

Department of Electrical and Computer Engineering

College of Engineering and Applied Sciences

WESTERN MICHIGAN UNIVERSITY



ECE 2510/4510/5530

HC12 Microprocessor

D-Bug12 Functions:
Stepping Through Programs
& Performing I/O

Dr. Bradley J. Bazuin

Associate Professor

Department of Electrical and Computer Engineering

College of Engineering and Applied Sciences

Basic D-Bug12 Debugging Operations

- Executing a program starting at an instruction (g 122A)
- Executing a program until some critical instruction, go until (gt <addr>)
 - Also <RegisterName> and <RegisterName> <RegisterValue>
- Tracing program execution (t)
- Setting and clearing program breakpoints (br, nobr)
- Modifying (mm, mmw) and displaying (md, mdw) memory locations
 - Also a block fill (bf) command for multiple locations
 - Also move a block of memory (move)
- Modifying (rm) and displaying (rd) registers

D-Bug12 Command Summary (1)

• ALTCLK - Specify an alternate BDM communications rate.	22
• ASM - Single line assembler/disassembler.	24
• BAUD - Set the SCI communications BAUD rate.	28
• BDMDB - Enter the BDM command debugger.	29
• BDMPGMR - Load BDM Programmer Firmware Image Into MCU Flash	30
• BF - Block Fill user memory with data.	32
• BR - Set/Display user breakpoints.	33
• BS - Block Search, Search an Address Range For A Data Pattern	35
• BULK - Bulk erase on-chip EEPROM	37
• CALL - Execute a user subroutine, return to D-Bug12 when finished.	38
• DEVICE - Select/define a new target MCU device.	39
• EEBASE - Inform D-Bug12 of the target's EEPROM base address	42
• FBULK - Erase the target processor's on-chip Flash EEPROM	44
• FLOAD - Program the target processor's on-chip Flash EEPROM from S-Records	46
• FSERASE - Erase one or more sectors of target Flash	49
• G - Go. Begin execution of user program.	51
• GT - Go Till. Set a temporary breakpoint and begin execution of user program.	52
• HELP - Display D-Bug12 command set and command syntax.	53
• LOAD - Load user program in S-Record format.	55
• MD - Memory Display. Display memory contents in hex bytes/ASCII format.	57
• MDW - Memory Display Words. Display memory contents in hex words/ASCII format.	58
• MM - Memory Modify. Interactively examine/change memory contents.	59
• MMW - Memory Modify Words. Interactively examine/change memory contents.	60

D-Bug12 Command Summary (2)

• MOVE - Move a block of memory.	61
• NOBR - Remove one/all user breakpoints.	62
• PCALL - Execute a user subroutine in expanded memory, return to D-Bug12 when finished.	63
• RD - Register Display. Display the CPU register contents.	64
• REGBASE - Inform D-Bug12 of the target's I/O register's base address	65
• RESET - Reset the target CPU	67
• RM - Register Modify. Interactively examine/change CPU register contents.	68
• SETVFP - Sets nominal Vfp Voltage to 12.0 or 12.6 volts.	69
• SO - Step over subroutine calls.	71
• STOP - Stop the execution of user code in the target processor and place the target processor in background mode.	72
• T - Trace. Execute an instruction, disassemble it, and display the CPU registers.	73
• TCONFIG - Configure target before erasing or programming target Flash	74
• UPLOAD - Display memory contents in S-Record format.	75
• USEHBR - Use EVB/Target Hardware breakpoints.	77
• VER - Display the running version of D-Bug12	78
• VERF - Verify memory contents against S-Record Data.	79
• <RegisterName> <RegisterValue> - Set CPU <RegisterName> to <RegisterValue>	81

D-Bug12 Operation G

- Go, execute the code starting at the current or a specified address (execute until swi, reset or a breakpoint)
 - Execute the following line after the code has been downloaded
G 122A<CR>
 - D-Bug12 will execute program from 122A until an swi in the code, the reset button is pressed, or a user set breakpoint is encountered. It will then stop, print the register values, with the next PC value and next instruction to be executed.

