Some Thoughts on a 68xxx ROM Monitor

Karl Lunt
7349 W. Canterbury
Peoria, AZ 85345 (602)-878-0305

My newest computer is a 68010 board I purchased surplus for $40. One of the first things a user of such a surplus board must do is to rewrite the original ROMs so the board will do something useful for its new owner. That means designing and writing a new ROM monitor.

I had already written several ROM monitors for my 6809 systems and learned several things that I wanted to change in the 68xxx monitor I was preparing to write.

The new ROM had to support a string output routine much more sophisticated than the PSTRING utility normally found in the 6809 monitors. Specifically, I wanted the string utility to support multiple arguments and formatted hex/decimal output.

The monitor had to permit on-the-fly changes to the primitive input and output routines. This was necessary to permit either of the two serial ports to use the monitor as needed. Ideally, changing from serial port 0 to serial port 1 should be no more difficult than changing a couple of RAM vectors.

The new monitor had to use NO local RAM variables. This was to permit more than one user to be running the monitor at the same time, in turn allowing the monitor to work in a multi-user fashion. As it turned out, the only local RAM variables used by the monitor are the I/O vectors in RAM, described above, and a memory-test flag used at RESET time.

The monitor had to provide some simple system checks upon power-up or reset, but also had to be smart enough to recognize when a user program was already installed in memory and not overwrite that program with a memory check just because someone hit the RESET switch.

This monitor also had to be callable from a user program. This means that a user should be able to execute a JSR MONITOR to enter the monitor from his program, make full use of the monitor's capabilities, then EXIT back to his program and continue as if never having left the program.

A monitor that does all of the above, plus considerably more, is presently running on my 68010 board. I will make the source listings available to 68MJ readers for use in their own systems or for any other non-commercial use they might have. Credit to different sources of information appear throughout the source listings; if you use this code, please leave the notices intact so the original thinkers get the credit they deserve.

The 680x0 code for the monitor was developed on a PC/XT compatible running the 2500 A.D. 68000 cross assembler and linker system. I used Sidekick as my text editor, principally because it is fast, RAM-based and I already knew how to use it. In the early stages of development, I had to burn different versions of the monitor into ROM, install the ROMs in the board, try the software, then repeat the cycle as necessary. The current version of the ROM supports downloading of S19 records, considerably easing the development of new software or monitor extensions. I have been using Crosstalk to communicate between the host PC and the target 68010 system, but just about any good PC public-domain modem program should work fine.

The first order of business is the formatted string output routine, PRINTF. The code for this routine can be found in the book "68000 Assembly Language," by Stanley and Krantz (published by Addison- Wesley). This book is an absolute gold-mine of information for the 680x0 programmer; I consider it a "must-have" book and have nearly destroyed my copy from overuse. (Don, how about a review of this book?) *(o.k. Karl, we'll be looking forward to it - DMW)*

The PRINTF function, like its C namesake, permits the output of formatted string and numeric information. All arguments to PRINTF are placed on the stack and must be popped from the stack by the calling program after the return from PRINTF.

Each call to PRINTF must have at least one argument; the LONG address of the formatted string to be printed. If this formatted string requires any arguments of its own (such as a decimal number to be printed in the body of the string), those arguments must also be pushed onto the stack. These secondary arguments must be pushed in reverse order of need, prior to pushing the address of the formatted string. A short example might make this more clear.

Consider a formatted string to print the amount of RAM available to a user and the first address of that RAM. It would make most sense if the first value was displayed in decimal and the second value in hex. An example of how you might want the display to appear could be:

```
AVAILABLE RAM: 129300 BYTES  FIRST RAM LOCATION: $42000
```

The following segment of 680x0 code, using the PRINTF function, will do just that.

```
MOVE.L      A0,-(A7)             FIRST RAM LOC IN A0 TO
                                 STACK
MOVE.L      D0,-(A7)             AMOUNT OF RAM IN D0 TO
                                 STACK
MOVE.L      #FMTSTRG,-(A7)       PUSH ADDRESS OF STRING
JSR         PRINTF               DISPLAY THE STRING
ADD.L       #12,A7               PULL 12 BYTES FROM STACK
BRA         ——                   REST OF PROGRAM FOLLOWS

 FMTSTRG:
BYTE        $0D,$0A
BYTE        'AVAILABLE RAM: %D
BYTES       '
BYTE        'FIRST RAM LOCATION: $%X'
BYTE        $0D,$0A
BYTE        0                    REQUIRED NULL TERMINATOR
```

As you can see, using the PRINTF routine is only slightly more complicated than the 6809 PSTRING routine, and far more powerful. Besides handling LONG decimal and LONG hex values, the PRINTF routine can handle a variety of other formatted data. The full list is:

```
WORD decimal
%d    displays 16-bit signed decimal LONG decimal
%D    displays 32-bit signed decimal WORD decimal
%u    displays 16-bit unsigned decimal LONG decimal
%U    displays 32-bit unsigned decimal WORD hex
%x    displays 16-bit hex LONG hex
%X    displays 32-bit hex string
%s    displays null-terminated string character
%c    displays single literal char
```

Although the routine is named PRINTF, its formatting convention differs significantly from the C version of PRINTF; full details on how the routine works can be found in "68000 Assembly Language" and by looking throught the examples in the ROM monitor code.

