Introduction

This module presents the student with the tools to construct programs under an OpenVMS system. Students will be presented with the advanced concepts involved in compiling, linking, running, and debugging programs. Students will also learn the conventions used in defining OpenVMS symbols, using them as an aid in writing programs. Students will learn how program sections are used to build programs. They will be able to read listing and map files.

Objectives

After completing this module, students should be able to:

- Interpret the symbols used by OpenVMS to describe structures.
- Compile C modules under OpenVMS.
- Explain the use of Program Sections by OpenVMS.
- Identify the cost for accessing misaligned data under the Alpha architecture.
- Perform basic debugging of a program.
- Produce programs using the Linker.
- Explain the general method used by OpenVMS to call procedures.

Topics

- OpenVMS Symbolic Naming Conventions
- Producing Programs
- Compiling
- Program Sections
- Data Alignment Issues
- Debugging
- Linking
- Alpha Calling Standard Mechanics

OpenVMS Symbolic Naming Conventions

OpenVMS uses a standard convention for describing data cells, objects (data structures), procedures, error messages, etc. This convention is intended to provide uniform names which will be distinct from customer defined symbols.

These conventions are referred to as SDL (Structure Definition Language).

General data cell format

Symbol	Meaning
facility$Gs_*name*	Global variable
facility$As_*name*	Global array
facility$Ps_*name*	Global pointer (NOTE: could be structure member)
facility$Is_*name*	Global integer (NOTE: could be structure member)

- Examples: EXE$GQ_SYSTIME, SCH$AQ_COMQH, HWPCB$IQ_KSP

Procedure name format

facility$name

- Examples: LIB$DATE_TIME, SYS$QIO, OTS$CVT_TZ_L

Constants

Symbol	Meaning
facility$_*name*	Return value
facility$K_*name*	Constant
facility$C_*name*	Constant
facility$M_*name*	Bit mask
facility$V_*name*	Bit position
facility$S_*name*	Size in bits of bit field (may also be string size)

- Examples: SS$_NORMAL, ACC$K_LENGTH, IO$M_READLBLK

Structure offsets/member names

facility$s_name Structure member name

- Examples: `DSC$W_LENGTH, ACC$L_PID`
- The *facility* identifies the subsystem/system service/object providing the name and *s* identifies the size of the object.
- Size fields are represented using the following letter designations:

Letter	Size
A	Pointer
B	Byte
C	Character (could be constant)
D	Double precision floating point
F	Single precision floating point
G	G-float
H	H-float
I	Integer extensions (IS == software determined integer size, IH == hardware determined integer size)
L	Longword (4 bytes)
N	Numeric string
O	Octaword (16 bytes)
P	Packed string
Q	Quadword (8 bytes)
T	Text string
U	Smallest unit of addressable storage
W	Word (2 bytes)

Names for structure offsets are defined by:

- Macros of the form: **$*xyz*def** for MACRO32 and BLISS in the macro libraries:

```
SYS$LIBRARY:STARLET.MLB (MACRO automatic)
SYS$LIBRARY:LIB.MLB (MACRO)
SYS$LIBRARY:STARLET.REQ (BLISS)
SYS$LIBRARY:CLIMAC.REQ (BLISS)
SYS$LIBRARY:LIB.REQ (BLISS)
SYS$LIBRARY:ARCH_DEFS.REQ (BLISS)
SYS$LIBRARY:TPAMAC.REQ (BLISS)
```

- Header files of the form: ***xyzdef*** for C in the text libraries:

SYS$LIBRARY:STARLETSD.TLB	(Run-Time Library definitions/prototypes)
SYS$LIBRARY:SYS$STARLET_C.TLB	(System/RMS services definitions/prototypes)
SYS$LIBRARY:SYS$LIB_C.TLB	(Kernel structures and procedure prototypes)

- Function prototype include *headers* of the form:

```
facility$routines.h
```

- C headers are also stored in ASCII form in SYS$COMMON:[DECC$LIB.REFERENCE.SYS$STARLET_C] and in SYS$COMMON:[DECC$LIB.REFERENCE.DECC$RTLDEF].

Locating Symbol Names For C

```
$
$ library/extract=iledef/output=sys$output sys$library:sys$starlet_c.tlb
/***************************************************************************
    ************************************************/
/* Created: 29-NOV-1995 19:48:14 by OpenVMS SDL EV1-49     */
/* Source:  29-NOV-1995 19:46:28 $64$DUA3250:[STARLET_H.SRC]STARDEFFL.SDI;1 */
/***************************************************************************
    ************************************************/
/*** MODULE $ILEDEF ***/
#ifndef __ILEDEF_LOADED
#define __ILEDEF_LOADED 1

#pragma __nostandard                    /* This file uses non-ANSI-Standard
    features */
#pragma __member_alignment __save
#pragma __nomember_alignment
#ifdef __INITIAL_POINTER_SIZE                   /* Defined whenever ptr size
    pragmas supported */
#pragma __required_pointer_size __save          /* Save the previously-defined
    required ptr size */
#pragma __required_pointer_size __short               /* And set ptr size default
    to 32-bit pointers */
#endif

#ifdef __cplusplus
    extern "C" {
#define __unknown_params ...
#define __optional_params ...
#else
#define __unknown_params
#define __optional_params ...
#endif
```

Locating Symbol Names For C (continued)

```
#if !defined(__VAXC)
#define __struct struct
#define __union union
#else
#define __struct variant_struct
#define __union variant_union
#endif

/*+                                                                    */
/* Define the Item List Entry 3 structure                              */
/*-                                                                    */

#ifdef __NEW_STARLET
typedef struct _ile3 {
    unsigned short int ile3$w_length;   /* Length of buffer in bytes      */
    unsigned short int ile3$w_code;     /* Item code value                */
    void *ile3$ps_bufaddr;              /* Buffer address                 */
    unsigned short int *ile3$ps_retlen_addr; /* Address of word for returned
     length */
    } ILE3;
#else     /* __OLD_STARLET */
typedef struct _ile3 {
    unsigned short int ile3$w_length;   /* Length of buffer in bytes      */
    unsigned short int ile3$w_code;     /* Item code value                */
    void *ile3$ps_bufaddr;              /* Buffer address                 */
    unsigned short int *ile3$ps_retlen_addr; /* Address of word for returned
     length */
    } ile3;
```

Locating Symbol Names For C (continued)

```
#endif    /* #ifdef __NEW_STARLET */
#define ILE3$K_LENGTH 12                /* Length of ILE3                    */
#define ILE3$C_LENGTH 12                /* Length of ILE3                    */
/*+                                                                         */
/* Define the Item List Entry 2 structure                                   */
/*-                                                                         */
#ifdef __NEW_STARLET
typedef struct _ile2 {
    unsigned short int ile2$w_length;   /* Length of buffer in bytes         */
    unsigned short int ile2$w_code;     /* Item code value                   */
    void *ile2$ps_bufaddr;              /* Buffer address                    */
    } ILE2;
#else     /* __OLD_STARLET */
typedef struct _ile2 {
    unsigned short int ile2$w_length;   /* Length of buffer in bytes         */
    unsigned short int ile2$w_code;     /* Item code value                   */
    void *ile2$ps_bufaddr;              /* Buffer address                    */
    } ile2;
#endif    /* #ifdef __NEW_STARLET */
#define ILE2$K_LENGTH 8                 /* Length of ILE2                    */
#define ILE2$C_LENGTH 8                 /* Length of ILE2                    */
```

Locating Symbol Names For C (continued)

```
#pragma __member_alignment __restore
#ifdef __INITIAL_POINTER_SIZE                    /* Defined whenever ptr size
    pragmas supported */
#pragma __required_pointer_size __restore            /* Restore the previously-
    defined required ptr size */
#endif
#ifdef __cplusplus
    }
#endif
#pragma __standard

#endif /* __ILEDEF_LOADED */

$
```

Locating Symbol Names for Bliss

```
$ editx/edt/read sys$library:starlet.req
    1       ! Version:  'X-7'
*'$iledef'
 6421       !*** MODULE $ILEDEF ***
*.:6440
 6421       !*** MODULE $ILEDEF ***
 6422       ! +
 6423       !  Define the Item List Entry 3 structure
 6424       ! -
 6425       literal ILE3$S_ILE3 = 12;
 6426       macro ILE3$W_LENGTH = 0,0,16,0 %;       !  Length of buffer in bytes
 6427       macro ILE3$W_CODE = 2,0,16,0 %;         !  Item code value
 6428       macro ILE3$PS_BUFADDR = 4,0,32,1 %;     !  Buffer address
 6429       macro ILE3$PS_RETLEN_ADDR = 8,0,32,1 %; !  Address of word for
    returned length
 6430       literal ILE3$K_LENGTH = 12;             !  Length of ILE3
 6431       literal ILE3$C_LENGTH = 12;             !  Length of ILE3
 6432       ! +
 6433       !  Define the Item List Entry 2 structure
 6434       ! -
 6435       literal ILE2$S_ILE2 = 8;
 6436       macro ILE2$W_LENGTH = 0,0,16,0 %;       !  Length of buffer in bytes
 6437       macro ILE2$W_CODE = 2,0,16,0 %;         !  Item code value
 6438       macro ILE2$PS_BUFADDR = 4,0,32,1 %;     !  Buffer address
 6439       literal ILE2$K_LENGTH = 8;              !  Length of ILE2
 6440       literal ILE2$C_LENGTH = 8;              !  Length of ILE2
*
 6422       ! +
*quit
$
```