Go (G) Command

G - Begin execution of user code

- Command Line Format

G [<Address> | <PPAGENum>:<PPAGEWinAddr>]

- Parameter Description

<Address> A 16-bit hexadecimal number or simple expression

<PPAGENum> - An 8-bit hexadecimal number or simple expression

<PPAGEWinAddr> - A 16-bit hexadecimal number or simple expression

- Command Description

- The G command is used to begin the execution of user code in real time. Before beginning execution of user code, any breakpoints set using the BR command are placed in memory. Execution of the user program will continue until a user breakpoint is encountered, a CPU exception occurs or the reset switch on the HC12EVB is pressed. When user code halts for one of these reasons and control is returned to D-Bug12, a message is displayed explaining the reason for program termination. In addition, D-Bug12 displays the CPU12's register contents, disassembles the instruction at the current PC address, and waits for the next DBug12 command to be entered by the user.

- If a starting address is not supplied in the command line parameter, program execution will begin at the address defined by the current value of the Program Counter.

D-Bug12 Operation T

- Trace program execution
 - First, set the PC to the program start address \$122a
PC 122A<CR>
 - Type in T and <CR> and D-Bug12 will execute the instruction at the PC and then print the register values, with the next PC value and next instruction to be executed.
 - Repeat a desired
PC 122A<CR>
T<CR>
 - To trace multiple instructions at a time, type T (number)<CR>
PC 122A<CR>
T 10<CR>

Trace (T) Command

T - Trace (Execute) CPU12 Instruction(s)

- Command Line Format

T [<Count>]

- Parameter Description

<Count> An 8-bit decimal number in the range 1..255

- Command Description

- The Trace command is used to execute one or more user program instructions beginning at the current Program Counter (PC) location. As each program instruction is executed, the CPU12's register contents are displayed and the next instruction to be executed is displayed.
- A single instruction may be executed by entering the trace command followed immediately by a carriage.

D-Bug12 Operation GT

- Go until a specific address is reached (a breakpoint)
 - First, set the PC to the program start address \$122a
PC 122A<CR>
 - Type in GT (command address) <CR> and D-Bug12 will execute program from 122A until the specified address. It will then stop, print the register values, with the next PC value and next instruction to be executed. You may then trace (T) or repeat GT.
 - Go Until a desired address
PC 122A<CR>
GT 1234<CR>

Go Until (GT) Command

GT - Go Until, Execute user code until temporary breakpoint is reached

- Command Line Format

GT <Address> | <PPAGENum>:<PPAGEWinAddr>

- Parameter Description

<Address> A 16-bit hexadecimal number or simple expression

<PPAGENum> - An 8-bit hexadecimal number or simple expression

<PPAGEWinAddr> - A 16-bit hexadecimal number or simple expression

- Command Description

- The GT command is similar to the G command except that a temporary breakpoint is placed at the address supplied on the command line. Any breakpoints set by the BR command are NOT placed in the user's code before program execution begins. Program execution begins at the address defined by the current value of the Program Counter. When user code reaches the temporary breakpoint and control is returned to D-Bug12, a message is displayed explaining the reason for user program termination. In addition, D-Bug12 displays the CPU12's register contents, disassembles the instruction at the current PC address, and waits for the next D-Bug12 command to be entered by the user.

D-Bug12 Operations

BR and NOBR

Helpful debugging hints:

- Breakpoints that cause the code to stop at one or more specific locations may be set.
 - To set a breakpoint, provide the appropriate instruction address
BR 1234<CR>
BR 1242<CR>
G 122A
 - D-Bug12 will execute program from 122A until a breakpoint address or swi is encountered.
- One or all breakpoints can be removed using NOBR
NOBR 1234<CR>
NOBR <CR>

Breakpoint (BR) Command

BR - Set/Display User Breakpoints

- Command Line Format
BR [<Address> | <PPAGENum>:<PPAGEWinAddr>...]
- Parameter Description
 - <Address> A 16-bit hexadecimal number or simple expression
 - <PPAGENum> - An 8-bit hexadecimal number or simple expression
 - <PPAGEWinAddr> - A 16-bit hexadecimal number or simple expression
- Command Description
 - The BR command is used to set a breakpoint at a specified address or to display any previously set breakpoints. The function of a breakpoint is to halt user program execution when the program reaches the breakpoint address. When a breakpoint address is encountered, D-Bug12 will disassemble the instruction at the breakpoint address, print the CPU12's register contents, and wait for the next D-Bug12 command to be entered by the user.
 - Breakpoints are set by entering the breakpoint command followed by one or more breakpoint addresses. Entering the breakpoint command without any breakpoint addresses will display all the currently set breakpoints.
 - A maximum of 10 breakpoints may be set at one time when using software breakpoints (default).