The implementation of I/O vectors is based somewhat on the method used by FLEX for redirecting I/O. The way the monitor handles the I/O, however, is more flexible.

All calls to the general purpose I/O routines are to ROM entry points named GETC, PUTC and KEYPRSD. For example, if your program wants to output a character to the terminal, it pushes the character (as a WORD) onto the stack and JSRs to PUTC. When control returns from PUTC, your program must then pop the character off of the stack before continuing.

Each of these primitive routines, however, is routed through a RAM- based vector to the appropriate I/O routine, depending on which user port is active at the time. For example, if user port 0 is active, GETC will jump through a RAM vector that is aimed at GETC0, the routine that gets a character from serial port 0.

Although the code for both GETC and GETC0 reside in ROM, the link between the two is in RAM and may be modified at any time. This link could just as easily point to GETC1, the ROM-based routine that services serial port 1. In this case, GETC would return a character input from port 1. A full set of GETC, PUTC and KEYPRSD routines exist in ROM for both of the serial ports.

This concept of RAM-based links for the primitive I/O routines can be extended even farther. If a user's program wants to talk to a printer (as an example) or a virtual terminal, the RAM links to the I/O functions can be changed to permit the user's interface routines to run. Because PRINTF and all other higher-level monitor routines use only the general GETC, PUTC and KEYPRSD entry points, the full monitor will run on any set of drivers installed with the RAM links.

But how do you write a monitor that uses NO global RAM variables? If you have never programmed on the 680x0 before, you are in for a real treat. Motorola provided the programmer with the LINK and UNLK instructions, specifically designed for creating stack-based variables. The structure created by these instructions is called a frame and it is crucial to much of the power in this ROM monitor.

The customary way for a routine to handle variables used to involve setting aside a small, fixed area of RAM for storage of things such as flags, counters, pointers and other data structures. If you tried to use the same piece of code to service two different users, however, one user's variables would get overwritten by the other's values, causing the routine to lose track of what was going on.

The answer to this problem is to put the variables in the stack. Since each user would (hopefully) enter the routine with a different stack pointer value, the variables associated with that user would be safely stored out of the way and available only to the proper user.

This, however, leads to a different problem. Upon entry to a routine, the item on the top of the stack is the return address (something your routine is liable to need again soon). Behind that address (going upwards in memory) may be arguments needed by the routine itself. Behind those arguments will be other return addresses and eventually the "bottom" of the stack. This means that a routine cannot arbitrarily store local variables and data on the stack using a positive displacement (that is, using addresses higher than the current stack pointer).

Well, how about going downwards (using negative displacement) on the stack? While it is possible to store and retrieve variables using the stack pointer plus some negative offset, this technique is doomed to failure as soon as interrupt or exception activity begins. The reason for this should be obvious; when an interrupt occurs, the first thing the CPU does is push vital information onto the stack and race off to service the interrupt. If this interrupt occurs in a routine that is storing data on the stack using negative displacements, that data will occasionally be trashed.

Enter, then, the LINK instruction. LINK "freezes" the stack by first copying the current stack pointer value into a selected register (by convention this is usually A6), then moving the stack pointer downwards in memory a specified amount. This has the effect of reserving a block of the stack for the routine's use RELATIVE TO

THE A6 REGISTER. Now, references to stack-based variables relative to A6 can be safely made, regardless of interrupt activity. When it comes time to exit the routine, the UNLK instruction repairs the stack (it even restores the selected register) and leaves the return address on the stack, ready for the subsequent RTS.

The best way to get a handle on the LINK instruction and the concept of stack framing is to study the examples in the monitor code and to read the text in "68000 Assembly Language." This is an extremely valuable programming tool and is worth the time it takes to understand it.

The ROM monitor makes heavy use of stack framing. In fact, the only global RAM variables used by the monitor are the I/O links discussed above and a couple of flag LONGs used to determine the course of action to take on reset. The entire monitor, then, can be thought of as a huge subroutine. Any user can execute the monitor by simply treating it as a subroutine and JSRing to it. All of the monitor's text buffers, flags and other data will be safely tucked away in the calling program's stack.

The ROM monitor performs a few simple chores upon reset. It tests the mapping RAM (used to assign physical RAM pages to logical addresses) and it tests all of available RAM before it configures the RAM-based serial links and announces its presence.

