the digital group
po box 6528 denver, colorado 80206 (303) 777-7133

STEPPER 1.0

296-085

S T E P P E R 1 . 0

STEPPER is a software machine code debugger which serves much the same purpose as a hardware front panel, but in a more comprehensive manner. At all times, the screen displays all CPU registers and flags, the top twelve bytes of the stack, and a 64 byte window of memory centered on the location indicated by the Program Counter register. In STEPPER, options exist for
*Examining or modifying any register, flag, or memory byte
*Block moving any sized block of memory to any location
*Block filling any sized block of memory with any single byte
*Single-step executing any instruction
*Setting and removing a breakpoint
*Executing code from any point
*Setting input/output mode to ASCII, octal, or hex
*Relocating STEPPER anywhere in memory

STEPPER commands are all initiated by typing a single character. If additional information is required, the program will ask for it. Commands are broken down into four basic groups: Cursor Movement, Input/Output, Macro, and Executive. Note: In this documentation, the prefix 'ctl' means that the following letter is to be entered while holding down the Control key.

C U R S O R M O V E M E N T C O M M A N D S :

H — Set the high order byte of the Program Counter and move the cursor to the resulting Program Counter location.

Example:
```
>H
Address?
>06
```

Will set the high order byte of the Program Counter equal to 06. The low order byte will not be affected.

L — Set the low order byte of the Program Counter and move the cursor to the resulting Program Counter location.

Example:
```
>L
Address?
>4E
```

Will set the low order byte of the Program Counter equal to 4E. The high order byte will not be affected.

J — Set both high and low order bytes of the Program Counter and move the cursor to the resulting location.

Example:

```
>J
 Address?
>064E
```

Will set the Program Counter equal to 064E.

ctl L — Move the cursor right one byte and increment the Program Counter if the cursor points to the memory window; otherwise, if the cursor points to the registers or flags, move the cursor right to the next register or flag.

Example:

```
>ctl L
```

Will move the cursor right one byte.

ctl H — Move the cursor left one byte and increment the Program Counter if the cursor points to the memory window; otherwise, if the cursor points to the registers or flags, move the cursor left to the next register or flag.

Example:

```
>ctl H
```

Will move the cursor left one byte.

ctl J — Move the cursor down one line (8 bytes) and change the Program Counter accordingly if the cursor points to the window of memory; otherwise, if the cursor points to the registers or flags, this command is ignored.

Example:

```
>ctl J
```

Will move the cursor down one line.

ctl K — Move the cursor up one line (8 bytes) and change the Program Counter accordingly if the cursor points to the window of memory; otherwise, if the cursor points to the registers or flags, this command is ignored.

Example:

```
>ctl K
```

Will move the cursor up one line.

ctl D — Tab the cursor down four lines (32 bytes) and

change the Program Counter accordingly if the cursor points to the window of memory; otherwise, if the cursor points to the registers or flags, this command is ignored.

Example:
```
>ctl D
```

Will tab the cursor down four lines.

ctl U — Tab the cursor up four lines (32 bytes) and change the Program Counter accordingly if the cursor points to the window of memory; otherwise, if the cursor points to the registers or flags, this command is ignored.

Example:
```
>ctl U
```

Will tab the cursor up four lines.

R — Move the cursor to the bank of registers and flags.

Example:
```
>R
```
Will cause the cursor to point to the Accumulator.

P — Move the cursor to the memory window.

Example:
```
>P
```

Will cause the cursor to point to the byte indicated by the Program Counter.

I N P U T / O U T P U T C O M M A N D S :

Del (or Rub) — Shift input/output mode to ASCII, octal, or hex.

Example:
```
>Del
```

Will switch to ASCII if the current mode is hex, octal if the current mode is ASCII, or hex if the current mode is octal. Note: when in the ASCII mode, all commands except Del are disabled, so any character except Del will be entered directly into the memory location or register indicated by the cursor. Characters will be entered with parity (bit 7) set high.

ctl W Write data to audio cassette

Example:

```
>ctl W
 Writing
```

Will write an audio cassette copy of the memory pointed to by addresses in the Write routine. The high order byte of the following addresses (denoted nn) is the first page of STEPPER (normally page 01.)