No Breakpoints (NOBR) Command

NOBR - Remove one/all user breakpoints

- Command Line Format
NOBR [<Address> | <PPAGENum>:<PPAGEWinAddr>...]
- Parameter Description
 - <Address> A 16-bit hexadecimal number or simple expression
 - <PPAGENum> - An 8-bit hexadecimal number or simple expression
 - <PPAGEWinAddr> - A 16-bit hexadecimal number or simple expression
- Command Description
 - The NOBR command is used to remove one or more of previously entered breakpoints. If the NOBR command is entered without any arguments, all user breakpoints are removed from the breakpoint table.

D-Bug12 Operation MD

- Display memory in hexadecimal bytes and ASCII format
 - Display the values in memory 1000:100F while in the debug mode of operation
 - MD 1000 <CR>
 - The debugger will respond with the start address requested, the next 16 memory byte values in hex, and then the ASCII equivalents of the memory locations.
 - The ASCII fields are useful if the memory is supposed to be ASCII characters. If not, it looks like a mess
 - Alternate command
 - MD 1000 11FF <CR>
 - The debugger will respond with all memory bytes from 1000 to 11FF.

Memory Display (MD) Command

MD - Display memory in hexadecimal bytes and ASCII format

- Command Line Format

MD <StartAddress> [<EndAddress>]

- Parameter Description

<StartAddress> A 16-bit hexadecimal number or simple expression

<EndAddress> A 16-bit hexadecimal number or simple expression

- Command Description

- The memory display command displays the contents of memory in both hexadecimal bytes and ASCII, 16-bytes on each line. The <StartAddress> parameter must be supplied, however, the <EndAddress> parameter is optional. When the <EndAddress> parameter is not supplied, a single line is displayed (16 bytes).
- The number supplied as the <StartAddress> parameter is rounded down to the next lower multiple of 16. While the number supplied as the <EndAddress> parameter is rounded up to the next higher multiple of 16 - 1.

D-Bug12 Operation MM

- Modify memory bytes in hexadecimal format
 - Modify memory byte values beginning at a starting address
MM 1000 <CR>
 - The debugger will respond with the address requested and the byte value. The value can be modified by typing in a new value and hitting a <CR>. The memory will be modified and the next address will be printed to continue sequential modifications. If there are no more modification, type a period (,) <CR>
 - To control the modification point displayed next, use:
 - [<Data>]<CR> update current location and display the next location
 - [<Data>] / or = update current location and redisplay the current location
 - [<Data>] ^ or - update current location and display the previous location
 - [<Data>] . update current location and exit Memory Modify

Memory Display (MM) Command

MM - Modify memory bytes in hexadecimal format

- Command Line Format
MM <Address> [<data>]
- Parameter Description
<Address> A 16-bit hexadecimal number or simple expression.
<data> An 8-bit hexadecimal number.
- Command Description
 - The memory modify word command allows the contents of memory to be examined and/or modified as 8-bit hexadecimal data. If the 8-bit data parameter is present on the command line, the byte at memory location at <Address> is replaced with <data>. If not, D-Bug12 will enter the interactive memory modify mode. In the interactive mode, each byte is displayed on a separate line following the data's address.
 - Once the memory modify command has been entered, several sub-commands are used for the modification and verification of memory contents. These sub-commands have the following format:
 - [<Data>]<CR> Optionally update current location and display the next location
 - [<Data>] / or = Optionally update current location and redisplay the current location
 - [<Data>] ^ or - Optionally update current location and display the previous location
 - [<Data>] . Optionally update current location and exit Memory Modify

D-Bug-12 Operation RD

- Display CPU12 Register Contents
 - Display the contents of the CPU12 registers.
RD <CR>
 - The debugger response 1st line is the register names, the second line is the register contents, and the third line is the current PC, machine language command and disassembled command.
 - It tells you what is in the registers and what the next instruction to be executed is.

Register Display (RD) Command

RD - Display CPU12 Register Contents

- Command Line Format

RD

- Parameter Description

Not applicable

- Command Description

- The Register Display command is used to display the CPU12's registers. The registers are displayed in the same format used when a breakpoint is encountered.

D-Bug-12 Operation RM

- Interactively Modify CPU12 Register Contents
 - Modify the contents of the CPU12 registers.
RM <CR>
 - The debugger response sequentially with each of the registers. If you want to modify it, provide a value. If not, step to the next one with a <CR> or exit the modification mode with a period (.)