But this testing of RAM, even though it is important the first timeyou apply power, can be a real pain if the program you are testing runs away, forcing a RESET. All of your patching and testing can get wiped out by the RAM check, costing you a lot of work.

To prevent this from happening, the monitor writes a key value in a specific location of RAM after it has successfully tested memory. On the next reset, it first checks this location to see if the key value is still there. If it is, the RAM test is skipped, leaving the user's program intact.

As I have already mentioned, the monitor's heavy use of stack framing permits it to be treated as a giant subroutine. In fact, the code itself reveals that there is a huge routine called MONITOR$ and a very small outer loop that exists solely to call MONITOR$. If you want to access the monitor from within a user program, you need only JSR to the ROM address named MONITOR. At that time the active port will display the monitor's prompt and you may enter any legal monitor commands. If you then use the monitor's EXIT command, the monitor will execute a RTS, returning you to your original program.

This provides some very powerful capabilities to someone trying to develop software on such a system. Adding a special keyboard sequence into your program code (for example) permits you to activate the monitor, view or modify memory as desired, then return to your program to continue with your testing.

This is only a part of the resources available to a developer using this monitor ROM. Other goodies in the code include the obligatory memory dump (DU), the ability to upload Motorola S19 records with an optional address offset (RL), an imbedded CASE support routine available to a calling program, and much more.

If there is enough interest in this ROM monitor, I can continue this discussion in subsequent issues of 68MJ. For example, I already have code running (not yet in the monitor ROM) that allows more than one user to execute the monitor simultaneously, providing true multi-user power at the monitor.

Editor's Note: If you want more of this please let me know, or communicate directly with the author. With the availability of "bargain" and surplus hardware, this type of information is like a lamp-post in the night! Takes me back a few years, how 'bout you?

