

```
BBBBBBBBBBBBBB  AAAAAAAAAA  SSSSSSSSSSSSS  RRRRRRRRRRRR  TTTTTTTTTTTTTT  LLL
BBBBBBBBBBBBBB  AAAAAAAAAA  SSSSSSSSSSSSS  RRRRRRRRRRRR  TTTTTTTTTTTTTT  LLL
BBBBBBBBBBBBBB  AAAAAAAAAA  SSSSSSSSSSSSS  RRRRRRRRRRRR  TTTTTTTTTTTTTT  LLL
BBB          BBB  AAA          AAA  SSS          RRR          RRR  TTT          LLL
BBB          BBB  AAA          AAA  SSS          RRR          RRR  TTT          LLL
BBB          BBB  AAA          AAA  SSS          RRR          RRR  TTT          LLL
BBB          BBB  AAA          AAA  SSS          RRR          RRR  TTT          LLL
BBB          BBB  AAA          AAA  SSS          RRR          RRR  TTT          LLL
BBB          BBB  AAA          AAA  SSS          RRR          RRR  TTT          LLL
BBBBBBBBBBBBBB  AAA          AAA  SSS          RRR          RRR  TTT          LLL
BBBBBBBBBBBBBB  AAA          AAA  SSS          RRR          RRR  TTT          LLL
BBBBBBBBBBBBBB  AAA          AAA  SSS          RRR          RRR  TTT          LLL
BBB          BBB  AAAAAAAAAAAAAA  SSS          RRR          RRR  TTT          LLL
BBB          BBB  AAAAAAAAAAAAAA  SSS          RRR          RRR  TTT          LLL
BBB          BBB  AAAAAAAAAAAAAA  SSS          RRR          RRR  TTT          LLL
BBB          BBB  AAA          AAA  SSS          RRR          RRR  TTT          LLL
BBB          BBB  AAA          AAA  SSS          RRR          RRR  TTT          LLL
BBB          BBB  AAA          AAA  SSS          RRR          RRR  TTT          LLL
BBBBBBBBBBBBBB  AAA          AAA  SSS          RRR          RRR  TTT          LLL
BBBBBBBBBBBBBB  AAA          AAA  SSS          RRR          RRR  TTT          LLL
BBBBBBBBBBBBBB  AAA          AAA  SSS          RRR          RRR  TTT          LLL
```

```
BBBBBBBB      AAAAAA      SSSSSSSS      MM      MM      AAAAAA      TTTTTTTTTT      AAAAAA      DDDDDDDD      DDDDDDDD
BBBBBBBB      AAAAAA      SSSSSSSS      MM      MM      AAAAAA      TTTTTTTTTT      AAAAAA      DDDDDDDD      DDDDDDDD
BB      BB      AA      AA      SS      MMMM      MMMM      AA      AA      TT      AA      AA      DD      DD      DD      DD
BB      BB      AA      AA      SS      MMMM      MMMM      AA      AA      TT      AA      AA      DD      DD      DD      DD
BB      BB      AA      AA      SS      MM      MM      AA      AA      TT      AA      AA      DD      DD      DD      DD
BBBBBBBB      AA      AA      SSSSSS      MM      MM      AA      AA      TT      AA      AA      DD      DD      DD      DD
BBBBBBBB      AA      AA      SSSSSS      MM      MM      AA      AA      TT      AA      AA      DD      DD      DD      DD
BB      BB      AAAAAAAAAA      SS      MM      MM      AAAAAAAAAA      TT      AAAAAAAAAA      DD      DD      DD      DD
BB      BB      AAAAAAAAAA      SS      MM      MM      AAAAAAAAAA      TT      AAAAAAAAAA      DD      DD      DD      DD
BB      BB      AA      AA      SS      MM      MM      AA      AA      TT      AA      AA      DD      DD      DD      DD
BB      BB      AA      AA      SS      MM      MM      AA      AA      TT      AA      AA      DD      DD      DD      DD
BBBBBBBB      AA      AA      SSSSSSSS      MM      MM      AA      AA      TT      AA      AA      DDDDDDDD      DDDDDDDD
BBBBBBBB      AA      AA      SSSSSSSS      MM      MM      AA      AA      TT      AA      AA      DDDDDDDD      DDDDDDDD
.....
.....
.....
.....
```

```
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      II      SS
LLLLLLLLLL      IIIIII      SSSSSSSS
LLLLLLLLLL      IIIIII      SSSSSSSS
```

BASSMAT_ADD
Table of contents

N 14

15-SEP-1984 23:39:02 VAX/VMS Macro V04-00

Page 0

(2) 68
(4) 353

DECLARATIONS
BASSMAT_ADD - Add 2 arrays giving a third

```
0000 1 .TITLE BASSMAT_ADD
0000 2 .IDENT /1-017/ ; File: BASMATADD.MAR Edit: DG1017
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 adds 2 arrays of any dtype and stores the result in a
0000 34 : third array.
0000 35 :
0000 36 : ENVIRONMENT: User Mode, AST Reentrant
0000 37 :
0000 38 :--
0000 39 : AUTHOR: R. Will, CREATION DATE: 18-Jun-79
0000 40 :
0000 41 : MODIFIED BY:
0000 42 :++
0000 43 : 1-001 - Original
0000 44 : 1-002 - Set IV flag in entry mask. RW 2-Oct-79
0000 45 : 1-003 - Add dtypes byte, g and h floating. PLL 11-Sep-81
0000 46 : 1-004 - More modifications for new data types. PLL 24-Sep-81
0000 47 : 1-005 - Changed external references to G^ RNH 25-Sep-81
0000 48 : 1-006 - Substitute a macro for the calls to the array fetch and store
0000 49 : routines. This should speed things up. PLL 6-Nov-81
0000 50 : 1-007 - STORE macro must be modified to handle g & h floating. PLL 11-Nov-81
0000 51 : 1-008 - Reserve space on stack for hfloat source. PLL 17-Nov-81
0000 52 : 1-009 - Correct a run-time expression in the FETCH and STORE macros.
0000 53 : PLL 20-Jan-82
0000 54 : 1-010 - Correct FETCH, STORE again. PLL 23-Feb-82
0000 55 : 1-011 - Don't list macro expansions. PLL 16-Mar-82
0000 56 : 1-012 - Fix CASEB statements. PLL 13-Apr-82
0000 57 : 1-013 - Remove FETCH and STORE macros; they are now located in macro
```

```
0000 58 : Library MATRIXMAC.OLB. Add code to support arrays of descriptors.
0000 59 : LEB 13-Jun-82
0000 60 : 1-014 - Change own storage to stack storage. LEB 9-Jul-1982
0000 61 : 1-015 - Allow gfloat results to be stored in a double destination, and
0000 62 : vice versa. PLL 7-Oct-1982
0000 63 : 1-016 - Use G^ for ALL externals. Don't list macro expansions.
0000 64 : SBL 16-Nov-1982
0000 65 : 1-017 - Correct stack offsets when storing in LONG array. DG 10-Jan-1984
0000 66 :--
```

DECLARATIONS

```

0000 68      .SBTTL  DECLARATIONS
0000 69      :
0000 70      : INCLUDE FILES:
0000 71      :
0000 72      :
0000 73      $DSCDEF      ; define descriptor offsets
0000 74      $SFDEF      ; use to get scale
0000 75      :
0000 76      :
0000 77      : EXTERNAL DECLARATIONS:
0000 78      :
0000 79      :
0000 80      .DSABL  GBL      ; Prevent undeclared
0000 81      :              ; symbols from being
0000 82      :              ; automatically global.
0000 83      .EXTRN  BASSK_ARGDONMAT ; signalled if all 3 blocks
0000 84      :              ; not present in array desc
0000 85      :              ; or dimct = 0
0000 86      .EXTRN  BASSK_DATTYPERR ; signalled if dtype of array
0000 87      :              ; isn't word long float double
0000 88      .EXTRN  BASSK_MATDIMERR ; signalled if # of dims on
0000 89      :              ; source arrays don't agree
0000 90      .EXTRN  BASSK_ARRMUSSAM ; signalled if upper and lower
0000 91      :              ; bnds not same on src arrays
0000 92      .EXTRN  BASSSTO_FA_W_R8 ; array element store for word
0000 93      .EXTRN  BASSSTO_FA_L_R8 ; array element store for long
0000 94      .EXTRN  BASSSTO_FA_F_R8 ; array element store - float
0000 95      .EXTRN  BASSSTO_FA_D_R8 ; array element store - double
0000 96      .EXTRN  BASSSTO_FA_B_R8 ; array element store - byte
0000 97      .EXTRN  BASSSTO_FA_G_R8 ; array element store - gfloat
0000 98      .EXTRN  BASSSTO_FA_H_R8 ; array element store - hfloat
0000 99      .EXTRN  BASSFET_FA_W_R8 ; array element fetch - word
0000 100     .EXTRN  BASSFET_FA_L_R8 ; array element fetch - long
0000 101     .EXTRN  BASSFET_FA_F_R8 ; array element fetch - float
0000 102     .EXTRN  BASSFET_FA_D_R8 ; array element fetch - double
0000 103     .EXTRN  BASSFET_FA_B_R8 ; array element fetch - byte
0000 104     .EXTRN  BASSFET_FA_G_R8 ; array element fetch - gfloat
0000 105     .EXTRN  BASSFET_FA_H_R8 ; array element fetch - hfloat
0000 106     .EXTRN  BASSMAT_REDIM   ; check if redimensioning of
0000 107     :              ; dest array is necessary, if
0000 108     :              ; so, do it
0000 109     .EXTRN  BASS$SCALE_R1   ; scale for double precision
0000 110     .EXTRN  MTH$DINT_R4     ; truncate dbl precision number
0000 111     .EXTRN  BASS$STOP      ; signal fatal errors
0000 112     .EXTRN  BASS$FETCH_DESC ; fetch addr of descriptor
0000 113     .EXTRN  BASS$FETCH_BFA
0000 114     .EXTRN  BASS$STORE_BFA
0000 115     :
0000 116     : MACROS:
0000 117     :
0000 118     :
0000 119     : SBASSMAT_ADD  add loop algorithm, see next page
0000 120     : FETCH      fetch an element from an array (found in macro library
0000 121     :              MATRIXMAC.OLB.
0000 122     : STORE      store an element into an array (found in macro library
0000 123     :              MATRIXMAC.OLB.
0000 124     :

