


```

BBBBBBBBB      AAAAAA      SSSSSSSS      MM      MM      AAAAAA      TTTTTTTTTT      IIIIII      DDDDDDDD      NN      NN
BBBBBBBBB      AAAAAA      SSSSSSSS      MM      MM      AAAAAA      TTTTTTTTTT      IIIIII      DDDDDDDD      NN      NN
BB      BB      AA      AA      SS      MMMM      MMMM      AA      AA      TT      II      DD      DD      NN      NN
BB      BB      AA      AA      SS      MMMM      MMMM      AA      AA      TT      II      DD      DD      NN      NN
BB      BB      AA      AA      SS      MM      MM      AA      AA      TT      II      DD      DD      NNNN      NN
BB      BB      AA      AA      SS      MM      MM      AA      AA      TT      II      DD      DD      NNNN      NN
BBBBBBBBB      AA      AA      SSSSSS      MM      MM      AA      AA      TT      II      DD      DD      NN      NN
BBBBBBBBB      AA      AA      SSSSSS      MM      MM      AA      AA      TT      II      DD      DD      NN      NN
BB      BB      AAAAAAAAAA      SS      MM      MM      AAAAAAAAAA      TT      II      DD      DD      NN      NN
BB      BB      AAAAAAAAAA      SS      MM      MM      AAAAAAAAAA      TT      II      DD      DD      NN      NN
BB      BB      AA      AA      SS      MM      MM      AA      AA      TT      II      DD      DD      NN      NN
BB      BB      AA      AA      SS      MM      MM      AA      AA      TT      II      DD      DD      NN      NN
BBBBBBBBB      AA      AA      SSSSSSSS      MM      MM      AA      AA      TT      IIIIII      DDDDDDDD      NN      NN
BBBBBBBBB      AA      AA      SSSSSSSS      MM      MM      AA      AA      TT      IIIIII      DDDDDDDD      NN      NN

```

```

LL      IIIIII      SSSSSSSS
LL      IIIIII      SSSSSSSS
LL      II      SS
LL      II      SS
LL      II      SS
LL      II      SS
LL      II      SSSSSS
LL      II      SSSSSS
LL      II      SS
LL      II      SS
LL      II      SS
LL      IIIIII      SSSSSSSS
LLLLLLLLLL      IIIIII      SSSSSSSS

```

(2) 61
(3) 128

DECLARATIONS
BASSMAT_IDN - Initialize a matrix to identity matrix

```
0000 1 .TITLE BASSMAT_IDN
0000 2 .IDENT /1-012/ ; File: BASSMATIDN.MAR Edit: MDL1012
0000 3
0000 4 *****
0000 5 *
0000 6 * COPYRIGHT (c) 1978, 1980, 1982, 1984 BY *
0000 7 * DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS. *
0000 8 * ALL RIGHTS RESERVED. *
0000 9 *
0000 10 * THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED *
0000 11 * ONLY IN ACCORDANCE WITH THE TERMS OF SUCH LICENSE AND WITH THE *
0000 12 * INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR ANY OTHER *
0000 13 * COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY *
0000 14 * OTHER PERSON. NO TITLE TO AND OWNERSHIP OF THE SOFTWARE IS HEREBY *
0000 15 * TRANSFERRED. *
0000 16 *
0000 17 * THE INFORMATION IN THIS SOFTWARE IS SUBJECT TO CHANGE WITHOUT NOTICE *
0000 18 * AND SHOULD NOT BE CONSTRUED AS A COMMITMENT BY DIGITAL EQUIPMENT *
0000 19 * CORPORATION. *
0000 20 *
0000 21 * DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE OR RELIABILITY OF ITS *
0000 22 * SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL. *
0000 23 *
0000 24 *
0000 25 *****
0000 26
0000 27
0000 28 ++
0000 29 : FACILITY: BASIC code support
0000 30
0000 31 : ABSTRACT:
0000 32
0000 33 : This module initializes a matrix to have zeros everywhere except
0000 34 : ones on the diagonal.
0000 35
0000 36 : ENVIRONMENT: User Mode, AST Reentrant
0000 37
0000 38 --
0000 39 : AUTHOR: R. Will, CREATION DATE: 29-May-79
0000 40
0000 41 : MODIFIED BY:
0000 42 ++
0000 43 : 1-001 - Original
0000 44 : 1-002 - Reference bounds as signed, not unsigned. RW 7-Jun-79
0000 45 : 1-003 - Add support for byte, g and h floating. PLL 17-Sep-81
0000 46 : 1-004 - More modifications for new data types. PLL 24-Sep-81
0000 47 : 1-005 - Changed shared external references to G^ RNH 25-Sep-81
0000 48 : 1-006 - Substitute a macro for the calls the array store
0000 49 : routines. This should speed things up. PLL 6-Nov-81
0000 50 : 1-007 - STORE macro must handle g & h floating. PLL 12-Nov-81
0000 51 : 1-008 - Correct a run-time expression in the STORE macro.
0000 52 : PLL 20-Jan-82
0000 53 : 1-009 - Changed macro STORE to handle arrays of descriptors.
0000 54 : Also added check in mainline code to handle arrays of
0000 55 : descriptors. LEB 28-JUN-1982.
0000 56 : 1-010 - Fixed bug in STORE macro. LEB 4-JUL-1982.
0000 57 : 1-011 - Change own storage to stack storage. LEB 9-Jul-1982
```

