HCS12 Instruction Examples

- We will divide the instruction set to several groups by function to develop a feel for HCS12 instructions.

The Load and Store Instructions:

Load Instructions		
Mnemonic	Function	Operation
LDAA	Load A	(M) → A
LDAB	Load B	(M) → B
LDD	Load D	(M:M+1) → (A:B)
LDS	Load SP	(M:M+1) → SP
LDX	Load Indexed Register X	(M:M+1) → X
LDY	Load Indexed Register Y	(M:M+1) → Y
LEAS	Load Effective Address into SP	Effective Address → SP
LEAX	Load Effective Address into X	Effective Address → X
LEAY	Load Effective Address into Y	Effective Address → Y

Store Instructions

Mnemonic	Function	Operation
STAA	Store A	(A) → M
STAB	Store B	(B) → M
STD	Store D	(A) → M, (B) → M+1
STS	Store SP	(SP) → M:M+1
STX	Store X	(X) → M:M+1
STY	Store Y	(Y) → M:M+1

Examples:

```
LDAA    0,X       ;Eff Add = [X]+0, [Eff Add] → A
LDAB    $1024     ;[$1024] → B
STX     $8000     ;[X(15:8)] →$8000, [X(7:0)] → $8001
LEAS    125,X     ;Eff Add = [X] + 125, Eff Add → SP
```

The Load and Store Instructions *continued …*

Messages | Terminal

```
D-Bug12 4.0.0b24
Copyright 1996 - 2002 Motorola Semiconductor
For Commands type "Help"

>X 1100


PP  PC    SP    X     Y    D = A:B   CCR = SXHI NZVC
38 0000  3C00  1100  0000     00:00        1001 0000
xx:0000  FF0000        LDS   $0000
>ASM 1000

xx:1000  32            PULA                       >LEAS 125,X


xx:1000  1BE07D        LEAS  125,X
xx:1003  5DB3          STY   $00B3                >.

>PC 1000


PP  PC    SP    X     Y    D = A:B   CCR = SXHI NZVC
38 1000  3C00  1100  0000     00:00        1001 0000
xx:1000  1BE07D        LEAS  125,X
>T


PP  PC    SP    X     Y    D = A:B   CCR = SXHI NZVC
38 1003  117D  1100  0000     00:00        1001 0000
xx:1003  5DB3          STY   $00B3

>
```

1: 1 | | Insert

Transfer and Exchange Instructions:

Transfer Instructions		
Mnemonic	Function	Operation
TAB	Transfer A to B	(A) → B
TAP	Transfer A to CCR	(A) → CCR
TBA	Transfer B to A	(B) → A
TFR	Transfer Register to Register	(A,B,CCR,D,X,Y,or SP) → A,B,CCR,D,X,Y,or SP
TPA	Transfer CCR to A	(CCR) → A
TSX	Transfer SP to X	(SP) → X
TSY	Transfer SP to Y	(SP) → Y
TXS	Transfer X to SP	(X) → SP
TYS	Transfer Y to SP	(Y) → SP

Exchange Instructions		
Mnemonic	Function	Operation
EXG	Exchange Register to Register	(A,B,CCR,D,X,Y,or SP) ↔ A,B,CCR,D,X,Y,or SP
XGDX	Exchange D with X	(D) ↔ X
XGDY	Exchange D with Y	(D) ↔ y

Sign Extension Instructions

Mnemonic	Function	Operation
SEX	Sign Extended 8-bit Operand	(A, B, CCR) $\leftrightarrow$ X, Y, or SP

Examples:

TAB		;B $\leftarrow$ [A]
MUL		;A:B $\leftarrow$ [A] $\times$ [B], gives you $[A]^2$
TFR	A,X	;X $\leftarrow$ \$00:[A], A is zero extended to 16-bit
TFR	X,B	;B $\leftarrow$ X[7:0], B receives bits 7 to 0 of X
EXG	Y,A	;Y $\leftarrow$ \$00:[A], A $\leftarrow$ Y[7:0]
EXG	B,X	;B $\leftarrow$ X[7:0], X $\leftarrow$ \$00:[B]
SEX	A,X	;If X=0000 & A=85, after execution, X=FF85 & A=85

Transfer Instructions

Mnemonic	Function	Operation
MOVB	Move byte (8-bit)	$[M_1] \rightarrow M_2$
MOVW	Move word (16-bit)	$[M_1:M_1+1] \rightarrow M_2:M_2+1$

Add and Subtract Instructions:

Add Instructions		
Mnemonic	Function	Operation
ABA	Add B to A	(A) + (B) → A
ABX	Add B to X	(B) + (X) → X
ABY	Add B to Y	(B) + (Y) → Y
ADCA	Add with Carry to A	(A) + (M) + C → A
ADCB	Add with Carry to B	(B) + (M) + C → B
ADDA	Add without Carry to A	(A) + (M) → A
ADDB	Add without Carry to B	(B) + (M) → B
ADDD	Add without Carry to D	(A:B) + (M:M+1) → A:B

Subtract Instructions		
Mnemonic	Function	Operation
SBA	Subtract B from A	(A) - (B) → A
SBCA	Subtract with Borrow from A	(A) - (M) - C → A
SBCB	Subtract with Borrow from B	(B) - (M) - C → B
SUBA	Subtract memory from A	(A) - (M) → A
SUBB	Subtract memory from B	(B) - (M) → B
SUBD	Subtract memory from D	(D) - (M:M+1) → D

Example 1:

➢ Write an instruction sequence to subtract the content of accumulator B from the 16-bit word at $1000-$1001 and store the difference at $1100-$1101.

