Introduction to BLISS

Session LT097
Friday, 11 June 1993
11:00 - 12:00

Matthew Madison
TGV, Incorporated
Santa Cruz, CA 95060
madison@tgv.com

Systems Implementation Language

- Originally developed at CMU for PDP-10's
- Designed for building O/S software
- Native implementations for TOPS, VAX systems
- Cross-compilers for PDP-11, AXP systems

This session will concentrate on BLISS-32 (the VAX implementation), BLISS-32E (the 32-bit AXP implementation), and BLISS-64 (the 64-bit AXP implementation).

“Mid-Level” Language

Low-level features

- No built-in I/O statements
- Direct access to instruction set and registers
- Control over psects
- Control over subroutine linkage
- Macro facility

High-level features

- Modular, block-structured
- Common 3GL control constructs
- Macro facility

Data

- No data “types”
- All calculations performed on *fullwords*
  - largest natural storage unit
  - 32-bit longword on VAX systems
  - 16-bit word in PDP-11 systems
- Data segments can be any number of addressable units in size
- Bit field references supported
- Structures supported

Fetch Operator

In BLISS, a variable name always represents the *address* of a data segment. To obtain the value stored at that address, you must use the *fetch* operator.

In:

```
B + 1
```

one is added to data segment B's address.

In:

```
.B + 1
```

the value of B is fetched and one is added.

Field References

Field references extract bit fields from data segments. Format:

```
.X<start-bit,bit-count,sign-extend>
```

Sign-extend field is either 0 (unsigned) or 1 (signed).

Example:

```
.X<0,8,0>
```

extracts the low-order byte (8 bits) from X.

An “Expression Language”

All BLISS executable constructs are expressions, even loops and blocks.

For example, in:

```
.A + (B = .C + 1))
```

the value of the assignment expression (which is .C + 1) is added to the value stored at A.

The semicolon (;) is used not only to separate expressions but also to discard an expression’s value. In

```
B = .C + 1;
```

the value of the assignment expression is not used.

Blocks

Blocks are used to form a single program unit out of several statements. They can return values, just like other expressions. Example:

```
LOVAL =
  BEGIN
    LOCAL TEMP;
    TEMP = .ARRAY [0];
    INCR I FROM 1 TO 9 DO
     IF .TEMP GTR .ARRAY [.I] THEN
       TEMP = .ARRAY [.I];
    .TEMP
  END
```


Declarations

All names must be declared!

```
LOCAL
    STATUS;
```

declares STATUS as a local automatic variable, one fullword in size.

```
OWN
  ARRAY : VECTOR [10,BYTE]
    INITIAL (REP 10 OF (0));
```

declares ARRAY as a static array of 10 bytes, all initialized to zero.

Other declarations

- **Routines**
- **Processor-specific functions**
- **Macros**
- **Literals**

Literals

Literals (constants) can be declared and named:

```
LITERAL
    BUF_SIZE = 1024;
EXTERNAL LITERAL
    CLI$_PRESENT;
```

External literals let you reference globalvalues, like condition values.

Structures

BLISS pre-declares common structures:

- **VECTOR [n, au]**
- **BLOCK [bsize, au]**
- **BLOCKVECTOR [n, bsize, au]**
- **BITVECTOR [n]**

“au” is allocation unit (LONG, BYTE, etc.)

BLOCK references are similar to field references:

```
LOCAL
    STRDSC : BLOCK [8,BYTE];
STRDSC [0,0,16,0] = .STRING_LEN;
STRDSC [2,0,8,0]  = 14;
STRDSC [3,0,8,0]  = 1;
STRDSC [4,0,32,0] = .STRING_ADDR;
```


Field Declarations

Field declarations can be used to simplify BLOCK references:

```
FIELD DSC_FIELDS =
SET
  DSC_W_LENGTH  = [0,0,16,0],
  DSC_B_DTYPE   = [2,0,8,0],
  DSC_B_CLASS   = [3,0,8,0],
  DSC_A_POINTER = [4,0,32,0]
TES;

LOCAL
   STRDSC : BLOCK [8,BYTE]
            FIELD (DSC_FIELDS);

STRDSC [DSC_W_LENGTH] = .STRING_LEN;
...
```

Macros can also be used for this.