Locating Symbol Names for Macro32

```
$ type ile.mar
     $ILEDEF GLOBAL
     .end
$ mac ile
$ link/map/share ile
$ edit/edt/read ile.map
    1       <FF>
*'symbol'
   20                                                    ! Symbols By Name !
*.:.+17
   20                                                    ! Symbols By Name !
   21                                                    +-----------------+
   22
   23       Symbol                Value       Symbol    Value       Symbol
     Value
   24       ------                -----       ------    -----       ------
     -----
   25       ILE2$C_LENGTH         00000008

   26       ILE2$K_LENGTH         00000008
   27       ILE2$PS_BUFADDR       00000004
   28       ILE2$S_ILE2           00000008
   29       ILE2$W_CODE           00000002
   30       ILE2$W_LENGTH         00000000
   31       ILE3$C_LENGTH         0000000C
   32       ILE3$K_LENGTH         0000000C
   33       ILE3$PS_BUFADDR       00000004
   34       ILE3$PS_RETLEN_ADDR   00000008
   35       ILE3$S_ILE3           0000000C
   36       ILE3$W_CODE           00000002
   37       ILE3$W_LENGTH         00000000
*quit
$
```

Producing Programs

The **compiler/assembler** translates source into binary representations of data and machine code representations of data and stores these representations in an object module. Object modules have a default file type of **.OBJ**.

The **linker** produces an executable program (image). Executables have a default file type of **.EXE**. The linker is responsible for:

- Resolving references to global symbols.
- Assigning virtual address space and section attributes to code and data.
- Initializing data and code sections of the image body.
- Performing **JSR** to **BSR** optimizations.

The program is executed using the **$RUN** command. The **$RUN** command:

- Maps the program to virtual address space.
- Calls the program, allowing the Pager to perform demand loading.

Producing Programs Example

```
$ type simple.c
#include     <stdio.h>
#include     <stdlib.h>
int main(void)
{
     printf("Goodbye, Cruel World!\n");
     printf("Hello, OpenVMS!!!!!!!\n");
     return(EXIT_SUCCESS);
}
$ cc simple
$ dir simple

Directory _DKA300:[SILLE]

SIMPLE.C;1          SIMPLE.OBJ;1

Total of 2 files.
$ link simple
$ dir simple

Directory _DKA300:[SILLE]

SIMPLE.C;1          SIMPLE.EXE;1        SIMPLE.OBJ;1

Total of 3 files.
$ r simple
Goodbye, Cruel World!
Hello, OpenVMS!!!!!!!
$
```

Program Sections

- *Program Sections (PSECTs)* are used by the assembler to group sections of code and data by name and attributes. Program sections are generated, implicitly, by compilers.
- Understanding the grouping of code and data in an image, later on, will hinge on a basic understanding of psects. The following table provides an overview of psect attributes.

Attribute	Characteristic	Description
BYTE,WORD,LONG, QUAD,OCTA,PAGE	Alignment	Specifies alignment options available in .ALIGN directive
PIC/NOPIC	Position Independence	Unused on OpenVMS Alpha
OVR/CON Concatenated	Overlaid/OVR	Used to overlay C externs FORTRAN COMMON
REL/ABS Absolute addresses	Relocatable/ Absolute	Used for constants
GBL/LCL	Global/Local	If GBL, gather psects from all clusters into same section in image
SHR/NOSHR	Shareability	Applicable to shareable images
EXE/NOEXE	Executability	Used for potential warning of executing data and by /SECTION_BINDING Linker qualifier
WRT/NOWRT	Writability	Writable vs readonly at run-time
VEC/NOVEC	Protected Vectors	Contains change mode vectors to be protected for higher access modes
SOLITARY	Solitary	Group in separate image section
NOMOD/MOD	Unmodified	Set by compiler/assembler to identify data, contains uninitialized data.
COM RD USR/LIB	Miscellaneous	Either Reserved or special use

Program Section Example

```
$ type psect1.mar
     .psect   data   wrt,noexe
lotsa_data:
     .blkl    10000
init_data:
     .long    42

     .psect   code   nowrt,exe
     .entry   main,^m<r2>
     moval    lotsa_data,r2
     .rept    1000
     movl     init_data,(r2)+
     .endr
     movl     #ss$_normal,r0
     ret
     .end     main
$
$ mac psect1
$ link psect1
$ dir/size psect1.*;
Directory _DKA300:[SILLE]
PSECT1.EXE;1              140
PSECT1.MAR;2                1
PSECT1.OBJ;1               57
Total of 3 files, 198 blocks.
```

Program Section Example (continued)

```
$ type psect2.mar
    .psect  dzro_data    wrt,noexe
lotsa_data:
    .blkl   10000
    .psect  data  wrt,noexe
init_data:
    .long   42

    .psect  code  nowrt,exe
    .entry  main,^m<r2>
    moval   lotsa_data,r2
    .rept   1000
    movl    init_data,(r2)+
    .endr
    movl    #ss$_normal,r0
    ret
    .end    main
$ mac psect2
$ link psect2
$ dir/siz psect2.*;
Directory _DKA300:[SILLE]
PSECT2.EXE;1              39
PSECT2.MAR;1               1
PSECT2.OBJ;1              33
Total of 3 files, 73 blocks.
$
```

Producing Programs: Compiling/Assembling

Compilers and assemblers provide many common qualifiers to aid in debugging code and modify characteristics of the final image.

Some DEC C qualifiers

Qualifier	Function
/[NO]TRACEBACK	Include traceback handler, for debugging purposes
/[NO]DEBUG	Include local symbols for debugger
/[NO]OPTIMIZE	May be useful debugging
/LIST [= listing_file]	Generate listing file
/MACHINE	Generate machine code listing
/[NO]MEMBER_ALIGNMENT	Byte vs. padded alignment => Compatibility vs. performance
/SHOW [= [ALL,[NO]EXPANSION, [NO]HEADER,[NO]INCLUDE, [NO]SOURCE]]	Show various options in listing
/PSECT_MODEL=[NO]MULTILANGUAGE	Non-padded externs for compatability
/PREPROCESS_ONLY	Generate expanded listing without compiling
/POINTER_SIZE=[SHORT,LONG]	Defines size of pointer variables and enables use of #pragma to change size

Data Alignment Issues

AXP systems load from memory only with longword and quadword data types. The data path on an AXP system is 64 bits wide and memory fetches are performed on quadword aligned boundaries. Accesses to memory which are not aligned on natural boundaries, i.e. longword aligned for longword fetches and quadword aligned for quadword fetches, will impose significant performance penalties. True accesses to unaligned data wil cause an exception to be generated.

- As a side note, accessing data which is not naturally aligned will also impose performance penalties on most VAX systems.

XXXXXX	XXXXXX	XXXXXX	XXXXXX				
	XXXXXX	XXXXXX	XXXXXX	XXXXXX			
YYYYYY	YYYYYY	YYYYYY	YYYYYY	YYYYYY	YYYYYY	YYYYYY	YYYYYY
YYYYYY	YYYYYY	YYYYYY	YYYYYY	YYYYYY	YYYYYY		
						YYYYYY	YYYYYY

Illustration of Data accesses which are not naturally aligned.

- The VAX and DEC C compilers will align non-structure member data on natural boundaries as part of its optimization process.
- The DEC C compiler will naturally align structure members on natural boundaries, the VAX C compiler will not.
- If you need to have non-naturally aligned data members in structures, due to file formats, you can override this DEC C optimization by using the **/NOMEMBER_ALIGNMENT** qualifier. However, you will pay a performance penalty for doing so.

▲ **Note**

Compilers will typically generate multiple load instructions and use extract instructions to access the data. An unaligned load attempt will cause an exception, forcing the operating system to correct the fetch.