Solution 1:

```
LDAA   $1001      ; get LS Byte
SBA               ; subtract from B
STAA   $1101      ; store in LS Byte of result
LDAA   $1000      ; load MS Byte
SBCA   #0         ; subtract carry from it
STAA   $1100      ; store it back
```

Solution 2:

```
STAB   $1101      ; store B as LS Byte
CLR    $1100      ; clear MS Byte
LDD    $1000      ; load D w/16-bit number
SUBD   $1100      ; subtract from it 16-bit number
STD    $1100      ; store it back
```

Example 2:

➢ Write an instruction sequence to swap the 16-bit word stored at $1000-$1001 with the 16-bit word stored at $1100-$1101.

Solution 1:

```
LDD   $1000              ; pick up the word from $1000
MOVW  $1100,$1000        ; move a word from $1100 to $1000
STD   $1100              ; replace $1100 word with $1000
```

Solution 2:

```
LDD   $1000              ; pick up the word from $1000
LDX   $1100              ; pick up the word from $1100
STD   $1100              ; replace $1100 word with $1000
STX   $1000              ; replace $1000 word with $1100
```

Example 3:

➢Give an instruction that can store the contents of indexed register Y at the memory location with an address smaller than the contents of X by 10.

Solution:

```
STY   -10,X              ; pick up the word from $1000
```

Example 4:

➢Write a program to subtract the hex numbers stored at \$1010-\$1013 from the hex number stored at \$1000-\$1003 and save the difference at \$1020-\$1023.

Solution 1:

	ORG	\$1000		LDAA	num1+2
num1	DB	\$D9,\$2A,\$CC,\$85		SBCA	num2+2
	ORG	\$1010		STAA	res+2
num2	DB	\$9C,\$72,\$B9,\$AA		LDAA	num1+1
	ORG	\$1020		SBCA	num2+1
res	RMB	16		STAA	res+1
				LDAA	num1
	ORG	\$1030		SBCA	num2
	LDAA	num1+3		STAA	res
	SUBA	num2+3		SWI	
	STAA	res+3		END	

Different Solutions:

```
        ORG     $1000
num1    DB      $D9,$2A,$CC,$85
        ORG     $1010
num2    DB      $9C,$72,$B9,$AA
        ORG     $1020
res      RMB    16

        ORG     $1030
        LDX     #num1+3   ; set pointer at LS-Byte
        LDAB    #4        ; set the counter
        CLC               ; no need for borrow on 1st subtraction
loop    LDAA    0,X       ; pick up byte
        SBCA    $10,X     ; subtract with carry same order byte of 2nd number
        STAA    $20,X     ; store the result of subtraction
        DEX               ; update pointer
        DECB              ; decrement counter
        BNE     loop      ; if not zero, continue subtracting
        SWI               ; stop execution and return control to monitor
        END
```

AsmIDE - C:\DRAGON12\examples\temp.asm

File Edit Build View Help

```
        ORG     $1000
num1    FCB     $D9,$2A,$CC,$85
        ORG     $1010
num2    DB      $9C,$72,$B9,$AA
        ORG     $1020
res     RMB     16

        ORG     $1030
        LDAA    num1+3
        SUBA    num2+3
        STAA    res+3
        LDAA    num1+2
        SBCA    num2+2
        STAA    res+2
        LDAA    num1+1
        SBCA    num2+1
        STAA    res+1
        LDAA    num1
        SBCA    num2
        STAA    res
        SWI
        END
```

Messages Terminal

```
>G 1030

User Bkpt Encountered

PP  PC    SP    X     Y     D = A:B   CCR = SXHI NZVC
38 1055  3C00  0000  0000      3C:00         1001 0000
xx:1055  00                BGND

>MD 1000 1040

1000   D9 2A CC 85 - 00 00 00 00 - 00 00 00 00 - 00 00 00 00     .*..............
1010   9C 72 B9 AA - 00 00 00 00 - 00 00 00 00 - 00 00 00 00     .r..............
1020   3C B8 12 DB - 00 00 00 00 - 00 00 00 00 - 00 00 00 00     <...............
1030   B6 10 03 B0 - 10 13 7A 10 - 23 B6 10 02 - B2 10 12 7A     ......z.#......z
1040   10 22 B6 10 - 01 B2 10 11 - 7A 10 21 B6 - 10 00 B2 10     ."......z.!.....
>
```

AsmIDE - C:\DRAGON12\examples\lect4_3.asm

File Edit Build View Help

```
        ORG     $1000
num1    FCB     $D9,$2A,$CC,$85        ;First number (MSByte 1st)
        ORG     $1010
num2    FCB     $9C,$72,$b9,$AA        ;Seconf number (MSByte 1st)

        ORG     $1030
        LDX     #num1+3                ;Set X as pointer to LSD of 1st number
        CLC                            ;Make sure borrow bit is clear
        LDAB    #4                     ;Set counter
loop    LDAA    0,X                    ;Load LSByte of 1st number and subtract
        SBCA    $10,X                   ;from LSByte of 2nd number and store
        STAA    $20,X                    ;the result into LSByte of result.
        DEX                            ;Move the pointer to next byte and
        DECB                            ;continue looping as long counter
        BNE     loop                     ;is not equal to zero.
        SWI                            ;Stop the operation and give the control
*                                       ;back to monitor.
        END
```