BASSMAT_IDN
1-012

G 11

15-SEP-1984 23:43:41 VAX/VMS Macro V04-00 Page 2
6-SEP-1984 10:29:18 [BASRTL.SRC]BASSMATIDN.MAR;1 (1)

0000 58 ; 1-012 - use G^ for ALL externals. MDL 26-May-1983
0000 59 ;--

DECLARATIONS

```

0000 61      .SBTTL  DECLARATIONS
0000 62      :
0000 63      : INCLUDE FILES:
0000 64      :
0000 65      :
0000 66      $DSCDEF
0000 67      $SFDEF
0000 68      :
0000 69      :
0000 70      : EXTERNAL DECLARATIONS:
0000 71      :
0000 72      .DSABL  GBL                ; Prevent undeclared
0000 73      :                          ; symbols from being
0000 74      :                          ; automatically global.
0000 75      .EXTRN  BASSK_ARGDONMAT    ; signalled if all 3 blocks
0000 76      :                          ; not present in array desc
0000 77      .EXTRN  BASSK_DATTYPERR    ; signalled if dtype of array
0000 78      :                          ; isn't word long float double
0000 79      .EXTRN  BASSK_MATDIMERR    ; array wasn't 2 dimensional
0000 80      .EXTRN  BASS$TO_FA_B_R8    ; array element store for byte
0000 81      .EXTRN  BASS$TO_FA_W_R8    ; array element store for word
0000 82      .EXTRN  BASS$TO_FA_L_R8    ; array element store for long
0000 83      .EXTRN  BASS$TO_FA_F_R8    ; array element store - float
0000 84      .EXTRN  BASS$TO_FA_D_R8    ; array element store - double
0000 85      .EXTRN  BASS$TO_FA_G_R8    ; array element store - gfloat
0000 86      .EXTRN  BASS$TO_FA_H_R8    ; array element store - hfloat
0000 87      .EXTRN  BASS$STOP          ; signal fatal errors
0000 88      .EXTRN  BASS$SCALE_R1      ; get the scale for double
0000 89      .EXTRN  BASS$STORE_BFA     ; store value in array
0000 90      :
0000 91      :
0000 92      : MACROS:
0000 93      :
0000 94      :
0000 95      : $BASSMAT_IDN  see below, defines entire identity init algorithm
0000 96      : STORE          store an element into an array
0000 97      :
0000 98      :
0000 99      : EQUATED SYMBOLS:
0000 100     :
0000 101     :
00000000 0000 102     lower_bnd2 = 0      ; stack offset for temp
00000004 0000 103     lower_bnd1 = 4      ; stack offset for temp
00000008 0000 104     upper_bnd1 = 8      ; stack offset for temp
0000000C 0000 105     value_desc = 12     ; output descriptor
00000000 0000 106     str_len = 12        ; length field within desc
0000000E 0000 107     dtype = 14         ; data type field in desc
0000000F 0000 108     class = 15         ; class field within desc
00000010 0000 109     pointer = 16
00000014 0000 110     data = 20           ; data
00000024 0000 111     one_cvt = 36        ; stack offset, converted one
0000001C 0000 112     dsc$l_l1_2 = 28     ; desc offset if 2 sub
00000020 0000 113     dsc$l_u1_2 = 32     ; desc offset if 2 sub
00000024 0000 114     dsc$l_l2_2 = 36     ; desc offset if 2 sub
00000028 0000 115     dsc$l_u2_2 = 40     ; desc offset if 2 sub
0000 116     :
0000 117     :

```

DECLARATIONS

```
0000 118 : OWN STORAGE:  
0000 119 :  
0000 120 :  
0000 121 :  
0000 122 : PSECT DECLARATIONS:  
0000 123 :  
00000000 124 : .PSECT _BASSCODE PIC, USR, CON, REL, LCL, SHR, -  
0000 125 : EXE, RD, NOWRT, LONG  
0000 126 :
```

