Some Thoughts on a 68xxx ROM Monitor

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

Karl Lunt

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 publicdomain 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 $680 \times 0 \, \text{code}$, using the PRINTF function, will do just that.

```
MOVE.L
          A0. - (A7)
                             FIRST RAM LOC IN AO TO
                             STACK
MOVE.L
          DO. - (A7)
                             AMOUNT OF RAM IN DO TO
MOVE 1.
          #FMTSTRG, - (A7)
                             PUSH ADDRESS OF STRING
                             DISPLAY THE STRING
ADD.L
                             PULL 12 BYTES FROM STACK
          #12,A7
                             REST OF PROGRAM FOLLOWS
DDA
 FMTSTRG:
BYTE
          SOD, SOA
           'AVAILABLE RAM: %D
DVTD
           'FIRST RAM LOCATION: SAX'
```

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

REQUIRED NULL TERMINATOR

SOD, SOA

DVTD

```
WORD decimal
    displays 16-bit signed decimal LONG decimal
*d
%D
    displays 32-bit signed decimal WORD decimal
*11
    displays 16-bit unsigned decimal LONG decimal
    displays 32-bit unsigned decimal WORD hex
9.11
%×
    displays 16-bit hex LONG hex
%X
    displays 32-bit hex string
85
    displays null-terminated string character .
20
    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 GETCO, 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 overwriten 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 tep 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 aroutine 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 pointier 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 only global RAM variables used by the monitor are the I/O links of the RAM of the

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 luge routine called MONITIORS and a very small outer loop that exists solely to call MONITIORS and a very small outer loop that exist solely to call MONITIOR. At that I'vo uwant to access the monitor from within a user program, you need only JSR to the ROM address named MONITIOR. At that much eachieve port vill 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 obliga-

tory 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 vou?

DMW

BUFFER

CHARS

· MONITOR

vector is aimed.

EXTERNAL.

Main routine in the 68010 EPROM.

Note that there is no ORG statment in this block. I

should be
* LIMKed to appear at address \$800100, as that is when
the COLDS

EXTERNAL PRINTF, SINIT, PUTC, GETC, GETLINE
EXTERNAL DUMPHUM, FILL, SWEEP
EXTERNAL PROMPT
EXTERNAL CASE, UPPER
EXTERNAL ACTEAN, TESTRAM

EXTERNAL S19, GO
EXTERNAL MONITOR, WARMS1
EXTERNAL DUMP_REG\$

PUBLIC WARMS\$, COLDS\$, PROCESS\$, MONITORS
PUBLIC WARMS1\$

FOU -256 STACK-BASED TEXT BUSEER

EQU -256 STACK-BASED TEXT BUFFER STACK MUST ALLOW FOR 256 REFER TO MONITOR

BY BOTH COLDS AND

68 Micro Journal

(BELOW) FOR USE

MON_SP: EQU SDF0 STACK POINTER
USED BY MONITOR.
THIS IS THE
VALUE WRITTEN TO A7

WARMS1.

COLDSS:

MOVE.W #53700,8450000 LEDS: G R
MOVE.L #5400000,A0 POINT AT START

MOVE.W #\$3700,\$450000 LEDS: G R MOVE.L #\$400000,A0 POINT AT START OF MAP RAM MOVE.W #\$3FF,DO 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

26

DOVE.W DOVE.L DOVE.L DOVE.L DOVE.L DOVE.W DOVE.L DOVE.W DO	DO, MAPTEST STARTUP RAME'AIL #\$3880, \$450000 #\$2000, \$450000 #\$2000, \$450000 #\$3500, \$450000 #\$3500, \$450000 #\$3500, \$450000 #\$35000, \$450000 #\$35000, \$450000 #\$35000, \$450000 #\$35000, \$450000 #\$35000, \$450000 #\$35000, \$450000 #\$35000, \$450000 #\$35000, \$450000 #\$35000, \$450000 #\$35000, \$450000 #\$35000, \$450000 #\$35000, \$450000 #\$35000, \$4500000 #\$35000, \$4500000 #\$35000, \$4500000 #\$35000, \$4500000 #\$35000, \$4500000 #\$35000, \$4500000 #\$35000, \$4500000 #\$35000, \$4500000 #\$35000, \$4500000 #\$35000, \$4500000 #\$35000, \$45000000000000000000000000000000000000	BRANCH IF FAILURE LEDS: G Y ACTIVATE ALL SYSTEM RAM SYSTEM RAM INITIALIZE THE SERIAL POST ESTATE MESSAGE DISPLAY IT ADJUST STACK LEDS: G 8 BEEN THROUGH HIS BEFORE? SKIP TEST IF SO TEST ALL	BYTE 'AN EXIT back to the ROM Monitor BYTE 200, 50A 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 for WARM Alternate entry is from a user program via the WARMS for WARM 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 ser- program to activate the monitor with a special ser- very program to activate the monitor with a special ser- program to activate the monitor with a special ser- within the program, use the monitor as if it had be activated by a warm-start, then return to the user program an EXIT monitor command. Entry by this technique should be the calling program with ALL registers preserved, though the CCR is not saved. MONITORS: LINK A6, BBUFFER SET THE TEXT MOVEN.L DO-D7/AG-A7,-GA7) SAVE EVENTIFING LOOF: BSS PROMET