```

DECLARATIONS

```
0000 125 :  
0000 126 : EQUATED SYMBOLS:  
0000 127 :  
0000 128 :  
00000000 0000 129 lower_bnd2 = 0 ; stack offset for temp  
00000004 0000 130 lower_bnd1 = 4 ; stack offset for temp  
00000008 0000 131 upper_bnd1 = 8 ; stack offset for temp  
0000000C 0000 132 save_src1 = 12 ; stack offset for temp  
00000010 0000 133 value_desc = 28 ; define an output desc  
0000001C 0000 134 str_len = 28 ; length field within desc  
0000001E 0000 135 dtype = 30 ; data type field in desc  
0000001F 0000 136 class = 31 ; class field in desc  
00000020 0000 137 pointer = 32 ; pointer to DATA  
00000024 0000 138 data = 36 ; data field (4 longwords)  
00000018 0000 139 dsc$l_l1_1 = 24 ; desc offset if 1 sub  
0000001C 0000 140 dsc$l_u1_1 = 28 ; desc offset if 1 sub  
0000001C 0000 141 dsc$l_l1_2 = 28 ; desc offset if 2 sub  
00000020 0000 142 dsc$l_u1_2 = 32 ; desc offset if 2 sub  
00000024 0000 143 dsc$l_l2_2 = 36 ; desc offset if 2 sub  
00000028 0000 144 dsc$l_u2_2 = 40 ; desc offset if 2 sub  
0000 145 :  
0000 146 :  
0000 147 : OWN STORAGE:  
0000 148 :  
0000 149 :  
0000 150 :  
0000 151 :  
0000 152 : PSECT DECLARATIONS:  
0000 153 :  
00000000 154 .PSECT _BASS$CODE PIC,USR,CON,REL,LCL,SHR,-  
0000 155 EXE,RD,NOWRT, LONG  
0000 156  
0000 157
```

DECLARATIONS

```

0000 159 :+
0000 160 : This macro contains the looping mechanism for accessing all elements of
0000 161 : an array. It also contains all the logic for all the combinations of data
0000 162 : types and scaling. A macro is used to make it easy to maintain the parallel
0000 163 : code for all the different data types.
0000 164 :-
0000 165 .MACRO $BASSMAT_ADD src1_dtype, src2_dtype ; add algorithm
0000 166
0000 167 :+
0000 168 : Loop through all the rows. Row and column upper and lower bounds have been
0000 169 : initialized on the stack.
0000 170 :-
0000 171
0000 172 LOOP_1ST_SUB'src1_dtype'src2_dtype':
0000 173     MOVL     lower_bnd2(SP), R11                ; R11 has 2nd lower bound
0000 174
0000 175 :+
0000 176 : Loop through all the elements (columns) of the current row. Column lower
0000 177 : bound is initialized in R11. Column upper bound is on the stack.
0000 178 : Distinguish array by data type so that the correct fetch routine can
0000 179 : retrieve the data, the correct add can be done and the correct
0000 180 : store routine can be called.
0000 181 :-
0000 182
0000 183 LOOP_2ND_SUB'src1_dtype'src2_dtype':
0000 184
0000 185 :+
0000 186 : Get the data from the first source array
0000 187 :-
0000 188
0000 189     MOVL     src1_matrix(AP), R0                ; pointer to 1st src array
0000 190     MOVL     lower_bnd1(SP), R1                ; current row
0000 191     MOVL     R11, R2                          ; current col
0000 192     FETCH   'src1_dtype'                      ; fetch data from src1 array
0000 193     MOV     'src1_dtype' R0, save_src1(SP)     ; store the 1st array element
0000 194
0000 195 :+
0000 196 : Get the data from the second source array
0000 197 :-
0000 198
0000 199     MOVL     src2_matrix(AP), R0                ; pointer to 2nd src array
0000 200     MOVL     lower_bnd1(SP), R1                ; current row
0000 201     MOVL     R11, R2                          ; current col
0000 202     FETCH   'src2_dtype'                      ; fetch data from src2 array
0000 203
0000 204 :+
0000 205 : If the data types of the 2 source arrays is the same, do the arithmetic
0000 206 : in that data type. Else convert the data to a common type and add.
0000 207 : If scaling is needed (ie if at least one but not both of the arrays is
0000 208 : double) convert integer to double. (Note that the integerize is not
0000 209 : necessary because only integers (not float) can be converted to double,
0000 210 : and the sum of 2 integers is guaranteed to be integer).
0000 211 :-
0000 212
0000 213     .IF     IDN     src1_dtype, src2_dtype ; src arrays are
0000 214     ADD     'src1_dtype'2          save_src1(SP), R0 ; same data type
0000 215     ADD     'src1_dtype'2          save_src1(SP), R0 ; add the source elements

```


DECLARATIONS

G 15

15-SEP-1984 23:39:02
6-SEP-1984 10:28:41

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

```

0000 216 BSBW DEST_CASE_'src1_dtype' ; go to store in dest
0000 217 .IFF
0000 218 .IF IDN src1_dtype, H ; source 1 is hfloat
0000 219 CVT'src2_dtype'H R0, R0
0000 220 ADDH2 save_src1(SP), R0
0000 221 BSBW DEST_CASE_H
0000 222 .IFF
0000 223 .IF IDN src2_dtype, H ; is 2nd source hfloat
0000 224 CVT'src1_dtype'H save_src1(SP), R2 ; cvt source 1 to hfloat
0000 225
0000 226 ADDH2 R2, R0
0000 227 BSBW DEST_CASE_H
0000 228 .IFF
0000 229 .IF IDN src1_dtype, G ; source 1 is gfloat
0000 230 .IF IDN src2_dtype, D ; special case gfloat + dbl
0000 231 CVT'src1_dtype'H save_src1(SP), R4 ; promote both operands to
0000 232 ; hfloat
0000 233 CVT'src2_dtype'H R0, R0
0000 234 ADDH2 R4, R0
0000 235 BSBW DEST_CASE_H ; go to store the dest
0000 236 .IFF
0000 237 CVT'src2_dtype'G R0, R0 ; cvt source 2 to gfloat
0000 238 ADDG2 save_src1(SP), R0
0000 239 BSBW DEST_CASE_G
0000 240 .ENDC
0000 241 .IFF
0000 242 .IF IDN src2_dtype, G ; is source 2 gfloat
0000 243 .IF IDN src1_dtype, D ; special case dbl + gfloat
0000 244 CVT'src1_dtype'H save_src1(SP), R4 ; promote both operands to
0000 245 ; hfloat
0000 246 CVT'src2_dtype'H R0, R0
0000 247 ADDH2 R4, R0
0000 248 BSBW DEST_CASE_H ; go to store the dest
0000 249 .IFF
0000 250 CVT'src1_dtype'G save_src1(SP), R2 ; cvt source 1 to gfloat
0000 251
0000 252 ADDG2 R2, R0
0000 253 BSBW DEST_CASE_G
0000 254 .ENDC
0000 255 .IFF
0000 256 .IF IDN src1_dtype, D ; src arrays different dtype
0000 257 ; source 1 is double
0000 258 ; (no need to check for gfloat
0000 259 ; because that case is handled
0000 260 ; above)
0000 260 CVT'src2_dtype'D R0, -(SP) ; cvt array2 to double & save
0000 261 MOVL SF$SAVE_FP(FP), R0 ; pass FP to get scale
0000 262 JSB G^BAS$$SCALE_R1 ; get scale in R0 & R1
0000 263 ; call a BLISS routine because
0000 264 ; the frame offsets are only
0000 265 ; defined for BLISS
0000 266 MULD2 (SP)+, R0 ; scale 2nd element
0000 267 JSB G^MTH$DINT R4 ; integerize
0000 268 ADDD2 save_src1(SP), R0 ; add 1st element & scaled 2nd
0000 269 BSBW DEST_CASE_D ; cvrt double sum to dest type
0000 270 .IFF ; 1st array not double
0000 271 .IF IDN src2_dtype, D ; is 2nd src double
0000 272 CVT'src1_dtype'D save_src1(SP), save_src1(SP)