BASSMAT_IDN - Initialize a matrix to i

```
0000 128 .SBTTL BASSMAT_IDN - Initialize a matrix to identity matrix
0000 129 :++
0000 130 : FUNCTIONAL DESCRIPTION:
0000 131 :
0000 132 : This routine initializes the input matrix to the identity matrix
0000 133 : by setting all diagonal elements to 1 and all the remaining elements
0000 134 : to zero. The algorithm is the same for all the supported
0000 135 : BASIC data types. In order to keep the code for the data types
0000 136 : the same and to simplify the reading, the code has been done as
0000 137 : a macro, which all the data types use varying only the letters
0000 138 : (B, W, L, F, D, G, H) in converting the ones and zeros, in passing the const
0000 139 : and calling the array store routines.
0000 140 :
0000 141 : CALLING SEQUENCE:
0000 142 :
0000 143 : CALL BASMAT_IDN (matrix.wx.da)
0000 144 :
0000 145 : INPUT PARAMETERS:
0000 146 :
0000 147 : NONE
0000 148 :
0000 149 : IMPLICIT INPUTS:
0000 150 :
0000 151 : Scaling from the callers frame (for the double precision one)
0000 152 :
0000 153 : OUTPUT PARAMETERS:
0000 154 :
00000004 0000 155 : matrix = 4
0000 156 :
0000 157 : IMPLICIT OUTPUTS:
0000 158 :
0000 159 : NONE
0000 160 :
0000 161 : FUNCTION VALUE:
0000 162 : COMPLETION CODES:
0000 163 :
0000 164 : NONE
0000 165 :
0000 166 : SIDE EFFECTS:
0000 167 :
0000 168 : This routine will call the BASIC array store routines and so may
0000 169 : cause any of their errors to be signalled. It may also signal any
0000 170 : of the errors listed in the externals section.
0000 171 :
0000 172 :--
0000 173
```

```

0000 175 :+
0000 176 : This macro is a substitute for calls to the array store
0000 177 : routines. It will call the BASS routines only if the array is a
0000 178 : virtual array. Otherwise, it will calculate the linear index into
0000 179 : the array via the INDEX instruction. (Note that BASIC programs must
0000 180 : be able to handle FORTRAN arrays, so the code must check for arrays
0000 181 : stored by column.) The INDEX instructions should provide a significant
0000 182 : performance improvement over calling a routine for each element of
0000 183 : the array.
0000 184 :-
0000 185 .MACRO STORE array_dtype,?L1,?L2,?L3,?L4,?L5,?L6,?L7,?L8,?L9,?L10,?L12,?L
0000 186
0000 187 .IF IDN 'array_dtype', H ; array is hfloat
0000 188 CMPB dsc$b_dtype(R4), #dsc$k_dtype_dsc ; descriptor?
0000 189 BNEQ L10
0000 190 MOVL 4(R4), R0 ; fetch addr of descriptor
0000 191 MOVB dsc$b_dtype(R0), dtype(sp) ; load in data type
0000 192 MOVB dsc$b_class(R0), class(sp) ; load in class field
0000 193 MOVAQ data(SP), pointer(SP) ; load in pointer field
0000 194 CMPB dsc$b_dimct(R4), #1 ; check # of dimensions
0000 195 BNEQ L12 ; branch if 2 dimensions
0000 196 PUSHL R5 ; value of 1st index
0000 197 PUSHL R4 ; addr of array desc
0000 198 PUSHAL value_desc+8(SP) ; addr of value desc
0000 199 CALLS #3,G^BASSSTORE_BFA
0000 200 BRW L9
0000 201 L12: PUSHL R6 ; value of 2nd index
0000 202 PUSHL R5 ; value of 1st index
0000 203 PUSHL R4 ; addr of array desc
0000 204 PUSHAL value_desc+12(SP) ; addr of value desc
0000 205 CALLS #4,G^BASSSTORE_BFA
0000 206 BRW L9
0000 207 L10: CMPB dsc$b_class(R4), #dsc$k_class_bfa ; virtual array?
0000 208 BNEQ L1 ; no
0000 209 JSB G^BASSSTO_FA_'array_dtype'_R8 ; yes, call store routine
0000 210 BRW L9 ; done
0000 211 L1: BBS #5, 10(R4), L2 ; br if stored row-wise
0000 212 INDEX R5, dsc$l_l1_2(R4), dsc$l_u1_2(R4), dsc$l_m2(R4), #0, R7 ; I * M2
0000 213 ; I * M2
0000 214 MOVZWL dsc$w_length(R4), R8 ; longword length for INDEX
0000 215 INDEX R6, dsc$l_l2_2(R4), dsc$l_u2_2(R4), R8, R7, R7 ; (J + (I * M2)) * length
0000 216 ; (J + (I * M2)) * length
0000 217 ADDL dsc$a_a0(R4), R7 ; compute addr of element
0000 218 MOV'array_dtype' R0, (R7) ; store element from R0
0000 219 BRW L9
0000 220 L2: INDEX R6, dsc$l_l2_2(R4), dsc$l_u2_2(R4), dsc$l_m1(R4), #0, R7 ; J * M1
0000 221 ; J * M1
0000 222 MOVZWL dsc$w_length(R4), R8 ; longword length for INDEX
0000 223 INDEX R5, dsc$l_l1_2(R4), dsc$l_u1_2(R4), R8, R7, R7 ; (I + (J * M1)) * length
0000 224 ; (I + (J * M1)) * length
0000 225 ADDL dsc$a_a0(R4), R7 ; compute addr of element
0000 226 MOV'array_dtype' R0, (R7) ; store element from R0
0000 227 .IFF
0000 228 .IF IDN 'array_dtype', G ; array is gfloat
0000 229 CMPB dsc$b_dtype(R2), #dsc$k_dtype_dsc ; descriptor?
0000 230 BNEQ L20
0000 231 MOVL 4(R2), R0 ; fetch addr of descriptor