DMW

* MONITOR

```
*
*   Main routine in the 68010 EPROM.
*
*   Note that there is no ORG statment in this block.  It
should be
*   LINKed to appear at address $800100, as that is where
the COLDS
*   vector is aimed.
*

        EXTERNAL  PRINTF,SINIT,PUTC,GETC,GETLINE
        EXTERNAL  DUMPMEM,FILL,SWEEP
        EXTERNAL  PROMPT
        EXTERNAL  CASE,UPPER
        EXTERNAL  ACTRAM,TESTRAM
        EXTERNAL  CHANGE
        EXTERNAL  S19,GO
        EXTERNAL  MONITOR,WARMS1
        EXTERNAL  DUMP_REG$

        PUBLIC    WARMS$,COLDS$,PROCESS$,MONITOR$
        PUBLIC    WARMS1$

BUFFER  EQU       -256        STACK-BASED TEXT BUFFER
*                             STACK MUST ALLOW FOR 256
CHARS
*                             REFER TO MONITOR
                              (BELOW) FOR USE
*

MON_SP: EQU       $DF0           STACK POINTER
                                 USED BY MONITOR.
*                                THIS IS THE
                                 VALUE WRITTEN TO A7
*                                BY BOTH COLDS AND
                                 WARMS1.

COLDS$:
        MOVE.W    #$3700,$450000   LEDS:  _ _ G _ R
        MOVE.L    #$400000,A0      POINT AT START
                                   OF MAP RAM
        MOVE.W    #$3FF,D0         SET A COUNTER
MAPTEST:
        MOVE.W    #$01A5,(A0)         WRITE A WORD
        MOVE.W    (A0)+,D1            READ IT BACK
        AND.L     #$01FF,D1           MASK OUT
                                      CONTROL BITS
```

```
        CMP.W     #$01A5,D1            NOW TEST IT
        BNE       RAMFAIL              BRANCH IF
                                       FAIL
        DBF       D0,MAPTEST           COUNT THIS
                                       WORD
        BRA       STARTUP              BRANCH IF ALL
                                       GOOD

RAMFAIL:
        BRA       RAMFAIL              DIE HERE WITH
                                       LEDS ON

STARTUP:
        MOVE.W    #$3E00,$450000     LEDS: R _ G _ _
        MOVE.W    #$2000,$400000     SWITCH ON
                                     LOWEST 4K BLOCK
        MOVE.L    #MON_SP,A7         PUT STACK IN
                                     LOWEST RAM
        MOVE.L    #$A5A5A5A5,$0FFC   TEST A LOCATION
        CMP.L     #$A5A5A5A5,$0FFC
        BNE       RAMFAIL              BRANCH IF
                                       FAILURE
        MOVE.W    #$3B00,$450000     LEDS: _ _ G Y _
        BSR       ACTRAM             ACTIVATE ALL
                                     SYSTEM RAM
        MOVE.W    #$3900,$450000     LEDS: _ Y G Y _
        BSR       SINIT              INITIALIZE THE
                                     SERIAL PORT
        MOVE.W    #$3D00,$450000     LEDS: _ Y G _ _
        MOVE.L    #HELLO,-(A7)       GET FIRST
                                     MESSAGE
        BSR       PRINTF             DISPLAY IT
        ADDQ.L    #4,A7              ADJUST STACK
        MOVE.W    #$3F00,$450000     LEDS: _ _ G _ _

        CMP.L     #$A5A5A5A5,$FF8    BEEN THROUGH
                                     THIS BEFORE?
        BEQ       WARMS1$            SKIP TEST IF SO
        BSR       TESTRAM            TEST ALL
                                     AVAILABLE RAM
        MOVE.L    D0,-(A7)           PUSH SIZE OF
                                     RAM
        MOVE.L    D0,-(A7)           PUSH IT AGAIN
        MOVE.L    #SIZE,-(A7)        PUSH MESSAGE
        BSR       PRINTF             DISPLAY IT
        ADD.L     #12,A7             REPAIR STACK
        MOVE.L    #$A5A5A5A5,$FF8    WRITE FLAG TO
                                     SHOW WE'VE BEEN
*                                    THROUGH HERE
                               ALREADY (SEE STARTUP)
HELP$:
        MOVE.L    #FIRST,-(A7)    GET SIGN-ON MESSAGE
        BSR       PRINTF          PRINT IT
        ADDQ.L    #4,A7           FIX THE STACK

WARMS1$:
        MOVE.L    #MON_SP,A7      ENTRY POINT THAT
                                  RESETS THE SP

WARMS$:
        BSR       DUMP_REG$       REINSTALL THE DUMP
                                  REGS VECTOR
        BSR       MONITOR$        BRANCH TO THE
                                  MONITOR WITHOUT
*                                 RESETTING THE STACK
                                  POINTER.

*
        MOVE.L    #EXITMSG,-(A7)  TELL USER EXIT
                                  ISN'T GOING TO WORK
        BSR       PRINTF          PRINT IT
        ADDQ.L    #4,A7           FIX THE STACK
        BRA       WARMS$          AN ENDLESS LOOP

EXITMSG:
        BYTE      $0D,$0A,$0A
        BYTE      'An EXIT back to the ROM Monitor
                   is'
        BYTE      ' pretty pointless, you know.'
        BYTE      $0D,$0A
        BYTE      0
*
*  Main routine for the 68010 ROM monitor.
*
*  Normal entry is from a COLD start following reset.
Alternate
*  entry is from a user program via the WARMS (or WARMS1
entry.
*
*  A user program may also enter via the MONITOR jump
vector in
*  the ROM jump table.  Entering through this point
permits a user
*  program to activate the monitor with a special se-
quence from
*  within the program, use the monitor as if it had been
activated
*  by a warm-start, then return to the user program with
an EXIT
*  monitor command.  Entry by this technique should be
with a JSR.
*  Leaving the monitor this way will return to the
calling program
*  with ALL registers preserved, though the CCR is not