STRUCTURE Declarations

You can also define your own structures, defining the allocation and access algorithm for it:

```
STRUCTURE
    FTN_ARRAY [ROW, COL; ROWS, COLS] =
    [ROWS*COLS*4]
    (FTN_ARRAY+((COL-1)*ROWS+(ROW-1)*4);
```

Defines an array structure, 1-based (BLISS predeclared vectors are 0-based), accessed like FORTRAN arrays in column-major rather than row-major order.

Operators

Besides the fetch operator, BLISS provides these operators:

- **arithmetic: +, -, * /, MOD**
- **bit shift: ^**
- **comparison: EQL, NEQ, GTR, LSS, LEQ, GEQ**
- **logical: AND, OR, XOR, NOT**

Only integer arithmetic is supported

Character-handling

BLISS provides a large collection of character-handling functions, only some of which are:

- **CH$MOVE (n, source, dest)**
- **CH$EQL (n1, ptr1, n2, ptr2) (also NEQ, GTR,...)**
- **CH$FILL (fill, n, ptr)**
- **CH$COPY (srcn1, src1, srcn2, src2, ..., fill, destn, dest)**
- **CH$FIND_CH (n, ptr, char)**

Note that CH$MOVE and CH$COPY usually translate into VAX MOVC3 and MOVC5 instructions.

Control Expressions

- Conditional (IF-THEN-ELSE)
- CASE, SELECT, and SELECTONE
- Loops: WHILE, UNTIL
- Iterative loops: INCR, DECR
- Other: LEAVE, EXITLOOP

Note: no GOTO

Routines

All routines must be declared before they are used:

```
ROUTINE SUB1 (PARAM1, PARAM2) =
BEGIN
   ...
END;
```

- **Ordinary routines are visible only to other blocks within a module. GLOBAL routines are visible program-wide.**
- **EXTERNAL ROUTINE can be used to declare routines located in other modules.**
- **FORWARD ROUTINE can be used to pre-declare routine names that will be declared later in the module.**

Addressing Modes

BLISS gives you control over the addressing mode used to access data segments. Typical are:

- **WORD_RELATIVE is the default for BLISS-32. Uses base + word displacement addressing.**
- **LONG_RELATIVE uses base + longword displacement addressing.**
- **GENERAL is equivalent to MACRO's G^.**

You can specify which mode should be used by default for a module in the MODULE declaration. You can override the default on any declaration.

Linkage

BLISS-32 uses the VAX calling standard's CALL-type linkage by default. You can also declare JSB-type linkages, which are handy when you're calling VMS executive routines from inner-mode code:

```
LINKAGE
  IOCRTN = JSB (REGISTER=1, REGISTER=2,
                REGISTER=3; REGISTER=1);
EXTERNAL ROUTINE
  IOC$SEARCHDEV : IOCRTN
                  ADDRESSING_MODE (GENERAL);
```

This declares IOC$SEARCHDEV as a routine with JSB-type linkage that uses registers R1, R2, and R3 as input and register R1 as output.

Including External Definitions

There are two mechanisms:

- **REQUIRE ’file-spec’;**
- **LIBRARY ’file-spec’;**

REQUIRE is for including straight source text. Libraries are pre-compiled files that are much faster loading but have restrictions on the kinds of declarations they can include.

SYS$LIBRARY:STARLET.REQ is the REQUIRE file that contains the VMS system service definitions, constants, etc. It is shipped with VMS.

Macro Facility

BLISS provides an extensive lexical processing facility, with four types of macros (ordinary, key-word, conditional, and interative) and several lexical functions.

