NASA-TM-86289 #

NASA Technical Memorandum 86289

NASA-TM-86289 19840026128

SOFTWARE IMPLEMENTED FAULT-TOLERANT (SIFT) USER'S GUIDE

David F. Green, Jr.
Daniel L. Palumbo
Daniel W. Baltrus

AUGUST 1984

NASA
National Aeronautics and
Space Administration
Langley Research Center
Hampton, Virginia 23665

CONTENTS

N84-34199#

SIFT USER'S GUIDE

INTRODUCTION

Research in a Software Implemented Fault-Tolerant (SIFT) computer system is conducted at the NASA Langley Research Center's AIRLAB facility. Program development for this system is accomplished using a DEC VAX-11 to interface with eight Bendix BDX 930 flight control processors. The interface software which provides this SIFT program development capability was written by AIRLAB personnel. This document describes the application and design of this software in detail, and is intended to assist both the user in performance of SIFT research and the systems programmer responsible for maintaining and/or upgrading the SIFT programming environment.

The main section of this guide describes the commands (hereafter referred to as SIFT commands) which are used to load SIFT programs and data to the BDX 930s, read and write BDX memory and registers, set and read the program counter, run and halt processors, debug programs, and other tasks needed in the development of SIFT systems. The command SIFT, which starts the SIFT session, is listed first. All other commands are listed in alphabetical order for easy reference.

Appendix A describes the SIFT Schedule Generating Utility. Coding all permutations of the SIFT schedule can be tedious and error prone. The schedule generator allows the user to define his schedule with simple command syntax.

Appendix B describes the command syntax required to operate the BDX 930 Relocatable Assembler, Symbolic Assembler, and Linkage Editor. Also included is a description of programs which modify the output files produced by the assembler/linker programs.

Appendix C, Systems Programming for the SIFT Environment, describes the installation and programming concepts for the SIFT functions in detail. It is applicable only to systems programmers responsible for maintaining and/or upgrading the functions outlined in this document.

The user should be advised that, because SIFT is a dynamically evolving system, modifications to existing functions/commands or new procedures can be expected and will not be immediately reflected in this guide. Amendments or revisions will be published periodically as required to bring this publication up to date. Current information can be obtained through the VAX/VMS help facility which will be kept updated to reflect the latest changes.

SIFT COMMANDS

SIFT

Format: SIFT (no parameters)

Starts the SIFT session and SIFT display of BDX 930 processor status. The prompt SIFT$ will replace the $ prompt indicating that all SIFT commands can be used. Note, however, that all VAX DCL commands can also be entered at the SIFT$ prompt.

SIFT display: (see figure 1 for example display)

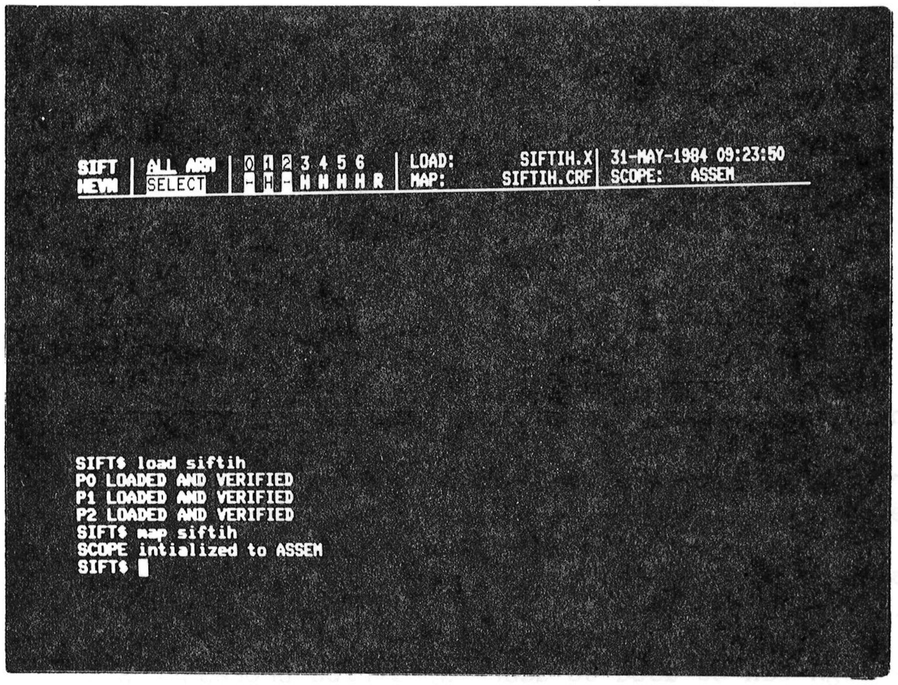
A display will appear at the top of the screen showing the current processors which are not allocated to other users. Processor numbers already allocated to other users will not appear in the display. Information is also displayed for LOAD:, MAP:, and SCOPE which is explained in the descriptions of LOAD and MAP commands in this section.

Processor status (ARMED, ALLOCATED, SELECTED, HALTED, and RUNNING) is indicated by various combinations of character attributes. For example, processors allocated are shown with the processor number underlined. Processors selected are shown with the processor number in reverse video. Halted processors are shown with the letter H under the processor number and conversely, running processors are shown with the letter R. Armed processors are indicated by bold (or brighter) letters H or R, as applicable. If the letter H is shaded it means that the user has executed a HLT (halt) command. If the letter H is not shaded it means the processor has halted for other reasons (e.g., a halt instruction).

ADDRESS - Find Hex Address

Format: ADD*RESS $[scope.]label;..

Translates the labels used in the user's program to their associated hex addresses. Labels must be mapped using the MAP command before the ADDRESS command can be used. See MAP command for the definition of scope and label.



```
SIFT | ALL ARM | 0 1 2 3 4 5 6   | LOAD:       SIFTIH.X | 31-MAY-1984 09:23:50
HEVN | SELECT  | - H - H H H H R | MAP:      SIFTIH.CRF | SCOPE:   ASSEM
```

```
SIFT$ load siftih
P0 LOADED AND VERIFIED
P1 LOADED AND VERIFIED
P2 LOADED AND VERIFIED
SIFT$ map siftih
SCOPE intialized to ASSEM
SIFT$ █
```

Figure 1. SIFT display

ALP - Allocate Processor(s)

```
Format:  ALP  Pn .. Pm
         ALP NONE
```

Allocates BDX930 processor(s) Pn through Pm, where n/m = 0..6. The second format deallocates all processors currently allocated. (Note: no operations can be done on a processor unless it is first allocated.)

Processors are required to be entered on the command line and can be separated by spaces, commas, or nothing.

```
  Examples:   ALP P0 P1 P5     ALP P0,P1,P5     ALP P0P1P5
```

A minus sign can be used to deallocate specified processor(s):

```
  Example:   ALP P0 -P1 P5   (allocates P0,P5 and deallocates P1)
```

The ALP command automatically does a SELect (see SEL) for each processor allocated. The allocated and selected processors will be shown in the SIFT display at the top of the screen with processor numbers underlined (allocated) and reverse video (selected).

ARM - Arm Processor(s)

```
Format:  ARM [ Pn .. Pm ]
         ARM NONE
```

Arms the interrupt of BDX930 processor(s) Pn through Pm, where m/n = 0..6. The second format dearms all allocated processors which were currently armed.

Processors can be separated by spaces, commas, or nothing.

```
  Examples:  ARM P3 P5     ARM P3,P5     ARM P3P5
```

If processors are not specified, the default is SELected set (see SEL command).

The minus sign can be used to dearm specified processor(s).

```
  Example:  ARM -p3, p4    (dearms P3 and arms P4)
```

The SIFT display will show the armed processors with bold (brighter) letters or brighter reverse video for the letters H,R which appear just below the processor numbers.

BRF - Breakpoint Fast Debugging

```
Format:  BRF [ hexaddress ]
         BRF [ $[scope.]label ]
```

Sets a breakpoint at the hexaddress or symbolic location (scope and/or label) for debugging programs on the BDX930 processor(s).

To use the second format, labels must first be mapped using the MAP command. For a definition of scope and label, see MAP command.

```
Examples:  BRF 05FF    BRF $VOTE    BRF $SIFTO.VOTE
```

If hexaddress or label or scope.label are omitted, you will be prompted for the information.

By default, the breakpoint is set on SELected processors. You cannot designate specific processors in the command line but you can specify one or more processors for debugging purposes by using the SEL command.

When the BRF command completes, the program automatically goes to the single step mode of operation (see SST command) for debugging purposes. The BDX program counter will point to the breakpoint address given in the BRF command unless the program halts for some other reason before it reaches the breakpoint, in which case a message will indicate the actual stopping point. Once in the single step mode (if BRF completes successfully) the current program counter will be displayed along with the machine code instruction in hex and disassembly code for the instruction (see figure 5).

CMP - Compute

```
Format:  CMP [number or expression]
         CMP  <RETURN>   (prompts for input)
```

Converts a number or the result of an arithmetic expression to its equivalent in decimal, hexadecimal, octal and binary. Accepts integers or floating point decimal numbers, but only the integer portion of the floating point number is converted. If entries are typed on the command line, the program exits after the computation. If there is no command line, the program prompts for the information and prompts are repeated after each computation.

Radix syntax: %D (decimal), %X (hex), %O (octal), %B (binary).
Initial default radix is decimal. Radix can be changed by using %.

```
  Example:  456 + %O 747 + 640 + %D123
            dec     oct     oct    dec
```

Arithmetic expressions can have parentheses and the binary operators + - * / ^ for add, subtract, multiply, divide, and exponent. The unary operators + - must be separated from binary operators by parentheses or radix operators.
Example: -243 - (-465) + %o -777

In the repeat mode (no initial command line) the result of the last computation is stored in the variable P which can be used in the next computation.
  Example: first computation: 2+2; 2nd computation: P+2 (result 6)

DISPLAY - Display Memory or Register Contents

```
                                       / $[scope.]label,.. \
 Format:  DISP*LAY/qualifiers         <  hex,..              >  [ON Pn..Pm]
                                       \ Rn,..Rm           /
          DISP*LAY REDRAW
```

Displays labels, hexaddresses, or registers and their contents from specified processors. If processors are not specified, default is SELected processors. Labels and hexaddresses can be on the same command line but registers must be specified on a separate command line. The screen can display up to six groups (or boxes) which show the processor number, current program counter, memory location (either hex or label as requested in the DISPLAY format), register number and the memory or register contents. Memory locations and registers are currently displayed in different groups. The groups are numbered from 1 to 6. See figures 2a and 2b for example displays.