```

DECLARATIONS

H 15

15-SEP-1984 23:39:02
6-SEP-1984 10:28:41

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

Page 7
(3)

```

0000 273
0000 274      MOVD   R0, -(SP)
0000 275      MOVL   SF$L_SAVE_FP(FP), R0
0000 276      JSB    G^BAS$$SCALE_R1
0000 277
0000 278
0000 279
0000 280      MULD2  save_src1+8(SP), R0
0000 281
0000 282      JSB    G^MTH$DINT_R4
0000 283      ADDD2  (SP)+, R0
0000 284      BSBW   DEST_CASE_D
0000 285      .IFF
0000 286      .IF    IDN    src1_dtype, F
0000 287      CVT'src2_dtype'F      R0, R0
0000 288      ADDF2  save_src1(SP), R0
0000 289      BSBW   DEST_CASE_F
0000 290      .IFF
0000 291      .IF    IDN    src2_dtype, F
0000 292      CVT'src1_dtype'F      save_src1(SP), R1
0000 293      ADDF2  R1, R0
0000 294      BSBW   DEST_CASE_F
0000 295      .IFF
0000 296      .IF    IDN    src1_dtype, L
0000 297      CVT'src2_dtype'L      R0, R0
0000 298      ADDL2  save_src1(SP), R0
0000 299      BSBW   DEST_CASE_L
0000 300      .IFF
0000 301      .IF    IDN    src2_dtype, L
0000 302      CVT'src1_dtype'L      save_src1(SP), R1
0000 303      ADDL2  R1, R0
0000 304      BSBW   DEST_CASE_L
0000 305      .IFF
0000 306      .IF    IDN    src1_dtype, B
0000 307      CVT'src2_dtype'B      R0, R0
0000 308      ADDB2  save_src1(SP), R0
0000 309      BSBW   DEST_CASE_B
0000 310      .IFF
0000 311      CVT'src1_dtype'B      save_src1(SP), R1
0000 312
0000 313      ADDB2  R1, R0
0000 314      BSBW   DEST_CASE_B
0000 315
0000 316      .ENDC
0000 317      .ENDC
0000 318      .ENDC
0000 319      .ENDC
0000 320      .ENDC
0000 321      .ENDC
0000 322      .ENDC
0000 323      .ENDC
0000 324      .ENDC
0000 325      .ENDC
0000 326      .ENDC
0000 327      .ENDC
0000 328
0000 329      ;+

```

```

: yes, make src1 double & save
: save the data
: pass FP to get scale
: get scale in R0 & R1
: call a BLISS routine because
: the frame offsets are only
: defined for BLISS
: scale, (+8 because src2 is
: double and saved on stack
: integerize
: compute the sum
: cvrt double sum to dest type
: no double operands try float
: is 1st element float
: make 2nd element float
: add
: cvrt float sum to dest type
: 1st array not float
: is 2nd array float
: yes-make 1st element float
: add
: cvrt float sum to dest type
: no double or float, try long
: is 1st array long
: make 2nd element long
: add
: convrt long sum to dest type
:
: source 2 is long
: cvt src1 to long
: add
: convrt long sum to dest type
: source 1 is byte
: cvt source 2 to byte
: src2 must be byte, so cvt src1

```

DECLARATIONS

```
0000 330 ; Have stored that element. Now see if it was the last column. If not,  
0000 331 ; continue with the next column. Otherwise continue to next row.  
0000 332 ;-  
0000 333  
0000 334 INCL R11 ; get next column  
0000 335 CMPL R11, R9 ; see if last column done  
0000 336 BGTR 5$  
0000 337 BRW LOOP_2ND_SUB'src1_dtype'src2_dtype' ; no, continue inner loop  
0000 338  
0000 339 ;+  
0000 340 ; Have completed entire row. See if it was the last row. If not,  
0000 341 ; continue with next row.  
0000 342 ;-  
0000 343  
0000 344 5$: INCL lower_bnd1(SP) ; get next row  
0000 345 CMPL lower_bnd1(SP), upper_bnd1(SP) ; see if last row done  
0000 346 BGTR 10$  
0000 347 BRW LOOP_1ST_SUB'src1_dtype'src2_dtype' ; no, continue outer loop  
0000 348  
0000 349 10$: RET ; yes, finished  
0000 350  
0000 351 .ENDM
```

BASSMAT_ADD

- Add 2 arrays giving a thi

.SBTTL BASSMAT_ADD - Add 2 arrays giving a third

0000 353
0000 354
0000 355
0000 356
0000 357
0000 358
0000 359
0000 360
0000 361
0000 362
0000 363
0000 364
0000 365
0000 366
0000 367
0000 368
0000 369
0000 370
0000 371
0000 372
0000 373
00000004 0000 374
00000008 0000 375
0000 376
0000 377
0000 378
0000 379
0000 380
0000000C 0000 381
0000 382
0000 383
0000 384
0000 385
0000 386
0000 387
0000 388
0000 389
0000 390
0000 391
0000 392
0000 393
0000 394
0000 395
0000 396
0000 397
0000 398
0000 399
0000 400
0000 401
4FFC 0000 402
0002 403
0002 404
0002 405
0002 406
0002 407
0002 408
0002 409

++
FUNCTIONAL DESCRIPTION:
Add 2 arrays giving a third. Signal an error if the 2 arrays to be added do not have the same number of dimensions and the same upper and lower bounds for those dimensions. Redimension the output array to have the same upper bounds as the input arrays. Initialize all the necessary looping information on the stack. Conversions may have to be done so that the sources are the same data type, so divide the looping portion according to the data types. Conversion to the correct destination data type will be done by a JSB to a routine, instead of multiplying the number of possible combinations by 4.
CALLING SEQUENCE:
CALL BASSMAT_ADD (src1_array.rx.da, src2_array.rw.da, dest_matrix.wx.da)
INPUT PARAMETERS:
src1_matrix = 4
src2_matrix = 8
IMPLICIT INPUTS:
Scale from the callers frame to scale double precision.
OUTPUT PARAMETERS:
dest_matrix = 12
IMPLICIT OUTPUTS:
NONE
FUNCTION VALUE:
COMPLETION CODES:
NONE
SIDE EFFECTS:
This routine calls the redimensioning routine and the array element fetch and store routines and therefore may signal any of their errors. It may also signal any of the errors listed in the externals section. It may also cause the destination array to have different dimensions.
--
.ENTRY BASSMAT_ADD, ^M<R2,R3,R4,R5,R6,R7,R8,R9,R10,R11,IV>
+
REGISTER USAGE
R0 - R8 destroyed by store routines
R9 upper bound for 2nd subscript
R10 pointer to dest array descriptor

BASSMAT_ADD

- Add 2 arrays giving a thi

```

0002 410 : R11 current value of 2nd subscript
0002 411 :-
0002 412 :-
0002 413 :+
0002 414 : Put routine arguments into registers for ease of use.
0002 415 : If block 2 of array descriptor (multipliers) is not present then error.
0002 416 :-
0002 417 :-
1F 52 04 AC DO 0002 418 MOVL src1_matrix(AP), R2 ; ptr to src1 array descr
0A A2 07 E1 0006 419 BBC #DSC$V_FL_BOUNDS, DSC$B_AFLAGS(R2), ERR_ARGDONMAT ;
000B 420 ; exit if block 3 not
000B 421 ; present in descriptor
16 53 08 AC DO 000B 422 MOVL src2_matrix(AP), R3 ; ptr to src2 array descr
0A A3 07 E1 000F 423 BBC #DSC$V_FL_BOUNDS, DSC$B_AFLAGS(R3), ERR_ARGDONMAT ;
0014 424 ; exit if block 3 not
0014 425 ; present in descriptor
5A 0C AC DO 0014 426 MOVL dest_matrix(AP), R10 ; ptr to dest descriptor
7E 7C 0018 427 CLRQ -(SP) ; save space for DATA
7E 7C 001A 428 CLRQ -(SP) ; and
7E 7C 001C 429 CLRQ -(SP) ; VALUE_DESC
7E 7C 001E 430 CLRQ -(SP) ; reserve space to save src1
7E 7C 0020 431 CLRQ -(SP) ; src1 may be hfloat
0022 432 :-
0022 433 :+
0022 434 : Set up limits for looping through all elements
0022 435 :-
0022 436 :-
01 0B A2 91 0022 437 CMPB DSC$B_DIMCT(R2), #1 ; determine # of subscripts
0F 13 0026 438 BEQLU INIT_ONE_SUB ; 1 sub, go init
59 1A 0028 439 BGTRU INIT_TWO_SUBS ; >=2 subs, go init
002A 440 ; 0 subs, fall into error proc
002A 441
002A 442 ERR_ARGDONMAT:
00000000'8F DD 002A 443 PUSHL #BASSK ARGDONMAT ; signal error, 0 for dimct
00000000'GF 01 FB 0030 444 CALLS #1, G^BASS$STOP ; or block 2 or 3 absent
0037 445
0037 446 :+
0037 447 : There is only 1 subscript. Redimension the destination array.
0037 448 : Make both upper and lower bound for 2nd
0037 449 : subscript a 1. A second subscript will be passed to and ignored by the
0037 450 : store routine. Put bounds for 1st subscript on stack.
0037 451 :-
0037 452 :-
0037 453 INIT_ONE_SUB:
0B A3 01 91 0037 454 CMPB #1, DSC$B_DIMCT(R3) ; do src arrays have same
003B 455 ; number of dimensions
1C A3 1C A2 91 003B 456 BNEQU ERR_MATDIMERR ; no, error
0042 457 CMPB dsc$l_u1_1(R2), dsc$l_u1_1(R3) ; do src arrays have the same
0042 458 ; upper bounds
18 A3 18 A2 91 0042 459 BNEQU ERR_ARRMUSSAM ; no, error
0044 460 CMPB dsc$l_l1_1(R2), dsc$l_l1_1(R3) ; do src arrays have the same
0049 461 ; lower bounds
0049 462 BNEQU ERR_ARRMUSSAM ; no, error
1C A3 DD 004B 463 PUSHL dsc$l_u1_1(R3) ; get bound for redim
00000000'GF 5A DD 004E 464 PUSHL R10 ; pointer to dest array desc
1C A3 02 FB 0050 465 CALLS #2, G^BASSMAT REDIM ; redimension the dest
0057 466 PUSHL dsc$l_u1_1(R3) ; 1st upper bound

