Date: Jan 31, 1989

Subject: Discovery Monitor ERS

Document Version Number: 00.05

Revision History

Ver	Date	Changes or Additions
Draft !!!	April 5, 88	
00.01	Sept 21, 88	Alpha ROM release
00.02	Oct 31, 88	Add in new vectors for setting system speed and slot arbitration.
00.03	Dec 02 ,88	More descriptions for step/trace. Add in 'Go', display control and registers display option commands for step/trace. Error code for memory mover. Go/Execute/Resume and Control_T commands explanations.
00.04	Dec 21, 88	Enhance the stack management for doing single-step and trace in emulation mode.
00.05	Jan 31, 89	Change the step/trace 'G or g' option commands to 'J or j' option commands.

Introduction

This document only describes the differences between the IIgs and Discovery system Monitor firmware. See //gs firmware reference for details about the monitor.

Differences between IIgs and Discovery Monitor

I. Enahnce the Monitor memory commands so that it can cross bank automatically.
II. Add in Step/Trace debugging functions.
III. Memory Mover.
IV. Change to NEWBELL routine.
V. More routines and vectors.
VI. Go/Execute/Resume and Control_T commands
VII. Miscellaneous addition and changes.

I. Commands for viewing and modifying memory

(a) Display single memory location:
Syntax: {bank/address} Return, No change from IIgs.

(b) Examining consecutive memory locations:
Syntax: {from_address}.{to_address}
The difference from IIgs is that you can specify the next higher bank address. It is not limited to a specific bank but it is limited to the highest bank of $ff.

01/1000.03/ffff will display the memory contents from $1000 of bank $01 to $ffff of bank $03. You can terminate the display with Control-X as in IIgs.

(c) Terminate memory-range display:
Command: Control-X, No change from IIgs.

(d) Modify consecutive memory:
Syntax: {destination}:{val} {val} {"literal ASCII"} {'flip ASCII'} {val}
The main difference from IIgs is that you can continue modify memory at the bank boundary. When you reach the boundary of a bank, it will flip automatically to the next higher bank and continue on.

01/fffe: 11 22 33 44 55 66 Return
It will modify fffe/ffff of bank $01 with 11 22 and 0000/0003 of bank $02 with 33 44 55 66.

(e) Move data in memory:
Syntax: {destination}<{from_address}.{to_address}M
If either the destination or source address crosses the bank boundary, then the firmware will help to move across the bank and continue with the higher bank. It is assumed that the source and destination blocks must not overlap.

Case 1. 02/1000<05/0000.ffff, cross bank occurs in destination memory blocks

Case 2. 02/1000<05/f000.06/ffff, cross bank occurs in source memory blocks

In either case, you don't have to worry about the bank crossing stuffs.

(f) Verify memory contents:
Syntax: {destination}<{from_address}.{to_address}V

Same to IIgs except it can verify across the bank.

(g) Fill memory (zap):
Syntax: {destination}<{from_address}.{to_address}Z
Same to IIgs except it can get across to higher bank.

(h) Pattern Search:
```
Syntax: \{val}\<{from_address}.{to_address}P
        \{'123t'}\<{from_address}.{to_address}P
        \{listeral ASCII }\<{from_address}.{to_address}P
        \{val16 }\<{from_address}.{to_address}P
```
Same to IIgs except it can search across the bank.

II. Single-Step and Trace Modes (Mini-Debugger)

The TRACEVECTOR, $e10074-0077, and STEPVECTOR, $e10078-007b are still availble for external powerful step/trace routines. When invoking step/trace from the monitor, it will call these vectors to check if there are any external routines, if the user's routine sets carry, the monitor step/trace will be started.

You can step through your program one instruction at a time or continuously, with the Monitor single-step and trace functions. Executing your program in this fashion gives you maximum control over the process, allowing you to stop at any point and examine the contents of the registers or your program's display.