DVE.W DVE.L DVE.L WE SVE.W SVE	RAMFAIL ##3280, \$430000 ##2000, #400000 ##2000, #400000 ##800, \$F, A7 ##8ASASASAS, \$0F ##8ASASASAS, \$7F ##8ASASASAS, \$7F	BRANCH IF ALL GOOD DIE HERE WITH LEDS ON LEDST R O SMITCH O SMITCH O SMITCH O SMITCH O SMITCH O SMITCH O TEST ALCATION COMEST RAM LEDST G G ACTIVATE ALL SYSTEM RAM G INITIALIZE THE SERIAL PORT LEDST G G ET FIRST LEDST G G ET FIRST LEDST G SMITCH ORD SMITCH ORD THIS BEFORE SKIP TEST IF SO TEST ALL	BYTE 900.50A BYTE 00 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 MARMS for MANI entry A user program may also enter via the MONITOR jump vector in the ROM jump table. Entering through this point permits a user rougher to activate the monitor with a special service within the program, use the monitor as if it had be activated by a warm-start, then return to the user program wan EXIT LINK activated the monitor with a special service within the program, use the monitor as if it had be activated by a warm-start, then return to the user program wan EXIT LINK activated the monitor this way will return to the calling program with ALL registers preserved, though the CCR is not saved. MONITORS: LINK AG, BUMPFER SET THE TEXT MONITORS: LOPE SER PROMET IN STACK SPACE MOVEN.L. DO-D7/AG-A7,-GA7) SAVE EVENTIFING
DVE.W DVE.L DVE.L DVE.L DVE.L SR DVE.W SR DVE.W SR DVE.W SR DVE.W SR DVE.W SR DVE.W SR	#\$3800,\$450000 #\$2000,\$40000 #\$000,\$40000 #\$3535,\$05,\$05 #\$3535,\$05 #\$3500,\$45000 ACTRAM #\$3900,\$45000 \$1011 #\$3900,\$45000 #HELLO,-(A7) PRINTF #4,A7 #53900,\$450000	LEDS ON LEDS: R C SMITCH ON LOWEST 4K BLOCK PUT STACK IN LOWEST RAM LOWEST R	Normal entry is from a COLD start following reset. Alternate Alter
DVE.W DVE.L DVE.L DVE.L DVE.L SR DVE.W SR DVE.W SR DVE.W SR DVE.W SR DVE.W SR DVE.W SR	#\$3800,\$450000 #\$2000,\$40000 #\$000,\$40000 #\$3535,\$05,\$05 #\$3535,\$05 #\$3500,\$45000 ACTRAM #\$3900,\$45000 \$1011 #\$3900,\$45000 #HELLO,-(A7) PRINTF #4,A7 #53900,\$450000	LEDS ON LEDS: R C SMITCH ON LOWEST 4K BLOCK PUT STACK IN LOWEST RAM LOWEST R	Normal entry is from a COLD start following reset. Alternate Alter
DVE.W DVE.L DVE.L DVE.L WE DVE.W SR DVE.W SR DVE.W	#\$2000, \$400000 #MON_SP, A7 #\$5A5A5A5, SOF #\$5A5A5A5A5, SFF #\$A5A5A5A5A5, SFF	SWITCH ON LOWEST 4K BLOCK PUT STACK IN PUT STACK PUT STACK LOWE IN LEDS: Y G GET FIRST MESSAGE IT DAUJUST STACK LEDS: G MESSAGE IT PUT STACK LEDS: G MESS	e entry is from a user program via the WARMS (or WANI 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 activated by a warm-start, then return to the user program an EXIT monitor command. Entry by this technique should by calling program with ALI registers preserved, though the CCR is not saved. MONITORS: LINK A6, BUUFFER SET THE TEXT MONITORS: LOPE BSE PROMET LOPE BSE PROMET LOPE BSE PROMET BSE PROMET BSE PROMET BSE PRO
DVE.W DVE.L DVE.L DVE.L WE DVE.W SR DVE.W SR DVE.W	#\$2000, \$400000 #MON_SP, A7 #\$5A5A5A5, SOF #\$5A5A5A5A5, SFF #\$A5A5A5A5A5, SFF	SWITCH ON LOWEST 4K BLOCK PUT STACK IN PUT STACK PUT STACK LOWE IN LEDS: Y G GET FIRST MESSAGE IT DAUJUST STACK LEDS: G MESSAGE IT PUT STACK LEDS: G MESS	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 sequence from within the program, use the monitor as if it had be activated an EXIT monitor command. Entry by this technique should be with a JSR. Loaving the monitor this way will return to the calling program with ALL registers preserved, though the CCR is not severe. MONITORS: LINK AG. BUFFER IN STACK SPACE MOVEN.L DO-D7/AG-A7, GATS SAVE EVERYTHING
DVE.L MP.L ME DVE.W SR DVE.W SR DVE.W SR DVE.W SP DVE.W DVE.W SP DVE.W D	#\$A\$A\$A\$A\$, \$0F #\$A\$A\$A\$A\$, \$0F #\$A\$A\$A\$A\$, \$0F #\$3800, \$450000 ACTRAM #\$3900, \$450000 \$INIT #\$4, A7 #4, A7 #45, A7 #45, A7 #45, A7	COMEST RAM	vector in the ROM jump table. Entering through this point program to activate the monitor with a special sequence from within the program, use the monitor as if it had be activated as EXIT monitor command. Entry by this technique should be with a JSR. Laaving the monitor this way will return to the Laaving the monitor this way will return to the succession of the second o
MP.L VE.W SR OVE.W SR OVE.W SR OVE.W MP.L EQ	#\$A\$A\$A\$A\$, \$OF #\$3800, \$450000 ACTRAM #\$3800, \$450000 SINIT #\$3800, \$450000 ##BLLO, - (A7) PRINTF #4, A7 #45, A7 #53900, \$450000 #\$A\$A\$A\$A\$, \$FF	BRANCH IF FAILURE LEDS: G Y ACTIVATE ALL SYSTEM RAM SYSTEM RAM INITIALIZE THE SERIAL POST ESTATE MESSAGE DISPLAY IT ADJUST STACK LEDS: G 8 BEEN THROUGH HIS BEFORE? SKIP TEST IF SO TEST ALL	permits a user porgram to activate the monitor with a special sequence from program to activate the monitor as if it had be activated by a warm-start, then return to the user program wan EXIT monitor command. Entry by this technique should be with a JSR. Leaving the monitor this way will return to the calling program start and the monitor this way will return to the saved. MONITORS: LINK A6. BUFFER SET THE TEXT BUFFER IN STACK SPACE MOVEN.L DO-D7/AO-A7, CAT) SAVE EVERYTHING LOOF: BSR PROMET