saved.
*

MONITOR$:
        LINK      A6,#BUFFER          SET THE TEXT
                                BUFFER IN STACK SPACE
        MOVEM.L   D0-D7/A0-A7,-(A7) SAVE EVERYTHING
LOOP:
        BSR       PROMPT
        LEA       BUFFER(A6),A0      POINT A0 AT
                                     TEXT STRING
        BSR       GETLINE
        BSR       PROCESS$
        BRA       LOOP
*
*  NOTE:  Exit from this routine is via the monitor EXIT
command,
*  whose code follows in the case structure below and is
labled
*  EXIT$.
*



*
*  PROCESS
*
*  This is the core of the ROM monitor.  It executes the
command
*  found in the first two character positions of the lin
pointed
*  at by A0.  Note that this routine is available exter-
nally via
*  the jump table.  This permits a user to load a string
with a
*  monitor command, put the string's address in A0 and
JSR here
*  so the monitor can process the command.  As is custom-
ary, any
*  string submitted to PROCESS must be terminated with a
null byte.
*  Additionally, any monitor command must be two charac-
ters long
*  and must begin in column one.
*
```

```
PROCESS$:
        CMP.B       #0,(A0)       ANYTHING ON THE LINE?
        BEQ         PROCX         BRANCH IF NOT
        MOVE.L      A0,A1         YES, MOVE POINTER
                                  INTO A1
        BSR         UPPER         CONVERT TO UPPERCASE
        MOVE.W      (A1),D0       GET THE FIRST TWO
                                  CHARACTERS
        MOVE.L      #PROC_TBL,A0  GET ADDR OF PROCESS
                                  TABLE
        BRA         CASE          DO THE COMMAND

PROCX:
        RTS                        RETURN TO MAIN LOOP

*
* JUMP TABLE FOR THE CASE SWITCH ABOVE.  THIS TABLE IS
NEEDED SO
* THE LINKER WILL PROPERLY RESOLVE THE ADDRESSES IN THE
CASE SWITCH
* TABLE.
*

DUMPMEM$: BSR       DUMPMEM
          BRA       PROCX

SWEEP$:   BSR       SWEEP
          BRA       PROCX

FILL$:    BSR       FILL
          BRA       PROCX

CHANGE$:  BSR       CHANGE
          BRA       PROCX

S19$:     BSR       S19
          BRA       PROCX

GO$:      BSR       GO
          BRA       PROCX

EXIT$:
          ADDQ.L  #4,A7     POP THE PROCESS RTN ADDR
          MOVEM.L    (A7)+,D0-D7/A0-A7   RESTORE
                                         EVERYTHING
          UNLK       A6   REMOVE THE FRAME
          RTS             LEAVE THE MONITOR ROUTINE


*
* WHAT
*
* This is the default CASE arm.  It just tells the user
that the
* input was not too good.
*

WHAT:
        MOVE.L      #WHAT_MSG,-(A7)
        BSR         PRINTF
        ADDQ.L      #4,A7
        BRA         PROCX


WHAT_MSG:
        BYTE        $0D,$0A
        BYTE        ' Beats me what you want.  Try
                    again.'
        BYTE        0

SIZE:
        BYTE        $0D,$0A
        BYTE        ' Available RAM: %D bytes.
                    First non-RAM address: %X.'
        BYTE        $0D,$0A
        BYTE        0

PROC_TBL:
        WORD        8
        BYTE        'D','U'             DUMP COMMAND
        LONG        DUMPMEM$

        BYTE        'S','W'             SWEEP COMMAND
        LONG        SWEEP$

        BYTE        'F','I'             FILL COMMAND
        LONG        FILL$

        BYTE        'C','H'             CHANGE
                                        COMMAND
        LONG        CHANGE$

        BYTE        'H','E'             HELP COMMAND
        LONG        HELP$

        BYTE        'R','L'             RAM LOAD
                                        (S19) COMMAND
        LONG        S19$

        BYTE        'G','O'             GO COMMAND
        LONG        GO$

        BYTE        'E','X'             EXIT COMMAND
        LONG        EXIT$

        LONG        WHAT                DEFAULT CASE

HELLO:
        BYTE        $0D,$0A,$0A
        BYTE        '       68010 ROM MONITOR  V1.2'
        BYTE        $0D,$0A
        BYTE        '       Written for the Convergent
                            Technologies'
        BYTE        ' Mini-Frame'
        BYTE        $0D,$0A
        BYTE        0

FIRST:
        BYTE        $0D,$0A,$0A
        BYTE        'All commands are two characters,
                    followed '
        BYTE        'by any arguments.'
        BYTE        $0D,$0A
        BYTE        'Separate all arguments (in hex)
                    by at least '
        BYTE        'one space.  Available'
        BYTE        $0D,$0A
        BYTE        'commands are:'
        BYTE        $0D,$0A,$0A
        BYTE        '     Dump memory       DU  <addr>'
        BYTE        $0D,$0A
        BYTE        '     Sweep memory      SW  <start>
                                    <stop> [times]'
        BYTE        $0D,$0A
        BYTE        '     Change memory     CH  <addr>
                              <data>...<data> or'
        BYTE        $0D,$0A
        BYTE        '                       CH  <addr>'
        BYTE        $0D,$0A
        BYTE        '     Fill memory       FI  <start>
                                    <stop> <data>'
        BYTE        $0D,$0A
        BYTE        '     RAM Load (S1-S9) RL  <offset
                                         addr>'
        BYTE        $0D,$0A
```

```
        BYTE        '    Goto location      GO
                                   <transfer addr>'
        BYTE        $0D,$0A
        BYTE        '     Exit monitor         EX'
        BYTE        $0D,$0A,$0A
        BYTE        0

          END
```