In single-step mode, you can step through your program one instruction at a time. As each instruction is executed, the registers, processor status and machine current states are displayed after the instruction.
In trace mode, it will step through each instruction in succession; other than being free-running, trace mode is identical to single-step mode. Use the following commands from the monitor mode to initiate single-step and trace modes:

Command	**Action**
S or s	Enter single-step mode at the current instruction. The current instruction is the **last-opened location** of K/PC register. The K/PC is updated each time an instruction is executed in single-step or trace modes.
*address*S or s	Enter single-step mode at address *address*. The K/PC is set to *address* and the instruction at *address* appears on the screen; press Space bar to execute it, or press Return to enter trace mode.
T or t	Enter trace mode at the current instruction (as indicated by the last-opened K/PC location). It will continue to execute instructions until you press ESC to stop it or until it reaches a BRK instruction.
*address*T or t	Enter trace mode at address *address*. The K/PC is set to *address* and the instruction at address appears on the scren; it will continue to execute instruction until you press ESC to stop or until it reaches a BRK instruction.

Once you are in either single-step or trace mode, you can use any of the following keypress commands.

Command	**Action**
Esc	Terminate trace or single-step mode and return to the monitor *.
Space bar	Single-step one instruction.
Return	Start continuous tracing.
R or r	Trace until the next RTS, RTI, or RTL. This command allows you to trace through one subroutine at a time.
X or x	If the current instruction (the next to be executed) is a JSR or JSL, execute in **real time** until the matching RTS or RTL that returns to the following instruction. If the next instruction is not JSR or JSL, this command is ignored.

J or j	Jump to the code at the current K/PC. Before execution, the screen display is restored is restored depends upon the new display mode byte set by the user. In default, it is set to text page 1 and if real BRK has occurred, then the display mode where BRK occurred will be stored to this mode byte and used to restore if 'J' or 'j' is executed.
O or o	Toggle the registers display on and off following execution of each instruction.
	Skip to the next instruction - do not execute current instruction.
<--	Change to the slow trace rate.
-->	Change to the fast trace rate.
T or t	Change the display to text mode.
M or m	Change the display to mixed text and graphics mode.
L or l	Change the display to low-resolution graphics mode.
H or h	Change the display to high-resolution graphics mode.
D or d	Change the display to double high/low-resolution mode.
S or s	Change the display to super higi-resolution graphic mode.

Executing code

Code can be executed in trace and single-step modes. In both modes, the code is displayed in the disassembly as it is being executed. The current instruction (the one that is about to be executed) is displayed first and when executed, the results may be shown in the next line depending upon the option command, O or o, and following the display of the next instruction to be executed.

If, while in trace or singl-step mode, a BRK is met then it will stay at the BRK and make a beep and force to display the registers contents. At this point, one can either skip this BRK with down arrow key or ESC to terminate the trace or single-step.

Both tracing and single-stepping recognize two PRODOS entry points. These are PRODOS 8 Machine Language Interface (MLI) entry point, $00bf00 and the PRODOS 16 entry point at $e100a8. Whenever these calls are met, the trace and single-step will adjust the user's program counter and point to the next opcode following the the PRODOS call and continue on. It doesn't whether these PRODOS calls are executed in real time (command 'x')or not but it is suggested to execute in real time since most of these calls are related to device input/output.

While tracing or single-step and it hits the end of the screen, code will continuously scroll up.

CAUTION: (1) Stack Usage
The mini-debugger uses the page 1 stack space and adjusts the user's stack space to $017f if the user stack space is in the range of $0180 to $01ff. If the user stack pointer is below $0110 then the single-step or trace mode would'nt be invoked and quit back to the monitor. The user can set the stack pointer with the monitor command, {val}=S.

If the users program run in native mode and having the stack out of the regular page 1 stack space, then there is no adjustment to the user stack space.

If, while in trace or single-step mode, a program causes the stack pointer to be 16 bytes to the end (or say $010f-$0100) of the regular stack (page 1 stack area), execution of the code will be immediately halted and the mini-debugger returns back to the monitor. To continue operation, the user must change the value of the stack pointer so that it is outside of the mini-debugger stack range

If, while in trace or single-step and real time code execution ('X'or 'x' for JSx routine, 'J' or 'j' for jumping to the users' k/pc) and a deliberate 'BRK' is executed. It breaks to the monitor and would like to continue on the trace or single-step, if the program is using the page 1 stack space and the stack pointer falls within the ranges described above, then the stack point is not real because it is being adjusted.