Labels can be augmented. For example:

```
  DISP $L1,$L2+4          (will display contents of L1 and L2+4)
  DISP $L1,$L2,+,+,+      (will display contents of L1,L2,L2+1,L2+2,L2+3)
```

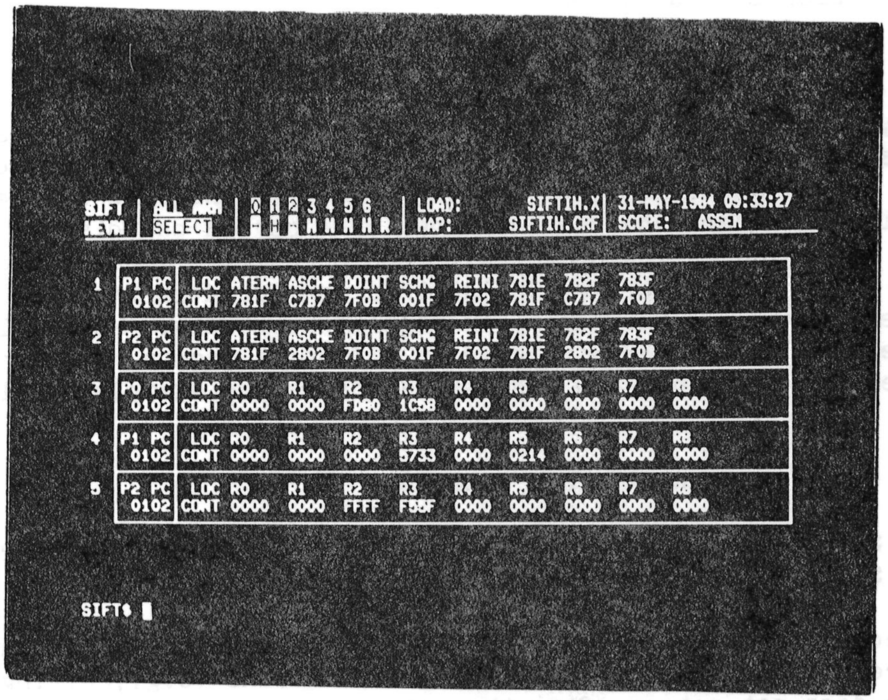
The special format, DISP*LAY REDRAW, can be used to redraw the display if the screen has been cleared for other operations (e.g.,editing) and the display does not return after the operation.

The DISP*LAY command without parameters will update the group(s) that are already on the screen. That is, the register or memory locations currently displayed will be updated.

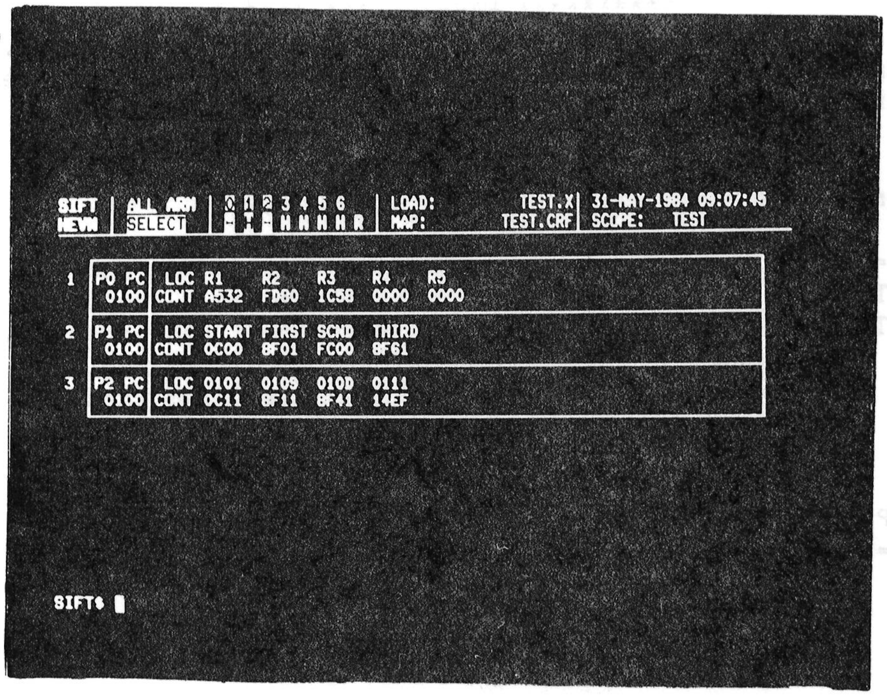
********** WARNING **********
Use of DISPLAY may disrupt operation of SIFT processors.
Unknown amounts of skew will be injected into each processor during a read. Current active task might timeout.
****

Qualifiers:

/SCOPE=s	Defines 's' as new scope.
/DEL*ETE=n,..m	Deletes groups n through m.
/DEL*ETE hex,..	Deletes specified addresses
Rn,..,Rm	Deletes specified registers
$[scope.]label,..	Deletes specified labels
/OFF	Disables display. Erases screen.
/ON	Enables display. Redraws screen.
/KEEP	Display is permanent.
/NOKEEP	Display is scrolled.
/GROUP=n	Uses group n as display.



```
SIFT | ALL ARM | 0 1 2 3 4 5 6 | LOAD:      SIFTIH.X| 31-MAY-1984 09:33:27
NEVN | SELECT  | - H - H H H H R | MAP:     SIFTIH.CRF| SCOPE:   ASSEN

  1 |P1 PC|  LOC ATERM ASCHE DOINT SCHG  REINI 781E  782F  783F
    | 0102|CONT 781F  C7B7  7F0B  001F  7F02  781F  C7B7  7F0B

  2 |P2 PC|  LOC ATERM ASCHE DOINT SCHG  REINI 781E  782F  783F
    | 0102|CONT 781F  2802  7F0B  001F  7F02  781F  2802  7F0B

  3 |P0 PC|  LOC R0    R1    R2    R3    R4    R5    R6    R7    R8
    | 0102|CONT 0000  0000  FD80  1C58  0000  0000  0000  0000  0000

  4 |P1 PC|  LOC R0    R1    R2    R3    R4    R5    R6    R7    R8
    | 0102|CONT 0000  0000  0000  5733  0000  0214  0000  0000  0000

  5 |P2 PC|  LOC R0    R1    R2    R3    R4    R5    R6    R7    R8
    | 0102|CONT 0000  0000  FFFF  F55F  0000  0000  0000  0000  0000

SIFT$ █
```



```
SIFT | ALL ARM | 0 1 2 3 4 5 6 | LOAD:        TEST.X| 31-MAY-1984 09:07:45
NEVN | SELECT  | - I - H H H H R | MAP:       TEST.CRF| SCOPE:   TEST

  1 |P0 PC|  LOC R1    R2    R3    R4    R5
    | 0100|CONT A532  FD80  1C58  0000  0000

  2 |P1 PC|  LOC START FIRST SCND  THIRD
    | 0100|CONT 0C00  8F01  FC00  8F61

  3 |P2 PC|  LOC 0101  0109  010D  0111
    | 0100|CONT 0C11  8F11  8F41  14EF

SIFT$ █
```

Figure 2. Displays of BDX 930 memory and register contents using DISPLAY command.

DMP - Dump Processor Memory

```
Format: DMP [hexaddress#numlocations] [FROM Pn,..,Pm]
        DMP [hexaddress#numlocations] [FROM Pn,..,Pm] TO filename[/SAVE]
```

The first format dumps contents of specified BDX memory locations to the screen. The second format dumps to a text file for printing or, if /SAVE option is used, to a binary file which can be reloaded using the LOAD command.

If hexaddress#numlocations is omitted from the command line, the DMP program prompts for the information. If FROM Pn,..,Pm is omitted the default is SELected set. If TO filename is omitted, a CRT dump is assumed. If /SAVE is omitted after the filename, a text file dump is assumed.

The # character is a required separator between hexaddress and numlocations if information is entered in the command line. Numlocations is entered in decimal.

The hexaddress and numlocations together is called an access group. When the first request has completed, the program will automatically prompt for additional access groups and/or processors to be dumped. Any number of access groups can be dumped and processors can be changed for each access group if desired. However, the /SAVE option uses a single processor and does not permit changing processors for each access group. If different processors are needed for the /SAVE option, the DMP command has to be reentered for each one. A null response (pressing <RETURN>) for all prompts will terminate the DMP command.

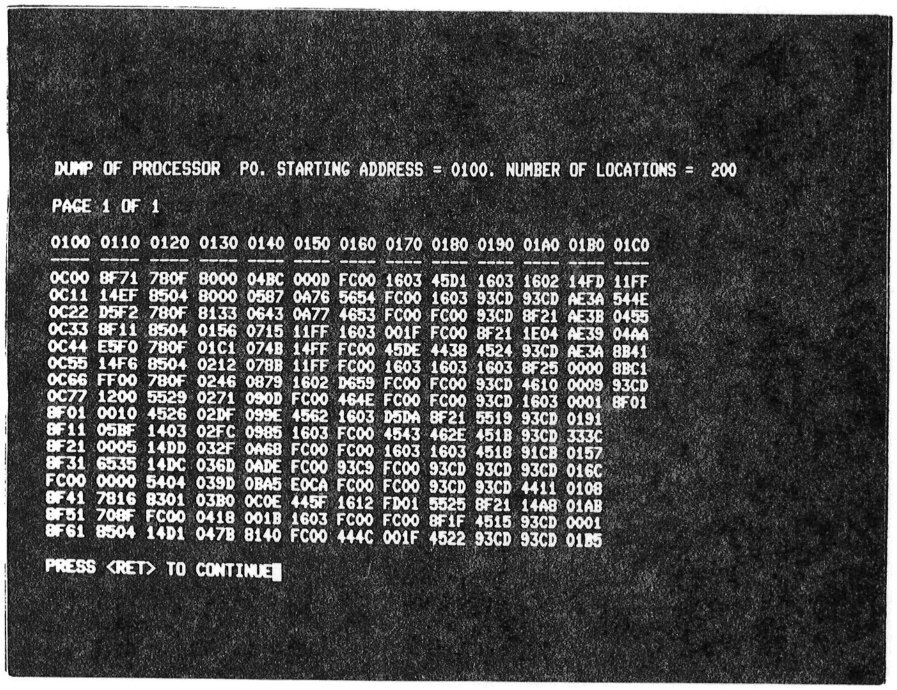
```
Examples: DMP 050A#100 FROM P0,P1
          DMP 050A#100 FROM P0 P1 TO PRTFILE.DAT
          DMP 0500#100 TO SVFILE.X/SAVE
```

Examples of formats for screen dump and text file dump are given in figures 3 and 4.