Non-Aligned Data Example

```
$
$ type align.c
#include     <stdlib.h>
struct       junk
{
    char     a;
    int      y;
    int      x;

};
main()
{
    int      sub(struct junk);

    struct   junk z ={0,0,0};
    int      i;

    for(i=0;i<100000000;i++)
    {
      z.x = 2*i - (i/2) + z.x;
      z.y +=1;
      z.a +=1;
      sub(z);
    }
    return(EXIT_SUCCESS);
}
```

Non-Aligned Data Example (continued)

```
$ type sub_align.c
struct junk
{
    char    a;
    int     y;
    int     x;
};
int  sub(struct junk temp)
{
    return(temp.x);
}
$
$ cc/nomember align
$ cc/nomember sub_align
$ link align,sub_align
$
```

Non-Aligned and Compiler Aligned Data Sample Runs

```
$
$ sh status
  Status on  24-AUG-1994 03:02:20.06     Elapsed CPU :    0 00:00:02.51
  Buff. I/O :      256    Cur. ws. :    2000    Open files :         0
  Dir. I/O :        50    Phys. Mem. :   928    Page Faults :     1707
$ ralign
$ sh status
  Status on  24-AUG-1994 03:03:23.11     Elapsed CPU :    0 00:01:02.35
  Buff. I/O :      269    Cur. ws. :    2000    Open files :         0
  Dir. I/O :        50    Phys. Mem. :   960    Page Faults :     1741
$
$
$
$
$ cc align
$ cc sub_align
$ link align,sub_align
$
$ sh status
  Status on  24-AUG-1994 03:04:45.38     Elapsed CPU :    0 00:01:03.48
  Buff. I/O :      371    Cur. ws. :    2000    Open files :         0
  Dir. I/O :        77    Phys. Mem. :   832    Page Faults :     2479
$ ralign
$ sh status
  Status on  24-AUG-1994 03:05:17.54     Elapsed CPU :    0 00:01:32.63
  Buff. I/O :      384    Cur. ws. :    2000    Open files :         0
  Dir. I/O :        77    Phys. Mem. :   928    Page Faults :     2517
$
```

Aligned Data Example

```
$ type align2.c
#include <stdlib.h>
struct junk
{
    int     y;
    int     x;
    char    a;

};
main()
{
    int     sub(struct junk);
    struct  junk z ={0,0,0};
    int     i;

    for(i=0;i<100000000;i++)
    {
      z.x = 2*i - (i/2) + z.x;
      z.y +=1;
      z.a +=1;
      sub(z);
    }
    return(EXIT_SUCCESS);
}
```

Aligned Data Example (continued)

```
$ type sub_align2.c
struct junk
{
    int     y;
    int     x;
    char    a;
};
int  sub(struct junk temp)
{
    return(temp.x);
}
$ cc/nomember align2
$ cc/nomember sub_align2
$ link align2,sub_align2
$
$ sh status
  Status on  24-AUG-1994 03:10:14.04     Elapsed CPU :   0 00:01:34.83
  Buff. I/O :      637    Cur. ws. :    2000    Open files :         0
  Dir. I/O :       122    Phys. Mem. :   896    Page Faults :     3801
$ r align2
$ sh status
  Status on  24-AUG-1994 03:10:47.72     Elapsed CPU :   0 00:02:03.99
  Buff. I/O :      650    Cur. ws. :    2000    Open files :         0
  Dir. I/O :       122    Phys. Mem. :   928    Page Faults :     3838
$
```

Using Traceback with Listings

The Condition Handling facility is defined as part of the OpenVMS calling procedure standard. It allows procedures to handle exceptions detected by hardware and software generated exceptions. The details of establishing condition handlers will be described later in the course.

- A default condition handler is enabled by the linker as part of the **EXE$IMGSTA** code, that gets called prior to entering the user program. This handler is the *traceback handler*. The traceback handler displays debug information allowing you to translate an error into a source code line.
- You may also activate the debugger through the DCL **$DEBUG** command.
- The traceback handler is automatically included unless you specifically link with the **/NOTRACE** qualifier.

Sample Program with Error

```
$ type prog.c
#include <stdlib.h>
#include <stdio.h>
#include     <string.h>

char *sub(char *);

int  main(void)
{
     char     *name = "B. B. King";
     printf("%s\n",sub(name));
     return(EXIT_SUCCESS);
}
$
$ typ sub.c
#include     <ctype.h>
```

Using Traceback with Listings (continued)

```
char *sub(char *str)
{
    char    *save = str;
    while(*str != '\0')
    {
      *str = toupper(*str);
      str++;
    }
    return(save);
}
$
```

Using Traceback with Listings (continued)

```
$ cc/list prog
$ cc/list sub
$ link /map prog,sub
$ run prog
%SYSTEM-F-ACCVIO, access violation, reason mask=00, virtual
    address=0000000000010020, PC=0000000000020168, PS=0000001B
%TRACE-F-TRACEBACK, symbolic stack dump follows
  image    module    routine    line      rel PC           abs PC
 PROG  SUB  sub                  288 0000000000000068 0000000000020168
 PROG  PROG  main                2545 00000000000000BC 00000000000200BC
 PROG  PROG  __main                 0 0000000000000054 0000000000020054
                                    0 FFFFFFFF81C62914 FFFFFFFF81C62914
$ type sub.lis
 Source Listing      17-JUL-1996 21:29:45    DEC C V5.2-003          Page 1
                                                                  17-JUL-1996
    16:48:38    _DKA300:[SILLE]SUB.C;2

          1 #include     <ctype.h>
        282
        283 char   *sub(char *str)
        284 {
        285 char   *save = str;
        286 while(*str != '\0')
        287 {
        288         *str = toupper(*str);
        289         str++;
        290 }
        291 return(save);
        292 }
...

$
```

Using Traceback with Listings (continued)

```
$ type prog.lis
 Source Listing                 17-JUL-1996 21:29:27    DEC C V5.2-003
    Page 1
                                17-JUL-1996 16:46:23    _DKA300:[SILLE]PROG.C;4


         1 #include <stdlib.h>
      1467 #include <stdio.h>
      2141 #include     <string.h>
      2539
      2540 char   *sub(char *);
      2541
      2542 int    main(void)
      2543 {
      2544  char   *name = "B. B. King ";
      2545  printf("%s\n",sub(name));
      2546  return(EXIT_SUCCESS);
      2547 }


Command Line
------- ----

CC/ANSI_ALIAS/ANALYSIS_DATA/ASSUME=(ACCURACY_SENSITIVE,ALIGNED_OBJECTS,NOWRITABL
    E_STRING_LITERALS)
...
$
```

Using Maps and Machine Code Listings

Production level code is normally linked **/NOTRACE**, since it is assumed to be debugged. Note that returning error status codes will cause traceback information to be displayed by the traceback handler.

- Images may be set up to run with elevated privileges or to be memory resident. These images must be linked **/NOTRACE**.
- When an image is linked without traceback, and a hardware detected exception (such as an access violation or an arithmetic exception) is generated and not handled, the last chance exception vector will be located, and the procedure **EXE$CATCH_ALL** will be called. The *catch all* handler will display raw information about the exception which may require map files and machine listings to analyze.

```
$ link/notrace prog,sub
$ run prog
%SYSTEM-F-ACCVIO, access violation, reason mask=00, virtual
    address=0000000000010020, PC=0000000000020168, PS=0000001B
  Improperly handled condition, image exit forced.
    Signal arguments:   Number = 0000000000000005
                        Name   = 000000000000000C
                                 0000000000010000
                                 0000000000010020
                                 0000000000020168
                                 000000000000001B
    Register dump:
    R0  = 0000000000000042  R1  = 000000007B5F0040  R2  = 00000000000100A0
    R3  = 0000000000010020  R4  = 0000000000010020  R5  = 000000007FFCF938
    R6  = 000000007FFAC9F0  R7  = 000000007FFAC9F0  R8  = 000000007FFAC208
    R9  = 000000007FFAC410  R10 = 000000007FFAD238  R11 = 000000007FFCE3E0
    R12 = 0000000000000000  R13 = 000000007B0A4140  R14 = 0000000000000000
    R15 = 000000007B0A37A0  R16 = 6C45206563757242  R17 = 0000000000000042
    R18 = 0000000000000043  R19 = 0000000000010021  R20 = 000000007B00FB68
    R21 = 000000007B00F590  R22 = FFFFFFFF805CA804  R23 = 000000007B00F620
    R24 = 0000000000000000  R25 = 0000000000000001  R26 = 0000000000020158
    R27 = 000000007B5BF138  R28 = 000000000000373B  R29 = 000000007B00FAE0
    SP  = 000000007B00FAE0  PC  = 0000000000020168  PS  = 200000000000001B
$
```