The above limitations only apply to those users who stack space is also in the regular stack space of the monitor.

(2) Absolute zero page
Since the monitor trace and single-step or mini-debugger uses absolute zero page extensively, therefore the programs that also use the absolute zero page can hardly use the mini-debugger to do the debugging, especially those locations defined for monitor to be used.

Example:
Get to the monitor and say there is a small program resided at 00/300 shown below

```
00/300: CLC           ;Clear carry
00/301: XCE           ;Set to native mode
00/302: REP #$30      ;16-bit m/x
00/304: LDA #$3344     ;load 'a' with $3344
00/307: PHA           ;save 'a' to stack
00/308: PLY
   .      .
   .      .
   .      .
   .      .
```

Then with single-stepping, the screen will display as follow:

```
*00/300s
Step
00/300: 18          CLC
A=0000 X=0000 Y=0000 S=0173 D=0000 P=30 B=00 K=00 M=0C Q=9E L=1 m=1 x=1 e=1
00/301: FB          XCE
A=0000 X=0000 Y=0000 S=0173 D=0000 P=31 B=00 K=00 M=0C Q=9E L=1 m=1 x=1 e=0
00/302: C2 30       REP #30
A=0000 X=0000 Y=0000 S=0173 D=0000 P=01B=00 K=00 M=0C Q=9E L=1 m=0 x=0 e=0
00/304: A9 44 33    LDA #3344
A=3344 X=0000 Y=0000 S=0173 D=0000 P=01 B=00 K=00 M=0C Q=9E L=1 m=0 x=0 e=0
00/307: 48          PHA
A=0000 X=0000 Y=0000 S=0171 D=0000 P=01 B=00 K=00 M=0C Q=9E L=1 m=0 x=0 e=0
00/308: 7A          PLY
A=0000 X=0000 Y=3344 S=0173 D=0000 P=01 B=00 K=00 M=0C Q=9E L=1 m=0 x=0 e=0
```

III. Memory Mover with vector at $E10200-0203

Memory Mover moves a block of data from a source location to a destination location. The following sequences need to be set up first for using memory mover:

On entry:
1. Place machine in full native mode (e=0, m=0, x=0)
2. Push high order word of source pointer onto stack
3. Push low order word of source pointer
4. Push high order word of destination pointer
5. Push low order word of destination pointer
6. Push high order word of transfer count
7. Push low order word of transfer count
8. Push command byte, explain below
9. Call through memory mover vector, jsl $e10200

Memory Mover command byte:

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

bits 15/14	=	reserved		
bits 13/12/11	=	move mode		
		000	=>	reserved
		001	=>	block move
		010 - 111	=>	reserved
bits 10/9/8/7	=	reserved		
bits 6/5/4	=	reserved		
bits 3/2	=	destination incrementer		
		00 (+0)	=>	constant destination
		01 (+1)	=>	increment destination by 1
		10 (-1)	=>	decrement destination by 1
		11	=>	reserved
bits 1/0	=	source incrementer		
		00 (+0)	=>	constant source
		01 (+1)	=>	increment source by 1
		10 (-1)	=>	decrement source by 1
		11	=>	reserved

On exit:
If no error then a=$0000 and c=0

If error occurred then
a=error code, $ffff, parameters range error
c=1
Data bank and direct register are preserved while x.y are scrambled.

IV. Change to NEWBELL

If sound volume is set to zero, then the border color would flash instead of bell.

