CS-280
Laboratory 1
Due 3/18/99
10 points

In this lab, you will enter, assemble, link, download, run, and simulate a program. The program you will be entering is shown in Appendix A. Enter the program with a text editor, and save it with the name `lab1.s` Remember, .s files signify the human-readable assembly language.

Next, assemble the program from a DOS box with the command (replace *X:* by the drive letter and *path* by the path to the assembler). In S-307 it will be K:\apps\

```
X:\path\as6811 -losz lab1.s
```

The letters `-losz` are command line options for the assembler. The `l` command causes the assembler to generate a listing file (in this case named `lab1.lst`). The `o` option tells the assembler to generate an object file (`lab1.rel`). The `s` option causes a symbol file to be generated (`lab1.sym`). The last option `z` causes the assembler to make the labels case sensitive. Examine each of these files.

Next, run the linker. The linker needs you to create the file called `lab1.lnk` shown in Appendix B. This file should be in the same directory as your other lab1 files. The command to run the linker is:

```
X:\path\aslink -f lab1
```

Looking at the `lab1.lnk` file, the first line is the linker options. The options mean:

```
m  - create a map file (lab1.map)
s  - create a Motorola compatible .s19 executable file
     (lab1.sl9)
z  - make the labels case sensitive
u  - create a relocated list file (lab1.rst)
```

The next lines starting with `-b` tell the linker where to place each of the areas in the assembly file. We will have the `CODE` area placed at 0xC000 (the start of memory in the briefcase), `STACK` at 0xF000 (Question: Where does the `DATA` area end up? Hint: Look in the .rst or .map file). The lines following the `-b` lines are a list of files to link. The `-e` at the end tells the linker it has reached the end of the .lnk file.

Now you are ready to use the executable (`lab1.s19`). The executable can be run on either a software simulator (Wookie) or in the actual briefcase hardware. The two methods are outlined below.

Wookie (simulation)

Start the Wookie executable (version 1.5 or greater). You will see a dialog box, pick briefcase and enter the starting address (in our case it is C000). Next you can bring in a listing file, which will show the line in the code which is the current line highlighted. Press the *view code* button on the left. Next pick *Load LST file*, and load the `lab1.rst` file (file type *GCC*, offset *0*). You can move the window after the file loads. Next press the *M68HC11 CPU* button. This will let you examine the registers. Now, press the *7 segment* button. This brings up the four character display. You can now single step through the program by repeatedly pressing the icon of the person walking. Run at full simulator speed by pressing the red dot (it turns green while running). Watch the registers and character display change.

Briefcase (hardware)

Start up WinBug11 (version 1.33 or greater). Boot the talker by pressing the icon on the far left (or a menu pick under the File menu). The talker should be `wintalk.boo`, also pick the COM port (1 or 2) you are using. Next pick *special test mode*. Press the reset button on the briefcase, and you should get a message that the talker initialized successfully. If the talker does not initialize, try again. You will not be able to do anything else until the talker initializes. Next open an .s19 file (file open). Once the file is loaded, you can execute by choosing program-execute.

The report for this lab will consist of your .rst file, and a narrative of any problems you had. Comment on the files generated, and also comment on your experiences.

Appendix A: lab1.s

```
; Lab 1
; written by S. Barnicki
; Writes a string to the 4 character display
; March 1999
;

        .globl  write_string    ;declare routine external
        .globl wait

        .area ports (abs)
ddrc    = 0x1007        ;data direction register for C
pioc    = 0x1002        ;for strobe B assertion change
portcl  = 0x1005        ;port C latched
portb   = 0x1004        ;port B

; set area for stack
        .area STACK (rel,con)
        .blkb 256       ;stack area
stckhi::                ;initial stack pointer value


; set area for code
        .area CODE (rel,con)
write_string::
        lds  #stckhi    ;set initial stack pointer value
        ldaa #0xFF      ;set for output
        staa ddrc       ;C output
        ldaa pioc       ;get current value
        anda #0xFE      ;reset bit 0 to 0
        staa pioc       ;send it out

        ldx  #string    ;get the string address
        ldab #0x4       ;position value

print:
        ldaa 0,x        ;get a value
        cmpa #0         ;is it a null?
        beq  end        ;done
        cmpb #0         ;off the end?
        bne  ok
        ldab #0x4       ;reset to 4
        dex
        dex
        dex             ;move pointer back
        jsr  wait
        bra  print      ;keep going

ok:     decb            ;position = position - 1
        stab portcl     ;set the position
        staa portb      ;set the character
```

```
        inx             ;point to next one
        bra  print      ;keep going

        .area CODE (rel,con)
; subroutine wait
; waits 250 ms
; modifies nothing
wait::
        psha            ;save A
        tpa             ;save condition codes
        psha
        pshx
        ldx #250        ;wait 250 milliseconds
1$:     jsr wait1ms     ;local label
        dex
        bne 1$
        pulx
        pula
        tap
        pula
        rts

wait1ms:
        psha            ;1 ms wait
        tpa
        psha
        pshx
        ldx #200
2$:     dex
        nop
        nop
        bne 2$
        pulx
        pula
        tap
        pula
        rts

end::   bra end

;
;data here
;
        .area DATA (rel,con)
string: .asciz "HELLO, IT WORKS!"       ;null terminated string
```

Appendix B: `lab1.lnk`

```
-mszu
-bCODE=0xC000
-bSTACK=0xF000
lab1
-e
```