```

BASSMAT_IDN - Initialize a matrix to i

```

0000 232      MOVB      dsc$b_dtype(R0), dtype(SP)      ; load in data type
0000 233      MOVB      dsc$b_class(R0), class(SP)      ; load in class field
0000 234      MOVAQ     data(SP), pointer(SP)          ; load in pointer field
0000 235      CMPB      dsc$b_dimct(R2), #1           ; check # of dimensions
0000 236      BNEQ      L22                          ; branch if 2 dimensions
0000 237      PUSHL     R3                            ; value of 1st index
0000 238      PUSHL     R2                            ; addr of array desc
0000 239      PUSHAL    value_desc+8(SP)              ; addr of value desc
0000 240      CALLS     #3,G^BASSSTORE_BFA
0000 241      BRW      L9
0000 242      L22:     PUSHL     R4                            ; value of 2nd index
0000 243      PUSHL     R3                            ; value of 1st index
0000 244      PUSHL     R2                            ; addr of array desc
0000 245      PUSHAL    value_desc+12(SP)             ; addr of value desc
0000 246      CALLS     #4,G^BASSSTORE_BFA
0000 247      BRW      L9
0000 248      L20:     CMPB      dsc$b_class(R2), #dsc$k_class_bfa ; virtual array?
0000 249      BNEQ      L3                            ; no
0000 250      JSB      G^BASSSTO_FA_'array_dtype'_R8 ; yes, call store routine
0000 251      BRW      L9                            ; done
0000 252      L3:      BBS      #5, 10(R2), L4          ; br if stored row-wise
0000 253      INDEX    R3, dsc$l_l1_2(R2), dsc$l_u1_2(R2), dsc$l_m2(R2), #0, R5
0000 254      ; I * M2
0000 255      MOVZWL    dsc$w_length(R2), R6           ; longword length for INDEX
0000 256      INDEX    R4, dsc$l_l2_2(R2), dsc$l_u2_2(R2), R6, R5, R5
0000 257      ; (J + (I * M2)) * length
0000 258      ADDL     dsc$a_a0(R2), R5               ; compute addr of element
0000 259      MOV      'array_dtype' R0, (R5)         ; store element from R0
0000 260      BRW      L9
0000 261      L4:      INDEX    R4, dsc$l_l2_2(R2), dsc$l_u2_2(R2), dsc$l_m1(R2), #0, R5
0000 262      ; J * M1
0000 263      MOVZWL    dsc$w_length(R2), R6           ; longword length for INDEX
0000 264      INDEX    R3, dsc$l_l1_2(R2), dsc$l_u1_2(R2), R6, R5, R5
0000 265      ; (I + (J * M1)) * length
0000 266      ADDL     dsc$a_a0(R2), R5               ; compute addr of element
0000 267      MOV      'array_dtype' R0, (R5)         ; store element from R0
0000 268      .IFF
0000 269      .IF      IDN      'array_dtype', D        ; array is double
0000 270      CMPB      dsc$b_dtype(R2), #dsc$k_dtype_desc ; descriptor?
0000 271      BNEQ      L30
0000 272      MOVL     4(R2), R0                      ; fetch addr of descriptor
0000 273      MOVB      dsc$b_dtype(R0), dtype(SP)      ; load in data type
0000 274      MOVB      dsc$b_class(R0), class(SP)     ; load in class field
0000 275      MOVAQ     data(SP), pointer(SP)          ; load in pointer field
0000 276      CMPB      dsc$b_dimct(R2), #1           ; check # of dimensions
0000 277      BNEQ      L32                          ; branch if 2 dimensions
0000 278      PUSHL     R3                            ; value of 1st index
0000 279      PUSHL     R2                            ; addr of array desc
0000 280      PUSHAL    value_desc+8(SP)              ; addr of value desc
0000 281      CALLS     #3,G^BASSSTORE_BFA
0000 282      BRW      L9
0000 283      L32:     PUSHL     R4                            ; value of 2nd index
0000 284      PUSHL     R3                            ; value of 1st index
0000 285      PUSHL     R2                            ; addr of array desc
0000 286      PUSHAL    value_desc+12(SP)             ; addr of value desc
0000 287      CALLS     #4,G^BASSSTORE_BFA
0000 288      BRW      L9