V. More New Vectors

1. $E10204-0207, TOSETSPEED
   This vector points to a routine that sets the system speed in preparation for calling a speed dependent device driver. Speed settings are in the least significant two bits of the 'A' defined as follows:

'A' =	xxxxxx00	1.0 mhz
	xxxxxx01	2.6 mhz
	xxxxxx10	>2.6 mhz
	xxxxxx11	not speed dependent (no change)

   Entry: via a 'JSL' and must be in 16 bit m/x native mode.
   Exit: 'A' = speed prior to requested change, no change to processor status, 'p'.

2. $E10208-020B, SLTARBITER
   This vector points to a routine that checks that a requested slot selection is valid. If so, the requested slot is selected and the appropriate screen holes are copied in prior to passing control back to the calling routine.

   Entry:
   via a 'JSL' and must be in 16 bit m/x native mode.
   'A' = requested slot

   Exit:
   If no error then a=$0000 and c=0, slot was granted
   If error then a=error code, $0010, slot not found and c=1

   Note: No dynamic slot selection at this time but could be in future.

VI. Go/Execute/Resume and Control_T commands

The monitor G/X/R commands are fully working no matter interrupt is on or off.
During a G/X command and a deliberate 'BRK' is hit, it can be resumed with the resume command, R.
During the 'BRK', the screen display I/O soft switches are stored and force to text page 1 display mode. When a resume is made, the screen display I/O switches are restored to the modes where 'BRK' occurred.
The Control_T command in original //gs is modified to flip between the text page 1 and the display where 'BRK' occurred. In //gs, it always force to text page 1 mode.

VII. Miscellaneous addition and changes.

1. Boot code and initialization, init vectors, set up AppleTalk

2. Mouse Firmware, include $c400 and bank $ff mouse firmware

3. Alternate Text Page2 shadowing, hardware for Discovery and software for //gs.

4. Interrupt handler changes for hardware text page2 shadowing bit, Midi interrupt polling

5. More batteryram parameters checking

6. Bugs fix: return of 6502 instruction length ($f88e), clear screen and display AppleIIgs logo ($fb60).

7. A new option flag, d, for disassembly has been added, when d=1 then the disassembly will not change the mode even though there is a mode change (m/x) with the instruction, REP/SEP. When d=0, then the disassembly will change mode in accordance with the new mode being set by the REP/SEP in the program.

8. Change Video Display mode, V
   Changes video display mode I/O soft switches to {val} for Resume/Step or Trace 'J'/'j' commands.
   If any 'BRK' occurred, then the video display mode or screen display mode I/O soft switches is saved to this value and therefore it provides a good way to resume to the previous mode. Normally, it is not necessary for the users to set this value if there is a deliberate BRK in the program for debugging. But in some cases, a BRK has been occurred before but the pre-stored screen display mode is not the one going to be used for the current RESUME/Step or Trace 'J'/'j' commnads.

   During power up or reset, it sets to default as text page 1 mode which is $40.

   {val}=V

   where val = bit7 - bit0 and defined as follows:
   bit7: 1/0 = super high resolution on/off
   bit6: 1/0 = text/graphic
   bit5: 1/0 = mixed mode on/off
   bit4: 1/0 = high resolution/low resolution
   bit3: 1/0 = text page 2/text page 0
   bit2: reserved, fill with 0
   bit1: reserved, fill with 0
   bit0: reserved, fill with 0

   Examples: setting text page 1, then $40=V
   setting super hi-res then $80=V

9. 'BRK' execution: When a BRK is executed, it not only services this software BRK interrupt but also preserves the screen display i/o soft switches states (but not the screen image itself) and also the user's stack pointer (display on the screen at s=xxxx) before the BRK occurred. Then, it forces to text page 1 and displays the registers and machine states. In doing this, the user will always get to the text mode when BRK occurred and if a RESUME command, R, is executed, it will also switch back the user's screen display mode.
   Also, the display of the stack pointer on the screen is the user's actual stack pointer before the BRK occurred.
   After the BRK, the user can also continue debugging the program with the mini-debugger

single-step and trace. If a 'J' or 'j' is invoked during single-step or trace, then it will also resume the screen display mode.
The screen display mode can also be set deliberately by the monitor new command, val=V where V stands for video mode and must be uppercase.