Messages | Terminal

```
>BF 1000 105F 0

>LOAD

*
>

>G 1030

User Bkpt Encountered

PP  PC     SP     X      Y     D = A:B    CCR = SXHI NZVC
38 1044   3C00   0FFF   0000      3C:00          1001 0100
xx:1044   00                BGND

>MD 1000 1040


1000   D9 2A CC 85 - 00 00 00 00 - 00 00 00 00 - 00 00 00 00    .*..............
1010   9C 72 B9 AA - 00 00 00 00 - 00 00 00 00 - 00 00 00 00    .r..............
1020   3C B8 12 DB - 00 00 00 00 - 00 00 00 00 - 00 00 00 00    <...............
1030   CE 10 03 10 - FE C6 04 A6 - 00 A2 E0 10 - 6A E0 20 09    ............j...
1040   53 26 F4 3F - 00 00 00 00 - 00 00 00 00 - 00 00 00 00    S&.?............
>
```

Using the D-Bug12 Functions to Perform I/O Operations

- The D-Bug12 monitor provides a few *functions* (*subroutines*) to support I/O operations.
- These *functions* can be used to facilitate program developments on a demo board.
- These user-accessible routines are written in C.
- All except the first parameter are passed to the user-callable *functions* on the stack.
- Parameters must be pushed onto the stack in the *reverse order* they are listed in the function declaration (*right-to-left*).
- The first parameter is passed to the function in accumulator D.
- If a function has only a single parameter, then the parameter is passed in accumulator D.
- Parameters of type *char* must be converted to an *integer* (16-bit).

Using the D-Bug12 Functions to Perform I/O Operations *continued …*

- Parameters of type *char* will occupy the lower-order byte (higher address) of a word pushed onto the stack or accumulator B if the parameter is passed in D.
- Parameters passed onto the stack before the function is called, remain on the stack when the function returns.
- It is the responsibility of the caller to **remove** passed parameters from the stack.
- All 8- and 16-bit function values are returned in the accumulator D.
- A value of type *char* returned in accumulator D is located in the 8-bit accumulator B.
- Boolean function returns a zero value for false and a nonzero value for true.
- None of the CPU register contents, except the stack pointer, are preserved by the called *functions*.

Using the D-Bug12 Functions to Perform I/O Operations *continued …*

➢ If any of the register values need to be preserved, they should be pushed onto the stack before any of the parameters have been pushed and restored after de-allocating the parameters.

Subroutine	**Function**	**Pointer Address**
far main ( )	Start of D-Bug12	$EE80
getchar ( )	Get a character from SCI0 or SCI1	$EE84
putchar ( )	Send a character out to SCI0 or SCI1	$EE86
printf ( )	Formatted string output – translates binary values to string	$EE88
far sscanhex ( )	Convert ASCII hex string to a binary integer	$EE8E
isxdigit ( )	Check if a character (in B) is a hex digit	$EE92
toupper ( )	Convert lower case characters to upper case	$EE94
isalpha ( )	Check if a character is alphabetic	$EE96
strlen ( )	Return the length of a NULL-terminated string	$EE98
strcpy ( )	Copy a NULL-terminated string	$EE9A
far out2hex ( )	Output 8-bit number as two ASCII hex characters	$EE9C
far out4hex ( )	Output 16-bit number as four ASCII hex characters	$EEA0
SetUserVector ( )	Set up a vector to a user’s interrupt service routine	$EEA4
far WriteEEByte ( )	Write a byte to the on-chip EEPROM memory	$EEA6

Calling D-Bug12 Functions from Assembly Language

- Calling the functions from the assembly language is a simple matter of pushing all the required parameters of the function onto the stack in proper order and loading the first parameter into accumulator D.

- The function can then be called with a JSR instruction.

- If the Function has no parameter or only one parameter, then there is no need to push any thing onto stack.

- If there are multiple parameters, then the first parameter will be pushed into D accumulator and the rest onto stack in *reverse order*.

- The code following the JSR instruction should remove any parameters pushed onto the stack.

- The LEAS instruction is the most efficient way to remove the parameters from stack.

- The addressing mode used in the JSR instruction is always of a form of index addressing that uses the address of address.

Example:

```
getchar EQU   $EE84
        LDX   getchar
        JSR   0,X
```

- This function retrieves a single character from the control terminal SCI0. If an unread character is not available in the receive data register when this function is called, it will wait until one is received.

- Because the character is returned as an integer, the 8-bit character is placed in accumulator B.

Example:

```
putchar EQU   $EE86
        LDD   #'A'
        LDX   putchar
        JSR   0,X
```

- This function outputs a single character to the control terminal SCI0. If the control SCI's transmit data register is full when the function is called, putchar( ) will wait until the transmit data register is empty before sending the character.

- putchar( ) returns the character that was sent.

