Micro Assembly Language Programming
Lab 8 - Using D-Bug12 Monitor User Accessible Routines

Name: _____ Lab Group: _____

Introduction
The purpose of this exercise is to write HCS12 programs that use the D-BUG12 monitor user accessible utility subroutines.

The D-Bug 12 monitor has a number of user accessible utility routines what we can use in our programs. In this exercise you will use some of the D-Bug12 utility routines to communicate with the terminal.

Part 1: Reviewing D-BUG12 Utility Routines
1. The end of this exercise has information on some of the D-Bug12 Monitor utility routines. Read this information carefully. Refer to the textbook and/or to the Reference Guide for D-Bug12 (Appendix C) for additional information.
2. Fill in the table on the Result Sheet for the selected routines. Show the subroutine name, the address of the subroutine, the type of pointer and the C prototype.

Part 2: Using the Built-in Subroutines
At the end of this lab there are 2 programs (Lab8a and Lab8b) that demonstrate the use of some of the built-in subroutines. Create these programs and run them. Examine the code so you understand how the utility routines are used.

Questions (Answer on the result sheet)
1. What is the purpose of the PSHB and PULB instructions in Lab8a?
2. What is the purpose of the DB CR,LF at the start of MSG2 in the first sample program, Lab8a.
3. What would happen if the LF, CR are not at the start of MSG2?
4. What is the purpose of the ANDB #%00001111 instruction in Lab8b and why does it work?
5. What other CPU instruction could you use that would do the same job as the ANDB #%00001111 instruction?

Part 3: Write a Program
Write an assembly language program that:
1. Prompts the user to enter a number (0 to 9) from the keyboard
2. Accepts a single digit input from the keyboard - Echoes the character on screen
3. Converts the character into the numerical value of the digit and save the value by pushing it on the Stack
4. Prints out a final message
5. Retrieves the number from Stack and displays the number that was entered

Testing (Answer on the result sheet)

Test your program. Verify that it does everything it is supposed to do.
1. Describe how you tested your program.
2. What happens if the user does not enter anything (just presses Enter)?
3. What happens if the user enters a character other than a digit?

You must demonstrate your working program and attach the printout of the source code to the Result Sheet.

Sample Programs

```
;*******************************************************
;*     Program Name: Lab8a.asm
;*     Course: ceng208
;*     Name: <your name>
;*     Date: <today's date>*
;*
;*   Demonstration code: Displays messages on the screen
;*   and outputs the numerical value in Accumulator B
;*   using  built-in subroutines of the D-Bug12 Monitor.
;*
;*******************************************************
;* Initialization of values
CR:       EQU   $0D
LF:       EQU   $0A
out2hex:  EQU   $EE9C   ;Output 8-bits as hex
printf:   EQU   $EE88

;* Define the data
          ORG   $1000
MSG1:     FCC   "The contents of accumulator B is: "
          DB    0
MSG2:     DB    CR,LF
          FCC   "All done"
          DB    CR,LF,0

;* Start of the programme
          ORG   $2000
          LDS   #$3C00        ;Initialize SP to end of RAM
          LDAB  #$12          ;Load a number into Acc B
          PSHB
          LDD   #MSG1         ;Point to first message and display
          JSR   [printf-*-4,PC]
          PULB
          call  [out2hex-*-4,PC];Send character in B

          LDD   #MSG2          ;Display second message
          JSR   [printf-*-4,PC]
          SWI                  ;Return to monitor
          END
```

Figure 8.1 - Sample Code for Output (Text, Number, and New Line) to Terminal

```
;*********************************************
;*    Program Name: Lab8b.asm
;*    Course: ceng208
;*    Name: <your name>
;*    Date: <today's date>
;*
;*   Demonstration code: Read a digit character
;*   from the keyboard and put the numeric value
;*   in Register B.  The program uses the
;*   built-in subroutines printf and getchar
;*
;*********************************************
;*Initialization of values
NULL:     EQU   $00         ;End of text
printf:   EQU   $EE88       ;Output message
getchar:  EQU   $EE84       ;Read ASCII character

;* Define the data
          ORG   $1000
MSG1:     FCC   "Enter a number : "
          DB    NULL

;* Start of the programme
          ORG   $2000
          LDS   #$3C00            ;Initialize SP to end of RAM
          ;Display prompt message
          LDD   #MSG1
          JSR   [printf-*-4,PC]

          JSR   [getchar-*-4,PC]  ;Get character
          ANDB  #%00001111        ;Convert ASCII code to value
          SWI                     ;Return to monitor
          END
```

Figure 8.2 Sample Code for Input from Terminal

D-Bug12 Monitor Utility Routines
The D-Bug12 Monitor contains routines that are available for use by programmer in application program.