Format for screen dump:

```
DUMP OF PROCESSOR P1. STARTING ADDRESS = 0500. NUMBER OF LOCATIONS = 100
PAGE 1 OF 1

0500 0510 0520 0530 0540 0550 0560
---- ---- ---- ---- ---- ---- ----
6837 6837 0026 1402 00F2 0F67 0013
7F02 3802 26F2 140A 7F06 0003 5E02
6837 6837 0026 1402 00F2 0F67 0013
7F02 3802 26F2 140A 7F06 0003 5E02
D5FA 000F 6F02 0037 1202 000F
8818 0C33 27EA 0012 FFFF 56F7
D5FA 000F 6F02 0037 1202 000F
8818 0C33 27EA 0012 FFFF 56F7
8F1F 0042 14F6 0053 0303 0053
E5F7 D5F7 D6E7 8F43 0382 0026
8F1F 0042 14F6 0053 0303 0053
E5F7 D5F7 D6F7 8F43 0382 0026
FF02 0026 55E2 26D7 0007 1402
1202 0E26 0063 0037 3802 1408
FF02 0026 55E2 2607 0007 1402
1202 0E26 0063 0037 3802 1408
```

The hexadecimal address is indicated at the top of each column and the contents are shown below each column. The first entry in the column is in address + 0, the second is in address + 1, the third is in address + 2, etc.. The last entry in a column is in address + 15. For example, to find the contents of address 051A, count from 0 to 10 down the column that starts with hex address 0510. The contents of 051A = 0042.



```
DUMP OF PROCESSOR  P0. STARTING ADDRESS = 0100. NUMBER OF LOCATIONS =  200

PAGE 1 OF 1

0100 0110 0120 0130 0140 0150 0160 0170 0180 0190 01A0 01B0 01C0
---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ----
0C00 8F71 780F 8000 04BC 000D FC00 1603 45D1 1603 1602 14FD 11FF
0C11 14EF 8504 8000 0587 0A76 5654 FC00 1603 93CD 93CD AE3A 544E
0C22 D5F2 780F 8133 0643 0A77 4653 FC00 FC00 93CD 8F21 AE3B 0455
0C33 8F11 8504 0156 0715 11FF 1603 001F FC00 8F21 1E04 AE39 04AA
0C44 E5F0 780F 01C1 074B 14FF FC00 45DE 4438 4524 93CD AE3A 8B41
0C55 14F6 8504 0212 078B 11FF FC00 1603 1603 1603 8F25 0000 8BC1
0C66 FF00 780F 0246 0879 1602 D659 FC00 FC00 93CD 4610 0009 93CD
0C77 1200 5529 0271 090D FC00 464E FC00 FC00 93CD 1603 0001 8F01
8F01 0010 4526 02DF 099E 4562 1603 D5DA 8F21 5519 93CD 0191
8F11 05BF 1403 02FC 0985 1603 FC00 4543 462E 451B 93CD 333C
8F21 0005 14DD 032F 0A68 FC00 FC00 1603 1603 4518 91CB 0157
8F31 6535 14DC 036D 0ADE FC00 93C9 FC00 93CD 93CD 93CD 016C
FC00 0000 5404 039D 0BA5 E0CA FC00 FC00 93CD 93CD 4411 0108
8F41 7816 8301 03B0 0C0E 445F 1612 FD01 5525 8F21 14A8 01AB
8F51 708F FC00 0418 001B 1603 FC00 FC00 8F1F 4515 93CD 0001
8F61 8504 14D1 047B 8140 FC00 444C 001F 4522 93CD 93CD 01B5

PRESS <RET> TO CONTINUE
```

Figure 3.   Screen dump format and display.

```
DUMP OF PROCESSOR  P3. STARTING ADDRESS = Ø5ØØ. NUMBER OF LOCATIONS =  1ØØ

ADDR      Ø     1     2     3     4     5     6     7     8     9     A     B     C     D     E     F
----    ----  ----  ----  ----  ----  ----  ----  ----  ----  ----  ----  ----  ----  ----  ----  ----
Ø5ØØ    77F8  ØØØF  6837  7FØ2  42ØF  ØØØF  D5FA  881B  56F9  Ø612  8F1F  E5F7  7FØ2  41ØF  FFØØ  12ØØ
Ø51Ø    ØØØ7  ØØ8Ø  6837  38Ø2  7FØ6  42ØF  ØØØF  ØC33  ØØ53  ØC22  ØØ42  D5F7  8F1F  ØØ61  ØØ24  ØE26
Ø52Ø    14Ø2  14ØA  ØØ24  26FØ  ØØ35  ØØ12  6FØØ  27EA  ØØ53  8F41  14F4  D6E7  8F21  ØØ42  55E2  ØØ61
Ø53Ø    ØØ24  ØE26  14Ø2  14ØA  ØØ24  26DE  ØØ35  ØØ12  6FØØ  27D8  ØØ53  8F41  14F4  D6D5  26D5  ØØ35
Ø54Ø    ØØ12  6FØØ  ØØFØ  7FØ6  41ØF  FFØØ  12ØØ  FFFF  8ØØØ  34ØØ  Ø3Ø1  Ø38Ø  77F9  1BBD  ØØØ7  38Ø2
Ø55Ø    74ØØ  6837  ØF67  ØØØ1  7FØ5  42ØF  ØØØF  56F7  ØØ42  ØC11  ØØ51  ØØ24  ØE25  14Ø3  14Ø2  14ØB
Ø56Ø    ØØ34  27EE  ØØ13  5EØØ
```

Figure 4. Format for dump to printer.

ENDSIFT - End SIFT Session

Format: `ENDSIFT    (no parameters)`

Deallocates processors, erases the screen and deletes SIFT command symbols. The DCL prompt ($) will reappear.

HLT - Halt Processor(s)

Format: `HLT [Pn .. Pm]`

Halts processors n through m.
If processors are not specified, the default is SELected processors.

The SIFT display will indicate the halted processors with the letter H in reverse video which appears below the processor number.

LOAD - Load processor(s)

Format: `LOAD filename [ ON Pn,..,Pm ]`

Loads an absolute executable image file to the BDX 930s.

Filename for LOAD cannot include a directory spec or logical name. The file is assumed to be located in LOAD:, which is assigned to the user's default directory at the start of the SIFT session. If a file to be loaded is located in another directory, the logical directory LOAD: must be reassigned using the DCL commands ASSIGN or DEFINE

ON is a required keyword if processor(s) are entered on the command line. If processor(s) are omitted the default is SELected set.

The SIFT display will show the name of the loaded file if the LOAD command was successfully executed. The file name appears next to the heading LOAD: in the display (see figure 1 for example).

Example: `LOAD STELO ON P2`
(Note - if the file extension is missing the default is .X)

MAP - Map Symbolic Names

Format: MAP filename

Associates (maps) symbolic names (scope and label) used in a SIFT program to BDX memory addresses. The MAP command must be executed before scope and label can be used in other SIFT commands (see definition of scope and label below).

The cross references between labels and addresses are contained in a keyed indexed file that is created during use of the symbolic assembler or linkage editor and has the default filename extension .CRF. However, the file.CRF will not be created unless the listfile (/L) option is specified in the ASM930 or LNK930 command line, or unless the parameter M is included the SIFTLNK command line. The MAP command cannot be used if these options are not specified. See Appendix B for a description of the listfile option when invoking the symbolic assembler and linkage editor. Also see the description of SIFTLNK command in this section.

Filename for the MAP command cannot include a directory spec or logical name. The file is assumed to be located in MAP:, which is assigned to the user's default directory at the start of the SIFT session. If a file to be used is located in another directory, the logical directory MAP: must be reassigned using the DCL commands ASSIGN or DEFINE.

Example: MAP STELO (Note - file extension .CRF is assumed)

The SIFT display will show the name of the map file beside the heading MAP: (see figure 1 for example).

Definition of scope and label: The scope is a symbolic name for a program module. The MAP command initializes the scope to the first module in the program and prints the scope name in the SIFT display. Labels (or names) are used to represent addresses and are defined only within a particular scope. Scopes and labels are used extensively in various SIFT commands to preclude the use of hex numbers and provide a symbolic reference to locations in the program. The syntax is as follows: $label or $scope.label. The $ is required as the first symbol. Scope is optional. If used it must come before label and separated with a dot. If scope is omitted, the default is the current SCOPE name shown in the SIFT display. The SCOPE name in the display can be changed using the command DISP/SCOPE=scopename as described under qualifiers for the DISPLAY command.

PC - Program Counter

```
Format:  PC  hexaddress [ON Pn .. Pm]
         PC  $[scope.]label [ON Pn .. Pm]
         PC  <RETURN>
```

Sets or reads the program counter on specified processor(s). Defaults to SELected set if processors are not specified on the command line.

The first format sets the PC to the hex address specified. The second format sets the PC to the address associated with the $label or $scope.label specified. To use this format labels must first be mapped using the MAP command. The third format (null command line) simply reads and displays the current program counter.

```
Examples:   PC 100 ON P2,P3  (PC set to 0100 hex on processors P2 and P3.)
            PC $START        (PC set to address associated with the label
                              START.  Set on SELected processors.)
            PC <RETURN>      (The current PC is displayed for SELected
                              processors.)
```

RDM - Read Memory

```
                / $[scope.]label\
Format:  RDM <                   >; ...
                \  hexaddress   /
```

Reads up to 8 BDX 930 memory locations of SELected processors. The locations can be entered in hex or using $scope.label and each location is separated with a semicolon. The memory address and contents are displayed on the terminal. See MAP command for a definition of scope and label.

```
Examples:   RDM A123; 53C0; 4269      RDM $A$35; $A$36
```

RDR - Read Register

```
Format:  RDR Rn; .. Rm
```

Reads up to 8 BDX 930 registers of SELected processors. The registers are separated by semicolons. Example: RDR R1; R2;R4. The register contents are displayed on the terminal.

```
Examples:   RDR R0;R1;R2      RDR R15
```

SEL - Select Processor(s)

Format: SEL [Pn .. Pm]
        SEL NONE

Selects or deselects processor(s) n through m for various SIFT functions such as debugging. Processors in the SELected set are used as the default in a number of other SIFT commands. The second format deselects all processors currently in the SELected set. If processors are not specified on the command line, all allocated processors will be put in the SELected set.

Processors specified in the command line can be separated by spaces, commas or no separation.

```
  Examples:  SEL p3 p5    sel p3,p5    SEL P3P5
```

The minus sign is used to deselect processors.

```
  Examples:  SEL -P3  (deselects P3)  SEL P3,-P4  (selects P3, deselects P4)
```

Selected processors are shown in the SIFT display with processor numbers in reverse video.

NOTE: The ALP (Allocate) command automatically SELects all processors that have been allocated. You must use the minus sign to deselect those processor(s) not wanted for a particular operation (e.g., debugging).

SIFTASM - Assemble Source File

Format: SIFTASM [assembler source filename] [L]

Calls the BDX 930 Relocatable Assembler to assemble a BDX 930 assembly code source file. If the file name is omitted the default edit file name is used. The output is a relocatable binary image file with extension .RX. If the parameter L is included in the command line, a listing will be generated with the extension .LIS. Appendix B gives additional options which are available for the Relocatable Assembler and the formal command-line syntax to invoke these options.