DOVE.W SR DOVE.W SR DOVE.W DOVE.L SR DOVE.W DOVE.L SR DOVE.W MP.L EQ	RAMFAIL \$3300,\$450000 \$453900,\$450000 \$151N17 \$453000,\$450000 \$HELLO,-(A7) PRINTF 44,A7 4539700,\$450000 \$5858585855,\$FF	BRANCH IF FAILURE LEDS: G Y ACTIVATE ALL SYSTEM RAM LEDS: _ Y G Y INITIALIZE THE SERIAL PORT LEDS: _ Y G GCFFIRST GCF	quence from within the program, use the monitor as if it had becivated activated an EXIT monitor command. Entry by this technique should by with a JSR. Leaving the monitor this way will return to the calling program with ALL registers preserved, though the CCR is not saved. MONITORS: LINK A6, #BUFFER BUFFER IN STACK SPACE MOVEM.L DO-D7/AG-A7, -GA7) SAVE EVENTHING LOOP: BSR PROMET
DOVE.W SR DOVE.W DOVE.L SR DOQ.L DOVE.W MP.L	#\$3900,\$450000 \$INIT #\$3900,\$450000 #HELLO,-(A7) PRINTF #4,A7 #53F00,\$450000 #\$A5A5A5A5,\$FF	ACTIVATE ALL SYSTEM RYG Y LEDS: YG Y INITIALIZE THE SERIAL PORT LEDS: YG G GET FIRST MESSAGE DISPLAY II ADJUST STACK LEDS: G HERN THROUGH THIS BEFORE? SKIP TEST IF SO TEST ALL	activated by a warm-start, then return to the user program was not been a consider 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. MONITORS: LINK A6, BBUFFER SET THE TEXT BUFFER IN STACK SPACE MOVEN.L DO-D7/A0-A7,-GAT) SAVE EVENTTHING LOG: BSR PROMET
DOVE.W DOVE.L SR DOQ.L DOVE.W MP.L SQ	#\$3D00,\$450000 #HELLO,-(A7) PRINTF #4,A7 #53F00,\$450000 #\$A\$A\$A\$A\$A\$,\$FF	LEDS: Y G Y INITIALIZE THE SERIAL PORT LEDS: Y G G GET FIRST MESSAGE DISPLAY IT ADJUST STACK LEDS: G G BEEN THROUGH THIS BEFORE? SKIP TEST IF SO TEST ALL	* 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 seven. MONITORS: LINK A6, BUFFER SET THE TEXT BUFFER IN STACK SPACE MOVEN.L D0-07/A0-A7,-GAT) SAVE EVERYTHING LOOP: BSR PROMET
DVE.W DVE.L SR DDQ.L DVE.W MP.L	#\$3000,\$450000 #HELLO,-(A7) PRINTF #4,A7 #\$3F00,\$450000 #\$A5A5A5A5,\$FF	SERIAL PORT LEDS: _Y G _ GET FIRST MESSAGE DISPLAY IT ADJUST STACK LEDS: _ G _ 8 BEEN THROUGH THIS BEFORE? SKIP TEST IF SO TEST ALL	with a JSR. Leaving the monitor this way will return to the calling program with ALL registers preserved, though the CCR is not saved. MONITORS: LINK A6, BUFFER SET THE TEXT BUFFER IN STACK SPACE MOVEM.L DOL-07.40-A7,-(A7) SAVE EVERYTHING LOOP: BSR PROMPT
DVE.L SR DDQ.L DVE.W MP.L SQ SR	#HELLO, - (A7) PRINTF #4, A7 #53F00, \$450000 #\$A5A5A5A5, \$FF WARMS1\$	GET FIRST MESSAGE DISPLAY IT ADJUST STACK LEDS: _ G 8 BEEN THROUGH THIS BEFORE? SKIP TEST IF SO TEST ALL	with ALL registers preserved, though the CCR is not saved. MONITORS: LINK A6, BBUFFER SET THE TEXT BUFFER IN STACK SPACE HOVEN.L. LOOP: BSR PROMET
SR DDQ.L DDQ.L DVE.W MP.L	PRINTF #4,A7 #33F00,S450000 #SA5A5A5A5,SFF WARMS1S	MESSAGE DISPLAY IT ADJUST STACK LEDS: _ G BEEN THROUGH THIS BEFORE? SKIP TEST IF SO TEST ALL	saved. MONITORS: LINK A6, BUFFER SET THE TEXT BUFFER IN STACK SPACE MOVEN.L D0-D7/A0-A7, -(A7) SAVE EVERYTHING LOOF: BSS PROMET
OVE.W MP.L SQ	#\$3F00,\$450000 #\$A5A5A5A5,\$FF WARMS1\$	LEDS: G 8 BEEN THROUGH THIS BEFORE? SKIP TEST IF SO TEST ALL	LINK A6, #BUFFER SET THE TEXT BUFFER IN STACK SPACE MOVEM.L D0-D7/A0-A7, -(A7) SAVE EVERYTHING LOOP: BSR PROMPT
MP.L SQ I	#SA5A5A5A5, SFF WARMS1\$	BEEN THROUGH THIS BEFORE? SKIP TEST IF SO TEST ALL	LINK A6, #BUFFER SET THE TEXT BUFFER IN STACK SPACE MOVEM.L D0-D7/A0-A7, -(A7) SAVE EVERYTHING LOOP: BSR PROMPT
EQ I	WARMS1\$	THIS BEFORE? SKIP TEST IF SO TEST ALL	MOVEM.L D0-D7/A0-A7,-(A7) SAVE EVERYTHING LOOP: BSR PROMPT
SR .		TEST ALL	BSR PROMPT
		AVAILABLE RAM	LEA BUFFER(A6), AO POINT AO AT
	DO,-(A7)	PUSH SIZE OF RAM	BSR GETLINE TEXT STRING
	DO,-(A7) #SIZE,-(A7)	PUSH IT AGAIN PUSH MESSAGE	BSR PROCESS\$ BRA LOOP
	PRINTF	DISPLAY IT	* DAW FOOL
D.L	#12,A7	REPAIR STACK 8 WRITE FLAG TO	* NOTE: Exit from this routine is via the monitor Elementary,
		THROUGH HERE	* whose code follows in the case structure below and labled
	AL	READI (SEE STARTUP)	EXITS.
DDQ.L	#4,A7	FIX THE STACK	* PROCESS
OVE.L			* This is the core of the ROM monitor. It executes to
			* found in the first two character positions of the !
SR I			* at by AO. Note that this routine is available extendily via
SR I		MONITOR WITHOUT	* the jump table. This permits a user to load a stri with a
			* monitor command, put the string's address in AO and JSR here
OVE.L			 so the monitor can process the command. As is cust ary, any
		ISN'T GOING TO WORK	* string submitted to PROCESS must be terminated with
			null byte. * Additionally, any monitor command must be two chara
			ters long * and must begin in column one.
יים	VE.L	ALL VE.L. #FIRST,-(A7) R PRINTY OJ.L. #4, A7 VE.L. #MON_SP, A7 R DUMP_REGS R MONITORS VE.L. #EXITHSG,-(A7) VE.L. #EXITHSG,-(A7) OJ.L. #4, A7	SHOW ME'VE BEEN ATHROMOGH HERE ATHRO

BY	TE	\$0D,\$0A			BYTE	\$0D, \$0A	addr>'
ZE:					BYTE	' RAM Load (S1	
BY	TE	0			BYTE	SOD, SOA	<stop> <data>'</data></stop>
BY	TE.	' Beats m again.'	e what you want. Try		BYTE	' Fill memory	FI <start></start>
BY		\$0D,\$0A			BYTE	\$0D, \$0A	
HAT_MSG:					BYTE	100,400	CH <addr>'</addr>
					BYTE	<da \$0D,\$0A</da 	ta> <data> or'</data>
					BYTE	' Change memor	
BR		PROCX			BYTE	\$0D, \$0A	
AD	DQ.L	#4,A7			5115	' Sweep memory	SW <start> <stop> [times]'</stop></start>
MO BS		#WHAT_MSG, PRINTF	- (A/)		BYTE	SOD, SOA	an
HAT:	VE.L	ANUAR MCC	4871		BYTE	' Dump memory	DU <addr>'</addr>
					BYTE	\$0D, \$0A, \$0A	
					BYTE	'commands are:	
	as not t	oo good.			BYTE	'one space. Avai \$0D,\$0A	Table.