AsmIDE - C:\DRAGON12\examples\put.asm

File Edit Build View Help

```
         ORG      $1000
         LDY      #num
loop     CLRA
         LDAB     0,Y
         PSHY
         LDX      putchar
         JSR      0,X
         PULY
         INY
         DEC      count
         BNE      loop
         SWI
num      DB       $38,$2C,$39,$35,$32,$2C,$30,$30,$30,CR,CR,LF
count    DB       12
putchar  EQU      $EE86
CR       EQU      $0D
LF       EQU      $0A
         END
```

Messages Terminal

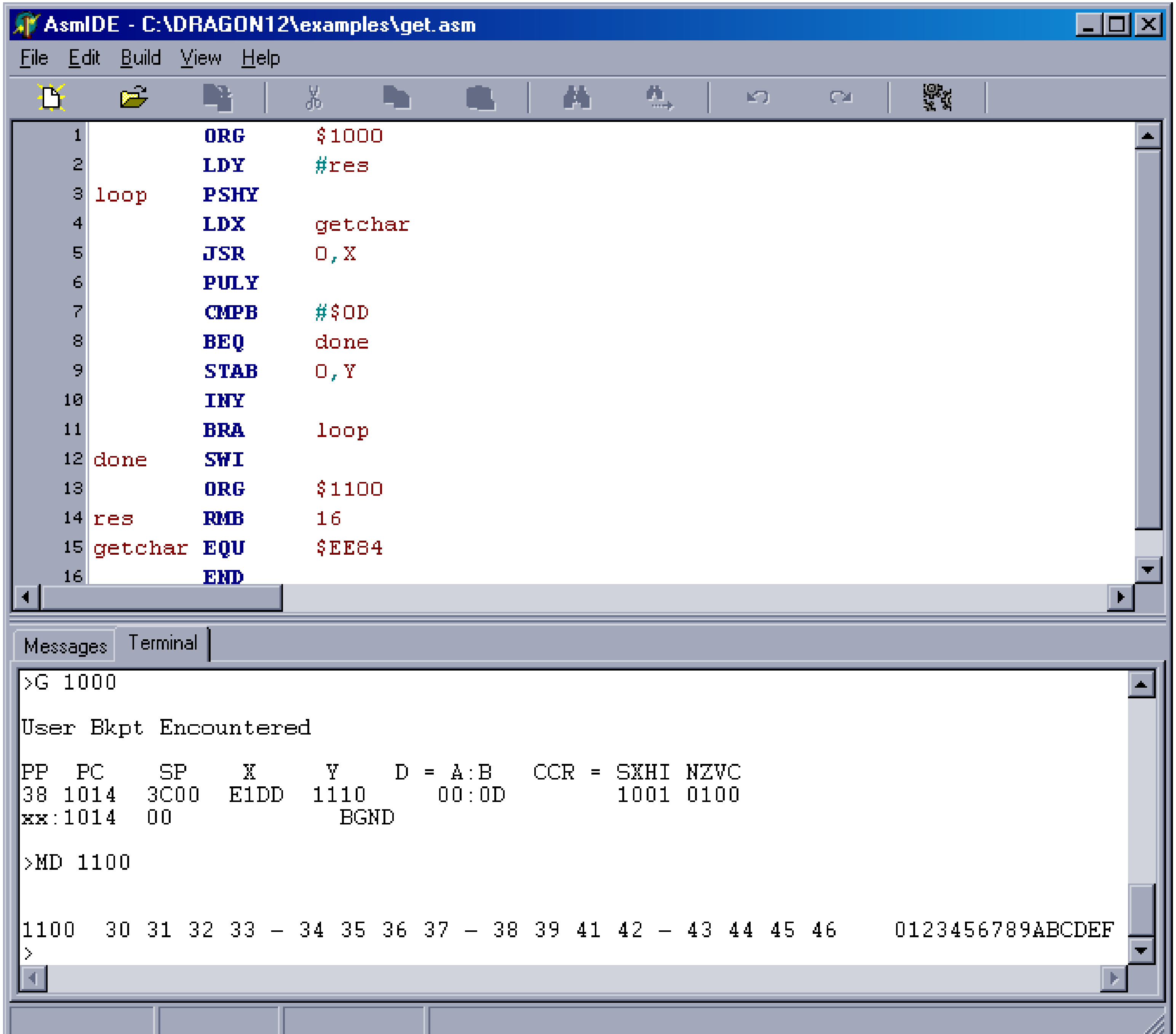
```
>G 1000

8,952,000

User Bkpt Encountered

PP  PC    SP    X     Y     D = A:B   CCR = SXHI NZVC
38 1014  3C00  E1A2  1020    00:0A          1001 0100
xx:1014  38            PULC

>
```


AsmIDE - C:\DRAGON12\examples\get.asm
File Edit Build View Help
ORG $1000
LDY #res
loop PSHY
LDX getchar
JSR 0,X
PULY
CMPB #$0D
BEQ done
STAB 0,Y
INY
BRA loop
done SWI
ORG $1100
res RMB 16
getchar EQU $EE84
END
Messages
Terminal
>G 1000
User Bkpt Encountered
PP PC SP X Y D = A:B CCR = SXHI NZVC
38 1014 3C00 E1DD 1110 00:0D 1001 0100
xx:1014 00 BGND
>MD 1100
1100 30 31 32 33 - 34 35 36 37 - 38 39 41 42 - 43 44 45 46 0123456789ABCDEF
>

Printf( ) Utility Function:

- This function is used to convert, format, and print its arguments on the standard output (the output device could be the monitor screen, printer, LCD, etc.) under the control of the format string pointed to by format.

- It returns the number of characters that were sent to standard output (sent through serial port SCI0).All except floating-point data types are supported.

