-bit instruction on DB[7:0]
- Toggle the E clock (rising edge followed by falling edge)

To send data:

- Set RS bit to 1
- Set R/W\* bit to 0
- Put 8 bits of data on DB[7:0]
- Toggle the E clock (rising edge followed by falling edge)

So how do you do this with the 68HC11 board? You have to understand how the 68HC11 is attached to each of the LCD's signals, usually by looking at the schematic...



Use the schematic to figure out the mapping of LCD signals to 68HC11 signals. For example, to toggle the E clock, set bit 0 of port C to 1 and then back to 0. As noted above, be sure to set Port C to be an output.

# EE 368 - Microprocessor Systems and Interfacing Laboratory

## Experiment 8

### Digital Clock on Serial Port

- **Objective:** To create a digital clock using the internal clock of the Motorola M68HC11EVB Board. The clock should be accurate to within  $\pm 0.1$  sec. The clock should be refreshed to the monitor every second and include the seconds, minutes, and hours.
- **Required Equipment:** The Motorola M68HC11EVB Evaluation Board with attached power supply.

- **Procedure:**

*Please note that the hardware addresses cited below may be different for some boards.*

1. Registers in use:

TMSK2 (\$1024): timer interrupt mask 2 register.

TFLG2 (\$1025): timer interrupt flag register 2.

PACTL (\$1026): pulse accumulator control register.

Please refer to Section 9: Timing System of the M68HC11 manual for more details. Also note that the "I" flag in the condition code register enables / disables all timer interrupts, and so I must be set to 0 for timer interrupts to occur (in addition to programming the three registers listed above).

2. Write three subroutines: INCR, DISP, and DELAY.

- a. The first subroutine, "INCR" will complete the necessary Binary Coded Decimal arithmetic to increment the clock variables by one second.

- b. The second subroutine, "DISP" will display the clock variables to the screen in an hour-minutes-seconds format.

- c. The third subroutine, "DELAY" will be calibrated to cause a one second delay between count cycles, more than one call to the delay subroutine may be necessary.

3. Write a main program (starting at \$8000) that creates a loop displaying the current time, wait one second, and calculate the next time.

4. Enter and cross-assemble your program using the AS11 cross-assembler.

5. Download and run your program on the M68HC11EVB board and have your instructor check its operation.

- **Report Contents:** Your report must follow the format found in the "Report Format" template.
- **Report Preparation:** All material included should be presented in a neat and orderly fashion. Use of a word processor and drawing package is required.

**EE 368 - Microprocessor Systems and Interfacing Laboratory**  
**Experiment 9**  
**Digital Clock on LCD**

- **Objective:** To set up the digital clock in Experiment 8 to display on the LCD display on Experiment 7.
- **Required Equipment:** The Motorola M68HC11EVB Evaluation Board with attached power supply.

- **Procedure:**

*Please note that the hardware addresses cited below may be different for some boards.*

1. Make sure that all of the time processing in Experiment 8 is contained in interrupt service routines and that the current time can be found in bytes of RAM.
  2. Modify the program in experiment 7 to write to the LCD repeatedly.
  3. Each time the LCD-write loop occurs, add a routine (or in-line code) to convert the current time to ASCII and then write it to the LCD. For example, if you declared a string in Experiment 8 to write to the LCD, you might convert the current time by writing into the string, then call the function to write the string to the LCD.
  4. Don't forget that you have to send a \$C0 or a \$80 command to the LCD to start a new line. This must be sent every time you start writing to the LCD.
  5. On the Motorola EVB boards, there are only 256 bytes of usable RAM, from address \$100 to address \$1FF. You must take special care to make sure that your code ends on or before address \$1FF.
  6. Show the incrementing-time display on the LCD to the instructor or teaching assistant.
- **Report Contents:** Your report must follow the format found in the "Report Format" template.
  - **Report Preparation:** All material included should be presented in a neat and orderly fashion. Use of a word processor and drawing package is required.

# EE 368 - Microprocessor Systems and Interfacing Laboratory

## Experiment 10

### A/D Conversion and Pulse-Width Modulation

- **Objective:** To read the A/D converter and create a PWM waveform that dims an LED circuit.
- **Required Equipment:** The Motorola M68HC11EVB Evaluation Board (Axiom board, starting address \$8000) with attached power supply. ALSO BRING the “bag of chips” from the 268 lab. You will need one 74LS04, one 7-segment display, and some wires. The instructor will supply a potentiometer and a resistor.
- **Procedure:**
  1. Connect the 68HC11 VREFHI (labeled VRH) to Vcc and connect VREFLO (VRL) to ground.
  2. Wire up the potentiometer so that one leg is connected to Vcc, one leg to Ground, and the middle leg (the “wiper”) is connected to PE1. With the board powered, confirm with a voltmeter that turning the potentiometer ranges the wiper voltage from 0 to Vcc.
  3. Wire up a 74LS04 to Vcc and ground.
  4. Connect the common signal of the 7-segment display to Vcc *by using a resistor* of about 600 ohms.
  5. Wire the output of one of the 74LS04 inverters to one of the LED segments.
  6. With the board powered, connect the input of the inverter to Vcc. You should see the LED segment light up. Then connect the input to the OC3 output.
  7. Download the software you have written.
  8. Demonstrate the software: When the potentiometer is turned, the LED should brighten and dim.
  9. Return the potentiometer and resistor to the instructor.
- **Report Contents:** Your report must follow the format found in the “Report Format” template.
- **Report Preparation:** All material included should be presented in a neat and orderly fashion. Use of a word processor and drawing package is required.