```

BASSMAT_IDN - Initialize a matrix to i 6-SEP-1984 10:29:18 [BASRTL.SRC]BASMATIDN.MAR;1

```
0000 289 L30:  CMPB    dsc$b_class(R2), #dsc$k_class_bfa ; virtual array?
0000 290          BNEQ    L5 ; no
0000 291          JSB     G^BASS$STO_FA_'array_dtype'_R8 ; call store routine
0000 292          BRW     L9 ; done
0000 293 L5:    BBS     #5, 10(R2), L6 ; br if stored col-wise
0000 294          INDEX   R3, dsc$l_l1_2(R2), dsc$l_u1_2(R2), dsc$l_m2(R2), #0, R5
0000 295          ; I * M2
0000 296          MOVZWL  dsc$w_length(R2), R6 ; longword length for INDEX
0000 297          INDEX   R4, dsc$l_l2_2(R2), dsc$l_u2_2(R2), R6, R5, R5
0000 298          ; (J + (I * M2)) * length
0000 299          ADDL    dsc$a_a0(R2), R5 ; compute addr of element
0000 300          MOV 'array_dtype' R0, (R5) ; store element from R0
0000 301          BRW     L9 ; done
0000 302 L6:    INDEX   R4, dsc$l_l2_2(R2), dsc$l_u2_2(R2), dsc$l_m1(R2), #0, R5
0000 303          ; J * M1
0000 304          MOVZWL  dsc$w_length(R2), R6 ; longword length for INDEX
0000 305          INDEX   R3, dsc$l_l1_2(R2), dsc$l_u1_2(R2), R6, R5, R5
0000 306          ; (I + (J * M1)) * length
0000 307          ADDL    dsc$a_a0(R2), R5 ; compute addr of element
0000 308          MOV 'array_dtype' R0, (R5) ; store element from R0
0000 309          .IFF ; array type other than double
0000 310          CMPB    dsc$b_dtype(R1), #dsc$k_dtype_dsc ; descriptor?
0000 311          BNEQ    L40
0000 312          MOVL    4(R1), R0 ; fetch addr of descriptor
0000 313          MOVB    dsc$b_dtype(R0), dtype(SP) ; load in data type
0000 314          MOVB    dsc$b_class(R0), class(SP) ; load in class field
0000 315          MOVAQ  data(SP), pointer(SP) ; load in pointer field
0000 316          CMPB    dsc$b_dimct(R1), #1 ; check # of dimensions
0000 317          BNEQ    L42 ; branch if 2 dimensions
0000 318          PUSHL  R2 ; value of 1st index
0000 319          PUSHL  R1 ; addr of array desc
0000 320          PUSHAL value_desc+8(SP) ; addr of value desc
0000 321          CALLS  #3, G^BASS$STORE_BFA
0000 322          BRW     L9
0000 323 L42:  PUSHL  R3 ; value of 2nd index
0000 324          PUSHL  R2 ; value of 1st index
0000 325          PUSHL  R1 ; addr of array desc
0000 326          PUSHAL value_desc+12(SP) ; addr of value desc
0000 327          CALLS  #4, G^BASS$STORE_BFA
0000 328          BRW     L9
0000 329 L40:  CMPB    dsc$b_class(R1), #dsc$k_class_bfa ; virtual array?
0000 330          BNEQ    L7 ; no
0000 331          JSB     G^BASS$STO_FA_'array_dtype'_R8 ; call store routine
0000 332          BRW     L9 ; done
0000 333 L7:    BBS     #5, 10(R1), L8 ; br if stored col-wise
0000 334          INDEX   R2, dsc$l_l1_2(R1), dsc$l_u1_2(R1), dsc$l_m2(R1), #0, R4
0000 335          ; I * M2
0000 336          MOVZWL  dsc$w_length(R1), R5 ; longword length for INDEX
0000 337          INDEX   R3, dsc$l_l2_2(R1), dsc$l_u2_2(R1), R5, R4, R4
0000 338          ; (J + (I * M2)) * length
0000 339          ADDL    dsc$a_a0(R1), R4 ; compute addr of element
0000 340          MOV 'array_dtype' R0, (R4) ; store element from R0
0000 341          BRW     L9 ; done
0000 342 L8:    INDEX   R3, dsc$l_l2_2(R1), dsc$l_u2_2(R1), dsc$l_m1(R1), #0, R4
0000 343          ; J * M1
0000 344          MOVZWL  dsc$w_length(R1), R5 ; longword length for INDEX
0000 345          INDEX   R2, dsc$l_l1_2(R1), dsc$l_u1_2(R1), R5, R4, R4
```

BASSMAT_IDN
1-012

N 11

15-SEP-1984 23:43:41
6-SEP-1984 10:29:18

VAX/VMS Macro V04-00
[BASRTL.SRC]BASSMATIDN.MAR;1

Page 9
(4)

BASSMAT_IDN - Initialize a matrix to i

```
0000 346  
0000 347 ADDL dsc$a_a0(R1), R4 ; (I + (J * M1)) * length  
0000 348 MOV 'array_dtype' R0, (R4) ; compute addr of element  
0000 349 .ENDC ; store element from R0  
0000 350 .ENDC  
0000 351 .ENDC  
0000 352 L9:  
0000 353 .ENDM  
0000 354
```

BASSMAT_IDN - Initialize a matrix to i

```

0000 356      .MACRO  SBASSMAT_IDN dtype          ; identity init algorithm
0000 357
0000 358  ::+
0000 359      REGISTER USAGE
0000 360      R0 - R8 destroyed by store routines
0000 361      R9      upper bound for 2nd subscript
0000 362      R10     pointer to array descriptor
0000 363      R11     current value of 2nd subscript
0000 364  :-
0000 365
0000 366  ::+
0000 367  :: Set up limits for looping through all elements
0000 368  :-
0000 369
0000 370      MOV 'dtype'      #1, -(SP)          ; make constant same data type
0000 371      ; as array, save on stack
0000 372      .IF      IDN      dtype, D        ; array is double
0000 373      MOVL     SF$SAVE_FP(FP), R0      ; pass FP to get scale
0000 374      JSB      G^BASS$SCALE_R1        ; get scale in R0 & R1
0000 375      ; call a BLISS routine because
0000 376      ; the frame offsets are only
0000 377      ; defined for BLISS
0000 378      MULD2    R0, (SP)                ; scale
0000 379      .ENDC
0000 380
0000 381      CLRQ     -(SP)                      ; alloc data
0000 382      CLRQ     -(SP)                      ; may be hfloat
0000 383      CLRQ     -(SP)                      ; alloc value_desc
0000 384
0000 385 1$:   CMPB     DSC$B_DIMCT(R10), #2    ; determine # of subscripts
0000 386      BEQLU   INIT_TWO_SUBS'dtype'      ; 2 subs, go init
0000 387      BRW     ERR_MATDIMERR             ; not 2 subs, error
0000 388
0000 389  ::+
0000 390  :: There are 2 subscripts. Put the upper bound for both subscripts on the
0000 391  :: stack and make sure that the lower bound for both subscripts will start
0000 392  :: at 1 (do not alter row or col 0 or any negative subscript)
0000 393  :-
0000 394
0000 395 INIT_TWO_SUBS'dtype':
0000 396      PUSHL    dsc$L_u1_2(R10)            ; 1st upper bound
0000 397      PUSHL    dsc$L_l1_2(R10)            ; 1st lower bound
0000 398      BGTR     1$                          ; not row 0 or neg, do cols
0000 399      MOVL     #1, (SP)                    ; start with row 1
0000 400 1$:   MOVL     dsc$L_u2_2(R10), R9      ; 2nd upper bound
0000 401      PUSHL    dsc$L_l2_2(R10)            ; 2nd lower bound
0000 402      BGTR     LOOP_2ND_SUB'dtype'        ; not col 0 or neg, go loop
0000 403      MOVL     #1, (SP)                    ; start with col 1
0000 404
0000 405  ::+
0000 406  :: Loop through all the rows. Row and column upper and lower bounds have been
0000 407  :: initialized on the stack.
0000 408  :-
0000 409
0000 410 LOOP_1ST_SUB'dtype':
0000 411      MOVL     lower_bnd2(SP), R11          ; R11 has 2nd lower bound
0000 412