Register Modify (RM) Command

RM - Interactively Modify CPU12 Register Contents

- Command Line Format

Rm

- Parameter Description

Not applicable

- Command Description

- The register modify command is used to examine and/or modify the contents of the CPU12's registers in an interactive manner. As each register and its contents is displayed, D-Bug12 allows the user to enter a new value for the register in hexadecimal. If modification of the displayed register is not desired, entering a carriage return causes the next CPU12 register and its contents to be displayed on the next line. Typing a period (.) as the first non space character on the line will exit the interactive mode of the register modify command and return to the D-Bug12 prompt.
- The registers are displayed in the following order, one register per line: PC, SP, X, Y, A, B, CCR.

<RegisterName> <RegisterValue> Command (1 of 2)

<RegisterName> - Modify a CPU12 Register Value

- Command Line Format
 - <RegisterName> <RegisterValue>
- Parameter Description
 - Where <RegisterName> is one of the following CPU12 register names:

Register	Name Description	Legal Range
• PC	Program Counter	\$0..\$FFFF
• SP	Stack Pointer	\$0..\$FFFF
• X	X-Index Register	\$0..\$FFFF
• Y	Y-Index Register	\$0..\$FFFF
• A	A Accumulator	\$0..\$FF
• B	B Accumulator	\$0..\$FF
• D	D Accumulator (A:B)	\$0..\$FFFF
• CCR	Condition Code Register	\$0..\$FF
• PP	PPAGE Register	\$0..\$FF

- For each of the CPU register names, <RegisterValue> may be a hexadecimal number or a simple expression.

<RegisterName> <RegisterValue> Command (2 of 2)

<RegisterName> - Modify a CPU12 Register Value

- Command Line Format
 - <RegisterName> <RegisterValue>
- Parameter Description
 - Each of the fields in the CCR may also be modified by using the following field Names:

CCR Bit	Name Description	Legal Range
• S	STOP Enable	0..1
• H	Half Carry	0..1
• N	Negative Flag	0..1
• Z	Zero Flag	0..1
• V	Overflow Flag	0..1
• C	Carry Flag	0..1
• IM	IRQ Interrupt Mask	0..1
• XM	XIRQ Interrupt Mask	0..1

- For the CCR bit names only a value of zero or one may be supplied for the <RegisterValue> parameter.

D-Bug-12 Operation ASM

- Single Line Assembler/Disassembler Command
 - Disassemble the machine language to assembly language with absolute addresses and values inserted for variables and labels
ASM 122A <CR>
 - This allows you to compare your original code to the code in the CPU12 that will execute.
 - All conveniences of the assembly language are removed. So the actual hexadecimal variable values, label addresses, and branch offset values are shown. This is what the machine is really running!

Assemble/Disassemble (ASM) Command (1)

ASM - Single Line Assembler/Disassembler Command

- Command Line Format
ASM <Address> | <PPAGENum>:<PPAGEWinAddr>
- Parameter Description
 - <Address> - A 16-bit hexadecimal number or simple expression
 - <PPAGENum> - An 8-bit hexadecimal number or simple expression
 - <PPAGEWinAddr> - A 16-bit hexadecimal number or simple expression
- Command Description
 - The assembler/disassembler is an interactive memory editor that allows memory contents to be viewed and altered using assembly language mnemonics. Each entered source line is translated into machine language code and placed into memory at the time of entry. When displaying memory contents, each instruction is disassembled into its source mnemonic form and displayed along with the hexadecimal machine code and any instruction operands.

Assemble/Disassemble (ASM) Command

(2)

ASM - Single Line Assembler/Disassembler Command

- **Disassembler Command Description**
 - When displaying memory contents, each instruction is disassembled into its source mnemonic form and displayed along with the hexadecimal machine code and any instruction operands.
 - When an instruction has been disassembled and displayed, the D-Bug12 prompt is displayed following the disassembled instruction. If a carriage return is entered immediately following the prompt, the next instruction in memory is disassembled and displayed on the next line.
- **Assembler Command Description**
 - Assembler mnemonics and operands may be entered in any mix of upper and lower case letters. Any number of spaces may appear between the assembler prompt and the instruction mnemonic or between the instruction mnemonic and the operand. By default, numeric values appearing in the operand field are interpreted as signed decimal numbers. Placing a \$ in front of a number will cause the number to be interpreted as a hexadecimal number.
 - If a CPU12 instruction is entered following the prompt, the entered instruction is assembled and placed in memory. The line containing the new entry is erased and the new instruction is disassembled and displayed on the same line. The contents of the next memory location(s) is disassembled and displayed on the screen.
 - The instruction mnemonics and operand formats accepted by the assembler follow the syntax as described in the M68HC12 Family CPU12 Reference Manual..