SIFTLNK - Link SIFT Operating System (OS) Modules

Format: SIFTLNK [OS binary filename] [M]

Calls the BDX 930 Linkage Editor to link a new OS file to standard SIFT modules. If the OS filename is omitted, the default edit filename will be used. Produces a loadable image file with extension .X. If parameter M is included in the command line, a map is generated with the extension .MAP and a cross reference file is created with the extension .CRF. File.MAP can be printed and provides a listing of module names, mnemonic names and associated hex addresses. File.CRF is a keyed indexed file used by various SIFT functions to retrieve hex addresses (see MAP command). Appendix B gives additional options which are available for the Linkage Editor and the formal command-line syntax to invoke these options.

SPO - Compile a Pascal Source File
SPL
SPN

Format: SPO [Pascal source file]
SPL [Pascal source file]
SPN [Pascal source file]

Compiles the Pascal source code file. If the filename is omitted, the default edit filename is assumed. SPO produces BDX assembler source code with file extension .SR as output. SPL produces a Pascal listing. SPN has no output and is used for a syntax check.

SST - Single Step Debugging

Format: SST (no parameters)

Single steps program instructions for processors in the SELected set for debugging purposes.

After the SST command is executed, a screen display will show the current program counter, machine code (hex) and disassembly of the instruction; and the SST> prompt will replace the SIFT$ prompt. See figure 5 for an example of the SST display.

SST Subcommands at the SST> prompt:

```
  Format:  SST> <RETURN>
           SST> P: Pn,..Pm
           SST> number from 1 to 999
           SST> EXIT
```

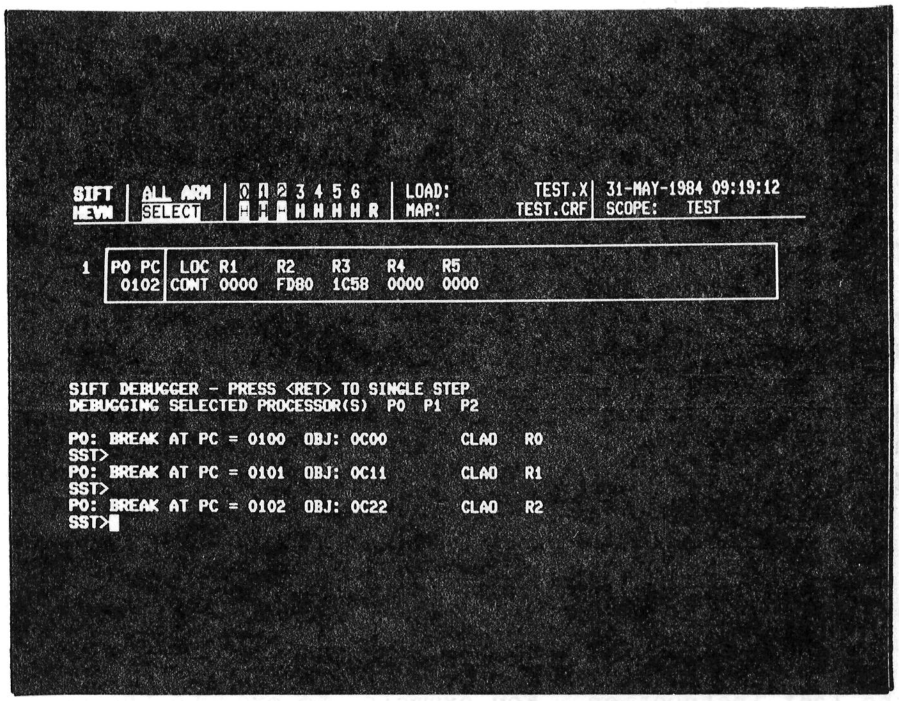
The first format will execute the single instruction pointed to by the program counter (PC). After the instruction has executed, the SST display will show the information for the next instruction. Note that, in addition to single steps by pressing <RETURN>, any DCL or SIFT command may be entered at the SST> prompt. For example, a specific breakpoint address can be set by using the BRF (breakpoint fast) command or the contents of a register can be changed by using the STR (store register) command. After the DCL/SIFT command is processed, the SST> prompt will reappear.

The second format designates which processors (of the SELected processors) will be shown in the SST display. By default the information displayed after single step executions will be for the first (lowest numbered) processor in the SELected set. To display other processors in the SELected set enter one or more processors according to the following examples:

```
SST> P:P1,P3   SST>P:P2 P4   SST>P:P0   SST>P:ALL   SST>P:NONE (no display)
```

The third format executes a number of instructions in one operation and only displays the last instruction. For example, to execute the next ten instructions enter the number 10 at the SST> prompt (example, SST> 10). Any number of instructions can be executed in this manner up to 999.

The forth format (EXIT) terminates single step and returns to the SIFT$ prompt.



```
SIFT | ALL ARM | 0 1 2 3 4 5 6   | LOAD:         TEST.X| 31-MAY-1984 09:19:12
HEVN | SELECT  | H H - H H H H R | MAP:        TEST.CRF| SCOPE:    TEST

 1 |PO PC| LOC R1    R2    R3    R4    R5
   | 0102|CONT 0000  FD80  1C58  0000  0000


SIFT DEBUGGER - PRESS <RET> TO SINGLE STEP
DEBUGGING SELECTED PROCESSOR(S)  PO  P1  P2

PO: BREAK AT PC = 0100  OBJ: 0C00        CLA0   R0
SST>
PO: BREAK AT PC = 0101  OBJ: 0C11        CLA0   R1
SST>
PO: BREAK AT PC = 0102  OBJ: 0C22        CLA0   R2
SST>█
```

Figure 5. Display of single step debugging.

START - Start Processor(s)

Format: START [Pn .. Pm]

Starts processors Pn through Pm. If processors are not specified, the default is SELected processors. Processors are started simultaneously.

The letter R in the SIFT display beneath the processor number indicates the processor is running. The letter H means it is halted.

STAT - Status of Processors

Format: STAT (no parameters)

Causes the SIFT display to be updated (header information at the top of the screen). The displays of memory and register contents, which are provided by the DISPLAY command, are not updated.

Use of the STAT command will not disturb operation of the system.

STM - Store Memory

```
                  / $[scope.]label\
Format:    STM  <                  >, hex_value; ...
                  \  hexaddress   /
```

Stores up to 8 hex_values in specified BDX 930 memory addresses of SELected processors. The address where the value is to be stored and the value to be stored are separated by a comma; additional pairs of these values are separated by semicolons.

The addresses can be specified in hex or by label (see MAP command for a description of scope and label).

Examples: STM 5C2A, 3F; 3884, A   STM $START,0CFF;$MOD1.A$3, 0

STR - Store Register

Format: STR Rn, hex_value; ...

Stores up to 8 hex_values in specified BDX 930 registers of SELected processors. Commas separate the register from the value to be stored and semicolons separate the register/hex-value pairs.

Example: STR R3, FF0C; R4, 8E3D

SIFT USER'S GUIDE

APPENDIX A

SIFT SCHEDULE GENERATING UTILITY

INTRODUCTION

The SIFT Schedule Utility (or scheduler for short) is designed to provide an interactive pre-processor to prepare the assembly macros for the Bendix Model 930 Flight Computers. The pre-processor syntax, which is described below, is intended to replace the direct preparation of the assembly macros with an English like set of instructions. In addition to preparing the assembly macros the pre-processor provides display options for presentation of the data structures used in the preparation of the BDX 930 assembly macros.

The SIFT Schedule Utility provides for automatic scheduling based upon the order of the task definition records processed. Alternatively, the researcher can override the auto schedule algorithm and input individual schedules for each processor and grouping of processors.

As an aid in development the scheduler logs all input during a session onto a file identified by the default name SCHED.TRN. This file is in a standard text format and can be edited and used as an input for subsequent sessions. The <FILE INPUT=file_name> command defines the file to be used.

The SIFT Schedule Utility produces an output file (defined by the command <FILE OUTPUT=file_name>) which is in macro assembly language for the Bendix Model 930 Flight Computer. Two items of special note are that the symbols must be a maximum of 5 characters, and that the assembler is sensitive to spacing on input lines. The symbols used by the scheduler are automatically truncated to 5 characters without a warning message, which allows the user to enter mnemonics of any length as long as the first 5 are unique. When editing a transaction file, note that the scheduler inserts tab characters between fields instead of spaces to avoid the spacing problems of the assembler.

SIFT SCHEDULER - SYNTAX SUMMARY

The SIFT Schedule Utility is designed to provide a simplified method of deriving the BDX 930 macro assembly language necessary to schedule tasks and voting. The syntax of the scheduler is given below in two parts. The first part, the definition syntax summary, describes the syntax to be used in the initial setup phase and for auto scheduling. The second part, the schedule edit syntax summary, describes the syntax to be used for the optional customizing of schedules by the user. The transition from the first syntax to the second is made by entering the SCHED EDIT command. To return to the main syntax use the EXIT command. The prompt for the main definition syntax is > and the prompt for the schedule edit syntax is *> . File SIFTDIR:SCHEDULE.DEF contains the SIFT default schedule and may be used as a base schedule.

SIFT SCHEDULER - DEFINITION SYNTAX SUMMARY

The following syntax is used in the normal mode (prompt >) to define system parameters and set up the automatic scheduling algorithm. This phase must preceed the manual EDIT SCHEDULE mode so that all tasks, groups and variables are defined prior to editing the schedules. To return to this mode type EXIT. To exit this program type EXIT again, the output is automatically copied to the file defined by the FILE OUTPUT=filename command.

```
* [ your comments ]
```

An asterisk denotes the beginning of a comment string. All characters after the asterisk are ignored. Comments are sent to the terminal if input is read from an alternate file.

```
ID integer_value
```

Allows user to enter the task ID. This must be done for each task or an assembly error will occur.

```
IN [var1[=buffer_location],[var2,...];]
```

Allows user to specify the input set for the object task; by typing "IN" alone the current input list for the object task is displayed. Note that the semi-colon is required to continue parameter entry on the same line. The buffer location parameter can also be defined by an OUT or VAR statement. Only one definition is required. The last value entered is always used.

```
DELETE [ taskname or groupname ]
```

Deletes the specified task or group from the schedule. Does not remove the task from the data structures. The object task becomes "null" upon execution of this command. If no task-name is specified the current object task is deleted.

```
DESTROY name
```

Removes the specified name (task or variable) from the schedule and the data structures. NOTE: All references are destroyed, therefore, variables cannot be destroyed if they are referenced by a task which is still valid.

```
DISPLAY [ name, ALL, VAR, TASK, SCHED or GROUP ]
```

Display information concerning a given name or all names, or display information for a class of names.

DURA [integer value]

Specifies duration in 1.6 ms cycles that execution of the object task requires. No default is used, and an error message will be generated during assembly code generation for each task for which a duration was not specified.

EDIT

This statement is the same as SCHED EDIT; it changes the program mode to the edit mode. See SCHED EDIT command for more information.

END_GROUP

This statement identifies the end of a group of tasks. It sets the current group to null. Note that a group can be reopened by entering the GROUP statement again with the same name.

EXIT

Causes the program to exit and save the output schedule and data structures created during the session, provided that an output file was previously specified and a schedule exist. An error message is written if a schedule cannot be generated. An EXIT command encountered during an alternate input read returns control to the terminal.