```

BASSMAT_IDN - Initialize a matrix to i

```

0000 413 :+
0000 414 : Loop through all the elements (columns) of the current row. Column lower
0000 415 : bound is initialized in R11. Column upper bound is on the stack.
0000 416 :-
0000 417
0000 418 LOOP_2ND_SUB'dtype':
0000 419
0000 420          CMPL    R11, lower_bnd1(SP)          ; see if diagonal element
0000 421          BEQL    1$                               ; yes, go put 1
0000 422          CLR'dtype'    R0                               ; no, zero to be stored
0000 423          BRB     2$                               ; continue
0000 424 1$:      MOV'dtype'    one_cvt(SP), R0           ; put scaled 1 into R0
0000 425                                     ; R0 & R1 for double
0000 426 :+
0000 427 : When passed by value, H takes 4 words, G and D take 2 words, and all
0000 428 : others take 1 word.
0000 429 :-
0000 430
0000 431 2$:      .IF      IDN      dtype, H                ; is datatype hfloat
0000 432          MOVL    R10, R4                            ; pointer to array desc
0000 433          MOVL    lower_bnd1(SP), R5                 ; current row
0000 434          MOVL    R11, R6                           ; current column
0000 435          .IFF
0000 436          .IF      IDN      dtype, G                ; datatype gfloat
0000 437          MOVL    R10, R2                            ; pointer to array desc
0000 438          MOVL    lower_bnd1(SP), R3                 ; current row
0000 439          MOVL    R11, R4                           ; current column
0000 440          .IFF
0000 441          .IF      IDN      dtype, D                ; datatype double
0000 442          MOVL    R10, R2                            ; pointer to array desc
0000 443          MOVL    lower_bnd1(SP), R3                 ; current row
0000 444          MOVL    R11, R4                           ; current column
0000 445          .IFF
0000 446                                     ; none of the above
0000 447          MOVL    R10, R1                            ; pointer to array desc
0000 448          MOVL    lower_bnd1(SP), R2                 ; current row
0000 449          MOVL    R11, R3                           ; current column
0000 450          .ENDC
0000 451          .ENDC
0000 452          .ENDC
0000 453          MOV'dtype'    R0, data(SP)                 ; code now same for all types
0000 454          STORE   'dtype'                            ; store in array
0000 455          INCL    R11                               ; get next column
0000 456          CMPL    R11, R9                           ; see if last column done
0000 457          BGTR    3$
0000 458          BRW     LOOP_2ND_SUB'dtype'              ; no, continue inner loop
0000 459
0000 460 :+
0000 461 : Have completed entire row. See if it was the last row. If not,
0000 462 : continue with next row.
0000 463 :-
0000 464
0000 465 3$:      INCL    lower_bnd1(SP)                     ; get next row
0000 466          CMPL    lower_bnd1(SP), upper_bnd1(SP)    ; see if last row done
0000 467          BGTR    5$
0000 468          BRW     LOOP_1ST_SUB'dtype'                 ; no, continue outer loop
0000 469

```

BASSMAT_IDN
1-012

D 12

BASSMAT_IDN - Initialize a matrix to i 15-SEP-1984 23:43:41 VAX/VMS Macro V04-00 Page 12
6-SEP-1984 10:29:18 [BASRTL.SRC]BASSMATIDN.MAR;1 (5)