This is	the def	aust CASE	arm. It just tells the t	ser	BYTE	by at least '	1-61-4
mbin i-			Th durah halls ::		BYTE	'Separate all arg	uments (in hex)
WHAT					BYTE	\$0D, \$0A	
					BYTE	'by any arguments	.*
						followed '	cao characters,
	RIS	LE	AVE THE MUNITOR ROUTINE		BYTE	\$0D,\$0A,\$0A 'All commands are	
	UNLK		MOVE THE FRAME AVE THE MONITOR ROUTINE	FIRST:	nuen		
			EVERYTHING			70	
	MOVEM.I		DO-D7/AO-A7 RESTORE		BYTE	0	
	ADDQ.L	#4, A7 P	OP THE PROCESS RTN ADDR		BYTE	\$0D,\$0A	
KITS:					BYTE	Technologi ' Mini-Frame'	es'
	BRA	PROCX			BYTE	Written fo	r the Convergent
0\$:	BSR	GO			BYTE .	\$0D,\$0A	
					BYTE		MONITOR V1.2'
	BRA	PROCX	4		BYTE	SOD, SOA, SOA	
19\$:	BSR	S19		HELLO:			
	BRA	PROCX			LONG	WHAT	DEFAULT CASE
IANGES:	BSR	CHANGE					
					LONG	EXIT\$	PULL COLLINE
	BRA	PROCX			BYTE	'E','X'	EXIT COMMAND
ILLS:	BSR	FILL			LONG	GO\$	
	BRA	PROCX			BYTE	'G','O'	GO COMMAND
WEEP\$:	BSR	SWEEP					
	2141	Linoun			LONG	S19\$	(S19) COMMAND
UMPMEMS:	BSR BRA	PROCX			BYTE	'R','L'	RAM LOAD
UMPMEMS:	BSR	DUMPMEM			nyan	104 414	
					LONG	HELP\$	
TABLE.					BYTE	'H','E'	HELP COMMAND
THE LIN		PROPERLY	RESOLVE THE ADDRESSES IN	THE	LONG	CHANGE\$	
EEDED SO	VDD 41	nnonnn:	RESOLVE THE ADDRESSES IN				COMMAND
	BLE FOR	THE CASE S	WITCH ABOVE. THIS TABLE	IS	BYTE	'C','H'	CHANGE
c:					LONG	FILLS	
RT	s		RETURN TO MAIN LOOP		BYTE	`F','I'	FILL COMMAND
ROCX:							
BR	^	CASE	DO THE COMMAND		LONG	'S','W' SWEEPS	SWEEP COMMAND
BR		CASE	TABLE DO THE COMMAND		BYTE	'S'.'W'	
MC	VE.L	#PROC_TBL,	A0 GET ADDR OF PROCESS		LONG	DUMPMEMS	COURT COLLINE
PAL		1411,00	CHARACTERS		BYTE	'D','U'	DUMP COMMAND
BS	R VE.W	UPPER (A1),D0	CONVERT TO UPPERCASE GET THE FIRST TWO	PROC_TB	L: WORD	8	
2.0	_		INTO A1				
MC	VE.L	A0, A1	YES, MOVE POINTER		BYTE	\$0D,\$0A 0	
					BYTE		
BE	P.B Q	#0, (A0) PROCX	ANYTHING ON THE LINE? BRANCH IF NOT			First non-RAM add	iress: %X.'

next-to las . l	prints pr	(x,y) - push x, the rint next character 3000 Assembly Langu as on page 234.	as literal	SKIPF: HO_CTL: EXIT: D_ARG: U_ARG: U_ARG: U1_ARG:	MOVE.L BSR BRA	#DISPATCH, AO CASE DO, - (A7) PUTC **2, A7 LOOP (A7) +, D0 - D6/AO - A: A6 (A2) +, D0 SIGN PRINTDEC (A2) +, D0 **50000FFFF, D0 PRINTDEC (A2) +, D0 SIGN PRINTDEC (A2) +, D0 SIGN PRINTDEC (A2) +, D0	GET CASE TABLE ADDR PUSH CHAR PRINT IT TRASH TRASH OD IT AGAIN GET VALUE, MOVE POINTE GET VALUE, MOVE POINT GET ABS() PRINT VALUE GET VALUE, MOVE POINT FRINT VALUE FRINT VALUE FRINT VALUE FRINT VALUE FRINT VALUE FRINT VALUE GET VALUE, MOVE POINT GET ABS() FRINT VALUE GET VALUE,
Contribute of the contribute o	CUISOF INTERIOR TO THE TO THE TOTAL THE T	(x,y) - push x, the function and the function that charge (2)4. 15	as literal age, by Krantz and age, by Krantz and age, by Krantz and age, by Krantz and companies and age, by Krantz and age, by	MO_CTL: EXIT:	BRA MOVE W BSR ADDQ L BRA MOVEM L UNLK RTS MOVE N EXT L BSR BRA MOVE W AND L BRA MOVE L BRA	CASE DO, - (A7) PUTC PUTC VEX.A7 LOOP (A7) +, D0 - D6/A0 - A: A6 (A2) +, D0 DO SIGN PRINTDEC (A2) +, D0 \$ \$ \$ \$ \$ \$ \$ \$ \$ \$ \$ \$ \$ \$ \$ \$ \$ \$ \$	TABLE ADDR TABLE ADDR PUSH CHAR PRINT IT TRASH PRAMETER DO IT AGAIN GET VALUE, MOVE POINTE CONVERT TO PRINT SIGN, GET ABS () GET VALUE, MOVE POINTE ZERO NIGH WERE GET VALUE, MOVE POINTE CARRELLE GET VALUE, MOVE POINTE PRINT VALUE PRINT SIGN, GET ABS () PRINT SIGN, GET ABS () PRINT SIGN, GET ABS ()