```

```

BASSMAT_ADD - Add 2 arrays giving a thi
18 A3 DD 005A 467          PUSHL  dsc$L_l1_1(R3)          ; 1st lower bound
   03 14 005D 468          BGTR   1$                    ; not 0 or neg, do 2nd sub
6E 01 D0 005F 469          MOVL  #1, (SP)                ; don't alter col 0
   01 DD 0062 470 1$:      PUSHL  #1                    ; dummy 2nd upper bound
59 01 D0 0064 471          MOVL  #1, R9                 ; dummy 2nd lower bound
   62 11 0067 472          BRB    SEPARATE_DTYPES          ; go loop
      0069 473
      0069 474 ERR_MATDIMERR:
00000000'8F DD 0069 475          PUSHL  #BASSK_MATDIMERR          ; Signal error, src arrays
00000000'GF 01 FB 006F 476          CALLS  #1, G^BASS$STOP          ; don't have same # dimensns
      0076 477
      0076 478 ERR_ARRMUSSAM:
00000000'8F DD 0076 479          PUSHL  #BASSK_ARRMUSSAM          ; Signal error, src arrays
00000000'GF 01 FB 007C 480          CALLS  #1, G^BASS$STOP          ; same bounds
      0083 481
      0083 482
      0083 483 :+
      0083 484 : There are 2 subscripts. Check and redimension the destination array if
      0083 485 : necessary. Put the upper bound for both subscripts on the
      0083 486 : stack and make sure that the lower bound for both subscripts will start
      0083 487 : at 1 (do not alter row or col 0)
      0083 488 :-
      0083 489 INIT_TWO_SUBS:
0B A3 02 91 0083 490          CMPB   #2, DSC$B_DIMCT(R3)          ; do src arrays have same
      0087 491          ; number of dimensions
      0087 492          BNEQU  ERR_MATDIMERR          ; no, error
20 A3 20 A2 91 0089 493          CMPB   dsc$L_u1_2(R2), dsc$L_u1_2(R3) ; do src arrays have the same
      008E 494          ; 1st upper bounds
      008E 495          BNEQU  ERR_ARRMUSSAM          ; no, error
1C A3 1C A2 91 0090 496          CMPB   dsc$L_l1_2(R2), dsc$L_l1_2(R3) ; do src arrays have the same
      0095 497          ; 1st lower bounds
      0095 498          BNEQU  ERR_MATDIMERR          ; no, error
28 A3 28 A2 91 0097 499          CMPB   dsc$L_u2_2(R2), dsc$L_u2_2(R3) ; do src arrays have the same
      009C 500          ; 2nd upper bounds
      009C 501          BNEQU  ERR_ARRMUSSAM          ; no, error
24 A3 24 A2 91 009E 502          CMPB   dsc$L_l2_2(R2), dsc$L_l2_2(R3) ; do src arrays have the same
      00A3 503          ; 2nd lower bounds
      00A3 504          BNEQU  ERR_ARRMUSSAM          ; no, error
      28 A3 DD 00A5 505          PUSHL  dsc$L_u2_2(R3)          ; 2nd upper bound
      20 A3 DD 00A8 506          PUSHL  dsc$L_u1_2(R3)          ; 1st upper bound
00000000'GF 5A DD 00AB 507          PUSHL  R10                ; dest array pointer
      20 A3 DD 00B4 509          PUSHL  dsc$L_u1_2(R3)          ; 1st upper bound
      1C A3 DD 00B7 510          PUSHL  dsc$L_l1_2(R3)          ; 1st lower bound
      03 14 00BA 511          BGTR   1$                    ; not row 0 or neg, do cols
      6E 01 D0 00BC 512          MOVL  #1, (SP)                ; start with row 1
59 28 A3 D0 00BF 513 1$:      MOVL  dsc$L_u2_2(R3), R9          ; 2nd upper bound
      24 A3 DD 00C3 514          PUSHL  dsc$L_l2_2(R3)          ; 2nd lower bound
      03 14 00C6 515          BGTR   SEPARATE_DTYPES          ; not col 0 or neg, go loop
      6E 01 D0 00C8 516          MOVL  #1, (SP)                ; start with col 1
      00CB 517
      00CB 518 :+
      00CB 519 : Algorithm now differs according to data types
      00CB 520 :-
      00CB 521
05 06 02 A2 8F 00CB 522 SEPARATE_DTYPES:
      00CB 523 4$: CASEB  DSC$B_DTYPE(R2), #DSC$K_DTYPE_B, #<DSC$K_DTYPE_D - DSC$K_DTYPE_B>

```

BASSMAT_ADD - Add 2 arrays giving a thi

```
0037' 00D0 524 2$: .WORD BYTE-2$ ; code for byte dtype
0E25' 00D2 525 .WORD WORD-2$ ; code for word dtype
1C13' 00D4 526 .WORD LONG-2$ ; code for long dtype
002A' 00D6 527 .WORD ERR_DATTYPERR-2$ ; quad not supported
2A01' 00D8 528 .WORD FLOAT-2$ ; code for float dtype
37EF' 00DA 529 .WORD DOUBLE-2$ ; code for double dtype
      00DC 530
1B 02 A2 91 00DC 531 CMPB DSC$B_DTYPE(R2), #DSC$K_DTYPE_G
      03 12 00E0 532 BNEQ 3$
      4600 31 00E2 533 BRW GFLOAT
      00E5 534
1C 02 A2 91 00E5 535 3$: CMPB DSC$B_DTYPE(R2), #DSC$K_DTYPE_H
      03 12 00E9 536 BNEQ 5$
      540B 31 00EB 537 BRW HFLOAT
      00EE 538
18 02 A2 91 00EE 539 5$: CMPB DSC$B_DTYPE(R2), #DSC$K_DTYPE_DSC ; descriptors?
      06 12 00F2 540 BNEQ ERR_DATTYPERR ; no - signal error
52 04 A2 D0 00F4 541 MOVL 4(R2),R2 ; Store addr of desc in R2
      D1 11 00F8 542 BRB 4$ ; CASE again for dtype in desc
      00FA 543
      00FA 544 ERR_DATTYPERR:
00000000'8F DD 00FA 545 PUSHL #BASS$K_DATTYPERR ; Signal error, unsupported
00000000'GF 01 FB 0100 546 CALLS #1, G^BASS$$STOP ; dtype in array desc
```

BASSMAT_ADD - Add 2 arrays giving a thi

```

0107 549 :+
0107 550 : Source array is a byte array. Differentiate on the destination type.
0107 551 :-
0107 552
05 06 02 A3 8F 0107 553 BYTE: CASEB DSC$B_DTYPE(R3), #DSC$K_DTYPE_B, #<DSC$K_DTYPE_D - DSC$K_DTYPE_B>
002D' 010C 554 1$: .WORD BYTE_TO_BYTE-1$ ; code for byte dtype
021B' 010E 555 .WORD BYTE_TO_WORD-1$ ; code for word dtype
040C' 0110 556 .WORD BYTE_TO_LONG-1$ ; code for long dtype
FFEE' 0112 557 .WORD ERR_DATTYPERR-1$ ; quad not supported
05FD' 0114 558 .WORD BYTE_TO_FLOAT-1$ ; code for float dtype
07EE' 0116 559 .WORD BYTE_TO_DOUBLE-1$ ; code for double dtype
0118 560
1B 02 A3 91 0118 561 CMPB DSC$B_DTYPE(R3), #DSC$K_DTYPE_G
03 12 011C 562 BNEQ 2$
09E2 31 011E 563 BRW BYTE_TO_GFLOAT
0121 564
1C 02 A3 91 0121 565 2$: CMPB DSC$B_DTYPE(R3), #DSC$K_DTYPE_H
03 12 0125 566 BNEQ 3$
0BD2 31 0127 567 BRW BYTE_TO_HFLOAT
012A 568
18 02 A3 91 012A 569 3$: CMPB DSC$B_DTYPE(R3), #DSC$K_DTYPE_DSC
06 12 012E 570 BNEQ 4$
53 04 A3 D0 0130 571 MOVL 4(R3),R3 ; R3 <-- addr of descriptor
D1 11 0134 572 BRB BYTE ; CASE again for dtype in desc
0136 573
FFC1 31 0136 574 4$: BRW ERR_DATTYPERR ; unsupported dtype
0139 575
0139 576 :+
0139 577 : Use the macro to generate the code for each case.
0139 578 :-
0139 579

```


BASSMAT_ADD
1-017

B 16

BASSMAT_ADD - Add 2 arrays giving a thi 15-SEP-1984 23:39:02 VAX/VMS Macro V04-00 Page 14
6-SEP-1984 10:28:41 [BASRTL.SRC]BASMATADD.MAR;1 (5)