- The format string can contain two basic types of objects: ASCII characters that are copied directly from format string to the display device, and conversion specifications that cause succeeding printf( ) arguments to be converted, formatted, and sent to the display device.

- Each conversion specification begins with a percent sign (**%**) and ends with a single conversion character.

- Optional formatting characters may appear between the percent sign and end with a single conversion character in the following order:

  [-] [<FieldWidth>] [.] [<Precision>] [h | l]

  These optional formatting characters are explained in the following table.

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 display argument will be padded on the left or right if necessary.
. (period)	Separates the field width from 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 display as a short.
l (letter ell)	To have an integer display as a long.

- The FieldWidth or precision field may contain an asterisk (*) character instead of a number.

- The asterisk will cause the value of the argument in the argument list to be used instead.
- The formatting characters supported by the printf( ) function are listed below.

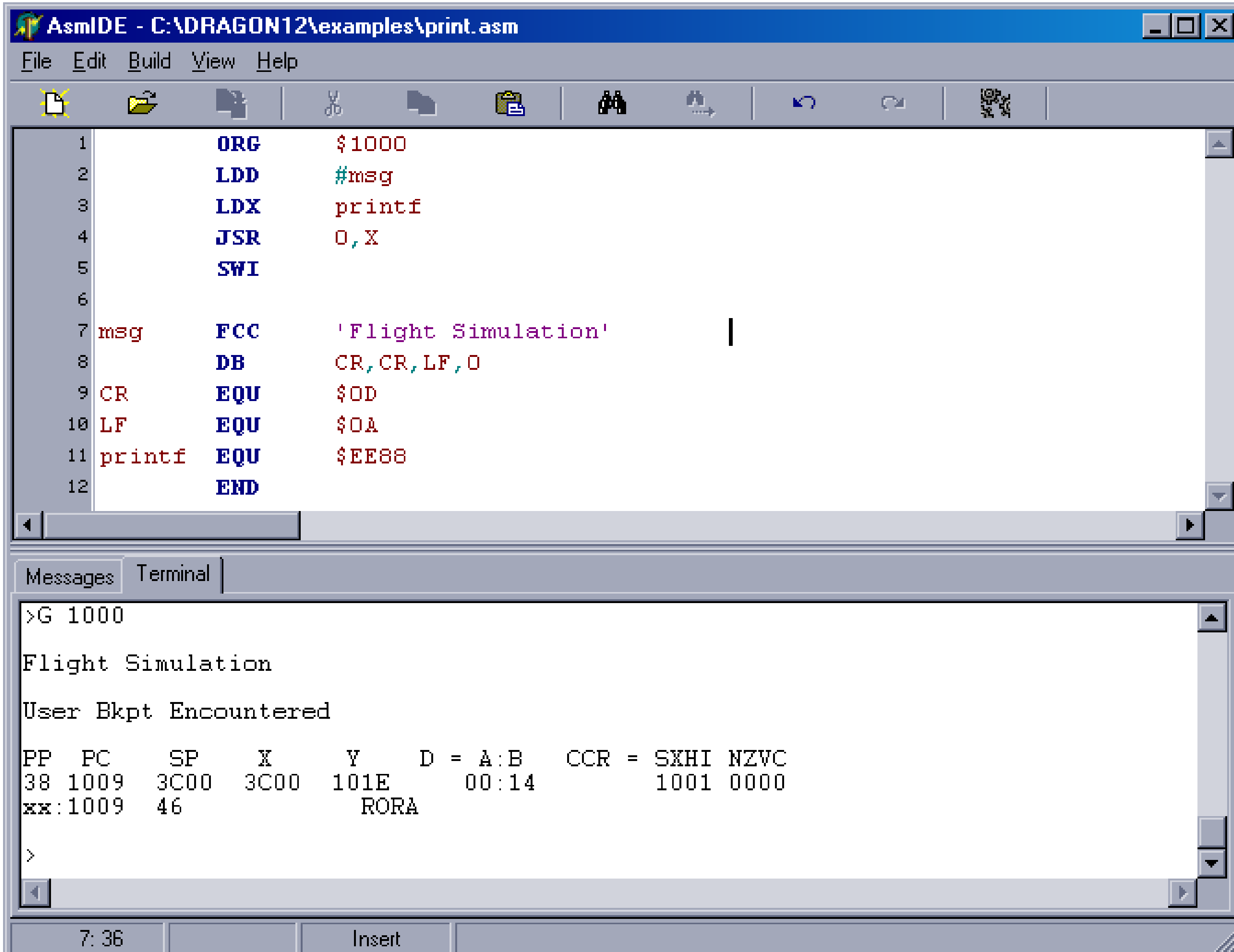
Printf( ) Conversion Characters

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 %

Example: Write instruction sequence that will cause the message 'Flight Simulation' to be displayed and the cursor will be moved to the beginning of the next line.

```
CR      EQU   $0D     EQU = Equate
LF      EQU   $0A
printf  EQU   $EE88
          ⋮
prompt  FCC   'Flight Simulation' FCC = form
        DB    CR,LF,0  DB = dfine byte
          ⋮
        LDD   #prompt   (# = imiddiate)
        LDX   printf  (address of address)
        JSR   0,X
```

