ECE 367 - Experiment #2
"Morse Code ID Tag"

Spring 2006 Semester

Introduction

Now that you are familiar with the basics of getting an assembly language program to run on the Technological Arts MicroStamp11 development system, the goal of this experiment is to increase your understanding of the Motorola 68HC11 microcontroller and its assembly language. As a result you will be able to write and demonstrate code that flashes the letters of your name on an LED in Morse Code.

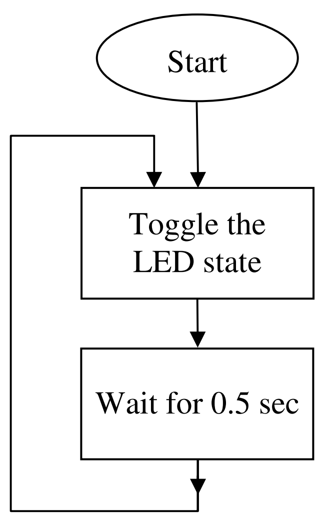
Required Hardware and Software

As in Experiment #1, only the MicroStamp11 module and its docking station hardware are required for this experiment.

Background

a) Software Delay Loops

Experiment #1 presented you with 68HC11 assembly language code to flash an LED on and off periodically. The basic flowchart for what took place there is shown here:



As you saw, the half-second delay caused the LED to blink at a rate of approximately 1 cycle/sec (ignoring the few clock cycles needed to jump back to the top and toggle the LED state). Now let's analyze the code to see how that was accomplished.

The basic idea is to load a register with an integer value and then decrement the register contents until zero is reached. By knowing how many clock cycles it takes to execute each assembly language instruction and the period of the clock, the delay of a software delay loop may be precisely calculated.

Consider this software delay loop:

```
            LDY   #1000        ; 4 clock cycles

A0:         DEY                ; 4 clock cycles
            BNE   A0           ; 3 clock cycles
Next:
```

In the code above, 16-bit register Y is first initialized to 1000. It takes 4 clock cycles to decrement Y, and another 3 clock cycles to compare the result to zero (BNE: Branch if result of the preceeding operation is Not Equal to zero[1]). When Y is not equal to zero, a jump to A0 takes place; this happens 1000 times. When Y finally equals zero no jump is made and program execution continues with the instruction that follows (at Next).

With an internal 2 MHz processor clock frequency (period = 0.5 usec), the total delay introduced by these three instructions is:

$$(4 + 1000 \cdot (4 + 3)) \text{ clock cycles} \times 0.5 \text{ usec}$$
$$= 7004 \times 0.5 \text{ usec} \approx 3.5 \text{ msec}$$

By changing the initial value of register Y, the three lines of code shown above will produce delays up to approximately 1/5 sec. Here is what may be done to achieve even longer delays:

- Insert lines of code within the delay loop that serve no purpose except to use up clock cycles. For example, the NOP (No Operation) instruction uses 2 cycles and the BRN (Branch Never) instruction uses 3 cycles of execution time.

  Example – here is a 1/2 sec software delay loop:

```
      LDY   #50000

A0:   NOP                ; 2 clock cycles
      NOP                ; 2 clock cycles
      BRN   A0           ; 3 clock cycles
      BRN   A0           ; 3 clock cycles
      BRN   A0           ; 3 clock cycles
      DEY                ; 4 clock cycles
      BNE   A0           ; 3 clock cycles
```

[1] as reflected by the state of the Z flag

- With nested loops very long software delays are possible.

  Example – here is a 1 min software delay loop:

```
     LDY   #17089     ; 4 cycles
A0:  PSHY             ; 5 cycles            \
          LDY #1000   ; 4 cycles             |
A1:       DEY         ; 4 cycles \ 7 x 1000  | 7022
          BNE   A1    ; 3 cycles /           |  x
     PULY             ; 6 cycles             | 17089
     DEY              ; 4 cycles             | cycles
     BNE A0           ; 3 cycles            /
```

Nested loops may be used to write software delays that last for *years*, but that is seldom necessary. (Later we will learn to use internal timer subsystems to achieve delays while allowing the microcontroller to execute other useful code at the same time.)

b) Anatomy of Program 1

Now that you know about software delay loops, let's take a look at the code that you ran in Experiment 1.

```
; Define symbolic constants

Regbas     EQU    $0000          ; Register block starts at $0000
PortA      EQU    $00            ; PortA Address (relative to Regbas)
Config     EQU    $3F            ; Configuration control register
```

The lines above define symbolic replacements for numbers that specify various addresses. "Regbas" is a nickname for 16-bit starting address of the 68HC11 register block (these are some CPU control and status registers that are accessed using memory read/write instructions, as if communicating with external memory). The MicroStamp11 has this register block configured for addresses $0000–$003F, instead of $1000–$103F as is usually the case for 68HC11 microcontrollers. But this minor difference is easily taken care of by equating Regbas to $0000 instead of to $1000.