0139 581 BYTE_TO_BYTE: \$BASSMAT_ADD B, B
0327 582

BASSMAT_ADD
1-017

C 16

BASSMAT_ADD - Add 2 arrays giving a thi 15-SEP-1984 23:39:02 VAX/VMS Macro V04-00 Page 15
6-SEP-1984 10:28:41 [BASRTL.SRC]BASMATADD.MAR;1 (5)

0327 584 BYTE_TO_WORD: SBASSMAT_ADD B, W
0518 585

BASSMAT_ADD
1-017

D 16

BASSMAT_ADD - Add 2 arrays giving a thi 15-SEP-1984 23:39:02 VAX/VMS Macro V04-00 Page 16
6-SEP-1984 10:28:41 [BASRTL.SRC]BASMATADD.MAR;1 (5)

0518 587 BYTE_TO_LONG: \$BASSMAT_ADD B, L
0709 588

BASSMAT_ADD
1-017

E 16

BASSMAT_ADD - Add 2 arrays giving a thi 15-SEP-1984 23:39:02 VAX/VMS Macro V04-00 Page 17
6-SEP-1984 10:28:41 [BASRTL.SRC]BASMATADD.MAR;1 (5)

0709 590 BYTE_TO_FLOAT: \$BASSMAT_ADD B, F
08FA 591

BASSMAT_ADD
1-017

F 16

BASSMAT_ADD - Add 2 arrays giving a thi 15-SEP-1984 23:39:02 VAX/VMS Macro V04-00 Page 18
6-SEP-1984 10:28:41 [BASRTL.SRC]BASMATADD.MAR;1 (5)

08FA 593 BYTE_TO_DOUBLE: \$BASSMAT_ADD B, D

BASSMAT_ADD
1-017

G 16

BASSMAT_ADD - Add 2 arrays giving a thi 15-SEP-1984 23:39:02 VAX/VMS Macro V04-00 Page 19
6-SEP-1984 10:28:41 [BASRTL.SRC]BASMATADD.MAR;1 (5)

OB03 595 BYTE_TO_GFLOAT: \$BASSMAT_ADD B, G
OCFC 596

BASSMAT_ADD
1-017

H 16

BASSMAT_ADD - Add 2 arrays giving a thi 15-SEP-1984 23:39:02 VAX/VMS Macro V04-00 Page 20
6-SEP-1984 10:28:41 [BASRTL.SRC]BASMATADD.MAR;1 (5)

OCFC 598 BYTE_TO_HFLOAT: SBASSMAT_ADD B, H
OEF5 599

BASSMAT_ADD - Add 2 arrays giving a thi

```

05 06 02 A3 8F 0EF5 601 :+
      002D 0EF5 602 : Source array is a word array. Now differentiate on the destination type.
      021E 0EF5 603 :-
      040C 0EF5 604
      F200 0EF5 605 WORD: CASEB DSC$B_DTYPE(R3), #DSC$K_DTYPE_B, #<DSC$K_DTYPE_D - DSC$K_DTYPE_B>
      05FD 0EF5 606 1$: .WORD WORD_TO_BYTE-1$ ; code for byte dtype
      07EE 0EF5 607 .WORD WORD_TO_WORD-1$ ; code for word dtype
      021E 0EFA 608 .WORD WORD_TO_LONG-1$ ; code for long dtype
      040C 0EFE 609 .WORD ERR_DATTYPERR-1$ ; quad not supported
      05FD 0F00 610 .WORD WORD_TO_FLOAT-1$ ; code for float dtype
      07EE 0F02 611 .WORD WORD_TO_DOUBLE-1$ ; code for double dtype
      0F06 612
1B 02 A3 91 0F06 613 CMPB DSC$B_DTYPE(R3), #DSC$K_DTYPE_G
      03 12 0FOA 614 BNEQ 2$
      09E2 31 0FOC 615 BRW WORD_TO_GFLOAT
      0F0F 616
1C 02 A3 91 0F0F 617 2$: CMPB DSC$B_DTYPE(R3), #DSC$K_DTYPE_H
      03 12 0F13 618 BNEQ 3$
      0BD2 31 0F15 619 BRW WORD_TO_HFLOAT
      0F18 620
18 02 A3 91 0F18 621 3$: CMPB DSC$B_DTYPE(R3), #DSC$K_DTYPE_DSC ; array of descriptors?
      06 12 0F1C 622 BNEQ 4$ ; branch if not
53 04 A3 D0 0F1E 623 MOVL 4(R3), R3 ; move addr of desc in R3
      D1 11 0F22 624 BRB WORD ; CASE again on dtype in desc
      0F24 625
      F1D3 31 0F24 626 4$: BRW ERR_DATTYPERR ; unsupported dtype
      0F27 627
      0F27 628 :+
      0F27 629 : Now type of source and destination arrays are known. Use the macro to
      0F27 630 : generate the code for each case
      0F27 631 :-
      0F27 632

```


BASSMAT_ADD
1-017

J 16

BASSMAT_ADD - Add 2 arrays giving a thi 15-SEP-1984 23:39:02 VAX/VMS Macro V04-00 Page 22
6-SEP-1984 10:28:41 [BASRTL.SRC]BASSMATADD.MAR;1 (5)

OF27 634 WORD_TO_BYTE: \$BASSMAT_ADD W, B
1118 635

BASSMAT_ADD
1-017

K 16

BASSMAT_ADD - Add 2 arrays giving a thi 15-SEP-1984 23:39:02 VAX/VMS Macro V04-00 Page 23
6-SEP-1984 10:28:41 [BASRTL.SRC]BASMATADD.MAR;1 (5)

1118 637 WORD_TO_WORD: \$BASSMAT_ADD W, W
1306 638

BASSMAT_ADD
1-017

L 16

BASSMAT_ADD - Add 2 arrays giving a thi 15-SEP-1984 23:39:02 VAX/VMS Macro V04-00 Page 24
6-SEP-1984 10:28:41 [BASRTL.SRC]BASMATADD.MAR;1 (5)

1306 640 WORD_TO_LONG: SBASSMAT_ADD W, L
14F7 641

BASSMAT_ADD
1-017

M 16

BASSMAT_ADD - Add 2 arrays giving a thi 15-SEP-1984 23:39:02 VAX/VMS Macro V04-00 Page 25
6-SEP-1984 10:28:41 [BASRTL.SRC]BASMATADD.MAR;1 (5)

14F7 643 WORD_TO_FLOAT: \$BASSMAT_ADD W, F
16E8 644

BASSMAT_ADD
1-017

B 1

BASSMAT_ADD - Add 2 arrays giving a thi 15-SEP-1984 23:39:02 VAX/VMS Macro V04-00 Page 26
6-SEP-1984 10:28:41 [BASRTL.SRC]BASMATADD.MAR;1 (5)

16E8 646 WORD_TO_DOUBLE: \$BASSMAT_ADD W, D
18F1 647

BASSMAT_ADD
1-017

C 1

BASSMAT_ADD - Add 2 arrays giving a thi 15-SEP-1984 23:39:02 VAX/VMS Macro V04-00 Page 27
6-SEP-1984 10:28:41 [BASRTL.SRC]BASMATADD.MAR;1 (5)

18F1 649 WORD_TO_GFLOAT: SBASSMAT_ADD W, G
1AEA 650

BASSMAT_ADD
1-017

D 1

BASSMAT_ADD - Add 2 arrays giving a thi 15-SEP-1984 23:39:02 VAX/VMS Macro V04-00 Page 28
6-SEP-1984 10:28:41 [BASRTL.SRC]BASMATADD.MAR;1 (5)

1AEA 652 WORD_TO_HFLOAT: \$BASSMAT_ADD W, H
1CE3 653

BASSMAT_ADD - Add 2 arrays giving a thi

```

1CE3 655 ;+
1CE3 656 ; Source array is a longword array. Now differentiate on the destination type
1CE3 657 ;-
1CE3 658
05 06 02 A3 8F 1CE3 659 LONG: CASEB DSC$B_DTYPE(R3), #DSC$K_DTYPE_B, #<DSC$K_DTYPE_D - DSC$K_DTYPE_B>
002D' 1CE8 660 1$: .WORD LONG_TO_BYTE-1$ ; code for byte dtype
021E' 1CEA 661 .WORD LONG_TO_WORD-1$ ; code for word dtype
040F' 1CEC 662 .WORD LONG_TO_LONG-1$ ; code for long dtype
E412' 1CEE 663 .WORD ERR_DATTYPERR-1$ ; quad not supported
05FD' 1CF0 664 .WORD LONG_TO_FLOAT-1$ ; code for float dtype
07EE' 1CF2 665 .WORD LONG_TO_DOUBLE-1$ ; code for double dtype
1CF4 666
1B 02 A3 91 1CF4 667 CMPB DSC$B_DTYPE(R3), #DSC$K_DTYPE_G
03 12 1CF8 668 BNEQ 2$
09E2 31 1CFA 669 BRW LONG_TO_GFLOAT
1CFD 670
1C 02 A3 91 1CFD 671 2$: CMPB DSC$B_DTYPE(R3), #DSC$K_DTYPE_H
03 12 1D01 672 BNEQ 3$
OBD2 31 1D03 673 BRW LONG_TO_HFLOAT
1D06 674
18 02 A3 91 1D06 675 3$: CMPB DSC$B_DTYPE(R3), #DSC$K_DTYPE_DSC ; array of descriptors?
06 12 1DOA 676 BNEQ 4$ ; branch if not
53 04 A3 D0 1D0C 677 MOVL 4(R3), R3 ; move addr of desc in R3
D1 11 1D10 678 BRB LONG ; CASE again on dtype in desc
1D12 679
E3E5 31 1D12 680 4$: BRW ERR_DATTYPERR ; unsupported dtype
1D15 681
1D15 682 ;+
1D15 683 ; Now type of source and destination arrays are known. Use the macro to
1D15 684 ; generate the code for each case
1D15 685 ;-
1D15 686