Contribute of the contribute o	CUISOF INTERIOR TO THE TO THE TOTAL THE T	(x,y) - push x, the function and the function that charge (2)4. 15	as literal age, by Krantz and age, by Krantz and millions - GET CONTROL STRING ADDR - GET POINTER STRING ADDR - GET POINTER STRING GHAR - OUT IF IT'S A NULL CLEAR HIGH BER - CONTROL - CLEAR HIGH BER - CONTROL - CLEAR HIGH HOT - CLEAR HIGH HOT - CLEAR HIGH HOT - CLEAR LEFT - HOT - CLEAR FIELD - WIDTH	EXIT: D_ARG: U_ARG:	BRA MOVE.W BSR ADDO.L BRA MOVEM.L UNLK RTS MOVE.W EXT.L BSR BRA MOVE.W AND.L BRA MOVE.L BSR	CASE DO, -(A7) PUTC PUTC PUTC PUTC PUTC PUTC PUTC PUTC	TABLE ADDR TO CASE PUSH CHAR PRINT IT TRASH PARAMETER PARAMETER OD IT AGAIN GET VALUE, MOVE POINTE CONVERT TO COMMON FORM PRINT SIGN, GET ABS() PRINT VALUE PRINT VALUE PRINT VALUE GET VALUE, MOVE POINTE PRINT VALUE GET VALUE, MOVE POINTE PRINT SIGN, GET VALUE PRINT SIGN, GET ASIGN, GET ASIG
Control lass to the contro	CURSOF INTELLIFICATION OF THE ACTION OF THE	(x,y) - push x, the first firs	as literal age, by Krantz and age, by Krantz and age, by Krantz and selection of the select	EXIT: D_ARG: U_ARG:	BRA MOVE.W BSR ADDQ.L BRA MOVEM.L UNLK RTS MOVE.N EXT.L BSR BRA MOVE.W AND.L BRA	CASE DO, - (A7) PUTC PUTC PUTC PUTC PUTC PUTC PUTC PUTC	TABLE ADDR TO CASE PUSH CHAR PRINT IT TRASH PARAMETER PARAMETER PARAMETER CONVERT TO CONVERT TO PRINT SIGN, GET ABLO GET VALUE, MOVE POINTE ZERO NIGH NORTH TO CONVERT TO PRINT VALUE GET VALUE, MOVE POINTE CERT VALUE, MOVE POINTE SERO NIGH MOVE POINTE CET VALUE, MOVE POINTE PRINT VALUE MOVE POINTE PRINT SIGN, GET VALUE MOVE POINTE PRINT SIGN, FOR THE PRINT SIGN, MOVE POINTE PRINT SIGN, MOVE POINTE PRINT SIGN,
Contribute of the contribute o	CUISOT fault P: from '68 ing appear pRINTE EXTERN VARIABLE EQU EQU LINK MOVE.L LEA MOVE.B BEQ EXT.W CMF.B	(x,y) - push x, the first that characteristic next characteristic	as literal age, by Krantz and age, by Krantz and millons * GET CONTROL STAING ADDR * GET POINTER TO PARAMETERS GET CONTROL STAING CHAR * OUT IF IT'S A NULL CLEAR HIGH BYTE ECONTROL ELEAR HIGH BYTE ECONTROL ELEAR HIGH BYTE ECONTROL STAING CHAR * OUT FI TY'S A NULL CLEAR HIGH BYTE ECONTROL STAING BYTE ECONTROL BYTE E	EXIT: D_ARG: U_ARG:	BRA MOVE.W BSR ADDQ.L BRA MOVEM.L UNLK RTS MOVE.N EXT.L BSR BRA MOVE.W AND.L BRA	CASE DO, - (A71) PUTC PUTC PUTC PUTC PUTC PUTC PUTC PUTC	TABLE ADDR TABLE ADDR PUSH CHAR PRINT IT TRASHER FARAMETER FOOT AGAIN GET VALUE, MOVE FOINT COMMON PRINT FRINT SIGN, GET VALUE, MOVE FOINT GET VALUE, MOVE FOINT WORD PRINT VALUE GET VALUE, MOVE FOINT WORD PRINT VALUE GET VALUE,
Control Contro	Cursor sfault Pro- fault Pro- from '66 ing appear PRINTE EXTERN L VARIABLE EQU EQU LINK MOVE.L LEA MOVE.B BEQ EXT.W	(x,y) - push x, the function text charge (x) - push x, th	as literal age, by Krantz and age, by Krantz and millions GET CONTROL STRING ADDR GET POINTER TO FARMARTERS GET CONTROL STRING CHAR OUT IF IT'S A NULL CLEAR HIGH CLEAR HIGH SEE IF IT'S SEE IF IT'S	EXIT: D_ARG: U_ARG:	BRA MOVE.W BSR ADDO.L BRA MOVEM.L UNLK RTS MOVE.N EXT.L BSR BRA MOVE.W BRA	CASE DO, - (A7) PUTC PUTC PUTC (A7) +, D0-D6/A0-A: A6 (A2) +, D0 DO SIGN PRINTDEC (A2) +, D0 \$50000FFFF, D0	TABLE ADDR TO CASE PUSH CHAR PRINT IT TRASHER FARAMETER FO IT AGAIN GET VALUE, MOVE FOINE COMMON FORMA PRINT SIGN, GET MASI FRINT VALUE GET VALUE, MOVE FOINE GET VALUE, MOVE FOINE MORD DITHE
Control Contro	Cursor sfault P: from '66 ng appear PRINTE EXTERN VARIABLE EQU EQU EQU LINK MOVE.L LEA MOVE.B BEQ	(x,y) - push x, the first firs	as literal age, by Krantz and age, by Krantz and wittions GET CONTROL STRING ADDR GET FOUNTER GET CONTROL STRING CORR GITTING CHAR OUT IF IT'S CLEAR HIGH	NO_CTL: EXIT: D_ARG:	BRA MOVE.W BSR ADDQ.L BRA MOVEM.L UNLK RTS MOVE.W EXT.L BSR BRA MOVE.W AND.L	CASE DO, - (A7) PUTC PUTC PUTC (A7) +, D0-D6/A0-A: A6 (A2) +, D0 DO SIGN PRINTDEC (A2) +, D0 \$50000FFFF, D0	TABLE ADDR TO CASE PUSH CHAR PRINT IT TRASHER FARAMETER FO IT AGAIN GET VALUE, MOVE FOINE COMMON FORMA PRINT SIGN, GET MASI FRINT VALUE GET VALUE, MOVE FOINE GET VALUE, MOVE FOINE MORD DITHE