(nn 24) low order byte of the starting address
(nn 27) high order byte of the starting address
(nn 2A) low order byte of the ending address
(nn 2D) high order byte of the ending address

ctl R Read in an audio cassette program

Example:

```
>ctl R
```

Will read in an audio cassette program starting at the location pointed to by the contents of (0118) and ending at the location pointed to by the contents of (011A). Note: this command will not function if your system is not Audio Cassette based.

M A C R O C O M M A N D S :

M Block move memory.

Example:

```
>M
 Old start?
>1610
 Old end?
>161F
 New start?
>1700
```

Will move the block starting with (1610) and ending with (161F) to (1700). Blocks may be moved up or down in memory, and may overlap without fear of data propagation.

ctl F Block memory fill.

Example:

```
>ctl F
 Data?
>00
 Start?
>1540
```

```
 End?
>16FF
```

Will change all bytes from (1540) through (16FF) inclusive to 00's.

ctl N — Relocate STEPPER code.

Example:

```
>ctl N
 New starting page?
>3C
```

Will move STEPPER so it starts at (3C00) and ends at (44FF). Note: when STEPPER is moved, it may be moved up or down in memory, but it must not overlap itself, so it must be moved a minimum of 9 pages.

Example:

```
>ctl N
 New starting page?
>0A
>ctl N
 New starting page?
>04
```

This is a sequence that could be used to move STEPPER from page 01 to page 04.

E X E C U T I V E C O M M A N D S :

escape — Vector to location 0000.

Example:

```
>esc
```

Will jump to 0000.

G — Execute code starting with the byte pointed to by the program counter.

Example:

```
>G
```

Will restore all registers and flags to the values specified on the screen and execute the code starting with the byte pointed to by the program counter.

ctl B — Sets the breakpoint at the location pointed to by the program counter.

Example:
>ctl B

Will set the breakpoint at the location pointed to by the program counter and remove any existing breakpoint. When the breakpoint is encountered by the CPU, control will be vectored to STEPPER, which will then display the status of the registers, flags, and stack at the time execution was halted.

ctl E — Remove any existing breakpoint.

Example:
>ctl E

Will remove the breakpoint if it exists. Note: ctl B and ctl E are most useful when used in conjunction with the G command.

S — Single-step execute one machine code instruction.

Example:
>S

Will execute the instruction pointed to by the program counter and update the display accordingly.

N — Single-step execute the instruction which has been replaced by the breakpoint.

Example:
>N

Will execute the instruction which has been replaced by the breakpoint. This command is valid only if the program counter points to the byte after the breakpoint, as is the case when the CPU encounters a breakpoint and vectors to STEPPER. This command is useful when used in conjunction with the ctl B, ctl E, and G commands.

D A T A E N T R Y :

Any character which is not recognized as a valid STEPPER command will simply be entered as data into the location indicated by the cursor, whether it is register, flag, or memory.

Example:
>2F

Would enter 2F into the location indicated by the cursor. To

alter data on the stack, the J command should be used to set the program counter equal to the stack area. Data on the stack can then be modified by changing memory in the method described above.

D E M O N S T R A T I O N P R O G R A M S :

*Example. Clear the screen and write an 'A'. First, the program counter must be set to 0A00. To do this, type the following:

```
>J
 Address?
>0A00
```

Then the program may be entered by typing in the following data (do not type the addresses!):

ADDRESS	DATA	EXPLANATION
0A00	CD	Call the subroutine 'HM ERS'
0A01	E6	
0A02	00	
0A03	3E	Load the accumulator with an 'A'
0A04	RUB	Change the mode to ASCII
	A	Enter the 'A'
	RUB	Switch the mode to octal...
	RUB	and back to hex.
0A05	CD	Call the subroutine 'TV' to display the character
0A06	FA	
0A07	00	
0A08	CD	Call the subroutine 'KEYBRD'; this will cause
0A09	A8	the 'A' to remain on the screen until any
0A0A	01	character is typed.
0A0B	ctl B	Set a breakpoint at the end of the program so we end up in STEPPER when the program is done.