```


BASSMAT_ADD
1-017

F 1

BASSMAT_ADD - Add 2 arrays giving a thi 15-SEP-1984 23:39:02 VAX/VMS Macro V04-00 Page 30
6-SEP-1984 10:28:41 [BASRTL.SRC]BASMATADD.MAR;1 (5)

1D15 688 LONG_TO_BYTE: \$BASSMAT_ADD L, B
1F06 689

BASSMAT_ADD
1-017

G 1

BASSMAT_ADD - Add 2 arrays giving a thi 15-SEP-1984 23:39:02 VAX/VMS Macro V04-00 Page 31
6-SEP-1984 10:28:41 [BASRTL.SRC]BASMATADD.MAR;1 (5)

1F06 691 LONG_TO_WORD: SBASSMAT_ADD L, W
20F7 692

BASSMAT_ADD
1-017

H 1
BASSMAT_ADD - Add 2 arrays giving a thi 15-SEP-1984 23:39:02 VAX/VMS Macro V04-00 Page 32
6-SEP-1984 10:28:41 [BASRTL.SRC]BASMATADD.MAR;1 (5)

20F7 694 LONG_TO_LONG: SBASSMAT_ADD L, L
22E5 695

BASSMAT_ADD
1-017

I 1

BASSMAT_ADD - Add 2 arrays giving a thi 15-SEP-1984 23:39:02 VAX/VMS Macro V04-00 Page 33
6-SEP-1984 10:28:41 [BASRTL.SRC]BASMATADD.MAR;1 (5)

22E5 697 LONG_TO_FLOAT: \$BASSMAT_ADD L, F
24D6 698

BASSMAT_ADD
1-017

J 1

BASSMAT_ADD - Add 2 arrays giving a thi 15-SEP-1984 23:39:02 VAX/VMS Macro V04-00 Page 34
6-SEP-1984 10:28:41 [BASRTL.SRC]BASMATADD.MAR;1 (5)

24D6 700 LONG_TO_DOUBLE: \$BASSMAT_ADD L, D
26DF 701

BASSMAT_ADD
1-017

K 1

BASSMAT_ADD - Add 2 arrays giving a thi 15-SEP-1984 23:39:02 VAX/VMS Macro V04-00 Page 35
6-SEP-1984 10:28:41 [BASRTL.SRC]BASMATADD.MAR;1 (5)

26DF 703 LONG_TO_GFLOAT: \$BASSMAT_ADD L, G
28D8 704

BASSMAT_ADD
1-017

L 1

BASSMAT_ADD - Add 2 arrays giving a thi 15-SEP-1984 23:39:02 VAX/VMS Macro V04-00 Page 36
6-SEP-1984 10:28:41 [BASRTL.SRC]BASMATADD.MAR;1 (5)

28D8 706 LONG_TO_HFLOAT: \$BASSMAT_ADD L, H
2AD1 707

BASSMAT_ADD - Add 2 arrays giving a thi

```

2AD1 709 :+
2AD1 710 : Source array is a floating array. Now differentiate on the destination type
2AD1 711 :-
2AD1 712
05 06 02 A3 8F 2AD1 713 FLOAT: CASEB DSC$B_DTYPE(R3), #DSC$K_DTYPE_B, #<DSC$K_DTYPE_D - DSC$K_DTYPE_B>
002D' 2AD6 714 1$: .WORD FLOAT_TO_BYTE-1$ ; code for byte dtype
021E' 2AD8 715 .WORD FLOAT_TO_WORD-1$ ; code for word dtype
040F' 2ADA 716 .WORD FLOAT_TO_LONG-1$ ; code for long dtype
D624' 2ADC 717 .WORD ERR_DATTYPERR-1$ ; quad not supported
0600' 2ADE 718 .WORD FLOAT_TO_FLOAT-1$ ; code for float dtype
07EE' 2AE0 719 .WORD FLOAT_TO_DOUBLE-1$ ; code for double dtype
2AE2 720
1B 02 A3 91 2AE2 721 CMPB DSC$B_DTYPE(R3), #DSC$K_DTYPE_G
03 12 2AE6 722 BNEQ 2$
09E2 31 2AE8 723 BRW FLOAT_TO_GFLOAT
2AEB 724
1C 02 A3 91 2AEB 725 2$: CMPB DSC$B_DTYPE(R3), #DSC$K_DTYPE_H
03 12 2AEF 726 BNEQ 3$
OBD2 31 2AF1 727 BRW FLOAT_TO_HFLOAT
2AF4 728
18 02 A3 91 2AF4 729 3$: CMPB DSC$B_DTYPE(R3), #DSC$K_DTYPE_DSC ; array of descriptors?
06 12 2AF8 730 BNEQ 4$ ; branch if not
53 04 A3 D0 2AFA 731 MOVL 4(R3), R3 ; move addr of desc in R3
D1 11 2AFE 732 BRB FLOAT ; CASE again on dtype in desc
2B00 733
DSF7 31 2B00 734 4$: BRW ERR_DATTYPERR ; unsupported dtype
2B03 735
2B03 736 :+
2B03 737 : Now type of source and destination arrays are known. Use the macro to
2B03 738 : generate the code for each case
2B03 739 :-
2B03 740

```


BASSMAT_ADD
1-017

N 1

BASSMAT_ADD - Add 2 arrays giving a thi 15-SEP-1984 23:39:02 VAX/VMS Macro V04-00 Page 38
6-SEP-1984 10:28:41 [BASRTL.SRC]BASMATADD.MAR;1 (5)

2B03 742 FLOAT_TO_BYTE: \$BASSMAT_ADD F, B
2CF4 743

B
1

BASSMAT_ADD
1-017

B 2

BASSMAT_ADD - Add 2 arrays giving a thi 15-SEP-1984 23:39:02 VAX/VMS Macro V04-00 Page 39
6-SEP-1984 10:28:41 [BASRTL.SRC]BASMATADD.MAR;1 (5)

2CF4 745 FLOAT_TO_WORD: SBASSMAT_ADD F, W
2EE5 746

BASSMAT_ADD
1-017

C 2

BASSMAT_ADD - Add 2 arrays giving a thi 15-SEP-1984 23:39:02 VAX/VMS Macro V04-00 Page 40
6-SEP-1984 10:28:41 [BASRTL.SRC]BASMATADD.MAR;1 (5)

2EE5 748 FLOAT_TO_LONG: \$BASSMAT_ADD F, L
30D6 749

BASSMAT_ADD
1-017

D 2

BASSMAT_ADD - Add 2 arrays giving a thi 15-SEP-1984 23:39:02 VAX/VMS Macro V04-00 Page 41
6-SEP-1984 10:28:41 [BASRTL.SRC]BASMATADD.MAR;1 (5)

30D6 751 FLOAT_TO_FLOAT: \$BASSMAT_ADD F, F
32C4 752

BASSMAT_ADD
1-017

E 2

BASSMAT_ADD - Add 2 arrays giving a thi 15-SEP-1984 23:39:02 VAX/VMS Macro V04-00 Page 42
6-SEP-1984 10:28:41 [BASRTL.SRC]BASMATADD.MAR;1 (5)

32C4 754 FLOAT_TO_DOUBLE: \$BASSMAT_ADD F, D
34CD 755

BASSMAT_ADD
1-017

F 2

BASSMAT_ADD - Add 2 arrays giving a thi 15-SEP-1984 23:39:02 VAX/VMS Macro V04-00 Page 43
6-SEP-1984 10:28:41 [BASRTL.SRC]BASMATADD.MAR;1 (5)

34CD 757 FLOAT_TO_GFLOAT: \$BASSMAT_ADD F, G
36C6 758

BASSMAT_ADD
1-017

G 2

BASSMAT_ADD - Add 2 arrays giving a thi 15-SEP-1984 23:39:02 VAX/VMS Macro V04-00 Page 44
6-SEP-1984 10:28:41 [BASRTL.SRC]BASMATADD.MAR;1 (5)