FILE [(INPUT,OUTPUT)=filename]

Informs processor of files to be used for input and output of the processor. If an external input file is used, control is automatically returned to the user when an EXIT or QUIT command is executed or at the end of the file. NOTE: no blanks should be used in the file specification.

GROUP [groupname]

Define a group name and start position. If no name is given then pertinent data regarding the current group is displayed.

INIT [ALL, SCHED]

Initializes all or part of the data structures used in the SIFT scheduling pre-processor.

LIST [ALL, OUT or SCHED]

List information concerning all tasks, variables and groups; or the resulting assembly output; or the current schedule. This output is copied to the file SCHEDULER.LIS which is then printed and deleted.

NAME asmb_name

Provides the processor with the name to be used in generating the assembly macros. This name is the TITLE parameter of the input to the BDX 930 assembler.

OUT [var1[=buffer_location],[var2,...];]

Allows user to enter the output variables for the object task, and set the buffer location to match the Pascal 'include' file. The semicolon (;) is required to terminate the output list if other parameters are to be entered on the same line. As with the IN and VAR statements, the buffer location only needs to be entered once.

QUIT

Causes the program to exit without saving the results of the session. Exception, if the processor assembly output file was created using "SCHED /OUT" command, this file remains intact.

REPEAT groupname

This command causes the group of tasks specified by groupname to be repeated in the schedule.

REPLIC [n1(,n2,n3..)]

Specifies the number of replicates of the object task which are required, or displays current value. If a task is to have defined levels of redundancy, they can be entered, e.g. [5,3,1].

SCHED [AUTO,ON,OFF,OUT,EDIT]

Provides control of and information concerning the SIFT schedule being generated by the processor. If no options are specified, the system generates the detailed schedules for subsequent DISPLAY SCHED or LIST SCHED commands. Options are:

AUTO	Sets the auto schedule flag true (default). This flag when true accepts the order of task and group definitions as the schedule order.
ON	Same as AUTO.
OFF	Sets the auto schedule flag false. When false, no entries are added to the schedule.
OUT	Takes the schedule created in AUTO mode or EDIT mode and generates the BDX 930 assembly language module.
EDIT	This command changes the mode of the program and switches to the syntax described below under the heading of SCHEDULE EDIT SYNTAX SUMMARY. The prompt for the edit mode changes to *> .

TASK [taskname[/display,/list]

Displays current object task, or alternately defines an object task, or provides display or list of the task parameters. An object task is that task which was last referenced, and on which the parameter commands (IN,OUT,etc) operate.

VAR [varname[=buffer_location][attributes]

Defines a variable to the scheduler. Used when a variable is used by a task before it is defined, or when special attributes are required. Allowable attributes are:

PRESET/NOPRESET for variables defined during initialization, the default is NOPRESET.

VOTE/NOVOTE for variables which may or may not be voted, the default value is VOTE.

The buffer location must be entered once for each variable, but may also be entered when specifying INput and OUTput variables during TASK definition.

SIFT SCHEDULER - SCHEDULE EDIT SYNTAX SUMMARY

The following synyax is provided as an option to allow the user to customize the schedule for tasks running on the BDX 930.

```
* [ your comments ]
```

An asterisk denotes the beginning of a comment string. All characters after the asterisk are ignored. Comments are sent to the terminal if input is read from an alternate file.

```
COPY Sij Smn
```

Copy schedule "Sij" to schedule "Smn", where i and m are digits representing the number of processors and j and n are the processor to be scheduled. Note that S00 is a master schedule and is defaulted to the schedule produced by the AUTO schedule algorithm if selected initially.

```
DELETE slot_number
```

Deletes the task at the given slot number and replaces it with the null task.

```
DISPLAY Sij [ or Si ]
```

Displays the schedule Sij on the terminal, or alternately displays the set of schedules for i processors.

```
EXIT
```

Exits the schedule editing mode and reverts to the definition mode.

```
INIT Sij [ or ALL ]
```

Initializes the specified schedule or optionally all schedules, except schedule S00 unless specifically requested.

```
INSERT slot_number task_name[ or group_name ]
```

Inserts the task or group specified at the slot given and adjusts all subsequent tasks already in the schedule down by one if inserting a task or by the number of tasks in the group.

LIST Si [or ALL]

Lists the schedules Si or alternately displays the set of all schedules.

PURGE slot_number

Purges the task at a given slot and moves all other tasks up in the schedule.

REPLACE slot_number [or task_name] new_task_name

Replaces the task at a given slot with a new task, or optionally replaces a given task with a new task globally.

SCHED Sij

Specifies which schedule is being edited.

SHOW Sij [or Si]

Displays the schedule Sij on the terminal or alternately displays the set of schedules for i processors.

SIFT USER'S GUIDE

APPENDIX B

IMPLEMENTATION OF THE BDX 930

RELOCATABLE ASSEMBLER
SYMBOLIC ASSEMBLER
LINKAGE EDITOR

1. INTRODUCTION

This appendix describes the operation of the BDX 930 relocatable assembler, symbolic assembler and linkage editor.

Section 2 gives an overview of the VAX-11 implementation and general rules of operation are covered. Sections 3, 4, and 5 contain specific instructions and command-line syntax for operating the symbolic assembler, the relocatable assembler and the linkage editor, respectively. The information in these sections was extracted from the documentation file BDX930.DOC prepared by Virginia Polytechnic and State University, September 1982, and is included in this appendix for ease of reference.

Section 6 contains a list of pertinent files, along with a short explanation of each file's contents.

Section 7 describes the programs which create files to load the BDX 930 computers directly from the VAX-11. The source code for these programs is in SIFTFILES.PAS and ASMFILES.PAS. The implementation of these programs is transparent to the user; however, they are described in this guide primarily for systems programming installation and maintenance purposes.

Additional information concerning installation, maintenance and testing for the symbolic assembler, relocatable assembler and linkage editor is contained in the documentation file BDX930.DOC.

2. VAX IMPLEMENTATION OVERVIEW

The topics covered in this section concern the VAX operating system interface. File specifications and general rules of operation will be discussed in relation to the BDX930 assembler/linker system. Hints useful to the user are also covered.

FILE SPECIFICATIONS

File specifications are of the form Dev:[Direct]File.Ext;Version where all but the file name is optional and a single file specification must not contain imbedded spaces or tabs. If the Device:[Directory] is omitted the default, SYS$LOGIN, is used. The Version number default depends on the context of its use. The most recent version of a file will be used in the case of an input specification. In the case of an output specification, a file having a version number one greater than the most recent will be created. If the file does not exist, the version number of 1 is used. The extension (Ext) may be omitted but the dot (.) must be included to terminate the file name if a null extension is desired. When the dot is omitted it indicates the use of a default extension. In particular, a specification of the form File (no dot) will be translated to File.DAT or File.CRF or File.X depending on the type of file (explained in sections 3, 4 and 5 of this appendix).

All three of the BDX930 programs impose a limitation on the length of a file specification, namely that a single file identifier must be less than 16 characters in length. A library search directory specification is the single exception to this rule; it is limited to 39 characters or less. The VAX logical name facility can be used if longer identifiers are required. These limitations do not include characters used for switch qualifiers. A further restriction is that wild card characters are NOT allowed in any file specification or library specification.

LOGICAL NAMES

VAX logical names may be used with any of the programs, subject to a few restrictions. An example will best demonstrate the use of logical names. Suppose the following command line is entered at monitor level:

```
ASSIGN Dra0:[BDX930.Source] BDX
```

This equates the name BDX to the logical device Dra0:[BDX930.Source]. The name may then be used as a device specification in a command line such as:

```
ASM930R BDX:File.ext
```

The logical name, BDX, must be delimited by a colon (:) to indicate a logical device. The above command line is functionally equivalent to:

```
ASM930R Dra0:[BDX930.Source]File.ext
```

The only difference is that this command line violates the 15 character file specification limit, while the other one does not.

The VAX logical name facility is more general than this example suggests. In fact, any part of or all of a file identifier may be assigned a logical name. The use of general logical names, however, is not recommended with these programs, primarily because of possible unexpected consequences and an inconsistant set of rules. Details can be found under the title NOTES, in the section on the particular program in question (sections 3, 4 and 5 of this appendix). It is important to note that logical devices, such as in the previous example, are not subject to these restrictions and can be used anywhere.

COMMAND LINE RULES

The command line used to invoke any of the programs conforms to VAX standards. If a command exceeds a single line, the continuation character - (dash) can be used. There is a 511 character maximum limit on the length of any command line imposed by these programs. Continuation characters (-) are not included in this limit. Note that this maximum may be superceeded by local operating system limitations. Finally, indirect file specification (@FILE) within a command line is not allowed. The command line rules for each of the programs are covered in more detail in sections 3, 4 and 5 of this appendix.

TEMPORARY FILES

All temporary files needed by the assembler/linker programs will be created in the user's login directory by default. While these files are deleted after use, they can temporarily consume a significant amount of disk space.

HINTS AND ERROR 99

There are a few details of the operation of these programs that may be useful. First, the VAX has a "null" I/O device (NL:) to which files can be sent, never to be seen again. This is sometimes useful in suppressing list file creation. Simply specify NL:/L in the command line and the list file will never be.

There is also a listing error number that is not included in the Bendix BDX930 user's manual (It was added to the VAX implementation). Error number 99 indicates an overflow of the location counter. This will occur any time a location (relative or absolute) greater than 32767 (decimal) is encountered. The error is not fatal and recovery is made simply by resetting the location counter to zero.

The list files produced by all three programs are formatted for a default page length of 58 lines/page (This default is set during system generation). The default may be changed at run-time with the /H switch. If n/H is entered as a command line parameter, the page size will be n lines/page.

3. SYMBOLIC ASSEMBLER

This section describes the calling sequence and specific instructions for the symbolic assembler, which will assemble individual source modules into an absolute executable image file.

CALLING SEQUENCE

```
ASM930{global options} {filespec/local option} ..... Input{/N} ...
```

GLOBAL OPTIONS		(Defaults)
/K	Octal list format	(Hexidecimal)
/M	Map (memory assignment listing)	(No map)
/P	Page zero symbol listing	(No listing)

LOCAL OPTIONS		(Defaults)
/L	Listfile	(NL:)
/O	Absolute image file	(Scratch file)
/E	Library search directory	(No search)
/N	Module Not to be included in listing	(Include module)
Input(s)	Input source module file(s)	(No default-error)
n/H	Format list file for n lines/page	(58 lines/page)

EXAMPLE

The command line,

```
ASM930/M/P Test.ls/L Test.ob/O Drc:[Sift]/E Testa.asm/N Testb.asm
```

causes the assembler to read in the source modules Testa.asm and Testb.asm. If either of these files contains unresolved EXTRN's, the directory Drc:[Sift] will be searched for files of the form "external symbol," "no extension." The list file produced will include a memory map with a page zero summary and will reside in the file Test.ls. Note that the source file, Testa.asm, will not be included in the listing. When the listfile option (/L) is specified as it is in this case, a cross reference file with the name File.CRF is automatically created during assembly. This is a keyed indexed file that is used later for mapping symbolic names used in the program to specific hex addresses (see MAP command in the main section of this guide). Also, if the extension is omitted from the executable image file (Test.ob in this example), the default extension .X will be used.