Now, to execute the program, we must start at the beginning, so type the following sequence of instructions to do so:

```
>L
 Address?
>00
```

This will reposition the program counter at the first byte of the program. Now, to execute it, type 'G' (for GO!). The screen should now be blank except for an 'A' in the upper left corner. To get back to STEPPER, type any character. The A, B, and C registers should all contain zeros. Now, on to bigger and better things. Try the following:

*Modify the above program to print an 'a'.

*Restore the program counter to 0A00 and single-step execute some of the program by using the 'S' command to see how it works.

*Print your name.

*Print your name in the middle of the screen using 'TV EDT'.

*Print your name in the middle of the screen, flashing on and off. (Hint - Use two 'DELAY' subroutine calls and an unconditional loop to program beginning loop).

*Print only the 128 possible characters and stop, using less than 20 bytes (Hint - Load Accumulator, Save, Display, Restore accumulator, modify accumulator, and loop if not end).

Score: Over 100 bytes - HA!
Over 30 bytes - Fair
20-25 bytes - Good
15-19 bytes - Giant
(Can be done in seven bytes if the characters are displayed in descending order)

S P E C I A L C O N S I D E R A T I O N S :

*The top item of the user's stack is used by STEPPER; thus, the user's stack data starts with the second item displayed.

*The user stack is displayed with the two bytes of each item reversed; this is to compensate for the fact that the CPU handles all sixteen bit numbers with the two bytes reversed.

*The stack pointer displayed must not point anywhere within STEPPER's code. If it does, executing a G, S, N, or ESC command will destroy STEPPER. When STEPPER is brought up on an Audio Cassette based system, the stack pointer invariably points within STEPPER, so before anything else is done, the following sequence of commands should be entered:

```
>R
>ctl H
>ctl H
>ctl H
>ctl H
>ctl H
>ctl H
>0A
>00
```

Note: This is only needed if STEPPER starts at page 01 and is booted up on an Audio Cassette based system!

*STEPPER is, for the time being, EROM dependant for CRT output routines (and audio cassette input routines, although this is not really a restriction).

S T E P P E R R O U T I N E S / S U B R O U T I N E S :

The following routines are located at the addresses given only if STEPPER starts at page 01.

ADDRESS	NAME	AFFECTED REGS	OBJECT REGS	COMMENTS
011C	MODE			Data byte; 0 = ASCII, 1 = octal, & 2 = hex
0120	WRITE	ABCDEHL		Audio cassette write routine
0177	DELAY	ABCDE		Wastes 0.1 x A seconds.
0193	EDITOR	ABCHL	M	Same as old TV EDT but uses skips instead of spaces
01A8	KEYBRD	A	A	Inputs one ASCII chr from keyboard
01B9	HMERAS	ABC		Home-erase CRT
01BE	MLTSKP	ABC	B	Outputs B skips to CRT
01C2	MLTSPC	ABC	B	Outputs B spaces to CRT
01CB	REGSTO			Register storage area (see table)
01EA	STACK			STEPPER stack area (01EA-01FF)
0200	CHROUT	AB'C'D'E'H'L'	A	Output A in ASCII/ octal/hex form
022B	CHOUT2	AB'C'D'E'H'L'	A	Output A in hex/ octal/hex form
025D	2CHRIN	ABC	BC	Input 2 hex/octal numbers in BC
0261	CHRINP	ABC	A	Input a hex/octal number in A
0329	START	all		STEPPER entry point

B R E A K D O W N O F R E G I S T E R S T O R A G E A R E A :

ADDRESS	REGISTER(S)
01CB	AF
01CD	BC
01CF	DE
01D1	HL
01D3	AF'
01D5	BC'
01D7	DE'
01D9	HL'
01DB	X index
01DD	Y index
01DF	Stack pointer
01E1	Program counter
01E3	R
01E4	I