Calling D-Bug12 Functions from Assembly Language *continued …*



Example: Suppose labels *m*, *n*, *gcd* represent three memory locations (2 bytes) started with the label *gcd* holds the greatest common divisor of two numbers stored at memory locations started with labels *m* and *n*. The following program illustrates the necessary instructions to display on the monitor both numbers and their GCD.

```
        LDD     gcd
        PSHD
        LDD     n
        PSHD
        LDD     m
        PSHD
        LDD     #msg
        LDX     printf
        JSR     0,X
        LEAS    6,SP
        SWI

CR      EQU     $0D
LF      EQU     $0A
printf  EQU     $EE88
msg     FCC     'The greatest common divisor of %u and %u IS %u'
        DB      CR,LF,0
gcd     DW      500
n       DW      64000
m       DW      1500
```

AsmIDE - C:\DRAGON12\examples\lect4_1.asm

```
         ORG     $1000
CR       EQU     $0D
LF       EQU     $0A
printf   EQU     $EE88

         LDD     gcd
         PSHD
         LDD     n
         PSHD
         LDD     m
         PSHD
         LDD     #prompt
         LDX     printf
         JSR     0,X
         LEAS    6,SP
         SWI

prompt   FCC     'The greatest common divisor of %u and %u is %u'
         DB      CR,LF,0
gcd      DW      400
m        DW      2400
n        DW      64000
         END
```

Running the Program on Terminal

Messages | Terminal

```
>bf 1000 1100 0

>load

*
>

>g 1000

The greatest common divisor of 2400 and 64000 is 400
User Bkpt Encountered

PP  PC    SP    X     Y     D = A:B   CCR = SXHI NZVC
38 1017  3C00  1044  1048     00:36         1001 0000
xx:1017  54              LSRB

>_
```

Multiplication and Division Instructions:

Multiplication & Division Instructions		
Mnemonic	**Function**	**Operation**
EMUL	Unsigned 16 by 16 multiply	$(D) \times (Y) \rightarrow Y{:}D$
EMULS	Signed 16 by 16 multiply	$(D) \times (Y) \rightarrow Y{:}D$
MUL	Unsigned 8 by 8 multiply	$(A) \times (B) \rightarrow A{:}B$
EDIV	Unsigned 32 by 16 divide	$(Y{:}D) \div (X)$ Quotient $\rightarrow$ Y Remainder $\rightarrow$ D
EDIVS	Signed 32 by 16 divide	$(Y{:}D) \div (X)$ Quotient $\rightarrow$ Y Remainder $\rightarrow$ D
FDIV	16 by 16 fractional divide	$(D) \div (X) \rightarrow X$ Remainder $\rightarrow$ D
IDIV	Unsigned 16 by 16 integer divide	$(D) \div (X) \rightarrow X$ Remainder $\rightarrow$ D
IDIVS	Signed 16 by 16 integer divide	$(D) \div (X) \rightarrow X$ Remainder $\rightarrow$ D

Example: Write a program to compute 1 + 2 + 3 + 4 + …….. + N, where N is a number less than 256. Store the result in 2-byte memory location Called ‘result’ and output the result to monitor appropriately.

```
            ORG     $1000
            LDAB    N           ; get the maximum number
            LDX     #0          ; clear the intermediate adder
LOOP        ABX                 ; start with max number, add it to intermediate result
            DECB                ; decrement the max number and add it again.
            BNE     LOOP        ; repeat the process till max number becomes zero
            STX     result      ; store the result in memory
            LDD     result      ; push the 2nd parameter onto stack
            PSHD                ;
            LDD     #outmsg     ; load the 1st parameter into D
            LDX     printf      ; send the result to the monitor
            JSR     0,X         ;
            LEAS    2,SP        ; balance the stack after returning from subroutine
            SWI

N           FCB     250         ; declare value of N
result      RMB     2           ; reserve two consecutive bytes of memory for result
outmsg      FCC     ‘The sum is equal to %u’
            DB      CR,LF,0

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

            END
```

AsmIDE - C:\DRAGON12\examples\lect4_4.asm

File Edit Build View Help

```
            ORG     $1000
            LDAB    N           ; get the maximum number
            LDX     #0          ; clear the intermediate adder
LOOP        ABX                 ; start with max number, add it to intermediate re
            DECB                  ; decrement the max number and add it again.
            BNE     LOOP            ; repeat the process till max number becomes z
            STX     result      ; store the result in memory
            LDD     result      ; push the sum onto the stack
            PSHD                 ;
            LDD     #outmsg     ; put the starting address of message into D
            LDX     printf      ; output the message
            JSR     0,X          ;
            LEAS    2,SP        ; balance the stack
            SWI

N           FCB     250         ; declare value of N
result      RMB     2           ; reserve two consecutive bytes of memory for resu
outmsg      FCC     'The sum is equal to %u'
            DB      CR,LF,0
printf      EQU     $EE88
CR          EQU     $0D
LF          EQU     $0A
```

Running the Program on Terminal

Messages Terminal

```
>bf 1000 1100 0

>load

*
>

>g 1000

The sum is equal to 31375
User Bkpt Encountered

PP  PC    SP    X     Y     D = A:B   CCR = SXHI NZVC
38 101D  3C00  1035  1039     00:1B         1001 0000
xx:101D  FA7A8F        ORAB  $7A8F

>
```