NOTES

1) If the library search mode is used and unresolved EXTRN's are present after processing the input module list, the directory specified will be searched for filenames having the EXTRN name with no extension. If the directory is not specified, SYS$LOGIN: is used.

2) There is no required order for the options to be specified in.

3) General logical names may be used for input file and list file specification. A logical device name used for a library specification must be terminated with a colon (:).

4) Read section 2 for more information.

4. RELOCATABLE ASSEMBLER

This section describes the calling sequence and specific instructions for the relocatable assembler, which will assemble individual source modules into individual relocatable object modules, having the same name and a .RX extension.

The object modules created can later be linked by the linkage editor into an absolute executable image file. Note that individual source modules must reside in separate files.

CALLING SEQUENCE

```
ASM930R{global options} {filespec/local option} .... Input{/N} ...
```

GLOBAL OPTIONS		(Defaults)
/K	Octal list format	(Hexidecimal)

LOCAL OPTIONS		(Defaults)
/L	Listfile	(NL:)
/N	Module Not to be included in listing	(Include module)
Input(s)	Input source module file(s)	(No default-error)
n/H	Format list file for n lines/page	(58 lines/page)

EXAMPLE

The command line,

```
ASM930R Test.ls/L Testa.asm/N Testb.asm
```

results in the creation of two relocatable object files, Testa.rx and Testb.rx. The list file, Test.ls will contain only the source listing for Testb.asm.

NOTES

1) A general logical name may be used to specify the list file. Any logical name used for an input specification must NOT be assigned to a file specification containing an extension or a version number. For example:

```
ASSIGN drb2:[system]inputfil.ext;100 Temp
ASM930R Temp
```

will not work properly, but

```
ASSIGN drb2:[system]inputfil Temp
ASM930R Temp:.ext;100
```

will work fine. A logical device may be used for a library specification if it is terminated with a colon (:).

2) There is no required order for specifying the options.

3) Read section 2 for more information.

5. LINKAGE EDITOR

This section describes the calling sequence and specific instructions for the linkage editor, which will link relocatable object modules assembled by the relocatable assembler into an absolute executable image file.

CALLING SEQUENCE

```
LNK930{global options} {filespec/local option} ..... Input{/N} ...
```

GLOBAL OPTIONS		(Defaults)
/K	Octal list format	(Hexidecimal)
/M	Map (memory assignment listing)	(No map)
/P	Page zero symbol listing	(No listing)

LOCAL OPTIONS		(Defaults)
/L	Listfile	(NL:)
/N	Module not to be included in listing	(Include module)
/O	Absolute image file	(Scratch file)
/E	Library search directory	(No search)
Input(s)	Input source module file(s)	(No default-error)
n/H	Format list file for n lines/page	(58 lines/page)

EXAMPLE

The command line,

```
LNK930/M/P Test.ls/L Test.ob/O Drc0:[Sift]/E Testa.rx Testb.rx
```

causes the linker to read in the relocatable object modules, Testa.rx and Testb.rx. If either of these files contains unresolved EXTRN's, the directory Drc0:[Sift] will be searched for files of the form "external symbol".RX. The list file produced by the linker will include a memory map with a page zero listing. The list file will reside in the user login directory in the file Test.ls. The absolute executable image will reside in Test.ob in the user login directory. The file Test.CRF will be created automatically and can be used later in the MAP command (see MAP command in main section of this document). If the extension .ob for the absolute image file had been omitted, the default file extension .X would have been used.

NOTES

1) If library search mode is used and unresolved EXTRN's are present after processing the input file list, the directory specified will be searched for filenames having the EXTRN name with a .RX extension. If the directory is not specified, SYS$LOGIN: is used.

2) General logical names may be used for any input file and the list file. A logical device may be used to specify the library search directory provided the name is terminated with a colon (:).

3) There is no required order for specifying the options.

4) Read section 2 for more information.

6. FILES LISTING

BDX930.DOC	VAX implementation document
ASM930S.FOR	Symbolic assembler source
ASM930R.FOR	Relocatable assembler source
LNK930.FOR	Linker source
SIFTFILES.PAS	VAX interface source program for linkage editor
ASMFILES.PAS	VAX interface source program for symbolic assembler
OPTIONS.FOR	Subroutine module linked with linkage editor and symbolic assembler (gets command line options for SIFTFILES and ASMFILES)
SIFTASM.COM	Provides a simplified method of invoking the relocatable assembler
SIFTLNK.COM	Provides a simplified method of invoking the linkage editor
IMAGEGEN.COM	Source compile/link command file
LOGINGEN.COM	Command verb definition generation command file
TMPDIRUSE.COM	Set temporary directory command file
LOCK.FOR	Module lock utility source
UNLOCK.FOR	Module unlock utility source
LOCKOB.COM	ASM930S module name list (for LOCK)
LOCKOR.COM	ASM930R module name list (for LOCK)
LOCKOL.COM	LNK930 module name list (for LOCK)
930TEST.ASM	Sample BDX930 source file (Processor self test)
930TEST.LIS	Listing of above produced by ASM930
930TEST.ABS	Executable image of source produced by ASM930

7. DESCRIPTION OF PROGRAMS SIFTFILES AND ASMFILES

The programs siftfiles and asmfiles, written in PASCAL, are actually modifications to the linkage editor and symbolic assembler programs, respectively. The primary purpose is to convert the absolute image files created by the linker and assembler to a format that can be used by the VAX-11 to load the BDX computers. The files created by the linker and assembler were designed for use by the Data General Eclipse but cannot be used by the VAX. The modification is achieved by calling siftfiles (or asmfiles) directly at the end of the linker (or assembler) program using the run time library routine, lib$do_command. The user does not have to invoke these programs since they are run automatically with the LNK930 or ASM930 command lines. The absolute image file produced by the linker or assembler is used as input, a new file is created and the original file is deleted.

Siftfiles and asmfiles also have additional functions. Siftfiles takes the listfile produced by the linker and makes a four column listing of the mnemonic name and address table. (Original version had a one column listing that used more paper.) Both siftfiles and asmfiles create a cross reference file (File.CRF) using the information contained in the listfile if the /L option is specified in the LNK930 or ASM930 command line. File.CRF is a keyed indexed file that associates symbolic names used in the program with hex addresses. It is used for the MAP command (see MAP command in the main section of this document) and allows retrieval of addresses using scopes and labels.

One other file is associated with this modification. OPTIONS.FOR is a fortran subroutine GET_OPTIONS which is linked with the linkage editor and symbolic assembler. Its purpose is to get the information from the LNK930 or ASM930 command line and put it in common (using lib$put_common) for use by siftfiles or asmfiles. Siftfiles and asmfiles get the information out of common (using lib$get_common) and determine what files to create from the options specified by the user.

For siftfiles and asmfiles to be invoked from the linkage editor and symbolic assembler, respectively, a command file must contain the following definitions:

```
$SIFTFILES :== $SIFTDIR:SIFTFILES
$ASMFILES :== $SIFTDIR:ASMFILES
```

SIFT USER'S GUIDE

APPENDIX C

SYSTEMS PROGRAMMING FOR THE SIFT ENVIRONMENT

1. INTRODUCTION

This appendix describes the software developed for the SIFT environment which is implemented on the VAX-11. The information is intended primarily for systems programmers required to maintain or upgrade the programs described herein, or to develop new programs for future requirements.

Section 2 contains a list of all the programs that make up the SIFT environment.

Section 3 describes the basic programming concepts used for SIFT and section 4 gives a functional description of each interface program.

Section 5 contains a summary of the device driver for the BDX 930 processors and the driver's function dependent operations.

2. LISTING OF APPLICABLE PROGRAMS

The software for the SIFT environment consist of five major packages (only source code files are listed):

a. SIFT Schedule Generating Utility (described in Appendix A).

b. BDX 930 Assembler and linkage programs (described in Appendix B).

c. SIFT Interface Programs:

(1) Command Procedures

- siftup.com
- makesyms.com
- runttdsp.com
- endsift.com
- alp2.com
- debugloop.com/calls sstoff.pas
- delsyms.com

(2) PASCAL modules containing common procedures or declarations

- gen.pas
- siftdec.pas

(3) PASCAL programs to implement the SIFT commands (Note: programs listed together are linked. The module gen is also linked with each group and must be listed last in the LINK command line.):

Main Program	Parser State Table	Associated Modules
sift.pas	genstate.mar	
ttdsp.pas	ttdspsta.mar,	vtdraw.pas
stopsift.pas	genstate.mar	
select.pas	procsta.mar	
arm.pas	procsta.mar	
alp1.pas	procsta.mar	
alp3.pas	procsta.mar	
unalp.pas	genstate.mar	
display.pas	dispsta.mar	dispsub.pas, loc.pas
mapsetup.pas	mapstate.mar	
load.pas	loadstate.mar	
start.pas	procsta.mar	
hlt.pas	procsta.mar	
pc.pas	pcstate.mar	loc.pas
bdxdump.pas	dmpstate.mar	diskfile.pas
stm.pas	stmstate.mar	loc.pas
str.pas	strstate.mar	
breakfast.pas	brfstate.mar	loc.pas
snglstep.pas	genstate.mar	
rdm.pas	rdmstate.mar	loc.pas
rdr.pas	rdrstate.mar	
status.pas	genstate.mar	
compute.pas		(not linked with gen)

d. Device Driver for the BDX 930 Flight Control Computers

e. SIFT Help Files

- sift.hlp	text file used for input to the VAX/VMS help facility
- sifthelp.com	command procedure for installing sift.hlp

3. Programming Concept for SIFT Environment

The overall programming concept for the SIFT environment is to provide a separate DCL command (and separate program) for each SIFT function. The DCL commands for SIFT comprise the SIFT command language. This language allows the user maximum flexibility and control of SIFT functions and permits the use of all DCL commands within the SIFT environment. It also provides a multi-user capability and the capability to add or modify functions easily to accommodate future requirements. Although not as user friendly as the menu driven approach, the SIFT operation can be implemented more effectively once a user becomes familiar with the SIFT command language.

To implement this concept, the initialization program (called SIFT) creates a global section in memory and spawns a subprocess. The subprocess runs a program (called TTDSP) whose function is to set up the SIFT display for a VT100 or VT125 terminal. The program TTDSP maps to the global section and the program SIFT exits. Control is returned to DCL and TTDSP runs as a subprocess concurrently with the user's main process. This allows the use of all DCL/SIFT commands while the SIFT display is on the screen. The global section continues to exist because TTDSP is mapped to it. All SIFT interface programs also map to the global section and can transfer information from one program (or task) to another by reading from and writing to the global section. Information in the global section is also used by TTDSP to update the SIFT display after each SIFT command. The global section is defined in the module gen (source code gen.pas) and the external declarations are in file siftdec.pas. All SIFT interface programs must link with gen and refer to siftdec.pas in an INCLUDE statement.