Using Maps and Machine Code Listings (continued)

```
$ type prog.map

17-JUL-1996 21:29        Linker A11-36                    Page    1

                                +-------------------------+
                                ! Object Module Synopsis !
                                +-------------------------+

Module Name     Ident           Bytes      File                    Creation Date      Creator
-----------     -----           -----      -----                   -------------

PROG            V1.0            420 _DKA300:[SILLE]PROG.OBJ;11     17-JUL-1996
SUB             V1.0            212 _DKA300:[SILLE]SUB.OBJ;5       17-JUL-1996

                                +---------------------------+
                                ! Program Section Synopsis !
                                +---------------------------+

Psect Name      Module Name      Base     End          Length          Align
     Attributes

----------      -----------      ----     ---          ------          -----

$LINK$                           00010000 000100CF 000000D0 (       208.) OCTA  4
     NOPIC,CON,REL,LCL,NOSHR,NOEXE,NOWRT,NOVEC,  MOD
                 PROG            00010000 0001009F 000000A0 (       160.) OCTA  4
                 SUB             000100A0 000100CF 00000030 (        48.) OCTA  4
$READONLY$                       000100D0 000100DF 00000010 (        16.) OCTA  4   PIC,CON,REL,LCL,
     SHR,NOEXE,NOWRT,NOVEC,  MOD
                 PROG            000100D0 000100DF 00000010 (        16.) OCTA  4
$CODE$                           00020000 000201A3 000001A4 (       420.) OCTA  4   PIC,CON,REL,LCL,
     SHR,  EXE,NOWRT,NOVEC,  MOD
                 PROG            00020000 000200F3 000000F4 (       244.) OCTA  4
                 SUB             00020100 000201A3 000000A4 (       164.) OCTA  4

                                +-----------------+
                                ! Symbols By Name !
                                +-----------------+
```

Using Maps and Machine Code Listings (continued)

Symbol	Value	Symbol	Value	Symbol
MAIN	00010000-R			
SUB	000100A0-R			
__MAIN	00010060-R			

Key for special characters above:

```
+--------------------+
! *  - Undefined     !
! A  - Alias Name    !
! I  - Internal Name !
! U  - Universal     !
! R  - Relocatable   !
! X  - External      !
! WK - Weak          !
+--------------------+
```

Using Maps and Machine Code Listings (continued)

```
_DKA300:[SILLE]PROG.EXE;6                                  17-JUL-1996 21:29
Linker A11-36                  Page     2

                                        +----------------+
                                        ! Image Synopsis !
                                        +----------------+

Virtual memory allocated:                00010000 0003FFFF 00030000 (196608. bytes,
384. pages)
Stack size:                                    20. pages
Image header virtual block limits:              1.        2. (    2. blocks)
Image binary virtual block limits:              3.        5. (    3. blocks)
Image name and identification:           PROG V1.0
Number of files:                                6.
Number of modules:                              4.
Number of program sections:                     9.
Number of global symbols:                    2268.
Number of image sections:                      14.
User transfer address:                   00010060
Debugger transfer address:               00000340
Number of code references to shareable images:  5.
Image type:                              EXECUTABLE.
Map format:                              DEFAULT in file _DKA300:[SILLE]PROG.MAP;2
Estimated map length:                    158. blocks
                                      +---------------------+
                                      ! Link Run Statistics !
                                      +---------------------+

Performance Indicators                  Page Faults      CPU Time        Elapsed
----------------------                  -----------      --------        -------
    Command processing:                          22     00:00:00.05     00:00:0
    Pass 1:                                     112     00:00:00.32     00:00:0
    Allocation/Relocation:                        3     00:00:00.03     00:00:0
    Pass 2:                                       4     00:00:00.04     00:00:0
    Map data after object module synopsis:        3     00:00:00.01     00:00:0
    Symbol table output:                          0     00:00:00.00     00:00:0
Total run values:                               144     00:00:00.45     00:00:0

Using a working set limited to 6800 pages and 1783 pages of data storage (excluding image)

Total number object records read (both passes):   360
    of which 0 were in libraries and 13 were DEBUG data records containing 831 bytes
339 bytes of DEBUG data were written,starting at VBN 6 with 1 blocks allocated

Number of modules extracted explicitly              = 0
    with 0 extracted to resolve undefined symbols

1 library searches were for symbols not in the library searched

A total of 0 global symbol table records was written

LINK/MAP PROG,SUB
$
```

Using Maps and Machine Code Listings (continued)

```
$ cc/list/mach prog
$ cc/list/mach sub
$
$ type prog

Source Listing                17-JUL-1996 21:32:46    D
                                                      17-JUL-1996 16:46:23    _

          1 #include <stdlib.h>
       1467 #include <stdio.h>
       2141 #include      <string.h>
       2539
       2540 char   *sub(char *);
       2541
       2542 int    main(void)
       2543 {
       2544        char    *name = "B. B. King ";
       2545        printf("%s\n",sub(name));
       2546        return(EXIT_SUCCESS);
       2547 }

                          Machine Code Listing          17-JUL-1996 21:32:46    DE
                          __MAIN                        17-JUL-1996 16:46:23    _D

                          .PSECT  $CODE$, OCTA, PIC, CON, REL, LCL, SHR,-
                                  EXE, NORD, NOWRT
             0000       __MAIN::
23DEFFC0     0000               LDA     SP, -64(SP)                  ; SP, -64(SP)
47E13419     0004               MOV     9, R25                       ; 9, R25
B77E0000     0008               STQ     R27, (SP)                    ; R27, (SP)
B75E0020     000C               STQ     R26, 32(SP)                  ; R26, 32(SP)
B45E0028     0010               STQ     R2, 40(SP)                   ; R2, 40(SP)
B7BE0030     0014               STQ     FP, 48(SP)                   ; FP, 48(SP)
47FE041D     0018               MOV     SP, FP                       ; SP, FP
A75B0030     001C               LDQ     R26, 48(R27)                 ; R26, 48(R27)
23DEFFE0     0020               LDA     SP, -32(SP)                  ; SP, -32(SP)
43A31001     0024               ADDL    FP, 24, R1                   ; FP, 24, R1
B7FE0000     0028               STQ     R31, (SP)                    ; R31, (SP)
47FB0402     002C               MOV     R27, R2                      ; R27, R2
43A21016     0030               ADDL    FP, 16, R22                  ; FP, 16, R22
A77B0038     0034               LDQ     R27, 56(R27)                 ; R27, 56(R27)
43A11017     0038               ADDL    FP, 8, R23                   ; FP, 8, R23
B43E0000     003C               STQ     R1, (SP)                     ; R1, (SP)
B6DE0008     0040               STQ     R22, 8(SP)                   ; R22, 8(SP)
B6FE0010     0044               STQ     R23, 16(SP)                  ; R23, 16(SP)
6B5A4000     0048               JSR     R26, DECC$MAIN               ; R26, R26
```

Using Maps and Machine Code Listings (continued)

```
2362FFA0    004C            LDA     R27, -96(R2)            ; R27, -96(R2)
D340000B    0050            BSR     R26, MAIN               ; R26, MAIN
A7420020    0054            LDQ     R26, 32(R2)             ; R26, 32(R2)
A7620028    0058            LDQ     R27, 40(R2)             ; R27, 40(R2)
47E00410    005C            MOV     R0, R16                 ; R0, R16
47E03419    0060            MOV     1, R25                  ; 1, R25
6B5A4000    0064            JSR     R26, DECC$EXIT          ; R26, R26
47FD041E    0068            MOV     FP, SP                  ; FP, SP
A75D0020    006C            LDQ     R26, 32(FP)             ; R26, 32(FP)
A45D0028    0070            LDQ     R2, 40(FP)              ; R2, 40(FP)
A7BD0030    0074            LDQ     FP, 48(FP)              ; FP, 48(FP)
23DE0040    0078            LDA     SP, 64(SP)              ; SP, 64(SP)
6BFA8001    007C            RET     R26                     ; R26

Routine Size: 128 bytes,    Routine Base: $CODE$ + 0000

            0080    MAIN::
                        ; 002542
23DEFFD0    0080            LDA     SP, -48(SP)             ; SP, -48(SP)
47E03419    0084            MOV     1, R25                  ; 1, R25
                        ; 002545
B77E0000    0088            STQ     R27, (SP)               ; R27, (SP)
                        ; 002542
B75E0008    008C            STQ     R26, 8(SP)              ; R26, 8(SP)
B45E0010    0090            STQ     R2, 16(SP)              ; R2, 16(SP)
B47E0018    0094            STQ     R3, 24(SP)              ; R3, 24(SP)
B7BE0020    0098            STQ     FP, 32(SP)              ; FP, 32(SP)
47FE041D    009C            MOV     SP, FP                  ; SP, FP
A75B0050    00A0            LDQ     R26, 80(R27)            ; R26, 80(R27)
                        ; 002545
43661001    00A4            ADDL    R27, 48, R1             ; R27, 48, R1
47FB0402    00A8            MOV     R27, R2                 ; R27, R2
                        ; 002542
43641010    00AC            ADDL    R27, 32, R16            ; R27, 32, R16
                        ; 002544
A77B0058    00B0            LDQ     R27, 88(R27)            ; R27, 88(R27)
                        ; 002545
47E10403    00B4            MOV     R1, R3                  ; R1, R3
6B5A4000    00B8            JSR     R26, SUB                ; R26, R26
A7420040    00BC            LDQ     R26, 64(R2)             ; R26, 64(R2)
A7620048    00C0            LDQ     R27, 72(R2)             ; R27, 72(R2)
43E00011    00C4            SEXTL   R0, R17                 ; R0, R17
```