TITLE PRINTF.ASM

```
*
*  PRINTF — Subset/superset of C printf standard I/O
function
*
*  Control args:
*     push last parameter in control string first, then
next-to-
*     last, etc.  Push addr of control string last.
*
*     %d  Print signed decimal word
*     %u  Print unsigned decimal word
*     %D  Print signed decimal longword
*     %U  Print unsigned decimal longword
*     %x  Print hexadecimal word
*     %X  Print hexadecimal longword
*     %s  Print null-terminated string
*     %c  Print character
*     %v  Cursor (x,y) - push x, then y as words
*     %default  Print next character as literal
*
*  Taken from '68000 Assembly Language,' by Krantz and
Stanley.
*  Listing appears on page 234.
*

 PUBLIC     PRINTF$
        EXTERN      CASE,PUTC,CURSOR
*
*  LOCAL VARIABLE DISPLACEMENT DEFINITIONS
*
LEFTJ   EQU         -2
FIELD   EQU         -4
SIGNF   EQU         -6
*
*
PRINTF$:
        LINK        A6,#-6
        MOVEM.L     D0-D6/A0-A2,-(A7)
        MOVE.L      8(A6),A1            * GET CONTROL
                                          STRING ADDR
        LEA         12(A6),A2           * GET POINTER
                                        TO PARAMETERS
LOOP:
        MOVE.B      (A1)+,D0            * GET CONTROL
                                          STRING CHAR
        BEQ         EXIT                * QUIT IF IT'S
                                          A NULL
        EXT.W       D0                  * CLEAR HIGH
                                          BYTE
        CMP.B       #'%',D0             * SEE IF IT'S
                                          CONTROL FLAG
        BNE         NO_CTL              * BRANCH IF
                                          NOT
        CLR.W       LEFTJ(A6)           * CLEAR LEFT
                                          JUSTIFY FLAG
        CLR.W       FIELD(A6)           * CLEAR FIELD
                                          WIDTH
        CLR.W       SIGNF(A6)           * CLEAR SIGN
                                          FLAG
LPOPF:
        MOVE.B      (A1)+,D0            * GET NEXT
                                        CTRL STRING CHAR
        EXT.W       D0                  * CLEAR HIGH
                                          BYTE
        CMP.B       #'-',D0             * IS IT MINUS?
        BNE         SKOPF               * NO, TRY
                                          AGAIN
        MOVE.W      #1,LEFTJ(A6)        * SET LEFT
                                          JUSTIFY FLAG
        BRA         LPOPF               * TRY FOR NEXT
                                          CHAR
SKOPF:
        CMP.B       #'0',D0             * IS IT < 0?
        BLT         SK1PF               * YES,CONTINUE
                                          PROCESSING
        CMP.B       #'9',D0             * IS IT > 9?
        BGT         SK1PF               * YES,CONTINUE
                                          PROCESSING
        MOVE.W      FIELD(A6),D1        * GET CURRENT
                                          FIELD SIZE
        MULU        #10,D1              * SHIFT LEFT
                                         ONE DEC DIGIT
        AND.W       #$000F,D0           * CONVERT
                                        ASCII TO NUMBER
        ADD.W       D0,D1               * ADD TO FIELD
                                          WIDTH
        MOVE.W      D1,FIELD(A6)        * SAVE IT
        BRA         LPOPF               * GET NEXT
                                          FORMAT CHAR
SK1PF:
        MOVE.L      #DISPATCH,A0        * GET CASE
                                          TABLE ADDR
        BRA         CASE                * DO CASE
NO_CTL:
        MOVE.W      D0,-(A7)            * PUSH CHAR
        BSR         PUTC                * PRINT IT
        ADDQ.L      #2,A7               * TRASH
                                          PARAMETER
        BRA         LOOP                * DO IT AGAIN
EXIT:
        MOVEM.L     (A7)+,D0-D6/A0-A2
        UNLK        A6
        RTS
*
*
D_ARG:
        MOVE.W      (A2)+,D0            * GET VALUE,
                                          MOVE POINTER
        EXT.L       D0                  * CONVERT TO
                                         COMMON FORMAT
        BSR         SIGN                * PRINT SIGN,
                                          GET ABS()
        BRA         PRINTDEC            * PRINT VALUE
*
*
U_ARG:
        MOVE.W      (A2)+,D0            * GET VALUE,
                                          MOVE POINTER
        AND.L       #$0000FFFF,D0       * ZERO HIGH
                                          WORD
        BRA         PRINTDEC            * PRINT VALUE
*
*
D1_ARG:
        MOVE.L      (A2)+,D0            * GET VALUE,
                                          MOVE POINTER
        BSR         SIGN                * PRINT SIGN,
                                          GET ABS()
        BRA         PRINTDEC            * PRINT VALUE
*
*
U1_ARG:
        MOVE.L      (A2)+,D0            * GET VALUE,
```