Control Contro	Cursor sfault P: from '66 ng appear PRINTE EXTERN VARIABLE EQU EQU LINK MOVE.L LLA MOVE.B	(x,y) - push x, the first that can be a considered as a constant of the consta	as literal age, by Krantz and MITIONS GET CONTROL STRING ADDR OET FOUNTER TO PARAMETERS OET FOUNTER OET OF OUT IF IT'S OUT IF IT'S OUT IF IT'S	NO_CTL: EXIT: D_ARG:	BRA MOVE.W BSR MOVEM.L UNLK RTS MOVE.W EXT.L BSR BRA MOVE.W	CASE DO, -(A7) PUTC #22, A7 LOOP (A7) +, D0 - D6/A0 - A: A6 (A2) +, D0 SIGN PRINTDEC (A2) +, D0	TABLE ADDR TO CASE PUSH CHAR PRINT IT TRASH PARAMETER DO IT AGAIN GET VALUE, WOVE POINTE COMMON FORM PRINT SIGN, GET ABS () PRINT VALUE GET VALUE, MOVE POINTE GET VALUE, MOVE POINTE GET ABS () PRINT VALUE GET VALUE, MOVE POINTE
Contr pusper to	Cursor pfault Pr from '68 ing appear PRINTF EXTERN VARIABLE EQU EQU EQU LINK MOVEM.L MOVE.L LEA	(x,y) - push x, the first that can be a considered as a constant of the consta	as literal age, by Krantz and wittons GET CONTROL STRING ADDR GET FOINTER TO PARAMETERS	NO_CTL: EXIT: D_ARG:	BRA MOVE.W BSR ADDQ.L BRA MOVEM.L UNLK RTS MOVE.W EXT.L BSR BRA	CASE DO, -(A7) PUTC PUTC PUTC PUTC PUTC PUTC PUTC PUTC	TABLE ADDR * PUSH CHAR * PRINT IT * TRASH PARAMETER PARAMETER DO IT AGAIN * GET VALUE, MOVE POINTE CONVERT TO COMMON FORM PRINT SIGN, GET ABS () * PRINT VALUE
Contr pus ext-to- las d d v v v v v v v v v v v v v v v v v	Cursor offault Pr ofform '68 ing appear PRINTE EXTERN VARIABLE EQU EQU EQU LINK MOVEM.L MOVE.L	(x,y) - push x, the first next chargin next next chargin next next next next next next next nex	as literal age, by Krantz and nitions GET CONTROL STRING ADDR GET POINTER	NO_CTL: EXIT: * D_ARG:	BRA MOVE.W BSR ADDQ.L BRA MOVEM.L UNLK RTS MOVE.W EXT.L BSR	CASE DD, -(A7) PUTC #2,A7 LOOP (A7)+,D0-D6/A0-A: A6 (A2)+,D0 D0 SIGN	TABLE ADDR DO CASE PUSH CHAR PRINT IT TAASH PRAMETER DO IT AGAIN GET VALUE, MOVE POINTE COMMON FORMA PRINT SIGN, GET ABL
Contr pus pus to- las td tu tb tb tc	Cursor ofault Pro ofform '66 ing appear PRINTE EXTERN VARIABLE EQU EQU EQU LINK MOVEM.L	(x,y) - push x, the int next character 1000 Assembly Langu 2s on page 234. 2s CASE, PUTC, CURSOR 2 DISPLACEMENT DEFI -2 -4 -6 A6, #-6 D0-D6/AO-A2, - (A7)	as literal age,' by Krantz and	NO_CTL:	MOVE.W BSR ADDQ.L BRA MOVEM.L UNLK RTS	CASE D0,-(A7) PUTC #2,A7 LOOP (A7)+,D0-D6/A0-A: A6 (A2)+,D0 D0	TABLE ADDR DO CASE PUSH CHAR PRINT IT TRASH PARAMETER DO IT AGAIN GET VALUE, MOVE POINTE CONVERT TO COMMON FORM PRINT SIGN, PRINT SIGN,
Contress of the contress of th	Cursor efault Prom '66 ing appear PRINTE EXTERN VARIABLE EQU EQU EQU	(x,y) - push x, the fint next character (stone and	as literal	NO_CTL:	MOVE.W BSR ADDQ.L BRA MOVEM.L UNLK RTS	DD, - (A7) PUTC #2, A7 LOOP (A7) +, D0-D6/A0-A: A6	TABLE ADDR DO CASE PUSH CHAR PRINT IT TRASH PARAMETER DO IT AGAIN GET VALUE, MOVE POINTE CONVERT TO
Contr pus pus to- las a a b b b b c s c s c c c c c c c c c c c c	Cursor efault Pa from '68 ing appear PRINTE EXTERN VARIABLE	(x,y) - push x, the rint next character 1000 Assembly Langu rs on page 234. S CASE, PUTC, CURSOR E DISPLACEMENT DEFI -2 -4	as literal	NO_CTL:	BRA MOVE.W BSR ADDQ.L BRA MOVEM.L UNLK RTS	DD, - (A7) PUTC #2, A7 LOOP (A7) +, D0-D6/A0-A: A6	TABLE ADDR * DO CASE * PUSH CHAR * PRINT IT * TRASH FARAMETER DO IT AGAIN * GET VALUE,
Contr pus ext-to- las a d d d d d d d d d d d d d d d d d d	Cursor efault Pa from '68 ing appear PRINTE EXTERN VARIABLE	(x,y) - push x, the rint next character 1000 Assembly Langu rs on page 234. S CASE, PUTC, CURSOR E DISPLACEMENT DEFI -2 -4	as literal	NO_CTL:	BRA MOVE.W BSR ADDQ.L BRA MOVEM.L UNLK	DD,-(A7) PUTC #2,A7 LOOP (A7)+,D0-D6/A0-A	TABLE ADDR DO CASE PUSH CHAR PRINT IT TRASH PARAMETER DO IT AGAIN