Using Maps and Machine Code Listings (continued)

```
Routine Size: 116 bytes,    Routine Base: $CODE$ + 0080

                            .PSECT  $LINK$, OCTA, NOPIC, CON, REL, LCL,-
                                    NOSHR, NOEXE, RD, NOWRT

                0000        ; Stack-Frame invocation descriptor
                            Entry point:          MAIN
                            Registers used:       R0-R3, R16-R27, FP, F0-F1, F10-F30
                            Registers saved:      R2-R3, FP
                            Fixed Stack Size:     48

42202442        0020        .ASCII  \B. B. King \<0>\<0>
694A2024        0024
0020676E        0028
000A7325        0030        .ASCII  \%s\<10><0>\<0>
                0040        .LINKAGE  DECC$GXPRINTF
                0050        .LINKAGE  SUB

                0060        ; Stack-Frame invocation descriptor
                            Entry point:          __MAIN
                            Registers used:       R0-R2, R16-R27, FP, F0-F1, F10-F30
                            Registers saved:      R2, FP
                            Fixed Stack Size:     64

                0080        .LINKAGE  DECC$EXIT
                0090        .LINKAGE  DECC$MAIN

                            .PSECT  $READONLY$, OCTA, PIC, CON, REL, LCL,-
                                    SHR, NOEXE, RD, NOWRT
                0000    _SIG_EMPTY_SET:
    00          0000            .BYTE   X^00, -
    00          0001                    X^00, -
    00          0002                    X^00, -
    00          0003                    X^00
    00          0004            .BYTE   X^00, -
    00          0005                    X^00, -
    00          0006                    X^00, -
    00          0007                    X^00
                0008    _SIG_FULL_SET:
    FF          0008            .BYTE   X^FF, -
    FF          0009                    X^FF, -
    FF          000A                    X^FF, -
    FF          000B                    X^FF
    FF          000C            .BYTE   X^FF, -
```

Using Maps and Machine Code Listings (continued)

```
Machine Code Listing            17-JUL-1996 21:32:46    DEC C V5.2-003                  Page 4

                                MAIN                                    17-JUL-1996 16:46:23
_DKA300:[SILLE]PROG.C;4

      FF     000D                    X^FF, -
      FF     000E                    X^FF, -
      FF     000F                    X^FF
Command Line
------- ----

CC/ANSI_ALIAS/ANALYSIS_DATA/ASSUME=(ACCURACY_SENSITIVE,ALIGNED_OBJECTS,NOWRITABLE_STRING_LITER
ALS)
/COMMENTS=SPACE/DIAGNOSTICS/DEBUG=(TRACEBACK)/ENDIAN=LITTLE/EXTERN_MODEL=RELAXED_REFDEF
/FLOAT=(G_FLOAT)/GRANULARITY=QUADWORD/INSTRUCTION_SET=FLOATING_POINT/LINE_CONTROL
/L_DOUBLE_SIZE=128/LINE_DIRECTIVES/LIST/MACHINE_CODE/MEMBER_ALIGNMENT/NAMES=UPPERCASE
/NESTED_INCLUDE_DIRECTORY=INCLUDE_FILE/OBJECT/OPTIMIZE=(LEVEL=4,TUNE=GENERIC,UNROLL=0)
/PREFIX=(ANSI_C89_ENTRIES)/PSECT_MODEL=NOMULTILANGUAGE/ROUNDING_MODE=NEAREST
/SHOW=(HEADER,SOURCE)/SIGNED_CHAR/STANDARD=RELAXED_ANSI89/STACK_CHECK/REENTRANCY=TOLERANT
/WARNINGS
$
$
$
$ type sub
                                Source Listing                  17-JUL-1996 21:32:55    DEC C
V5.2-003                  Page 1
                                                                17-JUL-1996 16:48:38
_DKA300:[SILLE]SUB.C;2

             1 #include      <ctype.h>
           282
           283 char   *sub(char *str)
           284 {
           285        char    *save = str;
           286        while(*str != '\0')
           287        {
           288                *str = toupper(*str);
           289                str++;
           290        }
           291        return(save);
           292 }

Machine Code Listing            17-JUL-1996 21:32:55    DEC C V5.2-003                  Page 2

SUB                             17-JUL-1996 16:48:38    _DKA300:[SILLE]SUB.C;2
```

Using Maps and Machine Code Listings (continued)

```
                        .PSECT  $CODE$, OCTA, PIC, CON, REL, LCL, SHR,-
                                EXE, NORD, NOWRT
                0000    SUB::
                            ; 000283
23DEFFD0        0000            LDA     SP, -48(SP)             ; SP, -48(SP)
B77E0000        0004            STQ     R27, (SP)               ; R27, (SP)
B75E0008        0008            STQ     R26, 8(SP)              ; R26, 8(SP)
B45E0010        000C            STQ     R2, 16(SP)              ; R2, 16(SP)
B47E0018        0010            STQ     R3, 24(SP)              ; R3, 24(SP)
B49E0020        0014            STQ     R4, 32(SP)              ; R4, 32(SP)
B7BE0028        0018            STQ     FP, 40(SP)              ; FP, 40(SP)
47FE041D        001C            MOV     SP, FP                  ; SP, FP
47FB0402        0020            MOV     R27, R2                 ; R27, R2
2C300000        0024            LDQ_U   R1, (R16)               ; R1, (R16)
                            ; 000286
47F00404        0028            MOV     R16, save               ; R16, R4
                            ; 000285
47F00403        002C            MOV     R16, str                ; R16, R3
                            ; 000283
483000C1        0030            EXTBL   R1, R16, R1             ; R1, R16, R1
                            ; 000286
E4200012        0034            BEQ     R1, L$3                 ; R1, L$3
                0038    L$4:
2E430000        0038            LDQ_U   R18, (R3)               ; R18, (R3)
                            ; 000288
22630001        003C            LDA     R19, 1(R3)              ; R19, 1(R3)
A7420020        0040            LDQ     R26, 32(R2)             ; R26, 32(R2)
47E03419        0044            MOV     1, R25                  ; 1, R25
A7620028        0048            LDQ     R27, 40(R2)             ; R27, 40(R2)
4A530F50        004C            EXTQH   R18, R19, R16           ; R18, R19, R16
4A071790        0050            SRA     R16, 56, R16            ; R16, 56, R16
6B5A4000        0054            JSR     R26, DECC$TOUPPER       ; R26, R26
2E030000        0058            LDQ_U   R16, (R3)               ; R16, (R3)
48030171        005C            INSBL   R0, R3, R17             ; R0, R3, R17
4A030050        0060            MSKBL   R16, R3, R16            ; R16, R3, R16
46110410        0064            BIS     R16, R17, R16           ; R16, R17, R16
3E030000        0068            STQ_U   R16, (R3)               ; R16, (R3)
40603003        006C            ADDL    str, 1, str             ; R3, 1, R3
                            ; 000289
447F0803        0070            XOR     R3, R31, R3             ; R3, R31, R3
2E430000        0074            LDQ_U   R18, (R3)               ; R18, (R3)
                            ; 000286
4A4300D2        0078            EXTBL   R18, R3, R18            ; R18, R3, R18
F65FFFEE        007C            BNE     R18, L$4                ; R18, L$4
                0080    L$3:
                            ; 000290
```

Using Maps and Machine Code Listings (continued)

```
47FD041E     0080              MOV     FP, SP                          ; FP, SP
                             ; 000291
A75D0008     0084              LDQ     R26, 8(FP)                      ; R26, 8(FP)
47E40400     0088              MOV     save, R0                        ; R4, R0
A45D0010     008C              LDQ     R2, 16(FP)                      ; R2, 16(FP)
A47D0018     0090              LDQ     R3, 24(FP)                      ; R3, 24(FP)
A49D0020     0094              LDQ     R4, 32(FP)                      ; R4, 32(FP)
A7BD0028     0098              LDQ     FP, 40(FP)                      ; FP, 40(FP)
23DE0030     009C              LDA     SP, 48(SP)                      ; SP, 48(SP)
6BFA8001     00A0              RET     R26                             ; R26
Routine Size: 164 bytes,    Routine Base: $CODE$ + 0000


                               .PSECT  $LINK$, OCTA, NOPIC, CON, REL, LCL,-
                                       NOSHR, NOEXE, RD, NOWRT

             0000              ; Stack-Frame invocation descriptor
                               Entry point:          SUB
                               Registers used:        R0-R4, R16-R27, FP, F0-F1, F10-F30
                               Registers saved:       R2-R4, FP

                                 Machine Code Listing          17-JUL-1996 21:32:55    DEC C
V5.2-003                   Page 3
                                 SUB                           17-JUL-1996 16:48:38
_DKA300:[SILLE]SUB.C;2

                               Fixed Stack Size:      48

             0020              .LINKAGE  DECC$TOUPPER


Command Line
------- ----

CC/ANSI_ALIAS/ANALYSIS_DATA/ASSUME=(ACCURACY_SENSITIVE,ALIGNED_OBJECTS,NOWRITABLE_STRING_LITER
ALS)
/COMMENTS=SPACE/DIAGNOSTICS/DEBUG=(TRACEBACK)/ENDIAN=LITTLE/EXTERN_MODEL=RELAXED_REFDEF
/FLOAT=(G_FLOAT)/GRANULARITY=QUADWORD/INSTRUCTION_SET=FLOATING_POINT/LINE_CONTROL
/L_DOUBLE_SIZE=128/LINE_DIRECTIVES/LIST/MACHINE_CODE/MEMBER_ALIGNMENT/NAMES=UPPERCASE
/NESTED_INCLUDE_DIRECTORY=INCLUDE_FILE/OBJECT/OPTIMIZE=(LEVEL=4,TUNE=GENERIC,UNROLL=0)
/PREFIX=(ANSI_C89_ENTRIES)/PSECT_MODEL=NOMULTILANGUAGE/ROUNDING_MODE=NEAREST
/SHOW=(HEADER,SOURCE)/SIGNED_CHAR/STANDARD=RELAXED_ANSI89/STACK_CHECK/REENTRANCY=TOLERANT
/WARNINGS
$
$
```

Debugging

The OpenVMS operating system provides a symbolic debugger, that understands the subtleties of the language under which a program was produced.