#### Using the A/D Converter

The 68HC11 has an analog-to-digital (A/D) converter. We will set it up to continuously convert four of the A/D inputs to digital values.

To activate the converter, set the most significant bit of register \$1039 (the ADPU bit of the OPTION register). Then wait 10 ms for the analog voltages in the converter to settle.

Next, the converter needs to be set up to “scan” the A/D inputs. In this lab, we will set up the A/D to scan the first four channels and put the results in the four A/D result registers. To set up the scan, set the SCAN bit to 1, the MULT bit to 1, and set CD and CC each to 0. This tells the A/D converter to convert continuously (SCAN=1), scan through four channels (MULT=1), and scan the first four channels (CD=CC=0). The bits of the ADCTL register are indicated below.

Address: \$1030

	Bit 7	6	5	4	3	2	1	Bit 0
Read:	CCF	0	SCAN	MULT	CD	CC	CB	CA
Write:								
Reset:	0	0	U	U	U	U	U	U

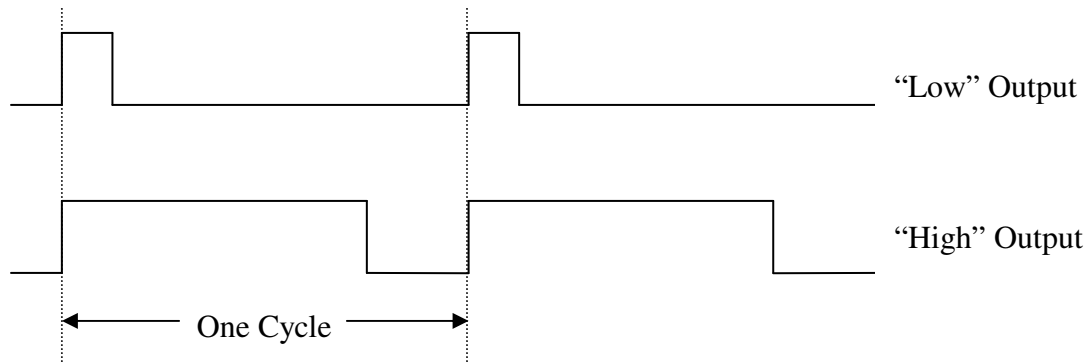
U = Unaffected

**Figure 12-6. A/D Control/Status Register (ADCTL)**

Once the A/D is programmed, scanning will begin. In the interest of debugging, set up the main loop of the program to print out the A/D results every 500 ms. The first result is at \$1031, the second at \$1032, etc. This will give you an indication that the potentiometer is wired correctly; the PE1 value should range between 00 and FF as the potentiometer is turned. (PE1 corresponds to the result register \$1032.)

**What is Pulse-Width Modulation?**

Pulse-Width Modulation (or PWM) is a very simple form of digital to analog conversion. With PWM, the duty cycle (the fraction of time that an output is high) is varied. A low duty cycle corresponds to a low voltage and a high duty cycle to a high voltage. In fact, a PWM output can be low-pass filtered with a resistor and capacitor to create a DC voltage that can be varied under computer control. An example is shown below.



**Using the OC Outputs**

The timer facility on the 68HC11 was designed specifically for (among other things) PWM. The process will work as follows. First, when the timer overflows (reaches a value of \$0000), the OC3 output is turned on (set high). Second, the OC3 register is set to a value equal to the A/D reading. This will make the OC3 output fall in an amount of time proportional to the A/D reading. For example, if the A/D reading is \$02, the output will fall when the timer reaches \$0200. If the A/D reading is \$EF, the output will fall when the timer reaches \$EF00.

To set all this up in software, the following needs to be done. First, enable the timer overflow interrupt and install an interrupt handler for the timer overflow. Second, inside the timer overflow interrupt handler, perform the following steps.

- Clear the TOF flag. (This is similar to the timer-based labs earlier, like labs 7 and 9.)
- Make Port A bit 5 (OC3 output) a general purpose output. (This is done by clearing \$1020.)
- Force Port A bit 5 high. (BSET memory location \$1000 with a mask of \$20.)
- Read A/D result 2 (found in \$1032)
- Store the result in the most significant byte of the OC3 register (\$101A). This is what indicates to the 68HC11 what “time” to turn off the output.
- Clear the least significant byte of the OC3 register (\$101B)
- Set the OC outputs back so that they turn off (go low) at the indicated time. (Write \$AA to \$1020.)
- Return from interrupt.

## Wiring Diagrams