PortA and Config refer to specific registers withing the register block.

```
ORG  $FF00        ; Place code in EEPROM starting at $FF00
```

This line directs the assembler to store the code that follows in EEPROM beginning at address $FF00 (the MicroStamp11 has 8K of EEPROM, ranging from $E000 to $FFFF).

```
Start:      LDS     #$00FF          ; Initialize stack pointer
            LDX     #Regbas         ; Initialize register base address ptr.
            LDAA    #$04
            STAA    Config,X        ; Disable "COP" watchdog timer
```

LDS loads address value $00FF into the stack pointer register so that it points to the top of RAM (MicroStamp11 RAM covers address range $0040 to $00FF).

"LDX #Regbas" initializes register X to the value equated with symbol Regbas, $0000, to serve as a reference base address. We will avoid using register X for anything else.

"STAA Config,X" copies the contents of Accumulator A ($04) to the Config register whose address is $003F. The Config register may only be written to within the first 64 clock cycles after a power-on reset, so we do this as soon as possible. What is being done here is to disable a COP (Computer Operating Properly) timer function that otherwise would interfere with normal program operation.

```
            LDAA    #$FF
    Loop:   STAA    PortA,X         ; Initialize output lines of PORT A to 1's
            EORA    #$FF            ; Toggles PortA values
            BSR     Delay
            JMP     Loop
```

In the code above we implement an endless loop to toggle Port A output pins (PA4, PA5 and PA6), call a ½ sec software delay procedure, then repeat. This results in 1 Hz output frequency. You already know how software delay loops work – confirm that the subroutine "Delay" introduces approximately ½ sec delay.

```
; Define Power-On Reset Interrupt Vector

            ORG     $FFFE           ; $FFFE, $FFFF = Power-On Reset Int. Vector Location
            FDB     Start           ; Specify instruction to execute on power up
```

Finally, the code above initializes memory locations {$FFFE and $FFFF} to the address of instruction at label Start ($FF00). This is called the Power-On Reset Interrupt Vector. When the microcomputer is first powered up, the processor fetches the address stored there (in nonvolotile EEPROM, same as program code) and begins execution of code that is found at that address.

c) Morse Code Basics

Letters of the alphabet may be represented using dots and dashes (short and long bursts of light or sound) in Morse Code – a system originally developed for the telegraph. You are asked to take what you know about basic 68HC11 program structure and software delay loops to design an assembly language program that outputs your name in Morse Code (first, last or both) from the PA6 LED on the docking module.

Here is the International Morse Code alphabet:

A	• _	N	_ •
B	_ • • •	O	_ _ _
C	_ • _ •	P	• _ _ •
D	_ • •	Q	_ _ • _
E	•	R	• _ •
F	• • _ •	S	• • •
G	_ _ •	T	_
H	• • • •	U	• • _
I	• •	V	• • • _
J	• _ _ _	W	• _ _
K	_ • _	X	_ • • _
L	• _ • •	Y	• _ • _
M	_ _	Z	_ _ • •

Forming the coded letters using short (dots) and long (dashes) pulses of light:

- make each dot between 1/8 and 1/3 sec in duration
- a dash is equal to three dots
- the space between parts of the same letter is equal to one dot
- the space between two letters is equal to one dash
- the space between two words is equal to five dots

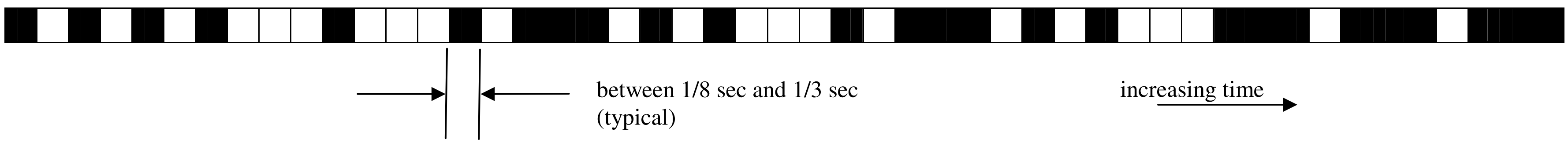
Suggested subroutines to write:

- Unit_Delay – delay between 1/8 and 1/3 sec
- Dot – turn on PA6 for one unit delay
- Dash – turn on PA6 for three unit delays
- Short_Space – turn off PA6 for one unit delay
- Long_Space – turn off PA6 for three unit delays

Write assembly language code to continuously display your name in Morse Code. We can call this product the "Morse Code ID Tag." Demonstrate the working microcontroller unit to your T.A. Submit a lab report as specified during lecture.

<u>Appendix</u>

Here is an example of how the word "hello" would be sent:



Sample code to generate the letter "L" (dot dash dot dot):

```
    ⋮
call Dot
call Short_Space
call Dash
call Short_Space
call Dot
call Short_Space
call Dot
call Long_Space
    ⋮
```