- The symbolic debugger provides a windows-oriented interface by default on a windows-based system. It also provides an extensive command line interface including extensive help.
- If you are planning to use the debugger you should compile with the **/DEBUG** and **/NOOPTIMIZE** qualifiers and link with the **/DEBUG** option. You will automatically enter the debugger when you run the program.

Basic Debugger commands

Note
Many other commands exist to tailor environment

Command	Function
CALL	Calls a routine.
EXAMINE	Examine a location supports qualifiers to define size and type of examine (/ASCII /BINARY /BYTE /HEX /LONG...)
GO	Run program to breakpoint, watchpoint, or completion
STEP	Single step through instructions
STEP/INTO	Step into procedure
SET BREAK	Set instruction/statement to stop execution
SET TRACE	Trace changes to a variable
SET WATCH	Stop execution when a variable changes
address-specifiers	%LINE module\routine\variable
SHOW CALLS	Show calls made to get to this point in the program

Sample Debug Session Using Command Interface

```
$
$ DEFINE DBG$DECW$DISPLAY""
$
$ cc/deb/noopt prog
$ cc/deb/noopt sub
$ link/deb prog,sub
$
$
$ run prog
        OpenVMS Alpha DEBUG Version V7.0-000
%DEBUG-I-INITIAL, Language: C, Module: PROG
%DEBUG-I-NOTATMAIN, Type GO to reach MAIN program
DBG> set mode screen




- OUT -output-----------------------------------------------------------------




- PROMPT -error-program-prompt-----------------------------------------------

%DEBUG-W-SCRNOSRCLIN, No source line for address: 0000000000020000
```

Sample Debug Session Using Command Interface (continued)

```
  2538: #endif  /* __STRING_LOADED */
  2539:
  2540: char    *sub(char *);
  2541:
  2542: int     main(void)
  2543: {
->2544:         char    *name = "B. B. King ";
  2545:         printf("%s\n",sub(name));
  2546:         return(EXIT_SUCCESS);
  2547: }
- OUT -output ------------------------------------------------------------
break at routine PROG\main
  2544:         char    *name = "B. B. King ";




- PROMPT -error-program-prompt-------------------------------------------
%DEBUG-W-SCRNOSRCLIN, No source line for address: 0000000000020000
DBG> go
```

Sample Debug Session Using Command Interface (continued)

```
  2538: #endif  /* __STRING_LOADED */
  2539:
  2540: char    *sub(char *);
  2541:
  2542: int     main(void)
  2543: {
  2544:         char    *name = "B. B. King ";
->2545:         printf("%s\n",sub(name));
  2546:         return(EXIT_SUCCESS);
  2547: }
- OUT -output-----------------------------------------------------------
break at routine PROG\main
  2544:         char    *name = "B. B. King";
stepped to PROG\main\%LINE 2545
  2545:         printf("%s\n",sub(name));



- PROMPT -error-program-prompt-----------------------------------------
DBG> go
```

Sample Debug Session Using Command Interface (continued)

```
DBG> step


   281: #endif /* __CTYPE_LOADED */
   282:
   283: char    *sub(char *str)
   284: {
-> 285:         char    *save = str;
   286:         while(*str != '\0')
   287:         {
   288:                 *str = toupper(*str);
   289:                 str++;
   290:         }
- OUT -output-----------------------------------------------------------
break at routine PROG\main
  2544:         char    *name = "B. B. King ";
stepped to PROG\main\%LINE 2545
  2545:         printf("%s\n",sub(name));
stepped to SUB\sub\%LINE 285
   285:         char    *save = str;

- PROMPT -error-program-prompt-----------------------------------------
DBG> step/into
%DEBUG-I-DYNMODSET, setting module SUB
DBG>
```

Sample Debug Session Using Command Interface (continued)

```
   282:
   283: char    *sub(char *str)
   284: {
   285:         char    *save = str;
-> 286:         while(*str != '\0')
   287:         {
   288:                 *str = toupper(*str);
   289:                 str++;
   290:         }
   291:         return(save);
- OUT -output-----------------------------------------------------------------
  2544:         char    *name = "B. B. King ";
stepped to PROG\main\%LINE 2545
  2545:         printf("%s\n",sub(name));
stepped to SUB\sub\%LINE 285
   285:         char    *save = str;
stepped to SUB\sub\%LINE 286
   286:         while(*str != '\0')
- PROMPT -error-program-prompt------------------------------------------------
%DEBUG-I-DYNMODSET, setting module SUB
```

Sample Debug Session Using Command Interface (continued)

```
DBG> step
DBG>

   283: char    *sub(char *str)
   284: {
   285:         char    *save = str;
   286:         while(*str != '\0')
   287:         {
-> 288:                 *str = toupper(*str);
   289:                 str++;
   290:         }
   291:         return(save);
   292: }
- OUT -output-----------------------------------------------------------
  2545:         printf("%s\n",sub(name));
stepped to SUB\sub\%LINE 285
   285:         char    *save = str;
stepped to SUB\sub\%LINE 286
   286:         while(*str != '\0')
stepped to SUB\sub\%LINE 288
   288:                 *str = toupper(*str);
- PROMPT -error-program-prompt-----------------------------------------
DBG> step
DBG> step
DBG>
```

Sample Debug Session Using Command Interface (continued)

```
   283: char    *sub(char *str)
   284: {
   285:         char    *save = str;
   286:         while(*str != '\0')
   287:         {
-> 288:                 *str = toupper(*str);
   289:                 str++;
   290:         }
   291:         return(save);
   292: }
- OUT -output-----------------------------------------------------------------
   285:         char    *save = str;
stepped to SUB\sub\%LINE 286
   286:         while(*str != '\0')
stepped to SUB\sub\%LINE 288
   288:                 *str = toupper(*str);
break on unhandled exception at SUB\sub\%LINE 288+56
   288:                 *str = toupper(*str);
- PROMPT -error-program-prompt-----------------------------------------------
%SYSTEM-F-ACCVIO, access violation, reason mask=00, virtual address=000000000001
0028, PC=00000000000201B4, PS=0000001B
DBG> exit
$
```

Object Modules

As mentioned earlier compilers produce object modules to describe the binary representation of data and machine level code.

- The object module is used to feed information about psects, global symbols, debug symbols, code, and data to the linker.
- Included with the data and code are several records, describing the object module:

Record type	Contents/Function
Main Module Header (EMH)	Module name, Version, Creation date and time, Language name
Global Symbol Directory (EGSD)	Used to build global symbol table and psect table
Text Information and Relocation (ETIR)	Used by linker to build binary content of image
End-of-Module (EOM)	Describes completion status of compile/assemble
Debugger Information (EDBG)	Used to pass local symbols to debugger
Traceback Information (TBT)	Provides traceback information

- These records may be viewed using **`$ANALYZE/OBJECT`**.

Linking

The linker produces an image file made up of an image header, image body, and a fixup vector.

- The image header describes where the program will be called after the image has been activated.
- Image activation is performed by reading image section descriptors and mapping the image into process virtual address space.
- Image sections are grouped based on major attributes of psects.
- Image sections have characteristics associated with them such as writability, demand zero pages, copy-on-reference, and executability.

Psect attributes affecting image sections for executables

Psect attributes	Attributes set for image section
NOWRT NOEXE NOVEC MOD	none
WRT NOEXE NOVEC MOD	WRT CRF
NOWRT EXE NOVEC MOD	EXE
WRT EXE NOVEC MOD	WRT CRF EXE
NOWRT NOEXE VEC MOD	VECTOR PROTECT
WRT NOEXE VEC MOD	WRT VECTOR PROTECT CRF
NOWRT EXE VEC MOD	VECTOR PROTECT EXE
WRT EXE VEC MOD	WRT VECTOR PROTECT EXE
NOWRT NOEXE NOVEC NOMOD	DZRO
WRT NOEXE NOVEC NOMOD	WRT DZRO

- Image Section layout and attributes may be examined using **`$ANALYZE/IMAGE`**.