36C6 760 FLOAT_TO_HFLOAT: SBASSMAT_ADD F, H
38BF 761

B
1

```

38BF 763 ;+
38BF 764 ; Source array is a double array. Now differentiate on the destination type.
38BF 765 ; -
38BF 766
05 06 02 A3 8F 38BF 767 DOUBLE: CASEB DSC$B_DTYPE(R3), #DSC$K_DTYPE_B, #<DSC$K_DTYPE_D - DSC$K_DTYPE_B>
002D' 38C4 768 1$: .WORD DOUBLE_TO_BYTE-1$ ; code for byte dtype
0231' 38C6 769 .WORD DOUBLE_TO_WORD-1$ ; code for word dtype
0435' 38C8 770 .WORD DOUBLE_TO_LONG-1$ ; code for long dtype
C836' 38CA 771 .WORD ERR_DATTYPERR-1$ ; quad not supported
0639' 38CC 772 .WORD DOUBLE_TO_FLOAT-1$ ; code for float dtype
083D' 38CE 773 .WORD DOUBLE_TO_DOUBL-1$ ; code for double dtype
38D0 774
1B 02 A3 91 38D0 775 CMPB DSC$B_DTYPE(R3), #DSC$K_DTYPE_G
03 12 38D4 776 BNEQ 2$
0A16 31 38D6 777 BRW DOUBLE_TO_GFLOA
38D9 778
1C 02 A3 91 38D9 779 2$: CMPB DSC$B_DTYPE(R3), #DSC$K_DTYPE_H
03 12 38DD 780 BNEQ 3$
0C0A 31 38DF 781 BRW DOUBLE_TO_HFLOA
38E2 782
18 02 A3 91 38E2 783 3$: CMPB DSC$B_DTYPE(R3), #DSC$K_DTYPE_DSC ; array of descriptors?
06 12 38E6 784 BNEQ 4$ ; branch if not
53 04 A3 D0 38E8 785 MOVL 4(R3), R3 ; move addr of desc in R3
D1 11 38EC 786 BRB DOUBLE ; CASE again on dtype in desc
38EE 787
C809 31 38EE 788 4$: BRW ERR_DATTYPERR ; unsupported dtype
38F1 789
38F1 790 ;+
38F1 791 ; Now type of source and destination arrays are known. Use the macro to
38F1 792 ; generate the code for each case
38F1 793 ; -
38F1 794

```


BASSMAT_ADD
1-017

I 2

BASSMAT_ADD - Add 2 arrays giving a thi 15-SEP-1984 23:39:02 VAX/VMS Macro V04-00 Page 46
6-SEP-1984 10:28:41 [BASRTL.SRC]BASMATADD.MAR;1 (5)

38F1 796 DOUBLE_TO_BYTE: \$BASSMAT_ADD D, B
3AF5 797

B
1

BASSMAT_ADD
1-017

J 2

BASSMAT_ADD - Add 2 arrays giving a thi 15-SEP-1984 23:39:02 VAX/VMS Macro V04-00 Page 47
6-SEP-1984 10:28:41 [BASRTL.SRC]BASMATADD.MAR;1 (5)

3AF5 799 DOUBLE_TO_WORD: \$BASSMAT_ADD D, W
3CF9 800

BASSMAT_ADD
1-017

K 2

BASSMAT_ADD - Add 2 arrays giving a thi 15-SEP-1984 23:39:02 VAX/VMS Macro V04-00 Page 48
6-SEP-1984 10:28:41 [BASRTL.SRC]BASMATADD.MAR;1 (5)

3CF9 802 DOUBLE_TO_LONG: \$BASSMAT_ADD D, L
3EFD 803

BASSMAT_ADD
1-017

L 2

BASSMAT_ADD - Add 2 arrays giving a thi 15-SEP-1984 23:39:02 VAX/VMS Macro V04-00 Page 49
6-SEP-1984 10:28:41 [BASRTL.SRC]BASMATADD.MAR;1 (5)

3EFD 805 DOUBLE_TO_FLOAT: \$BASSMAT_ADD D, F
4101 806

BASSMAT_ADD
1-017

M 2

BASSMAT_ADD - Add 2 arrays giving a thi 15-SEP-1984 23:39:02 VAX/VMS Macro V04-00 Page 50
6-SEP-1984 10:28:41 [BASRTL.SRC]BASMATADD.MAR;1 (5)

4101 808 DOUBLE_TO_DOUBL: \$BASSMAT_ADD D, D
42EF 809

BASSMAT_ADD
1-017

N 2

BASSMAT_ADD - Add 2 arrays giving a thi 15-SEP-1984 23:39:02 VAX/VMS Macro V04-00 Page 51
6-SEP-1984 10:28:41 [BASRTL.SRC]BASMATADD.MAR;1 (5)

42EF 811 DOUBLE_TO_GFLQA: \$BASSMAT_ADD D, G
44EC 812

BASSMAT_ADD
1-017

B 3

BASSMAT_ADD - Add 2 arrays giving a thi 15-SEP-1984 23:39:02 VAX/VMS Macro V04-00 Page 52
6-SEP-1984 10:28:41 [BASRTL.SRC]BASMATADD.MAR;1 (5)

44EC 814 DOUBLE_TO_HFLOA: \$BASSMAT_ADD D, H
46E5 815

BASSMAT_ADD - Add 2 arrays giving a thi

```

46E5 817 :+
46E5 818 : Source array is a gfloat array. Now differentiate on the destination type.
46E5 819 :-
46E5 820
05 06 02 A3 8F 46E5 821 GFLOAT: CASEB DSC$B_DTYPE(R3), #DSC$K_DTYPE_B, #<DSC$K_DTYPE_D - DSC$K_DTYPE_B>
002D' 46EA 822 1$: .WORD GFLOAT_TO_BYTE-1$ ; code for byte dtype
0227' 46EC 823 .WORD GFLOAT_TO_WORD-1$ ; code for word dtype
0421' 46EE 824 .WORD GFLOAT_TO_LONG-1$ ; code for long dtype
BA10' 46F0 825 .WORD ERR_DATTYPERR-1$ ; quad not supported
061B' 46F2 826 .WORD GFLOAT_TO_FLOAT-1$ ; code for float dtype
0815' 46F4 827 .WORD GFLOAT_TO_DOUBL-1$ ; code for dbl dtype
46F6 828
1B 02 A3 91 46F6 829 CMPB DSC$B_DTYPE(R3), #DSC$K_DTYPE_G
03 12 46FA 830 BNEQ 2$
09FE 31 46FC 831 BRW GFLOAT_TO_GFLOA
46FF 832
1C 02 A3 91 46FF 833 2$: CMPB DSC$B_DTYPE(R3), #DSC$K_DTYPE_H
03 12 4703 834 BNEQ 3$
0BF1 31 4705 835 BRW GFLOAT_TO_HFLOA
4708 836
18 02 A3 91 4708 837 3$: CMPB DSC$B_DTYPE(R3), #DSC$K_DTYPE_DSC ; array of descriptors?
06 12 470C 838 BNEQ 4$ ; branch if not
53 04 A3 D0 470E 839 MOVL 4(R3), R3 ; move addr of desc in R3
D1 11 4712 840 BRB GFLOAT ; CASE again for dtype in desc
4714 841
B9E3 31 4714 842 4$: BRW ERR_DATTYPERR ; unsupported dtype
4717 843
4717 844 :+
4717 845 : Now type of source and destination arrays are known. Use the macro to
4717 846 : generate the code for each case
4717 847 :-
4717 848

```


BASSMAT_ADD
1-017

D 3

BASSMAT_ADD - Add 2 arrays giving a thi 15-SEP-1984 23:39:02 VAX/VMS Macro V04-00 Page 54
6-SEP-1984 10:28:41 [BASRTL.SRC]BASMATADD.MAR;1 (5)

4717 850 GFLOAT_TO_BYTE: \$BASSMAT_ADD G, B
4911 851

BASSMAT_ADD
1-017

E 3

BASSMAT_ADD - Add 2 arrays giving a thi 15-SEP-1984 23:39:02 VAX/VMS Macro V04-00 Page 55
6-SEP-1984 10:28:41 [BASRTL.SRC]BASMATADD.MAR;1 (5)

4911 853 GFLOAT_TO_WORD: \$BASSMAT_ADD G, W
480B 854

BASSMAT_ADD
1-017

F 3

BASSMAT_ADD - Add 2 arrays giving a thi 15-SEP-1984 23:39:02 VAX/VMS Macro V04-00 Page 56
6-SEP-1984 10:28:41 [BASRTL.SRC]BASMATADD.MAR;1 (5)

4B0B 856 GFLOAT_TO_LONG: \$BASSMAT_ADD G, L
4D05 857

BASSMAT_ADD
1-017

G 3

BASSMAT_ADD - Add 2 arrays giving a thi 15-SEP-1984 23:39:02 VAX/VMS Macro V04-00 Page 57
6-SEP-1984 10:28:41 [BASRTL.SRC]BASMATADD.MAR;1 (5)

4D05 859 GFLOAT_TO_FLOAT: \$BASSMAT_ADD G, F
4EFF 860

BASSMAT_ADD
1-017

H 3

BASSMAT_ADD - Add 2 arrays giving a thi 15-SEP-1984 23:39:02 VAX/VMS Macro V04-00 Page 58
6-SEP-1984 10:28:41 [BASRTL.SRC]BASMATADD.MAR;1 (5)

4EFF 862
4EFF 863 GFLOAT_TO_DOUBL: \$BASSMAT_ADD G, D
50FD 864

BASSMAT_ADD
1-017

I 3

BASSMAT_ADD - Add 2 arrays giving a thi 15-SEP-1984 23:39:02 VAX/VMS Macro V04-00 Page 59
6-SEP-1984 10:28:41 [BASRTL.SRC]BASMATADD.MAR;1 (5)

50FD 866 GFLOAT_TO_GFLOA: SBASSMAT_ADD G, G
52F9 867

BASSMAT_ADD
1-017

J 3

BASSMAT_ADD - Add 2 arrays giving a thi 15-SEP-1984 23:39:02 VAX/VMS Macro V04-00 Page 60
6-SEP-1984 10:28:41 [BASRTL.SRC]BASMATADD.MAR;1 (5)