Continued On Page 36

```
                                             MOVE POINTER
        BRA         PRINTDEC               * PRINT VALUE
        PAGE
*
*
*   SIGN
*
*   PRINT SIGN IF NEEDED AND TAKE ABS() OF VALUE.
*
SIGN:
        TST.L       D0                     * IS IT
                                             NEGATIVE?
        BPL         SK0_SG                 * EXIT IF NOT
        MOVE.W      #-1,SIGNF(A6)          * FLAG SIGN
                                             NEEDED
        SUBQ.W      #1,FIELD(A6)           * TAKE AWAY
                                             ONE FOR SIGN
        NEG.L       D0                     * MAKE ABS()
SK0_SG:
        RTS
*
*
*   PRINTDEC
*
* COMMON DECIMAL OUTPUT ROUTINE.  VALUE IS IN D0 UPON
ENTRY.
*
PRINTDEC:
        CLR.W       D1              * OUTPUT DIGIT COUNT
LP0_PD:
        DIVU        #10,D0      * DIVIDE NUMBER BY 10
        BVS         O_FLOW      * NUMBER TOO LARGE
        SWAP        D0          * GET REMAINDER IN D0.W
        MOVE.W      D0,-(A7)    * PUSH DIGIT
        ADDQ.W      #1,D1       * BUMP DIGIT COUNT
        CLR.W       D0          * GET RID OF REMAINDER
        SWAP        D0          * PUT QUOTIENT IN D0.W
        TST.W       D0          * IF ZERO, ALL DONE
        BNE         LP0_PD      * LOOP IF NOT DONE
        MOVE.W      D1,D6       * USED FOR FIELD ADJUST
        BSR         PREFIX      * DO PRFIX SPACES
        SUBQ.W      #1,D1     * ADJUST LOOP INDEX CNTR
LP2_PD:
        ADD.W       #$30,(A7) * MAKE DIGIT ON TOS ->
                                  ASCII
        BSR         PUTC        * SEND TO OUTPUT
        ADDQ.L      #2,A7       * EAT DIGIT FROM TOS
        DBF         D1,LP2_PD * LOOP UNTIL ALL DONE
        BSR         POSTFIX     * DO POSTFIX SPACES
        BRA         LOOP        * EXIT TO CONTROL
                                  PARSER
O_FLOW:
        MOVE.L      #OFLOWSTR,-(A7)        * PUSH CONTROL
                                             STRING ADDR
        BSR         PRINTF$                * PRINT IT
        ADDQ.L      #4,A7                  * ADJUST STACK
        BRA         LOOP                   * CONTINUE
OFLOWSTR:
        DB          '*overflow*',0
        PAGE
*
*   PREFIX
*
*   OUTPUT ANY NEEDED PREFIX SPACES AND SIGN.
*
PREFIX:
        TST.W       FIELD(A6)  CHECK IF FIELD NONZERO
        BLE         CHKSIGN    IF ZERO, SKIP NEXT PART
        TST.W       LEFTJ(A6)  LEFT JUSTIFY SELECTED?
        BNE         CHKSIGN             BRANCH IF NOT
        SUB.W       D6,FIELD(A6)        DIGITS ALLOWED
                                        - ACTUAL
        BLE         CHKSIGN             BRANCH IF NO
```

```
                                                SPACES NEEDED
LP0_PR:
        MOVE.W      D0,-(A7)                    SAVE D0 ACROSS CAL
TO _PUT
        MOVE.W      #$2020,-(A7)                1 SPACE TO
                                                OUTPUT
        BSR         PUTC            SEND THE SPACE
        ADDQ.L      #2,A7           ADJUST STACK
        MOVE.W      (A7)+,D0        RETRIEVE D0
        SUBQ.W      #1,FIELD(A6)    DECREMENT LOOP INDEX
        BNE         LP0_PR          LOOP IF MORE SPACES
CHKSIGN:
        TST.W       SIGNF(A6)                   SIGN NEEDED?
        BEQ         CHKEXIT                     JUMP IF NOT
        MOVE.L      D0,D2                       SAVE D0 ACROSS CAL
TO _PUT
        MOVE.W      #$202D,-(A7)                PUSH SIGN
        BSR         PUTC                        SEND IT
        ADDQ.L      #2,A7                       ADJUST STACK
        MOVE.L      D2,D0                       RETRIEVE D0
CHKEXIT:
        RTS
        PAGE
*
*   POSTFIX
*
*   PRINTS ANY POSTFIX SPACES.
*
POSTFIX:
        SUB.W       D6,FIELD(A6)                DIGITS ALLOWED
                                                - ACTUAL
        BLE         SK1_PO                      EXIT IF NOT
                                                NEEDED
LP0_PO:
        MOVE.W      #$2020,-(A7)                SPACE TO SEND
        BSR         PUTC                        OUTPUT THE
                                                SPACE
        ADDQ.L      #2,A7                       ADJUST STACK
        SUBQ.W      #1,FIELD(A6)                COUNT THIS
                                                SPACE
        BNE         LP0_PO                      LOOP UNTIL
                                                DONE
SK1_PO:
        RTS
        PAGE
X_ARG:
        MOVE.W      #3,D1                       NUMBER OF
                                               DIGITS TO PRINT
        MOVE.W      #4,D6                       USED FOR FIELD
                                                ADJUST
        MOVE.W      (A2)+,D2                    TRANSFER
                                                OUTPUT VALUE
        SWAP        D2                          POSITION
                                                OUTPUT VALUE
        BRA         PRINTHEX                    DO IT
*
*
X1_ARG:
        MOVE.W      #7,D1                       NUMBER OF