$Analyze/Image Example

```
$ ana/image prog

Analyze Image                                  17-JUL-1996 21:53:21.74    Page 1
_DKA300:[SILLE]PROG.EXE;9
ANALYZ A05-19

This is an OpenVMS Alpha image file

IMAGE HEADER

    Fixed Header Information

      image format major id: 3, minor id: 0
      header block count: 2
      image type: executable (EIHD$K_EXE)
      I/O channel count: default
      I/O pagelet count: default
      Symbol Vector Virtual Address: %X'00000000'
      Symbol Vector Size: 0 bytes
      Virtual Memory Block Size: 65536 (BPAGE = 16)
      Fixup Section Virtual Address: %X'00030000'
      linker flags:
            (0)  EIHD$V_LNKDEBUG  1
            (1)  EIHD$V_LNKNOTFR  0
            (2)  EIHD$V_NOP0BUFS  0
            (3)  EIHD$V_PICIMG    1
            (4)  EIHD$V_P0IMAGE   0
```

$Analyze/Image Example (continued)

```
(5)  EIHD$V_DBGDMT     1
        (6)  EIHD$V_INISHR     0
        (7)  EIHD$V_XLATED     0
        (8)  EIHD$V_BIND_CODE  0
        (9)  EIHD$V_BIND_DATA  0

Image Activation Information

  first transfer address:  %X'00000340'
  second transfer address: %X'00010050'
  third transfer address:  %X'00000000'

Global Symbol Table & Debug Symbol Table Information

  debug symbol table VBN:  6, byte count: 2048
  global symbol table VBN: 11, record count: 5
  debug module/psect table VBN: 10, byte count: 64

Image Identification Information

  image name: "PROG"
  image file identification: "V1.0"
  image file build identification: ""
  link date/time: 17-JUL-1996 21:45:38.88
  linker identification: "A11-36"

Patch Information

  There are no patches at this time.
```

$Analyze/Image Example (continued)

```
    Image Section Descriptors (ISD)

Analyze Image                                  17-JUL-1996 21:53:22.03   Page 2
_DKA300:[SILLE]PROG.EXE;9 ANALYZ A05-19
      1)  image section descriptor (36 bytes)
            byte count: 512
            base virtual address: %X'00010000' (P0 space)
            page fault cluster size: default
            ISD flags:
                  (0)  EISD$V_GBL         0
                  (1)  EISD$V_CRF         1
                  (2)  EISD$V_DZRO        0
                  (3)  EISD$V_WRT         1
                  (4)  EISD$V_INITALCOD   0
                  (5)  EISD$V_BASED       0
                  (6)  EISD$V_FIXUPVEC    0
                  (7)  EISD$V_RESIDENT    0
                  (8)  EISD$V_VECTOR      0
                  (9)  EISD$V_PROTECT     0
                  (10) EISD$V_LASTCLU     1
                  (11) EISD$V_EXE         0
                  (12) EISD$V_NONSHRADR   1
            section type: EISD$K_NORMAL
            base VBN: 3
      2)  image section descriptor (36 bytes)
            byte count: 512
            base virtual address: %X'00020000' (P0 space)
            page fault cluster size: default
```

$Analyze/Image Example (continued)

```
ISD flags:
                    (0)  EISD$V_GBL        0
                    (1)  EISD$V_CRF        0
                    (2)  EISD$V_DZRO       0
                    (3)  EISD$V_WRT        0
                    (4)  EISD$V_INITALCOD  0
                    (5)  EISD$V_BASED      0
                    (6)  EISD$V_FIXUPVEC   0
                    (7)  EISD$V_RESIDENT   0
                    (8)  EISD$V_VECTOR     0
                    (9)  EISD$V_PROTECT    0
                    (10) EISD$V_LASTCLU    1
                    (11) EISD$V_EXE        1                    (12)
  EISD$V_NONSHRADR 0
          section type: EISD$K_NORMAL
          base VBN: 4
     3)  image section descriptor (36 bytes)
          byte count: 512
          base virtual address: %X'00030000' (P0 space)
          page fault cluster size: default
          ISD flags:
                (0)  EISD$V_GBL        0
                (1)  EISD$V_CRF        1
                (2)  EISD$V_DZRO       0
                (3)  EISD$V_WRT        1
                (4)  EISD$V_INITALCOD  0
                (5)  EISD$V_BASED      0
                (6)  EISD$V_FIXUPVEC   1
                (7)  EISD$V_RESIDENT   0
```

$Analyze/Image Example (continued)

```
Analyze Image                                   17-JUL-1996 21:53:22.11   Page 3
_DKA300:[SILLE]PROG.EXE;9
ANALYZ A05-19

                (8)  EISD$V_VECTOR     0
                (9)  EISD$V_PROTECT    0
                (10) EISD$V_LASTCLU    0
                (11) EISD$V_EXE        0
                (12) EISD$V_NONSHRADR  0
            section type: EISD$K_NORMAL
            base VBN: 5

        4)  image section descriptor (36 bytes)
            byte count: 10240
            base virtual address: %X'7FFF0000' (P1 space)
            page fault cluster size: default
        ISD flags:
                (0)  EISD$V_GBL        0
                (1)  EISD$V_CRF        0
                (2)  EISD$V_DZRO       1
                (3)  EISD$V_WRT        1
                (4)  EISD$V_INITALCOD  0
                (5)  EISD$V_BASED      0
                (6)  EISD$V_FIXUPVEC   0
                (7)  EISD$V_RESIDENT   0
                (8)  EISD$V_VECTOR     0
                (9)  EISD$V_PROTECT    0
                (10) EISD$V_LASTCLU    1
                (11) EISD$V_EXE        0
                (12) EISD$V_NONSHRADR  0
            section type: EISD$K_USRSTACK
```

$Analyze/Image Example (continued)

```
5)  image section descriptor (56 bytes)
      byte count: 1342976
      base virtual address: %X'00000000' (P0 space)
      page fault cluster size: default
      ISD flags:
            (0)  EISD$V_GBL        1
            (1)  EISD$V_CRF        0
            (2)  EISD$V_DZRO       0
            (3)  EISD$V_WRT        0
            (4)  EISD$V_INITALCOD  0
            (5)  EISD$V_BASED      0
            (6)  EISD$V_FIXUPVEC   0
            (7)  EISD$V_RESIDENT   0
            (8)  EISD$V_VECTOR     0
            (9)  EISD$V_PROTECT    0
            (10) EISD$V_LASTCLU    0
            (11) EISD$V_EXE        0
            (12) EISD$V_NONSHRADR  0
      section type: EISD$K_SHRPIC
      base VBN: 0
      global section major id: %X'01', minor id: %X'000001'
      match control: ISD$K_MATLEQ
      global section name: "DECC$SHR_001"
```

$Analyze/Image Example (continued)

```
Analyze Image                          17-JUL-1996 21:53:22.14    Page 4
_DKA300:[SILLE]PROG.EXE;9
ANALYZ A05-19

      6)  image section descriptor (64 bytes)
            byte count: 6656
            base virtual address: %X'00000000' (P0 space)
            page fault cluster size: default
            ISD flags:
                  (0)  EISD$V_GBL         1
                  (1)  EISD$V_CRF         0
                  (2)  EISD$V_DZRO        0
                  (3)  EISD$V_WRT         0
                  (4)  EISD$V_INITALCOD   0
                  (5)  EISD$V_BASED       0
                  (6)  EISD$V_FIXUPVEC    0
                  (7)  EISD$V_RESIDENT    0
                  (8)  EISD$V_VECTOR      0
                  (9)  EISD$V_PROTECT     0
                  (10) EISD$V_LASTCLU     0
                  (11) EISD$V_EXE         0
                  (12) EISD$V_NONSHRADR   0
            section type: EISD$K_PRVFXD
            base VBN: 0
            global section major id: %X'22', minor id: %X'3A6984'
            match control: ISD$K_MATEQU
            global section name: "SYS$PUBLIC_VECTORS_001"
```

$Analyze/Image Example (continued)

```
Analyze Image                          17-JUL-1996 21:53:22.17    Page 5
_DKA300:[SILLE]PROG.EXE;9
ANALYZ A05-19

IMAGE ACTIVATOR FIXUP SECTION


     Fixed Information

       Flags:
            (0)  EIAF$V_SHR        0

     EIAF$L_QRELFIXOFF     : 00000000
     EIAF$L_LRELFIXOFF     : 00000000
     EIAF$L_QDOTADROFF     : 00000000
     EIAF$L_LDOTADROFF     : 00000000
     EIAF$L_CODEADROFF     : 00000000
     EIAF$L_LPFIXOFF       : 00000058
     EIAF$L_CHGPRTOFF      : 00000078
     EIAF$L_SHLSTOFF       : 000000A0
     EIAF$L_SHRIMGCNT      : 00000002
     EIAF$L_SHLEXTRA       : 00000000
     EIAF$L_PERMCTX        : 00000000
     EIAF$L_LPPSBFIXOFF    : 00000000
```

$Analyze/Image Example (continued)

```
Shareable Image List

  0)  "DECC$SHR"
  1)  "SYS$PUBLIC_VECTORS"

Linkage Pair Reference Fixups (relative to %X'00010000')

  4 references to image 0:
        offset:   00000030  00000070  00000080  000000B0

Protection Change Fixups (relative to %X'00010000')

  address: %X'00020000', byte count: 512
  protection: PRT$C_UREW

  address: %X'00000000', byte count: 512
  protection: PRT$C_UR


The analysis uncovered NO errors.


ANA/IMAGE PROG
$
```

Linker Qualifiers

The linker provides several qualifiers which will affect the resultant image and link map. Qualifiers may be used to affect shareable images and library lookups. These qualifiers will be discussed later.