0000 470 5\$: RET
0000 471 .ENDM

; yes, finished

BASSMAT_IDN - Initialize a matrix to i
.ENTRY BASSMAT_IDN, ^M<R2,R3,R4,R5,R6,R7,R8,R9,R10,R11,IV>

```

4FFC 0000 473
      0002 474
      0002 475 :+
      0002 476 : Put routine arguments into registers for ease of use.
      0002 477 : If block 2 of array descriptor (multipliers) is not present then error.
      0002 478 :-
      0002 479
      0002 480
3F 5A 04 AC DO 0002 480      MOVL    matrix(AP), R10      ; ptr to array descr in R10
  0A AA 07 E1 0006 481      BBC     #DSC$V_FL_BOUNDS, DSC$B_AFLAGS(R10), ERR_ARGDONMAT
      000B 482      ; exit if block 3 not
      000B 483      ; present in descriptor
      000B 484
      000B 485 :+
      000B 486 : Algorithm now differs according to data types
      000B 487 :-
      000B 488
05 06 58 5A DO 000B 489      MOVL    R10, R8      ; save original pointer
  02 A8 8F 000E 490 4$: CASEB   DSC$B_DTYPE(R8), #DSC$K_DTYPE_B, #<DSC$K_DTYPE_D - DSC$K_DTYPE_B>
  0051' 0013 491 1$: .WORD    BYTE-1$      ; code for byte dtype
  0146' 0015 492      .WORD    WORD-1$      ; code for word dtype
  023B' 0017 493      .WORD    LONG-1$      ; code for long dtype
  002A' 0019 494      .WORD    ERR_DATTYPERR-1$      ; quad not supported
  0330' 001B 495      .WORD    FLOAT-1$      ; code for float dtype
  0425' 001D 496      .WORD    DOUBLE-1$      ; code for double dtype
      001F 497
      001F 498 :+
      001F 499 : G and H floating fall outside the range of the CASEB.
      001F 500 :-
      001F 501
1B 02 A8 91 001F 502      CMPB   DSC$B_DTYPE(R8), #DSC$K_DTYPE_G
      03 12 0023 503      BNEQ   2$
      0512 31 0025 504      BRW    GFLOAT      ; code for gfloat dtype
      0028 505
1C 02 A8 91 0028 506 2$: CMPB   DSC$B_DTYPE(R8), #DSC$K_DTYPE_H
      03 12 002C 507      BNEQ   3$
      0603 31 002E 508      BRW    HFLOAT      ; code for hfloat dtype
      0031 509
18 02 A8 91 0031 510 3$: CMPB   DSC$B_DTYPE(R8), #DSC$K_DTYPE_DSC
      06 12 0035 511      BNEQ   ERR_DATTYPERR
58 04 A8 DO 0037 512      MOVL   4(R8), R8      ; R8 <-- addr of descriptor
      D1 11 003B 513      BRB    4$      ; CASE again for dtype in desc
      003D 514
      003D 515 ERR_DATTYPERR:
00000000'8F DD 003D 516      PUSHL  #BASSK_DATTYPERR      ; Signal error, unsupported
00000000'GF 01 FB 0043 517      CALLS  #1, G^BASS$STOP      ; dtype in array desc
      004A 518
      004A 519 ERR_ARGDONMAT:
00000000'8F DD 004A 520      PUSHL  #BASSK_ARGDONMAT      ; signal error,
00000000'GF 01 FB 0050 521      CALLS  #1, G^BASS$STOP      ; block 2 or 3 absent
      0057 522
      0057 523 ERR_MATDIMERR:
00000000'8F DD 0057 524      PUSHL  #BASSK_MATDIMERR      ; signal error not 2 for dimct
00000000'GF 01 FB 005D 525      CALLS  #1, G^BASS$STOP
      0064 526

```

BASSMAT_IDN
1-012

F 12

BASSMAT_IDN - Initialize a matrix to i 15-SEP-1984 23:43:41 VAX/VMS Macro V04-00 Page 14
6-SEP-1984 10:29:18 [BASRTL.SRC]BASSMATIDN.MAR;1 (6)

0064 528 BYTE: \$BASSMAT_IDN B

; expand to byte operations

BASSMAT_IDN
1-012

G 12

BASSMAT_IDN - Initialize a matrix to i 15-SEP-1984 23:43:41 VAX/VMS Macro V04-00 Page 15
0159 530 WORD: \$BASSMAT_IDN W 6-SEP-1984 10:29:18 [BASRTL.SRC]BASMATIDN.MAR;1 (6)
; expand to word operations

BASSMAT_IDN
1-012

H 12

BASSMAT_IDN - Initialize a matrix to i 15-SEP-1984 23:43:41 VAX/VMS Macro V04-00 Page 16
6-SEP-1984 10:29:18 [BASRTL.SRC]BASSMATIDN.MAR;1 (6)

024E 532 LONG: SBASSMAT_IDN L

: expand to long operations

BASSMAT_IDN
1-012

I 12

BASSMAT_IDN - Initialize a matrix to i 15-SEP-1984 23:43:41 VAX/VMS Macro V04-00 Page 17
6-SEP-1984 10:29:18 [BASRTL.SRC]BASMATIDN.MAR;1 (6)

0343 534 FLOAT: \$BASSMAT_IDN F

; expand to float operations

BASSMAT_IDN
1-012

J 12

BASSMAT_IDN - Initialize a matrix to i 15-SEP-1984 23:43:41 VAX/VMS Macro V04-00 Page 18
6-SEP-1984 10:29:18 [BASRTL.SRC]BASSMATIDN.MAR;1 (6)

0438 536 DOUBLE: \$BASSMAT_IDN D

; expand to double operations

BASSMAT_IDN
1-012

K 12

BASSMAT_IDN - Initialize a matrix to i
053A 538 GFLOAT: SBASSMAT_IDN G

15-SEP-1984 23:43:41
6-SEP-1984 10:29:18

VAX/VMS Macro V04-00
[BASRTL.SRC]BASSMATIDN.MAR;1

Page 19
(6)

; expand to gfloat operations

BASSMAT_IDN
1-012

L 12

BASSMAT_IDN - Initialize a matrix to i

15-SEP-1984 23:43:41
6-SEP-1984 10:29:18

VAX/VMS Macro V04-00
[BASRTL.SRC]BASSMATIDN.MAR;1

Page 20
(6)