13: 34 Insert

Example: Now add necessary code to receive N from user.

```
        ORG     $1000
        LDX     #num
set     CLR     1,X+     ;initialize variables
        CPX     #count
        BNE     set
        MOVB    #3,count
        LDD     #msg1   ;message to get N
        LDX     printf
        JSR     0,X
L1      LDX     getchar  ;get a digit
        JSR     0,X
        CMPB    #CR      ;if CR put num in
        BEQ     cal      ;order
        SUBB    #$30     ;ASCII offset
        PSHB
        DEC     count
        BNE     L1    ;bring in up to 3 digits
cal     LDAA    count
        CMPA    #3
        BEQ     pac
        PULB
        STAB    num+2
        INCA
        CMPA    #3
        BEQ     pac
```

```
        PULB
        STAB    num+1
        INCA
        CMPA    #3
        BEQ     pac
        PULB
        STAB    num
pac     LDAB    num     ;convert to binary
        LDAA    #10
        MUL
        ADDB    num+1
        LDAA    #10
        MUL
        ADDB    num+2
        STAB    N
        SWI

num     DB      0,0,0
count   DB      3
N       RMB     1
msg1    FCC     'Enter a number up to 256:'
        DB      CR,LF,0
printf  EQU     $EE88
getchar EQU     $EE84
CR      EQU     $0D
LF      EQU     $0A
        END
```

AsmIDE - C:\DRAGON12\examples\lect4_5.asm

File Edit Build View Help

```
          ORG     $1000
          LDX     #num
set       CLR     1,X+
          CPX     #count
          BNE     set
          LDAB    #3
          STAB    count
          LDD     #msg1    ; send a message out to get
          LDX     printf    ; number N
          JSR     0,X        ;
L1        LDX     getchar  ; get digit/character
          JSR     0,X       ;
          CMPB    #CR      ; if CR encountered, then compact
          BEQ     cal       ; the number
          SUBB    #$30     ; if digit subtract ASCII offset and
          PSHB              ; store it onto stack
          DEC     count    ; as long as less than 3-digit
          BNE     L1        ; bring in the next digit
cal       LDAA    count    ; get the digits from stack and
          CMPA    #3        ; put them in memory in proper
          BEQ     pac        ; order
          PULB                ;
          STAB    num+2        ;
          INCA                   ;
          CMPA    #3               ;
          BEQ     pac                ;
          PULB                         ;
          STAB    num+1                  ;
```

AsmIDE - C:\DRAGON12\examples\lect4_5.asm

File Edit Build View Help

```
        STAB    num+1          ;
        INCA                    ;
        CMPA    #3               ;
        BEQ     pac               ;
        PULB                       ;
        STAB    num                 ;
pac     LDAB    num      ; change the decimal number to
        LDAA    #10       ; binary
        MUL                ;
        ADDB    num+1       ;
        LDAA    #10          ;
        MUL                   ;
        ADDB    num+2          ;
        STAB    N        ; store the number in location N
        SWI
        ORG     $1100
num     DB      0,0,0
N       DB      0        ; declare value of N
count   DB      3
msg1    FCC     'Enter a number up to 256 :'
        DB      CR,LF,0
getchar EQU     $EE84
printf  EQU     $EE88
CR      EQU     $0D
LF      EQU     $0A
        END
```

Running the Program on Terminal

Messages | Terminal

```
>BF 1000 1200 0

>LOAD

**
>

>G 1000

Enter a number up to 256 :
User Bkpt Encountered

PP  PC     SP     X      Y     D = A:B    CCR = SXHI NZVC
38 1058   3C00  DAD6   0038       00:F8          1001 1000
xx:1058   00              BGND

>MD 1100 113F


1100   02 04 08 F8 - 00 45 6E 74 - 65 72 20 61 - 20 6E 75 6D     .....Enter a num
1110   62 65 72 20 - 75 70 20 74 - 6F 20 32 35 - 36 20 3A 0D     ber up to 256 :.
1120   0A 00 00 00 - 00 00 00 00 - 00 00 00 00 - 00 00 00 00     ................
1130   00 00 00 00 - 00 00 00 00 - 00 00 00 00 - 00 00 00 00     ................
>
```

Take advantage of ‘*loop*’ to make source code more efficient.

```
        ORG     $1000
        LDX     #num
set     CLR     1,X+      ;initialize variables
        CPX     #count
        BNE     set
        MOVB    #3, count
        LDD     #msg1   ;message to get N
        LDX     printf
        JSR     0,X
L1      LDX     getchar  ;get a digit
        JSR     0,X
        CMPB    #CR       ;if CR put num in
        BEQ     cal       ;order
        SUBB    #$30      ;ASCII offset
        PSHB
        DEC     count
        BNE     L1     ;bring in up to 3 digits
cal     LDX     #num+2
        LDAA    count
 cont   CMPA    #3
        BEQ     pac
        PULB
        STAB    1,X-
        INCA
        BRA     cont
```

```
pac     LDX     #num
        CLRB
pac1    ADDB    1,X+     ;convert to binary
        LDAA    #10
        MUL
        CPX     #num+2
        BLO     pac1
        ADDB    0,X
        STAB    N
        SWI

        ORG     $1200
num     DB      0,0,0
count   DB      3
N       RMB     1
msg1    FCC     ‘Enter a number up to 256:’
        DB      CR,LF,0
printf  EQU     $EE88
getchar EQU     $EE84
CR      EQU     $0D
LF      EQU     $0A
        END
```

Example cont'd …

- Now, lets put all of it together:

  Write a program to compute 1 + 2 + 3 + 4 + …….. + N, where N is a number between 1 to 255 that user will enter. Store the result in 2-byte memory location called 'result' and output the result to monitor appropriately. User ought to be able to enter numbers as many time as he/she wishes.