DEBUG-12 SUBROUTINES

D-Bug12 Functions to Perform I/O

- D-Bug12 monitor provides a few subroutines to support I/O operations
 - I/O routines can be utilized to facilitate program developments on a demo board that contains the D-Bug12 monitor
- User accessible routines are written in C
 - All except the first parameter are passed to the user callable functions on the stack
 - Parameters are to be pushed in the reverse order they are listed in the function declaration
 - First parameter is passed to the function in D accumulator

User Accessible Utility Functions

Function	Description	Pointer Address
<code>far main()</code>	Start of D-Bug12	\$EE80
<code>getchar()</code>	Get a character from SCI0 or SCI1	\$EE84
<code>putchar()</code>	Send a character out SCI0 or SCI1	\$EE86
<code>printf()</code>	Formatted Output - Translates binary values to characters	\$EE88
<code>far GetCmdLine()</code>	Obtain a line of input from the user	\$EE8A
<code>far sscanfhex()</code>	Convert an ASCII hexadecimal string to a binary integer	\$EE8E
<code>isxdigit()</code>	Checks for membership in the set [0..9, a..f, A..F]	\$EE92
<code>toupper()</code>	Converts lower case characters to upper case	\$EE94
<code>isalpha()</code>	Checks for membership in the set [a..z, A..Z]	\$EE96
<code>strlen()</code>	Returns the length of a null terminated string	\$EE98
<code>strcpy()</code>	Copies a null terminated string	\$EE9A
<code>far out2hex()</code>	Displays 8-bit number as 2 ASCII hex characters	\$EE9C
<code>far out4hex()</code>	Displays 16-bit number as 4 ASCII hex characters	\$EEA0
<code>SetUserVector()</code>	Setup user interrupt service routine	\$EEA4
<code>far WriteEEByte()</code>	Write a data byte to on-chip EEPROM	\$EEA6
<code>far EraseEE()</code>	Bulk erase on-chip EEPROM	\$EEAA
<code>far ReadMem()</code>	Read data from the M68HC12 memory map	\$EEAE
<code>far WriteMem()</code>	Write data to the M68HC12 memory map	\$EEB2

Note: far name must use CALL others use JSR

D-Bug12 Functions to Perform I/O

- Parameter Restrictions
 - Parameters of type char must be converted to an integer
 - Parameters of type char will occupy the lower order byte of a word pushed onto the stack or accumulator B if the parameter is passed in D accumulator
 - All 8-bit and 16-bit function values are returned in accumulator D
 - A value of type char returned in accumulator D is located in the 8-bit accumulator B
- None of the CPU registers contents, except the stack pointer are preserved by the called functions
 - Register values needs to be pushed onto the stack before any parameters have been pushed onto the stack and should be restored after de-allocating the parameters

Calling D-Bug12 Functions from Assembly

- Calling the functions from assembly involves
 - Pushing the parameters onto the stack in the proper order
 - Loading the first or only function parameter into accumulator D
 - Calling the D-Bug12 function with a JSR instruction
 - Code following the JSR instruction should remove any parameters pushed onto the stack
- Addressing mode used in the JSR instruction is Indexed Indirect Addressing Mode

Ex: JSR [getchar,PCR] → PCR is an index register X

Functions Called using JSR

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
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
SetUserVector()	Setup user interrupt service routine	\$EEA4

- Basic IO functions with a few additions

getchar

Int getchar (void)

- Pointer Address: \$EE84
 - This function receives a single character from the control terminal (Serial Communication Interface or keyboard)
 - The character is returned as an integer, the 8-bit character is placed in accumulator B

Example of getchar Function

- Ex:

```
Getchar = $EE84
```

```
-----
```

```
-----
```

```
-----
```

```
Jsr [getchar,x]
```

```
-----
```

```
-----
```

- This code will read one character typed on the keyboard

putchar

Int putchar(int)

- Pointer Address: \$EE86
 - This function outputs a single character to the control terminal Serial Communication interface

Ex:

```
Putchar = $EE86
```

```
----
```

```
----
```

```
Ldd #32
```

```
Jsr [putchar,x]
```

```
---
```

- Output one ASCII character to the terminal (a 2 is printed)

printf

Int printf(char *format,-----)