0634 540 HFLOAT: SBASSMAT_IDN H
072F 541
072F 542 .END

; expand to hfloat operations

; end of BASSMAT_IDN

BASSMAT IDN
Symbol Table

```

BAS$$SCALE_R1          ***** X 00
BAS$$STOP             ***** X 00
BAS$$K_ARGDONMAT      ***** X 00
BAS$$K_DATTYPERR      ***** X 00
BAS$$K_MATDIMERR      ***** X 00
BASMAT IDN            00000000 RG 02
BAS$$STORE_BFA        ***** X 00
BAS$$STO_FA_B_R8      ***** X 00
BAS$$STO_FA_D_R8      ***** X 00
BAS$$STO_FA_F_R8      ***** X 00
BAS$$STO_FA_G_R8      ***** X 00
BAS$$STO_FA_H_R8      ***** X 00
BAS$$STO_FA_L_R8      ***** X 00
BAS$$STO_FA_W_R8      ***** X 00
BYTE                  00000064 R 02
CLASS                 = 0000000F
DATA                  = 00000014
DOUBLE                00000438 R 02
DSC$$A_AO             = 00000010
DSC$$B_AFLAGS         = 0000000A
DSC$$B_CLASS          = 00000003
DSC$$B_DIMCT          = 0000000B
DSC$$B_DTYPE          = 00000002
DSC$$K_CLASS_BFA     = 000000BF
DSC$$K_DTYPE_B        = 00000006
DSC$$K_DTYPE_D        = 0000000B
DSC$$K_DTYPE_DSC     = 00000018
DSC$$K_DTYPE_G        = 0000001B
DSC$$K_DTYPE_H        = 0000001C
DSC$$L_L1_2           = 0000001C
DSC$$L_L2_2           = 00000024
DSC$$L_M1             = 00000014
DSC$$L_M2             = 00000018
DSC$$L_U1_2           = 00000020
DSC$$L_U2_2           = 00000028
DSC$$V_FL_BOUNDS     = 00000007
DSC$$W_LENGTH         = 00000000
DTYPE                 = 0000000E
ERR_ARGDONMAT         0000004A R 02
ERR_DATTYPERR         0000003D R 02
ERR_MATDIMERR         00000057 R 02
FLOAT                 00000343 R 02
GFLOAT                0000053A R 02
HFLOAT                00000634 R 02
INIT_TWO_SUBSB        00000076 R 02
INIT_TWO_SUBSD        00000457 R 02
INIT_TWO_SUBSF        00000355 R 02
INIT_TWO_SUBSG        0000054D R 02
INIT_TWO_SUBSH        00000647 R 02
INIT_TWO_SUBSL        00000260 R 02
INIT_TWO_SUBSW        0000016B R 02
LONG                  0000024E R 02
LOOP_1ST_SUBB         0000008D R 02
LOOP_1ST_SUBD         0000046E R 02
LOOP_1ST_SUBF         0000036C R 02
LOOP_1ST_SUBG         00000564 R 02
LOOP_1ST_SUBH         0000065E R 02

```

```

LOOP_1ST_SUBL        00000277 R 02
LOOP_1ST_SUBW        00000182 R 02
LOOP_2ND_SUBB        00000090 R 02
LOOP_2ND_SUBD        00000471 R 02
LOOP_2ND_SUBF        0000036F R 02
LOOP_2ND_SUBG        00000567 R 02
LOOP_2ND_SUBH        00000661 R 02
LOOP_2ND_SUBL        0000027A R 02
LOOP_2ND_SUBW        00000185 R 02
LOWER_BND1           = 00000004
LOWER_BND2           = 00000000
MATRIX               = 00000004
ONE_CVT              = 00000024
POINTER              = 00000010
SF$L_SAVE_FP         = 0000000C
UPPER_BNDT           = 00000008
VALUE_DESC           = 0000000C
WORD                 00000159 R 02

```

! Psect synopsis !

PSECT name	Allocation	PSECT No.	Attributes
. ABS .	00000000 (0.)	00 (0.)	NOPIC USR CON ABS LCL NOSHR NOEXE NORD NOWRT NOVEC BYTE
\$ABSS	00000000 (0.)	01 (1.)	NOPIC USR CON ABS LCL NOSHR EXE RD WRT NOVEC BYTE
_BAS\$CODE	0000072F (1839.)	02 (2.)	PIC USR CON REL LCL SHR EXE RD NOWRT NOVEC LONG

! Performance indicators !

Phase	Page faults	CPU Time	Elapsed Time
Initialization	34	00:00:00.11	00:00:00.56
Command processing	130	00:00:00.63	00:00:02.71
Pass 1	224	00:00:06.64	00:00:12.61
Symbol table sort	0	00:00:00.32	00:00:00.59
Pass 2	112	00:00:01.93	00:00:04.24
Symbol table output	9	00:00:00.08	00:00:00.09
Psect synopsis output	3	00:00:00.03	00:00:00.03
Cross-reference output	0	00:00:00.00	00:00:00.00
Assembler run totals	514	00:00:09.75	00:00:20.84

The working set limit was 1350 pages.
35843 bytes (71 pages) of virtual memory were used to buffer the intermediate code.
There were 20 pages of symbol table space allocated to hold 220 non-local and 81 local symbols.
542 source lines were read in Pass 1, producing 17 object records in Pass 2.
30 pages of virtual memory were used to define 10 macros.

! Macro library statistics !

Macro library name	Macros defined
_\$255\$DUA28:[BASRTL.OBJ]BASRTL.MLB;1	0
-\$255\$DUA28:[SYSLIB]STARLET.MLB;2	5
TOTALS (all libraries)	5

223 GETS were required to define 5 macros.

There were no errors, warnings or information messages.

MACRO/ENABLE=SUPPRESSION/DISABLE=(GLOBAL,TRACEBACK)/LIS=LIS\$:BASMATIDN/OBJ=OBJ\$:BASMATIDN MSRC\$:BASMATIDN/UPDATE=(ENH\$:BASMATIDN)+LI