Contr pus lext-to- las	Cursor efault Pr from '68 ing appear PRINTE EXTERN VARIABLE	(x,y) - push x, the rint next character 0000 Assembly Langu s on page 234. 'S CASE, PUTC, CURSOR E DISPLACEMENT DEFI -2	as literal	NO_CTL:	BRA MOVE.W BSR ADDQ.L BRA MOVEM.L UNLK	DD,-(A7) PUTC #2,A7 LOOP (A7)+,D0-D6/A0-A	TABLE ADDR DO CASE PUSH CHAR PRINT IT TRASH PARAMETER DO IT AGAIN
Contr pus lext-to- las tau tau tau tax tax tax tax tax tax tax tax tax tax	Cursor efault Pr from '68 ing appear PRINTE	(x,y) - push x, the rint next character 3000 Assembly Langu s on page 234.	as literal	NO_CTL:	BRA MOVE.W BSR ADDQ.L BRA MOVEM.L	DD,-(A7) PUTC #2,A7 LOOP (A7)+,D0-D6/A0-A	TABLE ADDR DO CASE PUSH CHAR PRINT IT TRASH PARAMETER DO IT AGAIN
Contr pus lext-to- las tad tau tau tau tau tau tau tau tau tau tau	Cursor efault Pr from '68 ing appear	(x,y) - push x, the rint next character 3000 Assembly Langu s on page 234.	as literal	NO_CTL:	BRA MOVE.W BSR ADDQ.L	DO, - (A7) PUTC #2, A7	* DUSH CHAR * PUSH CHAR * PRINT IT * TRASH PARAMETER
Contr pus ext-to- las %d %u %D %X %X %S %c %v %dee Taken tanley.	Cursor efault Pr	(x,y) - push x, the rint next character 3000 Assembly Langu	as literal		BRA MOVE.W BSR	DO, - (A7)	* DO CASE * PUSH CHAR * PRINT IT
Contr pus las las tad tau tab tax tax tax tax tax tax tax tax tax tax	Cursor efault Pr	(x,y) - push x, the rint next character 3000 Assembly Langu	as literal		BRA MOVE.W	D0, - (A7)	* DO CASE * PUSH CHAR
Contrr pus- las las tu tu tu tu tu tu tu tu tu tu	Cursor fault Pr	(x,y) - push x, the rint next character	as literal				TABLE ADDR
Contr pus lext-to- las td tu tb tb tx tx tx tx tx tx tx	Cursor	(x,y) - push x, the		SK1PF:			TABLE ADDR
Contr pus text-to- las tad tu tab tau tab				SK1PF:	MOVE . L	#DISPATCH, AO	* GET CASE
Contr pus lext-to- las td td tau tau tau tau tau tau tau tau tau tau	LITUC DI		ny				
Contr pus ext-to- las das tu	Print he	exadecimal longword			DRA	LPUPF	FORMAT CHAR
Contr pus ext-to- las		signed decimal lon	gword		MOVE.W BRA	D1, FIELD (A6) LPOPF	* SAVE IT. * GET NEXT
Contr pus next-to- las		nsigned decimal wor igned decimal longw			ADD.W	D0, D1	* ADD TO FIEL WIDTH
Contr pus ext-to-		igned decimal word			AND.W	#\$000F,D0	* CONVERT ASCII TO NUMBE
Contr	t, etc.	Push addr of contr	ol string last.		MULU	#10,D1	* SHIFT LEFT ONE DEC DIGI
		rameter in control	string first, then		MOVE.W	FIELD(A6),D1	* GET CURRENT FIELD SIZE
					BGT	SK1PF	* YES, CONTINU PROCESSING
		t/superset of C pr	intf standard I/O		CMP.B	#'9',D0	* IS IT > 9?
TITLE	P	RINTF.ASM			CMP.B BLT	#'0',D0 SK1PF	* IS IT < 0? * YES, CONTINU
				SKOPF:			CHAR
	END				BRA	LPOPF	JUSTIFY FLAG * TRY FOR NEX
	BYTE	0			MOVE.W	#1, LEFTJ (A6)	AGAIN * SET LEFT
		' Exit monitor SOD, SOA, SOA	EX'		CMP.B BNE	#'-',D0 SKOPF	* IS IT MINUS * NO, TRY
	BYTE	\$0D, \$0A					* CLEAR HIGH BYTE
			<transfer addr="">'</transfer>		EXT.W	DO.	

	BRA	PRINTDEC	MOVE POINTER * PRINT VALUE	LPO_PR:			SPACES NEEDED
	PAGE			TO PUT	MOVE.W	DO, - (A7)	SAVE DO ACROSS
* SIGN				-	MOVE.W	#\$2020, - (A7)	1 SPACE TO OUTPUT
					BSR	PUTC	SEND THE SPACE
PRINT	SIGN IF	NEEDED AND TAKE	ABS() OF VALUE.		ADDQ.L MOVE.W	#2,A7 (A7)+,D0	ADJUST STACK RETRIEVE DO
IGN:					SUBQ.W	#1, FIELD (A6)	DECREMENT LOOP INDEX
	TST.L	D0	* IS IT NEGATIVE?	CHKSIGN	BNE	LPO_PR	LOOP IF MORE SPACES
	BPL	SK0_SG	* EXIT IF NOT		TST.W	SIGNF (A6)	SIGN NEEDED?
	MOVE.W	#-1, SIGNF (A6)	* FLAG SIGN NEEDED		BEQ MOVE.L	CHKEXIT DO,D2	JUMP IF NOT
	SUBQ.W	#1, FIELD (A6)	* TAKE AWAY	TO _PUT	MOVE.L		SAVE DO ACROSS
	NDO 1	D0	ONE FOR SIGN	_	MOVE.W BSR	#\$202D, - (A7)	PUSH SIGN
KO SG:	NEG.L	50	* MAKE ABS()		ADDQ.L	PUTC #2,A7	SEND IT ADJUST STACK
	RTS				MOVE . L	D2, D0	RETRIEVE DO
				CHKEXIT	RTS		
PRINT	DEC				PAGE		
COMMO	N DECIMAL	OUTPUT ROUTINE	. VALUE IS IN DO UPON	* POST	FIX		
NTRY.				*			
RINTDEC	::			* PRIN	TS ANY PO	STFIX SPACES.	