4. Functional Description of Programs and Sequence of Operations:

a. Programs to Initialize and Terminate the SIFT Environment:

The login procedure for SIFT users (members of the SIFT systems group [300,*]) contains global symbols and definitions necessary to operate the SIFT environment. This procedure defines the command SIFT:==@siftdir:siftup, where siftdir is the logical name for the directory containing all executable versions of the SIFT interface programs. The symbols sifting==0 and sifted==0 are also defined.

To start the SIFT session the user types the command SIFT (without parameters) which is defined in the system login procedure to call the command procedure siftup.com.

SIFTUP.COM, MAKESYMS.COM - Siftup.com starts the process by assigning the logical names MAP: and LOAD: to the user's default directory to locate the map and load files. It then calls the command file, makesyms.com, to define the symbols and command verbs that execute the SIFT tasks. Finally, it calls the program SIFT (sift.exe) to initiate the SIFT environment. When it returns from sift.exe it changes the DCL prompt to SIFT$.

SIFT.EXE - The primary functions of program SIFT are to create the global section in memory (calls the global procedure create_section declared in gen), initialize some arrays in the global section and spawn a subprocess by calling lib$spawn. Creating the global section requires a unique name which is constructed by getting the first 5 characters of the process name concatenated with '_GBL'. The subprocess also requires a unique name which is constructed by getting the first 5 characters of the process name concatenated with '_DSP'. An argument passed to lib$spawn is the command string which includes the command TTDSP and the name of the global section as the command line. The command TTDSP is defined in makesyms.com to call the command procedure runttdsp.com. Therefore, when lib$spawn is called runttdsp.com is invoked which in turn starts the program TTDSP and passes to TTDSP the name of the global section. When TTDSP is executed the program SIFT exits and the program TTDSP continues to run as a subprocess with the function of creating and updating the SIFT display.

TTDSP.EXE - The program TTDSP parses the name of the global section passed by runttdsp.com and then maps to the global section. It then calls the procedure vtdraw (in module vtdraw) to set up the display. Procedure vtdraw cycles up every second to update the display. Action is taken only if a SIFT command has signaled. Vtdraw queries the driver for a list of allocated processors, halted processors, and armed processors. The header is updated with this information and the load, map and scope data. This process doesn't disturb SIFT operation. Vtdraw can redraw the header (the startup sequence) or just update the header and display fields. The display is redrawn from the DISPLAY program. Updating the DISPLAY program's display can upset SIFT operation. Procedure up_disp queries the DISPLAY data structure, reads memory, registers and PC as required, and updates the screen. The section on the DISPLAY program has more information on the DISPLAY data structure. I/O to the terminal is by QIO. Output buffers are built holding up to 500 characters each. To insure contention for the terminal from the user process and vtdraw doesn't destroy the screen, the user process must wait until vtdraw is complete. This synchronization is done through the boolean update in the global section. Procedure set_update in module gen sets update to true (signaling vtdraw to begin), then waits for update to go false (signaling vtdraw complete).

To terminate the SIFT session the user types the command ENDSIFT (without parameters) which is defined in makesyms.com to call the command procedure endsift.com.

ENDSIFT.COM, UNALP.EXE, STOPSIFT.EXE, DELSYMS.COM - Endsift.com calls unalp.exe to deallocate processors, calls stopsift.exe to stop the subprocess, calls delsyms.com to delete all the command symbols and verbs, deassigns logical names LOAD: and MAP:, and finally restores the DCL $ prompt.

b. Programs which implement the SIFT commands:

General - All programs described in this section (except command procedures and the compute program) start by mapping to the global section and parsing the command line if applicable. This is normally done by one function called map_and_parse found in module gen. Map_and_parse calls lib$get_foreign (get foreign command line), sys$mgblsc (map global section system service) and lib$tparse (table driven finite state parser) to do its work. Lib$tparse uses a state table which is a MACRO-11 object module linked with the main program module (see parser state table column in section 2c(3) above). The state table lists the keys and action routines used by lib$tparse to parse the command line. The action routines are functions in the main program module.

The programs which implement the SIFT commands can be classified in two ways - programs to run and load the BDX computers, and programs for debugging.

(1) Programs for Running and Loading the BDX 930s:

ALP1.EXE, ALP2.COM, ALP3.EXE - This series of programs implements the ALP (allocate processor) command. The user types ALP with a command line listing the processors requested to be allocated or deallocated. The first program ALP1 maps to the global section and parses the command line (by calling the function map_and_parse in the module gen), and then executes ALP2.COM by calling lib$do_command. ALP2 does the actual allocation which is necessary because device units cannot be allocated permanently in a program. ALP2 then calls ALP3 which has the function of updating the global section and display. One other function of ALP3 is to put the allocated processors in the SELect set so it is not necessary for the user to do a separate SEL command after the ALP command.

SELECT.EXE - This program implements the SEL (select processor) command. The user types SEL with an optional command line listing processors selected or deselected from the allocated set. The command is not necessary after the ALP command since allocated processors are automatically selected. However, the user may want to work with only a portion of the processors allocated in which case the SEL command can be used for selection and deselection. The program does no I/O but merely writes to the global section variable which holds the selected set and updates the display.

ARM.EXE - This program implements the ARM (arm processor) command. The user types ARM with an optional command line listing processors to arm or dearm. If there is no command line the default is processors in the SELected set. The program calls map_and_parse and then calls $QIOW with the function code IO$_ARM or IO$_DISARM, as applicable. The last thing is to update the display.

HLT.EXE - This program implements the HLT (halt processor) command. The user types HLT with an optional command line listing processors to halt. If the command line is omitted, the default is SELected processors. After calling map_and_parse the only function is to call $QIOW with the function code IO$_HLT and update the display.

START.EXE - This program implements the START (start processor) command. The user types START with an optional command line listing processors to start. If the command line is omitted the default is SELected processors. After calling map_and_parse the only function is to call $QIOW with the function code IO$_START and update the display.

LOAD.EXE - This program implements the LOAD (load processor) command. The user types LOAD with the load filename on the command line. The program assumes the load file is located in the directory LOAD: and assumes a file extension of .X if no file extension is given. The primary functions listed in sequence are as follows:

1) Procedure map_loadfile - maps the loadfile to an array in VAX memory which allows the fastest transfer of data from the file.

2) Procedure load_proc - assigns channels, halts processors, gets additional information if the load size is greater than 32,000 words (procedure get_load_info) and calls procedure qio_load to do the actual loading from the array in VAX memory to the BDX memory. If the size of the load is greater than 32,000 words there has to be two separate load operations due to device driver limitations.

3) Procedure verify_load - if load_proc is successful verify_load does a procedure to read the BDX memory locations just loaded into a separate array in VAX memory, compares the read with the original load and, if there are errors, it makes up to 5 attempts to reload the processor(s).

4) Prints an appropriate message to the screen based on results of the foregoing operations. If the load is successful the name of the load file is printed in the SIFT display

(2) Programs for Debugging

BDXDUMP.EXE - This program implements the DMP (memory dump) command. The user types DMP and an optional command line. If the command line is omitted, the program prompts for information to do a dump to the screen. Besides screen dump two other dump operations are possible: dump to a text file for printing and dump to a binary file for subsequent loading using the LOAD command. Only one type of dump is possible each time the DMP command is used and the type of dump is determined by the command line syntax. The primary programming tasks are to determine the type of dump(performed by procedure get_dump_options), call the I/O driver to read specified BDX memory locations into an array in VAX memory (performed by procedure read_BDX), and process the output depending on the type of dump (performed by procedure process_output). The program also allows the user to enter additional access groups (address and number of locations) after the first access group has been processed (performed by procedure get_more_dumps). For text file and binary file dumps, the additional access groups are added to the file. Create_diskfile is a procedure in a separate module linked with the main program and is called to facilitate creation of the binary file if this option is selected. Create_diskfile calls $CRMPSC (create and map system service) which maps the array in VAX memory directly to a disk file thus avoiding numerous disk accesses to create the file.

BREAKFAST.EXE - This program implements the BRF (breakpoint fast) command. The user types BRF with a command line listing the memory location to set the breakpoint. If the command line is omitted the program prompts for the information. The program assumes processor(s) in the SELect set. The user cannot choose particular processor(s) for debugging with the BRF command but must use the SEL command for this purpose. The primary programming tasks are listed in sequence as follows:

1) Procedure get_breakpoint - gets information from the command line or prompts to determine the memory address for the breakpoint. The user can enter a hex address or a symbolic name (scope and label) to refer to the breakpoint. If scope and/or label is used the labels have to be mapped to a keyed indexed file which the user should have already done using the MAP command (see mapsetup.exe). If the file is mapped (indicated by a boolean in the global section), the function LOC (label) returns the hex address associated with the label. The function LOC is in module LOC which is linked with this program.

2) Procedure save_breakpoint - saves the contents of the breakpoint location by doing QIO to read the breakpoint and store the value in an array variable.

3) Procedure Control_Y_AST - calls lib$disable_ctrl to disable control_y for the command interpreter. Calls $QIOW with a special function code that establishes the control_y AST. The AST is the procedure exit_handler which restores the breakpoint if the user types control_y before the end of the program.

4) Procedure write_halt_inst - calls qio_write_memory to write a halt instruction (FC00) into the breakpoint location.

5) Procedure run_to_breakpoint - starts the processor(s) with qio function code IO$_START and runs until the processor(s) are halted at the breakpoint. Updates the SIFT display during this process. After each processor is halted the program checks the PC to see if it halted at the correct location. If not, an appropriate message is displayed.

6) Procedure restore_breakpoint - writes the original contents of the breakpoint location back into the breakpoint address and sets the program counter back to point to the breakpoint address (i.e., writes the breakpoint address to the PC).

7) Procedure set_update - updates the SIFT display.

8) If not already in SST mode (single step), the program calls the command procedure debugloop.com (using lib$do_command) to link this program with the single step program. Thus, when the breakpoint is reached the user can enter single step commands (see snglstep.exe).

SNGLSTEP.EXE - This program implements the SST (single step) command but is also entered automatically after the BRF command (see breakfast.exe above). To go directly into single step without setting a breakpoint in BRF, the user types SST without a command line. The program assumes processor(s) in the SELect set. The user cannot specify particular processor(s) for debugging with the SST command but must use the SEL command for this purpose. The SST command calls the command procedure debugloop.com. The function of debugloop.com is to establish a single step loop so that all DCL/SIFT commands can be entered from single step and control will always return to the single step program after the DCL/SIFT commands are executed. Debugloop.com also sets up a short (2 sec) delay in order to read any DCL or error messages printed on the screen. When SNGLSTEP is called from debugloop, a display is printed on the screen and the SIFT$ prompt is replaced by SST> indicating that single step commands as well as DCL/SIFT commands can be entered. The primary functions of program SNGLSTEP are listed in sequence as follows:

1) Procedure map_section - maps to the global section. Procedure map_and_parse is not used because there is no command line to parse.