- For most part, we have already written this program. All needs to be done is to put them all together and add the finishing touches.
- When we write a rather large program, it is a good practice to modularize it and write each module as a *subroutine* or *nested subroutines*.
- Once every function is written as an *independent subroutine*, it becomes portable and can be used or *included* in other programs.
- Next slide will demonstrate how to organize and modularize this problem.

This is the skeleton of the program to accomplish this task.

```
        ORG     $1000
start   JSR     INIT        ; initialize all the parameters and subsystems
        JSR     GET         ; get the number N from user and pack it
        JSR     ADD         ; find the result and store it
        JSR     OUT         ; output the result to screen
        LDD     #repmsg     ; ask if user wants to repeat the process
        LDX     printf      ;
        JSR     0,X         ;
        LDX     getchar     ;
        JSR     0,X         ;
        CMPB    #’Y’        ;
        BEQ     start       ;
        SWI
INIT    :
        RTS
GET     :
        RTS
ADD     :
        RTS
OUT     :
        RTS


repmsg  FCC     ‘Do you want to enter another number (Y/N)?’
        DB      CR,LF,0
printf  EQU     $EE88
CR      EQU     $0D
LF      EQU     $0A
        END
```

```
INIT    LDX     #num
set     CLR     1,X+        ;initialize variables
        CPX     #count
        BNE     set
        MOVB    #3,count
        RTS

GET     LDD     #msg1       ;message to get N
        LDX     printf
        JSR     0,X
L1      LDX     getchar     ;get a digit
        JSR     0,X
        PSHD                ; echo out the key
        LDX     putchar      ;
        JSR     0,X           ;
        PULD                   ;
        CMPB    #CR         ;if CR put num in
        BEQ     cal          ;order
        SUBB    #$30        ;ASCII offset
        PSHB
        DEC     count
        BNE     L1          ;bring in up to 3 digits
cal     LDX     #num+2
        LDAA    count
```

```
cont    CMPA    #3
        BEQ     pac
        PULB
        STAB    1,X-
        INCA
        JMP     cont
pac     LDX     #num
        CLRB
pac1    ADDB    1,X+     ;convert to binary
        LDAA    #10
        MUL
        CPX     #num+2
        BLO     pac1
        ADDB    0,X
        STAB    N
        RTS

ADD     LDAB    N        ; get the maximum number
        LDX     #0       ; clear the intermediate adder
LOOP    ABX              ; start with max number, add it to intermediate result
        DECB               ; decrement the max number and add it again.
        BNE     LOOP         ; repeat the process till max number becomes zero
        STX     result   ; store the result in memory
        RTS
```

```
OUT      LDD    result   ; push the 2nd parameter onto stack
         PSHD            ;
         CLRA
         LDAB   N
         PSHD
         LDD    #outmsg  ; load the 1st parameter into D
         LDX    printf   ; send the result to the monitor
         JSR    0,X      ;
         LEAS   4,SP     ; balance the stack after returning from subroutine
         RTS
         ORG    $1200
num      DB     0,0,0
count    DB     3
N        RMB    1
result   RMB    2
repmsg   FCC    'Do you want to enter another number (Y/N)?'
         DB     $0D,0
msg1     FCC    'Enter a number up to 256:'
         DB     $0D,0
outmsg   DB     $0D
         FCC    'The sum of 1 to %u is equal to %u .'
         DB     $0D,$0D,0
printf   EQU    $EE88
putchar  EQU    $EE86
getchar  EQU    $EE84
         END
```

Now putt all this together as a source file, assemble, and download. Running the program will produce output as follow:

Messages Terminal

```
>g 1000

Enter a number up to 256:
25

The sum is equal to 325

Do you want to enter another number (Y/N)?
Enter a number up to 256:
234
The sum is equal to 27495

Do you want to enter another number (Y/N)?
Enter a number up to 256:
2

The sum is equal to 3

Do you want to enter another number (Y/N)?
Enter a number up to 256:
255
The sum is equal to 32640

Do you want to enter another number (Y/N)?
User Bkpt Encountered

PP  PC     SP     X      Y     D = A:B    CCR = SXHI NZVC
38 1022   3C00   DBB2   006E      00:6E         1001 1001
xx:1022   CE1200          LDX    #$1200

>
```

5-digit Input – Output

1. Output a message to ask for a positive number up to 65536.
2. Store each digit in a reserved memory location. Carriage return or 5-digit should indicate end of number.
3. Check the number to make sure it is a genuine number between 0 and 65535. In case of error, display appropriate error message and redo the input process.
4. Compact the five digits, (in some cases less than 5-digit), and process them accordingly to pack them into a single 16-bit number.
5. Convert 16-bit binary number back to 5-digit ASCII. To accomplish this, divide the 16-bit number by 10000 and display the ten thousand-digit, divide by 1000 and display the thousand-digit, and so on. The **putchar** subroutine should be used up to five times. Your program should not display the leading zeros. All of this can be done in a subroutine.
6. Use appropriate message and utility subroutine **printf** to output HEX version of the same number.
7. Display a message to check if user wants to enter another number and act accordingly.