	CLR.W	D1 *	OUTPUT DIGIT COUNT	POSTFIX			
LPO_PD:	DIVU	#10.D0 * DI	VIDE NUMBER BY 10		SUB.W	D6, FIELD (A6)	DIGITS ALLOWED - ACTUAL
	BVS	O_FLOW * NU	MBER TOO LARGE		BLE	SK1_PO	EXIT IF NOT
	SWAP MOVE.W		T REMAINDER IN DO.W	LPO PO:			NEEDED
	ADDQ.W	#1,D1 * BU	MP DIGIT COUNT		MOVE.W	#\$2020,-(A7)	SPACE TO SEND
	CLR.W SWAP		T RID OF REMAINDER T QUOTIENT IN DO.W		BSR	PUTC	OUTPUT THE SPACE
	TST.W	D0 * IF	ZERO, ALL DONE		ADDQ.L	#2,A7	ADJUST STACK
	BNE MOVE.W		OP IF NOT DONE ED FOR FIELD ADJUST		SUBQ.W	#1, FIELD (A6)	COUNT THIS SPACE
	BSR	PREFIX * DO	PRFIX SPACES		BNE	LPO_PO	LOOP UNTIL
P2 PD:	SUBQ.W	#1,D1 * ADJ	UST LOOP INDEX CNTR	SK1 PQ:			DONE
e a_co:	ADD.W		KE DIGIT ON TOS ->	SKI_FQ:	RTS		
	BSR		CII ND TO OUTPUT	X ARG:	PAGE		
	ADDQ.L	\$2,A7 * EA	T DIGIT FROM TOS	v_u:	MOVE.W	#3,D1	NUMBER OF
	DBF BSR		OP UNTIL ALL DONE POSTFIX SPACES		MOVE.W	#4.D6	DIGITS TO PRINT USED FOR FIELD
	BRA	LOOP * EX	IT TO CONTROL				ADJUST
FLOW:		PA	RSER		MOVE.W	(A2)+,D2	TRANSFER OUTPUT VALUE
	MOVE.L	#OFLOWSTR, - (A7) * PUSH CONTROL STRING ADDR		SWAP	D2	POSITION OUTPUT VALUE
	BSR	PRINTF\$	* PRINT IT		BRA	PRINTHEX	DO IT
	ADDQ.L BRA	#4,A7 LOOP	* ADJUST STACK * CONTINUE	:			
FLOWSTR	t:		CONTINUE	X1_ARG:			
	DB PAGE	'*overflow*',0		535	MOVE.W	#7,D1	NUMBER OF DIGITS TO PRINT
					MOVE.W	# 8,D6	USED FOR FIELD
PREFI					MOVE.L	(A2)+,D2	ADJUST TRANSFER
OUTPU	JT ANY NEE	DED PREFIX SPACE	ES AND SIGN.		BRA	PRINTHEX	OUTPUT VALUE GO PRINT IT
REFIX:	TST.W	FIELD(A6) CHE	CK IF FIELD NONZERO		PAGE		
	BLE		ERO, SKIP NEXT PART	* PRIN	THEX		
	TST.W	LEFTJ(A6) LEF CHKSIGN	T JUSTIFY SELECTED? BRANCH IF NOT	* 01.22	ume uni	TH D2 TH 1****	D1 IS NUMBER OF
		D6, FIELD (A6)	DIGITS ALLOWED		TS TO PRI		DI IS NOWBER OF
	SUB.W						
	SUB.W BLE	CHKSIGN	- ACTUAL BRANCH IF NO	PRINTHE	٧.		

_

	Journal			lay '89			
		FOR	THOSE WI	HO NI	ED T	9 KNOW	68 MICRO JOURNAL
	BRA	LOOP	NEXT COMMAND	ſ			
	BSR ADDQ.L	PUTC #2,A7	SEND LITERAL CHAR DROP ARGUMENT		END		
ARG:	MOVE.W	(A2)+,-(A7)	GET ARGUMENT				HANDLED HERE
					LONG	DEFAULT	UNKNOWN CASES
	BRA	LPO_SA	CONTINUE		LONG	V_ARG	ADDRESS
	ADDQ.L	#2,A7	ADJUST STACK				X, Y
	BSR	D0, - (A7) PUTC	PUT ON STACK AND SEND IT		DB	0.'v'	NV SET CURSOR TO
	MOVE.W	(A0)+,D0	GET CHAR FROM STRING		DB LONG	0,'c' C_ARG	<pre>%c PRINT CHARACTER ADDRESS</pre>
K1_SA:						S_ARG	ADDRESS
	BSR	POSTFIX	ALL DONE, SEND FINAL SPACES JUMP OUT		LONG	0,'s'	%s PRINT NULL-TERM STRING
	BNE	SK1_SA	NO, KEEP PRINTING		LONG	X1_ARG	ADDRESS
0_SA:	TST.B	(A0)	END OF STRING?		DB	0,'X'	%X PRINT HEXADECIMAL L
	BSR	PREFIX	SEND LEADING SPACES		LONG	X_ARG	ADDRESS
	MOVE.L	(A2)+, A0	GET STRING ADDR AGAIN		DB	0,'x'	%× PRINT HEXADECIMAL W
0_SA:			CHAR		LONG	U1_ARG	ADDRESS
	BRA	SLEN	CHAR LOOK AT NEXT		DB	0,""	%U PRINT UNSIGNED DECIMAL L
	ADDO.W	\$1,D6	FOUND IT COUNT THIS		LONG	D1_ARG	DECIMAL L ADDRESS
	BEO	SKO_SA	TERMINAL NULL BRANCH IF		DB.	0,'D'	%D PRINT SIGNED
LEN:	TST.B	(A0) +	FIELD ADJ		LONG	U_ARG	DECIMAL W ADDRESS
	CLR.W	D6	ADDR FROM STACK GET STRLEN FOR		DB	0,'u'	%u PRINT UNSIGNED
_ARG:	MOVE.L	(A2), A0	GET STRING		LONG	D_ARG	DECIMAL W .
	DB PAGE	'0123456789ABCDEF'			DB	0,'d'	VALID OPTIONS %d PRINT SIGNED
EXDIGI	TS:		CONTROL CHAR	DISPATO	H: DW	9	NUMBER OF
	BRA	LOOP	SPACES DO NEXT	*		TABLE FOR P	RINTF.
	BSR	POSTFIX	DONE ADD TRAILING		PAGE		DO NEXT
	ADDQ.L DBF	#2,A7 D1,LPOPH	DROP PARAMETER LOOP UNTIL		ADDQ.L BRA	#2,A7 LOOP	ADJUST STACK DO NEXT
	BSR	PUTC	SEND IT		BSR	PUTC	
	MOVE.W	0 (A0, D0.W), D0 D0, - (A7)	GET DIGIT FROM TABLE PUT ON STACK	DEFAULT	MOVE W	DO,-(A7)	PRINT CHAR AS
	AND.W	#\$000F,D0	LEAVE ONLY LOW 4 BITS	:			
	MOVE.W	D2, D0	PUT IN WORKING REGISTER		BRA	LOOP	CONTINUE
			FOUR BITS		ADDQ.L	#4,A7	CURSOR DROP BOTH ARG
	ROL.L	#4,D2	TRANSLATE TABLE PUT MSD IN LOW		BSR	CURSOR	POSITON THE

BSR

MOVE.L

LPOPH:

PREFIX

#HEXDIGITS, A0

OUTPUT PREFIX SPACES

ADDRESS OF

V_ARG:

MOVE.L (A2)+,-(A7)

MOVE BOTH ARGS AT ONCE