2) Procedure disp_current_PC - displays the current PC, machine instruction code and disassembly of the instruction. This information is displayed for the first processor in the SELect set. The disassembly is obtained by calling the procedure disassemble.

3) The prompt SST> is printed on the screen. To single step the user presses <RETURN> or types a number to single step more than one time.

4) Procedure single_step - called when the user types <RETURN> at the SST> prompt. Single_step calls procedure qio_sst for each processor in the SELect set. Qio_sst does a QIOW with the function code IO$_SST which executes the single instruction pointed to by the PC. When qio_sst returns the new PC is read and the procedure disassemble is called. When disassemble returns the information is displayed in the same format as displayed by procedure disp_current_PC.

5) Procedure disassemble. This procedure is broken down into three parts: the main procedure which calls the QIO to read the machine instruction at the PC location, procedure disassemble_opcode which disassembles only the opcode portion of the machine instruction, and procedure disassemble_operand which disassembles the operand portion of the machine instruction. The data structures used for these procedures are arrays (or tables) to store the instruction mnemonics and a variant record type for the 16 bit instruction. The variant record allows a symbolic reference to any combination of bits in the instruction and greatly facilitates coding and testing/comparing bit fields. A separate procedure compute_effective_addr is called by disassemble_operand if a memory reference or skip/jump instruction is detected. The code which results from disassembly is put in a string variable and this plus the hex machine instruction (one or two words) are passed back to the calling procedure (display_current_PC or single_step).

6) Procedure multiple_SST - this procedure is called if the user enters a number at the SST> prompt. It does a loop which calls procedure single_step the number of times requested by the number entered. However, the display is not printed on the screen until the last single_step has been performed.

7) Procedure get_show_procs. This procedure allows the user to choose which processors (of those being debugged) are to be displayed and disassembled. By default the first processor in the SELect set is displayed but this may be changed by typing P: Pn,..,Pm or P: ALL or P: NONE. The effect is to show (and compare) instructions being executed simultaneously by the processors or eliminate the display entirely.

8) Lib$do_command passes DCL/SIFT commands typed at the SST> prompt and executes the command after which it returns to the SST> prompt. (Remember that debugloop.com performs this function of calling program SNGLSTEP in a loop.) The command EXIT entered at the SST> prompt will stop the loop, get out of single step mode and the prompt SIFT$ will reappear.

MAPSETUP.EXE - This program implements the MAP command to associate symbolic names (scopes and labels) used in a program with hex addresses. This cross reference information is in a keyed indexed file (file extension .CRF) that was created during assembly or linking if the listfile option was specified at that time (see Appendix B). The primary functions of program mapsetup are listed in sequence as follows:

1) The map file name passed by function map_and_parse is concatenated with the logical directory name MAP: and the file extension .CRF.

2) Lib$find_file is called with the map file name to determine if the file exists. If not, a message is printed on the screen.

3) If the map file exists (had been created during assembly or linking), the file is opened, several values in the global section are initialized and the display is updated with the map file name. Once the file is mapped in this manner, the user can use scopes and labels in other SIFT commands that refer to BDX memory locations. See MAP command in the main section of this guide for a definition of scope and label.

PC.EXE - This program implements the PC (program counter) command. The user types PC with an optional command line. The program has two basic functions - to set the PC to a specified address or to read the PC if no address is specified on the command line. Addresses can be specified on the command line by hex number or by scope and/or label (if previously mapped using the MAP command). In setting the PC processors can be listed on the command line or defaulted to the SELected set. In reading the PC processors cannot be listed and the Selected set is assumed. The write (set) PC and read PC functions are accomplished by QIOW calls with function codes IO$_WPC and IO$_RPC.

STAT.EXE - This short program (only 3 lines) implements the STAT command which causes the header information in the SIFT display to be updated. It has no effect on the memory and register displays from the DISPLAY command and will not disturb operation of the system. Function map_and_parse is called to map to the global section and procedure set_update is called to update the display. Both routines are in module gen which is linked with STAT.

RDM.EXE, RDR.EXE, STM.EXE, STR.EXE - These programs, all very similar in coding and functions, implement the commands RDM (read memory), RDR (read register), STM (store memory), and STR (store register). The functions of each are as implied by the command and they each call QIOW with the appropriate function code. However, RDM/STM differ from RDR/STR in that procedures must be included in RDM/STM for referencing memory locations by scopes and/or labels.

COMPUTE.EXE - This program implements the CMP (compute) command which converts a number or result of an expression to its equivalent in decimal, hexadecimal, octal and binary. It can be used as an on line calculator with additional capabilities of radix conversions and expression evaluation using standard operators and parentheses. The program operates in two modes: single computation mode when the user types a number or expression on the command line, and repeat computation mode when the user types <RETURN> after CMP. In repeat mode the program prompts for more numbers/expressions after the first is completed. Also, repeat mode remembers the result of the last computation and this result can be used in the next computation using the letter P. The main program is an algorithm which calls a procedure get_token to parse the expression and uses stack data structures to evaluate the expression.

DISPLAY.EXE - This program implements the DISPLAY command for displaying the contents of memory and registers. DISPLAY draws a display area for each group needed to present the data requested by the user. Up to six groups may be displayed at one time. A group is a row of up to 10 data items from a processor's memory or registers. Registers and memory locations cannot be displayed in the same group. The major data structure in the DISPLAY task is the 'display record' found in the global memory partition GEN. The display record contains an array of records each of which describes the composition of its associated group. These 'group descriptors' (display.proc_disp[i]) contain information about the screen display and the I/O queries needed to gather the required data from the SIFT processors. During command-line parse in DISPLAY, a temporary group descriptor is built. After the parse, for each processor identified in the command line, DISPLAY finds a group to hold its data. If the processor already has a group, that one is used. If not, another group area is drawn on the screen and is allocated to that processor. Once a group is found, then the temporary group descriptor is copied into it (the temporary descriptor is also used to define delete operations when that option is declared). After completing the update of the display record, DISPLAY then fills in the group areas on the screen with information from the group descriptor (e.g. processor number, labels, register numbers). Before exiting, DISPLAY signals for a screen update. Procedure VTDRAW uses the display record to update the screen with data from the SIFT processors.

5. DEVICE DRIVER

The device driver (source file SX.MAR) handles communication between the VAX and the device developed by Wyle Laboratories that controls operation of the eight Bendix BDX 930 processors. The driver is invoked by SIFT interface programs using QIOW system service calls and I/O function codes defined by the driver. The following device dependent commands are defined by the driver as follows:

ARM	arm processors
RDM	read memory
RDR	read register
WRM	write memory
WRR	write register
HLT	halt specified processors
WPC	write program counter
RPC	read program counter
START	start processors
SST	single step processors
DISARM	disarm processors
CLKWR	write to clock
CLKRD	read clock value
SENSEMODE	get allocate and halt status of all processors

The device driver returns information to the I/O status block (IOSB) in two formats depending on the function:

Format for HLT, START and SENSEMODE:

```
        word 1 = return status code
        word 2 = BDX status
        word 3
            byte 1 = set of processors allocated to anyone
            byte 2 = set of processors halted - actual status
        word 4
            byte 1 = mask of halted processors - those requested
            byte 2 = mask of armed processors
```

Format for all other functions:

```
        word 1 = return status code
        word 2 = BDX status returned
        word 3 = function dependent data returned (see below)
        word 4
            byte 1 = mask of halted processors
            byte 2 = mask of armed processors
```

The parameters required in the QIOW call and the function dependent output data returned in the IOSB is defined below for each I/O function code:

IO$_ARM	P1 = set of processors output = mask of armed processors in word 3 of IOSB
IO$_RDM	P1 = address of destination array P2 = number of bytes to be transferred P3 = number of access groups output = number of words transferred in word 3 of IOSB
IO$_RDR	P1 = address of destination array P2 = number of bytes to be transferred P3 = set of registers output = number of words transferred in word 3 of IOSB
IO$_WRM	P1 = address of source array of access groups P2 = number of bytes to be transferred P3 = number of access groups output = number of words transferred in word 3 of IOSB
IO$_WRR	P1 = address of source array of data P2 = number of bytes to be transferred P3 = register set. output = number of words transferred in word 3 of IOSB
IO$_HLT	P1 = processor set
IO$_WPC	P1 = source longword output = program counter in word 3 of IOSB
IO$_RPC	output = program counter in word 3 of IOSB
IO$_START	P1 = processor set
IO$_SST	output = program counter in word 3 of IOSB
IO$_DISARM	P1 = processor set output = mask of armed processors in word 3 of IOSB
IO$_CLKWR	P1 = value to put in clock CSR output = clock value in word 3 of IOSB
IO$_CLKRD	output = clock value in word 3 of IOSB
IO$_SENSEMODE	output = status of allocated and halted processors in word 3 of IOSB

1. Report No. NASA TM-86289	2. Government Accession No.	3. Recipient's Catalog No.
4. Title and Subtitle Software Implemented Fault-Tolerant (SIFT) User's Guide		5. Report Date August 1984
		6. Performing Organization Code 505-34-13-32
7. Author(s) David F. Green, Jr. Daniel L. Palumbo Daniel W. Baltrus		8. Performing Organization Report No.
		10. Work Unit No.
9. Performing Organization Name and Address NASA Langley Research Center Hampton, Virginia 23665		11. Contract or Grant No.
		13. Type of Report and Period Covered Technical Memorandum
12. Sponsoring Agency Name and Address National Aeronautics and Space Administration Washington, DC 20546		14. Sponsoring Agency Code

15. Supplementary Notes David F. Green, Jr., Kentron International, Inc., Hampton, VA 23666
Daniel W. Baltrus, Wyle Laboratories, Hampton, VA 23666
Daniel L. Palumbo, NASA Langley Research Center, Hampton, VA 23665

16. Abstract

Program development for a Software Implemented Fault-Tolerant (SIFT) computer system is accomplished in the NASA LaRC AIRLAB facility using a DEC VAX-11 to interface with eight Bendix BDX 930 flight control processors. The interface software which provides this SIFT program development capability was developed by AIRLAB personnel. This technical memorandum describes the application and design of this software in detail, and is intended to assist both the user in performance of SIFT research and the systems programmer responsible for maintaining and/or upgrading the SIFT programming environment.

17. Key Words (Suggested by Author(s)) SIFT Schedule Generating Utility Relocatable Assembler Symbolic Assembler Linkage Editor		18. Distribution Statement Unclassified - Unlimited Subject Category 62	
19. Security Classif. (of this report) Unclassified	20. Security Classif. (of this page) Unclassified	21. No. of Pages 52	22. Price* A04

N-305