Some of the more useful routines are summarized below. A more complete list is given in your text book or the Reference Guide for D-Bug12 (Appendix C).

Function	Description	Pointer Address
getchar()	Get a character from SCI0 or SCI1	$EE84
putchar()	Send a character out SCI0 or SCI1	$EE86
printf()	Formatted Output - Translates binary values to characters	$EE88
far GetCmdLine()	Obtain a line of input from the user	$EE8A
far sscanhex()	Convert an ASCII hexadecimal string to a binary integer	$EE8E
isxdigit()	Checks for membership in the set [0..9, a..f, A..F]	$EE92
toupper()	Converts lower case characters to upper case	$EE94
isalpha()	Checks for membership in the set [a..z, A..Z]	$EE96
strlen()	Returns the length of a null terminated string	$EE98
strcpy()	Copies a null terminated string	$EE9A
far out2hex()	Displays 8-bit number as 2 ASCII hex characters	$EE9C
far out4hex()	Displays 16-bit number as 4 ASCII hex characters	$EEA0

Table 2 – D-Bug12 Utility Routines

Accessing Functions
The addresses in the table are the location of pointers to the functions, not the location of the function. Indirect addressing must be used to access these function.

The regular functions can be accessed using the JSR instruction. This syntax works with AsmIDE:

```
jsr [getchar -*-4,pc]   "getchar-*-4" - calculates the offset of getchar from
                        the current value of the Program Counter. 4 is the size
                        of the jsr instruction.
```

Another more straight forward way is to use indirect indexed addressing using the X or Y registers:

```
ldx #0
jsr [getchar,x]
```

The far functions are located in the banked memory. The far function pointers are 24-bits long because they include the value of the Program Page (PP) register. Far functions must be accessed using the CALL instruction.

```
call [out2hex -*-4,pc]
```

The alternative using indirect indexed addressing with the X registers is:

```
ldx #0
call [out2hex,x]
```

Function Calling Convention
These functions were written in C and use the C calling convention. If there is a single argument it is passed in accumulator D. If there is more than one argument they are pushed on the stack in reverse order except for the first one which is passed in Register D.

The code following the CALL or JSR instruction must remove any arguments pushed onto the stack.

All 8 and 16-bit function results (e.g., getchar, out2hex, out4hex) are returned in accumulator D. char values returned in accumulator D are located in the 8-bit accumulator B. Boolean function results are zero for False and non-zero values for True.

Registers
The CPU registers are used by the functions and are not saved (except the Stack Pointer) by the functions. If any of the register values need to be saved, they should be pushed onto the stack before any of the arguments are pushed and restored after deallocating the arguments.

Examples
If you want to display the string located at address startMSG you should define printf as $EE88, load register D with the starting address of the null terminated string and output the message to the terminal as follows:

```
printf:    EQU  $EE88
CR:        EQU  $0D
LF:        EQU  $0A

startMSG:  fcc  "The program is starting"
           db   CR,LF,0
             :
           ldd  #startMSG
           jsr  [printf-*-4,pc]
             :
```

The equivalent C code is: printf("The program is starting\n").

A more complicated use of printf requires the arguments to be pushed on the stack in reverse order except for the pointer to the format string which is loaded into register D. For example:

```
; printf("Number1(decimal)=%d. Number 2(hex)=%x\n",num1,num2);
string:    fcc  "Number1(decimal)=%d. Number2(hex)=%x"
           db   CR,LF,0
num1:      dw   2043
num2:      dw   255
            :
           ldd  num2
           pshd
           ldd  num1
           pshd
           ldd  #string
           jsr  [printf-*-4,pc]
           puld             ;clean up stack
           puld
            :
```

Note that the printf function recognizes the standard format specifiers (e.g., %s, %d, %x, etc.) but does not recognize the special characters such as '\n'.