Simple conditional processing:

```
%IF DEBUG %THEN
   LIB$PUT_OUTPUT (%ASCID'in routine X');
%FI
```

Simple macro example:

```
MACRO
   DBGPRT (STRING) =
    %IF DEBUG %THEN
      LIB$PUT_OUTPUT (%ASCID STRING)
    %FI
   %;
...
   DBGPRT ('in routine X');
```


Example Module

This code gets the DECwindows display device for another process by translating the logical name DECW$DISPLAY in the process's job logical name table.

```
%TITLE 'DECW_DISPLAY'
MODULE DECW_DISPLAY (IDENT='V1.0-2') =
BEGIN
!+
! Copyright © 1993, Matthew D. Madison.
!                   All Rights Reserved.
!-
    LIBRARY 'SYS$LIBRARY:LIB';

    LINKAGE
        R0JSB = JSB (REGISTER=0) :
                PRESERVE(1,2,3,4,5)
                NOTUSED(6,7,8,9,10,11);

    FORWARD ROUTINE
        DECW_DISPLAY,
        GET_JIB;

%IF %BLISS(BLISS32E) %THEN
    MACRO
        EXE$EPID_TO_PCB = EXE$CVT_EPID_TO_PCB%;
%FI

    EXTERNAL ROUTINE
        EXE$EPID_TO_PCB : R0JSB
                ADDRESSING_MODE (GENERAL),
        LIB$SYS_FAO   : ADDRESSING_MODE (GENERAL),
        STR$COPY_R    : ADDRESSING_MODE (GENERAL),
        STR$FREE1_DX  : ADDRESSING_MODE (GENERAL);
```

Example

```
%SBTTL 'DECW_DISPLAY'
GLOBAL ROUTINE DECW_DISPLAY (PID, DISP_A, ACMODE) =
BEGIN
    BIND
        DISP    = .DISP_A    : BLOCK [,BYTE];

    LOCAL
        JIB             : VOLATILE,
        TABNAM          : BLOCK [DSC$K_S_BLN,BYTE],
        DISPLEN         : VOLATILE WORD,
        DISPBUF         : VOLATILE VECTOR [255,BYTE],
        LNMLST          : $ITMLST_DECL (ITEMS=1),
        ARGLST          : VECTOR [3],
        STATUS;

    ARGLST [0] = 2;
    ARGLST [1] = .PID;
    ARGLST [2] = JIB;

    STATUS = $CMKRNL (ROUTIN=GET_JIB, ARGLST=ARGLST);
    IF NOT .STATUS THEN RETURN .STATUS;

    $INIT_DYNDESC (TABNAM);
    LIB$SYS_FAO (%ASCID'LNM$JOB_!XL', 0, TABNAM, .JIB);

    $ITMLST_INIT (ITMLST=LNMLST,
        (ITMCOD=LNM$_STRING, BUFSIZ=%ALLOCATION (DISPBUF),
            BUFADR=DISPBUF, RETLEN=DISPLEN));

    STATUS = $TRNLNM (TABNAM=TABNAM,
                      LOGNAM=%ASCID'DECW$DISPLAY',
                      ACMODE=ACMODE,
                      ITMLST=LNMLST);
    STR$FREE1_DX (TABNAM);

    IF .DISPBUF [0] EQL %C'_' THEN
          DISPLEN = .DISPLEN-1;

    IF .STATUS THEN
        STR$COPY_R (DISP, DISPLEN,
         (IF .DISPBUF [0] EQL %C'_' THEN
                DISPBUF [1] ELSE DISPBUF));

    .STATUS
END; ! DECW_DISPLAY
```

Example

```
%SBTTL 'GET_JIB'
ROUTINE GET_JIB (PID, JIB_A) =
BEGIN
    LOCAL
        PCB : REF BLOCK [,BYTE];

    .JIB_A = 0;
    PCB = EXE$EPID_TO_PCB (.PID);
    IF .PCB NEQA 0 THEN
    BEGIN
        .JIB_A = .PCB [PCB$L_JIB];
        RETURN SS$_NORMAL;
    END;

    SS$_NONEXPR

END; ! GET_JIB

END
ELUDOM
```