                                               DIGITS TO PRINT
        MOVE.W      #8,D6                       USED FOR FIELD
                                                ADJUST
        MOVE.L      (A2)+,D2                    TRANSFER
                                                OUTPUT VALUE
        BRA         PRINTHEX                    GO PRINT IT
        PAGE
*
*   PRINTHEX
*
*   OUTPUTS VALUE IN D2 IN HEX.  D1 IS NUMBER OF
    DIGITS TO PRINT.
*
PRINTHEX:
```

```
         BSR      PREFIX        OUTPUT PREFIX SPACES
LPOPH:
         MOVE.L   #HEXDIGITS,A0    ADDRESS OF
                                   TRANSLATE TABLE
         ROL.L    #4,D2            PUT MSD IN LOW
                                   FOUR BITS
         MOVE.W   D2,D0            PUT IN WORKING
                                   REGISTER
         AND.W    #$000F,D0        LEAVE ONLY LOW
                                   4 BITS
         MOVE.B   0(A0,D0.W),D0    GET DIGIT FROM
                                   TABLE
         MOVE.W   D0,-(A7)         PUT ON STACK
         BSR      PUTC             SEND IT
         ADDQ.L   #2,A7            DROP PARAMETER
         DBF      D1,LPOPH         LOOP UNTIL
                                   DONE
         BSR      POSTFIX          ADD TRAILING
                                   SPACES
         BRA      LOOP             DO NEXT
                                   CONTROL CHAR
HEXDIGITS:
         DB       '0123456789ABCDEF'
         PAGE
S_ARG:
         MOVE.L   (A2),A0          GET STRING
                                   ADDR FROM STACK
         CLR.W    D6               GET STRLEN FOR
                                   FIELD ADJ
SLEN:
         TST.B    (A0)+            LOOK FOR
                                   TERMINAL NULL
         BEQ      SK0_SA           BRANCH IF
                                   FOUND IT
         ADDQ.W   #1,D6            COUNT THIS
                                   CHAR
         BRA      SLEN             LOOK AT NEXT
                                   CHAR
SK0_SA:
         MOVE.L   (A2)+,A0         GET STRING
                                   ADDR AGAIN
         BSR      PREFIX           SEND LEADING
                                   SPACES
LP0_SA:
         TST.B    (A0)             END OF STRING?
         BNE      SK1_SA           NO, KEEP
                                   PRINTING
         BSR      POSTFIX          ALL DONE, SEND
                                   FINAL SPACES
         BRA      LOOP             JUMP OUT
SK1_SA:
         MOVE.B   (A0)+,D0         GET CHAR FROM
                                   STRING
         MOVE.W   D0,-(A7)         PUT ON STACK
         BSR      PUTC             AND SEND IT
         ADDQ.L   #2,A7            ADJUST STACK
         BRA      LP0_SA           CONTINUE
*
*
C_ARG:
         MOVE.W   (A2)+,-(A7)      GET ARGUMENT
         BSR      PUTC             SEND LITERAL
                                   CHAR
         ADDQ.L   #2,A7            DROP ARGUMENT
         BRA      LOOP             NEXT COMMAND
*
*
```

```
V_ARG:
         MOVE.L   (A2)+,-(A7)      MOVE BOTH ARGS
                                   AT ONCE
         BSR      CURSOR           POSITON THE
                                   CURSOR
         ADDQ.L   #4,A7            DROP BOTH ARGS
         BRA      LOOP             CONTINUE
                                   COMMANDS
*
*
DEFAULT:
         MOVE.W   D0,-(A7)         PRINT CHAR AS
                                   IS
         BSR      PUTC
         ADDQ.L   #2,A7            ADJUST STACK
         BRA      LOOP             DO NEXT
         PAGE
*
*  CASE DISPATCH TABLE FOR PRINTF.
*
DISPATCH:
         DW       9                NUMBER OF
                                   VALID OPTIONS
         DB       0,'d'         %d PRINT SIGNED
                                   DECIMAL W
         LONG     D_ARG         ADDRESS

         DB       0,'u'         %u PRINT UNSIGNED
                                   DECIMAL W
         LONG     U_ARG         ADDRESS

         DB       0,'D'         %D  PRINT SIGNED
                                   DECIMAL L
         LONG     D1_ARG        ADDRESS

         DB       0,'U'         %U  PRINT UNSIGNED
                                   DECIMAL L
         LONG     U1_ARG        ADDRESS

         DB       0,'x'         %x PRINT
                                   HEXADECIMAL W
         LONG     X_ARG         ADDRESS

         DB       0,'X'         %X  PRINT
                                   HEXADECIMAL L
         LONG     X1_ARG        ADDRESS

         DB       0,'s'         %s  PRINT NULL-TERM
                                   STRING
         LONG     S_ARG         ADDRESS

         DB       0,'c'         %c  PRINT CHARACTER
         LONG     C_ARG         ADDRESS

         DB       0,'v'         %v  SET CURSOR TO
                                   X,Y
         LONG     V_ARG         ADDRESS

         LONG     DEFAULT       UNKNOWN CASES
                                HANDLED HERE

         END
```