- Pointer Address: \$EE88
 - This function is used to convert, format and print its argument on the standard output monitor, LCD under the control of the format string pointed to by format.
 - All except floating-point data types are supported.
- The format string contains two basic types of objects:
 - ASCII characters which will be copied directly to the display device.
 - Conversion specifications. Each conversion specification begins with a percent sign (%).

printf Optional Formatting

- Conversion specifications. Each conversion specification begins with a percent sign (%).
- Optional formatting characters may appear between the percent sign and ends with a single conversion character in the following order: [-][<FieldWidth>][.][<Precision>][h | l]

Table 4.3 Optional formatting characters

Character	Description
-(minus sign)	Left justifies the converted argument.
FieldWidth	Integer number that specifies the minimum field width for the converted argument. The argument will be displayed in a field at least this wide. The displayed argument will be padded on the left or right if necessary.
.(period)	Separates the field width from the precision.
Precision	Integer number that specifies the maximum number of characters to display from a string or the minimum number of digits for an integer.
h	To have an integer displayed as a short.
l(letter ell)	To have an integer displayed as a long.

printf Conversion Characters

Table 4.4 Printf() conversion characters

character	Argument type; displayed as
d, i	int; signed decimal number
o	int; unsigned octal number (without a leading zero)
x	int; unsigned hex number using abcdef for 10...15
X	int; unsigned hex number using ABCDEF for 10...15
u	int; unsigned decimal number
c	int; single character
s	char *; display from the string until a '\0' (NULL)
p	void *; pointer (implementation-dependent representation)
%	no argument is converted; print a %

- This set is universal to applications based on C (e.g. MATLAB)

printf Example

```
printf = $EE88
prompt: .asciz "Flight Simulation\n"
----
----
ldd #prompt
jsr [printf,x]
----
----
```

- This instruction sequence will cause the message “Flight Simulation” to be displayed and cursor will be moved to the beginning of next line.

printf Example 2

- Suppose labels m, n and gcd represent three memory locations and gcd holds greatest common divisor and m ,n represent the two numbers

```
printf = $EE88
```

```
prompt .asciz "The Greatest Common Divisor of %d and %d is %d\n"
```

```
ldd gcd
```

```
pshd
```

```
ldd n
```

```
pshd
```

```
ldd m
```

```
pshd
```

```
ldd #prompt
```

```
jsr [printf,x]
```

```
leas 6,SP
```

GetCmdLine

Int far GetCmdLine(char *cmdLineStr, int CmdLineLen)

- Pointer Address: \$EE8A
 - This function is used to get line of input from the user
 - GetCmdLine accepts inputs from the user one character at a time by calling getchar() function
 - The received characters are placed in the character array pointed by the cmdLineStr using putchar() instruction
 - A maximum of CmdLineLen-1 characters may be entered
 - Only printable ascii characters are accepted as inputs

GetCmdLine Example

```
Printf = $EE88
GetCmdLine = $EE8A
Prompt: .asciz "Please Enter a String\n"
Inbuf: .blkb 100
Cmdlinelen: .byte 100
        ldd Prompt
        jsr [printf,x]
        ldd Cmdlinelen
        pshd
        ldd Inbuf
        call [GetCmdLine,x]
```

toupper

Int toupper(int c)

- Pointer Address: \$EE94
 - If c is a lowercase character, toupper() will return the corresponding uppercase letter
 - If the character is in uppercase, it simply returns c

- Example:

```
        toupper = $EE94
C_buf:  .asciz "a"
        ldab c_buf
        clra
        jsr [toupper,x]
```

isalpha

Int isalpha(int c)

- **Pointer Address: \$EE96**
 - This function tests the character passed in c for membership in character set (a-z, A-Z)
 - If the character c is in this set, the function returns a non-zero value, else returns a zero value
 - This function is useful for validating an input string
- **Example:**

```
Isalpha = $EE96
c_buf:  .ascii "b"
ldab c_buf
clra
jsr [isalpha,x]
```

strlen

Unsigned int strlen(const char *cs)

- Pointer Address: \$EE98
 - The strlen() function returns the length of the string pointed to by cs

- Example:

```
        strlen = $EE98
cs:     .asciz "-----"
        ldd #cs
        jsr [strlen,x]
```

strcpy

Char *strcpy(char *s1, char *s2)

- Pointer Address: \$EE9A
 - This function copies the string pointed to by s2 into the string pointed to by s1
- Example:

```
strcpy = $EE9A
S2: .asciz "-----"
S1: .blkb ---
    ldd #S2
    pshd
    ldd #S1
    jsr [strcpy,x]
    leas 2,SP
```