Qualifiers affecting executable image

Qualifier	Effect
/BPAGE[=9,13,14,15,16]	Affects image section alignment in powers of 2. (512, 8KB, 16KB, 32 KB, and 64KB) Defaults to 64KB.
/[NO]CONTIGUOUS	Create image file on contiguous disk space
/DEBUG	Include debug symbol table
/[NO]DEMAND_ZERO	Allow/block demand zero sections
/EXECUTABLE[=*file_spec*]	Define name of executable
/NATIVE_ONLY	Block outgoing calls to translated images
options_file/OPTIONS	Include linker options (see options)
/[NO]REPLACE	Perform/block JSR->BSR optimizations
/SECTION_BINDING	Check coding practices which could block installing image /RESIDENT
/SELECTIVE_SEARCH	Include only referenced global symbols in global symbol table
/SYMBOL_TABLE	Produce a symbol table file (.STB) for use with the System Dump Analyzer
/SYSEXE	Link against system base image. For kernel level system programs
/[NO]TRACEBACK	Block/enable traceback handler

The following qualifiers affect the link map: **`/BRIEF`**, **`/CROSS_REFERENCE`**, **`/FULL`**, and **`/MAP`**.

Linker Options

The linker also provides options that affect the grouping of sections and the inclusion/production of shareable images. Shareable images will be discussed later.

Option	Effect
CASE_SENSITIVE=YES/NO	Enables/disables case sensitive symbol lookups
CLUSTER=(*name*,,[*pfc*],[*obj1*,...])	Used to order inclusion of sections and group modules close in virtual memory
COLLECT=(*name,psect1*[,*psect2*,...])	Used to group psects into same cluster
IDENTIFICATION=*id-name*	Define identification name
IOSEGMENT=*(size-in-pgelets*,[NO]P0BUFS)	Define size of RMS user mode buffer space (overrides SYSGEN parameter IMGIOCNT)
NAME=*name*	Sets image file name in image header
PROTECT=YES/NO	Blocks user/supervisor mode writability
PSECT_ATTR=(*psect,attribute_list*)	Overrides psect attributes for named psect
STACK=*n*	Defines size of stack in pagelets, defaults to 20 pagelets. Note: will automatically grow as needed at run-time
SYMBOL=(*name,value*)	Defines a global symbol set to absolute numeric value

Alpha Calling Standard Mechanics

The VAX supported a calling standard which used the **CALLS** and **CALLG** instructions to construct a call frame, set the Argument Pointer (**AP**), Frame Pointer (**FP**), and Stack Pointer(**SP**) registers, automatically save registers on the stack, and allow for a condition handler to be established. This mechanism and the **JSB**, **BSBW**, and **BSB** instructions were used to transfer control to subroutines.

- The Alpha systems supports only one call format and the **JSR** instruction to call subroutines. The OpenVMS Calling Standard defines, for Alpha systems, that:
- The first 6 arguments will be passed in **R16** through **R21**
  - Subsequent arguments will be passed on the stack
  - **R25** will contain a parameter count
  - Stack space allocation, call frame construction, and register saving must be performed by prologue and epilogue code in the calling routine or the called routine.
- The Linkage Section
  - The *linkage section* is a program section (psect) which supports the access of external variables, constants, linkage pairs, and procedure descriptors.
  - *Procedure descriptors* support condition handling and debugging. They contain information describing stack and register usage. Each routine has one unique procedure.
- *Linkage pairs* contain the address of the called procedure's procedure descriptor and entry point. Linkage pairs are maintained for external module calls in the caller's linkage section.

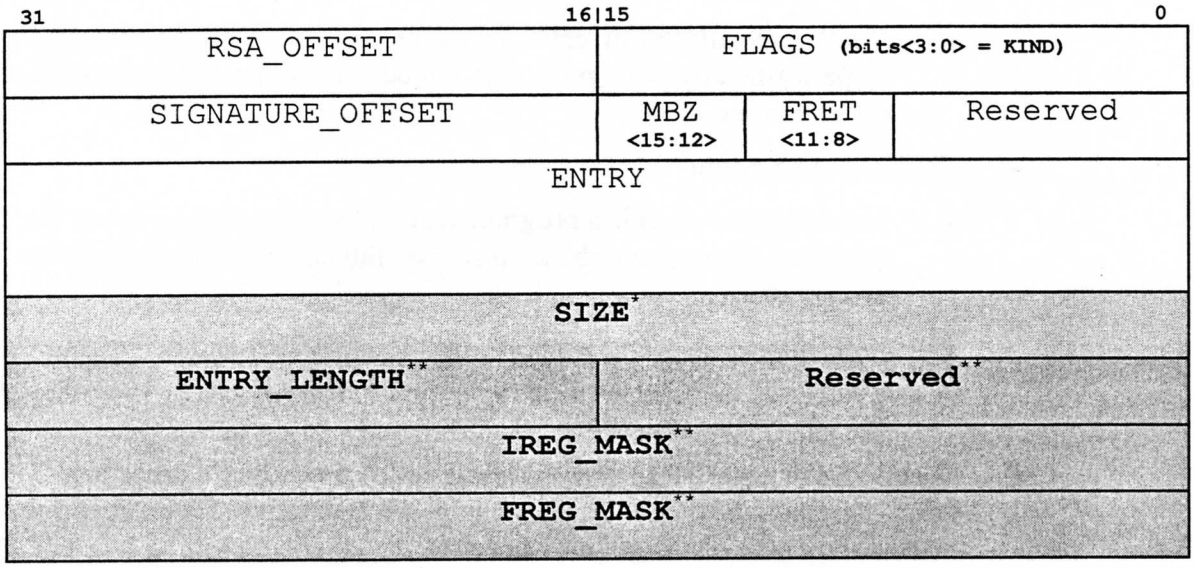
Procedure Descriptors and Values

The OpenVMS procedure calling standard supports 3 basic procedure descriptor formats, though stack frame procedures are most commonly used.

- The basic procedure descriptor types are:

Type	Context	Kind Value
Stack Frame	Context on stack	9
Register Frame	Context in registers	10
Null Frame	Executes in context of caller	8

Procedure descriptor formats:



Shade	Field is a member of
	All
*	**Stack and Register**
**	**Stack only**

- When the address of a function is accessed, the address of the function's procedure descriptor is returned. This address is, referred to as, the *procedure value* of the function.

Accessing Procedure Descriptor Example

```
$ type prog2.c
#include <stdlib.h>
#include <stdio.h>
#include     <string.h>
#include <pdscdef.h>
char *sub(char *);
int  main(void)
{
     char     name[] = "B. B. King ";
     struct pdscdef *proc_desc;

/* Locating and displaying procedure descriptor information. */
     proc_desc =  (struct pdscdef *)sub;
     printf("Procedure descriptor address for \"sub\" %p\n", proc_desc);
     printf("Flags: %04hx   RSA offset: %04hx\n",
       proc_desc->pdsc$w_flags,
       proc_desc->pdsc$w_rsa_offset);
     printf("Procedure entry: %p\n", proc_desc->pdsc$q_entry[0] );
     printf("Integer reg mask: %x\n", proc_desc->pdsc$l_ireg_mask);
     printf("Float reg mask: %x\n", proc_desc->pdsc$l_freg_mask);
     printf("%s\n",sub(name));
     return(EXIT_SUCCESS);
}
$
```

Accessing Procedure Descriptor Example (continued)

```
$ type sub.c
#include     <ctype.h>

char *sub(char *str)
{
     char     *save = str;
     while(*str != '\0')
     {
       *str = toupper(*str);
       str++;
     }
     return(save);
}
$
$ cc prog2
$ cc sub
$ link prog2,sub
$
$ run prog2
Procedure descriptor address for "sub" 100B0
Flags: 3089   RSA offset: 0008
Procedure entry: 20270
Integer reg mask: 2000001c
Float reg mask: 0
B. B. KING
$
```