EECS 388: Computer Systems and Assembly Language

Lab 7: Using MON12 Utility Subroutines for I/O

MON12 Utility Functions

Below are the descriptions of some of the utility functions, taken from the MON12 manual included with the Axiom CD (Documents/Mon12man.pdf).

UPCASE If character in accumulator A is lower case alpha, convert to upper case.
OUTA Output accumulator A ASCII character.
OUTlBYT Convert binary byte at address in index register X to two ASCII characters and output. Returns address in index register X pointing to next byte.
OUTCRLF Output ASCII carriage return followed by a line feed.
OUTSTRG Output string of ASCII bytes pointed to by address in index register X until character is an end of transmission ($04).
OUTSTRGO Same as OUTSTRG except leading carriage return and line feed is skipped.
INCHAR Input ASCII character to accumulator A and echo back. This routine loops until character is actually received.

MON12 Service Routine Jump Table

Below is a table of the MON12 utility functions and their addresses, taken from the CMLS12DP256 manual (Documents\manuals\CMLS12DP256man.pdf on the Axiom CD). It's also available in the "references" section of the class website.

ADDRESS
ff10 MAIN ; warm start
ff13 BPCLR ; clear breakpoint table
ff16 RPRINT ; display user registers
ff19 HEXBIN ; convert ascii hex char to binary
ff1c BUFFARG ; build hex argument from buffer
ff1f TERMARG ; read hex argument from terminal
ff22 CHGBYT ; modify memory byte at address in x
ff25 JMP CHGWORD ; modify memory word at address in x
ff28 READBUFF ; read character from buffer
ff2b INCBUFF ; increment buffer pointer
ff2e DECBUFF ; decrement buffer pointer
ff31 WSKIP ; find non-whitespace char in buffer
ff34 CHKABRT ; check for abort from terminal
ff37 JMP UPCASE ; convert to upper case
ff3a WCHEK ; check for white space
ff3d DCHEK ; check for delimeter
ff40 ONSCI0 ; initialize i/o device
ff43 INPUT ; low level input routine
ff46 OUTPUT ; low level output routine
ff49 OUTLHLF ; display top 4 bits as hex digit
ff4c OUTRHLF ; display bottom 4 bits as hex digit

ff4f OUTA ; output ascii character in A
ff52 OUT1BYT ; display the hex value of byte at X
ff55 OUT1BSP ; out1byt followed by space
ff58 OUT2BSP ; display 2 hex bytes (word) at x and a space
ff5b OUTCRLF ; carriage return, line feed to terminal
ff5e OUTSTRG ; display string at X (term with $04)
ff61 OUTSTRG0 ; outstrg with no initial carr ret
ff64 INCHAR ; wait for and input a char from term
ff67 VECINIT ; initialize RAM vector table

Examples:

To use a utility function, simply do a JSR to the memory address. For example, to output an ASCII character that's in accumulator A, you can use the following code. It's considered good form to label your functions, rather than just jumping to the memory locations.

```
OUTA    equ   $FF4F
        ldaa  #$67      ;Load A with a lowercase "g."
        jsr   OUTA      ;Print "g" to the monitor.
```

Ex1. Prompting the user:

Define your message text using FCC, and call the OUTSTRG function.

```
OUTSTRG equ   $FF5E
EOT     equ   $04
        ldx   #FIRST

        org   $4000
        jsr   OUTSTRG       ;Print the string until EOT.
        swi

        org $6000

FIRST   fcc 'Enter first number:'
        fcb   #EOT
```

Ex2. Reading a character from the keyboard:

Define INCHAR with the correct address, similarly to the previous example. This function waits for the user to enter a character, and then stores it in accumulator A.

```
        jsr     INCHAR  ;Take in a character
```

Accumulator A now has the ASCII value of the character.

Ex3. Passing arguments to a function via the stack:

You may simply push the arguments on the stack, and then call the function. Here, "ARITH" is your own function that you defined.

```
        psha            ;Push the argument.
        jsr     ARITH   ;Call the function.
```

Ex4. Retrieving arguments from the stack inside of a function:

Remember that calling the function using JSR puts the return value on top of the stack. You may retrieve arguments from the stack without popping them using indexed addressing and the stack pointer.

```
ARITH
        psha            ;Push A so that A is only modified locally.
        ldaa    3,sp    ;Retrieve the top argument from the stack.
```

The jsr has used two bytes on the stack for the return address, and we just pushed A, which uses another byte. So the argument that we pushed on the stack before calling the function will be at position 3 (counting from zero).

Ex5. Returning the result using the stack:

Building on the above example, we can use the same stack location for the input argument to return the value.

```
ARITH
        psha            ;Push A so that it's only modified locally.
        ldaa    3,sp    ;Get the argument.
        adda    #$01    ;Do some work…
                        ;Get ready to return the argument by storing it in the stack.
        staa    3,sp
        pula            ;Restore the A accumulator.
        rts             ;Return.
```

Now in the main program, you can retrieve the return value. The index is now zero for this item, because the PULA removed one byte, and the RTS removed two bytes from the stack. Actually, the bytes aren't "removed," but the stack pointer is moved.

```
        ldaa 0,sp       ;Retrieve the result.
```

Ex6. Printing a hex byte value:

Since OUTA will translate the value in A to an ASCII character, we can't use it to print the result of our arithmetic. Instead, we can use OUT1BYT, which prints a hex value to the screen.

```
TMP             equ     $5000
```

```
OUT1BYT     equ     $FF52

            org     $4000

            ldaa    #$41        ;ASCII value for 'A'
            staa    TMP         ;Store it.
            ldx     #TMP        ;Load the address in X.
            jsr     OUT1BYT     ;Print the ASCII value.
```

Problem 1:

Write a program to prompt the user for a string, and then determine whether the string is a palindrome or not. A palindrome is a word or phrase that is spelled the same forwards and backwards, such as: "A Santa, at NASA."

The program should convert the string to uppercase (using the MON12 function), and remove all non-alphabetic characters, before determining whether the string is a palindrome or not. Then it should print out the uppercase string, followed by a message indicating whether the string was a palindrome.

The program should then prompt the user for another string. If the user enters a period "." the program should terminate.

Example program run:

Enter a string:
A Santa, at NASA.
ASANTAATNASA is a palindrome.

Enter a string:
Frog
FROG is not a palindrome.

Enter a string:
.
(program ends).

Demonstrating Your Results:

To demonstrate your results, run the program and enter a palindrome, and a non-palindrome to show that both cases work.

Report Format and Grading:

Following the report format in your syllabus, include the following in your report:

1. Your name, student number, lab project number and title, course number, lab section number, and date.

2. Description of the lab in your own words. What did you learn? If your code did not work in the lab, explain why. (45% of report grade)

3. The source code for your program. Use comments to indicate what changes you made to the program template. (45% of report grade)

4. A short evaluation of the lab. What did you like about the lab? What could be improved? (10% of report grade)