52F9 869 GFLOAT_TO_HFLOA: SBASSMAT_ADD G, H
54F9 870

BASSMAT_ADD - Add 2 arrays giving a thi

```

54F9 872 :+
54F9 873 : Source array is an hfloat array. Now differentiate on the destination type.
54F9 874 :-
54F9 875
05 06 02 A3 8F 54F9 876 HFLOAT: CASEB DSC$B_DTYPE(R3), #DSC$K_DTYPE_B, #<DSC$K_DTYPE_D - DSC$K_DTYPE_B>
002D' 54FE 877 1$: .WORD HFLOAT_TO_BYTE-1$ ; code for byte dtype
0227' 5500 878 .WORD HFLOAT_TO_WORD-1$ ; code for word dtype
0421' 5502 879 .WORD HFLOAT_TO_LONG-1$ ; code for long dtype
ABFC' 5504 880 .WORD ERR_DATTYPERR-1$ ; quad not supported
061B' 5506 881 .WORD HFLOAT_TO_FLOAT-1$ ; code for float dtype
0815' 5508 882 .WORD HFLOAT_TO_DOUBL-1$ ; code for double dtype
550A 883
1B 02 A3 91 550A 884 CMPB DSC$B_DTYPE(R3), #DSC$K_DTYPE_G
03 12 550E 885 BNEQ 2$
09FA 31 5510 886 BRW HFLOAT_TO_GFLOA
5513 887
1C 02 A3 91 5513 888 2$: CMPB DSC$B_DTYPE(R3), #DSC$K_DTYPE_H
03 12 5517 889 BNEQ 3$
0BF1 31 5519 890 BRW HFLOAT_TO_HFLOA
551C 891
18 02 A3 91 551C 892 3$: CMPB DSC$B_DTYPE(R3), #DSC$K_DTYPE_DSC ; array of descriptors?
06 12 5520 893 BNEQ 4$ ; branch if not
53 04 A3 D0 5522 894 MOVL 4(R3), R3 ; move addr of desc in R3
D1 11 5526 895 BRB HFLOAT ; CASE again for dtype in desc
5528 896
ABCF 31 5528 897 4$: BRW ERR_DATTYPERR ; unsupported dtype
552B 898
552B 899 :+
552B 900 : Now type of source and destination arrays are known. Use the macro to
552B 901 : generate the code for each case
552B 902 :-
552B 903

```


BASSMAT_ADD
1-017

L 3

BASSMAT_ADD - Add 2 arrays giving a thi 15-SEP-1984 23:39:02 VAX/VMS Macro V04-00 Page 62
6-SEP-1984 10:28:41 [BASRTL.SRC]BASMATADD.MAR;1 (5)

552B 905 HFLOAT_TO_BYTE: \$BASSMAT_ADD H, B
5725 906

BASSMAT_ADD
1-017

M 3

BASSMAT_ADD - Add 2 arrays giving a thi 15-SEP-1984 23:39:02 VAX/VMS Macro V04-00 Page 63
6-SEP-1984 10:28:41 [BASRTL.SRC]BASSMATADD.MAR;1 (5)

5725 908 HFLOAT_TO_WORD: SBASSMAT_ADD H, W
591F 909

BASSMAT_ADD
1-017

N 3

BASSMAT_ADD - Add 2 arrays giving a thi 15-SEP-1984 23:39:02 VAX/VMS Macro V04-00 Page 64
6-SEP-1984 10:28:41 [BASRTL.SRC]BASMATADD.MAR;1 (5)

591F 911 HFLOAT_TO_LONG: \$BASSMAT_ADD H, L
5B19 912

BASSMAT_ADD
1-017

B 4

BASSMAT_ADD - Add 2 arrays giving a thi 15-SEP-1984 23:39:02 VAX/VMS Macro V04-00 Page 65
6-SEP-1984 10:28:41 [BASRTL.SRC]BASSMATADD.MAR;1 (5)

5B19 914 HFLOAT_TO_FLOAT: SBASSMAT_ADD H, F
5D13 915

BASSMAT_ADD
1-017

C 4

BASSMAT_ADD - Add 2 arrays giving a thi 15-SEP-1984 23:39:02 VAX/VMS Macro V04-00 Page 66
6-SEP-1984 10:28:41 [BASRTL.SRC]BASMATADD.MAR;1 (5)

5D13 917 HFLOAT_TO_DOUBL: \$BASSMAT_ADD H, D
5F0D 918

BASSMAT_ADD
1-017

D 4

BASSMAT_ADD - Add 2 arrays giving a thi 15-SEP-1984 23:39:02 VAX/VMS Macro V04-00 Page 67
6-SEP-1984 10:28:41 [BASRTL.SRC]BASMATADD.MAR;1 (5)

5F0D 920 HFLOAT_TO_GFLOA: \$BASSMAT_ADD H, G
610D 921

BASSMAT_ADD
1-017

E 4

BASSMAT_ADD - Add 2 arrays giving a thi 15-SEP-1984 23:39:02 VAX/VMS Macro V04-00 Page 68
6-SEP-1984 10:28:41 [BASRTL.SRC]BASSMATADD.MAR;1 (5)

610D 923 HFLOAT_TO_HFLOA: \$BASSMAT_ADD H, H

BASSMAT_ADD - Add 2 arrays giving a thi

```

6309 925 :+
6309 926 : Add has been in byte. Determine destination type to convert to dest.
6309 927 :-
6309 928
6309 929 DEST_CASE B:
05 06 55 5A DO 6309 930 30$: MOVL R10, R5 ; save original pointer
02 A5 8F 630C 931 31$: CASEB DSC$B_DTYPE(R5), #DSC$K_DTYPE_B, #<DSC$K_DTYPE_D - DSC$K_DTYPE_B>
019A' 6311 932 1$: .WORD STORE_BYTE-1$ ; no conversion needed
027A' 6313 933 .WORD DEST_B_TO_W-1$ ; code for word dtype
0389' 6315 934 .WORD DEST_B_TO_L-1$ ; code for long dtype
9DE9' 6317 935 .WORD ERR_DATTYPERR-1$ ; quad not supported
0498' 6319 936 .WORD DEST_B_TO_F-1$ ; code for float dtype
05A7' 631B 937 .WORD DEST_B_TO_D-1$ ; code for double dtype
631D 938
631D 939 :+
631D 940 : To avoid having to specify 'ERR_DATTYPERR' for all the cases in between
631D 941 : double and gfloat (dtypes 12 to 26), check for gfloat and hfloat separately.
631D 942 :-
631D 943
1B 02 A5 91 631D 944 CMPB DSC$B_DTYPE(R5), #DSC$K_DTYPE_G
03 12 6321 945 BNEQ 2$ ; dest not gfloat
070B 31 6323 946 BRW DEST_B_TO_G
6326 947
1C 02 A5 91 6326 948 2$: CMPB DSC$B_DTYPE(R5), #DSC$K_DTYPE_H
03 12 632A 949 BNEQ 3$ ; dest not hfloat
081A 31 632C 950 BRW DEST_B_TO_H
632F 951
18 02 A5 91 632F 952 3$: CMPB DSC$B_DTYPE(R5), #DSC$K_DTYPE_DSC ; array of descriptors?
06 12 6333 953 BNEQ 4$ ; branch if not
55 04 A5 DO 6335 954 MOVL 4(R5), R5 ; move addr of desc to R5
D1 11 6339 955 BRB 31$ ; CASE again for dtype in desc
633B 956
9DBC 31 633B 957 4$: BRW ERR_DATTYPERR ; if we get here, must be an
633E 958 ; unsupported data type
633E 959

```


BASSMAT_ADD - Add 2 arrays giving a thi

```

633E 961 :+
633E 962 : Add has been in word. Determine destination type to convert to dest.
633E 963 :-
633E 964
633E 965 DEST_CASE W:
05 06 55 02 5A DO 633E 966 32$:- MOVL R10, R5 ; save original pointer
06 02 A5 8F 6341 967 33$: CASEB DSC$B_DTYPE(R5), #DSC$K_DTYPE_B, #<DSC$K_DTYPE_D - DSC$K_DTYPE_B>
0136' 6346 968 1$: .WORD DEST_W_TO_B-1$ ; code for byte dtype
0274' 6348 969 .WORD STORE_WORD-1$ ; no conversion needed
0359' 634A 970 .WORD DEST_W_TO_L-1$ ; code for long dtype
9DB4' 634C 971 .WORD ERR_DATTYPERR-1$ ; quad not supported
0468' 634E 972 .WORD DEST_W_TO_F-1$ ; code for float dtype
0585' 6350 973 .WORD DEST_W_TO_D-1$ ; code for double dtype
6352 974
1B 02 A5 91 6352 975 CMPB DSC$B_DTYPE(R5), #DSC$K_DTYPE_G
03 12 6356 976 BNEQ 2$ ; dest not gfloat
06DC 31 6358 977 BRW DEST_W_TO_G
635B 978
1C 02 A5 91 635B 979 2$: CMPB DSC$B_DTYPE(R5), #DSC$K_DTYPE_H
03 12 635F 980 BNEQ 3$ ; dest not hfloat
07EB 31 6361 981 BRW DEST_W_TO_H
6364 982
18 02 A5 91 6364 983 3$: CMPB DSC$B_DTYPE(R5), #DSC$K_DTYPE_DSC ; array of descriptors?
06 12 6368 984 BNEQ 4$ ; branch if not
55 04 A5 DO 636A 985 MOVL 4(R5), R5 ; move addr of desc to R5
D1 11 636E 986 BRB 33$ ; CASE again for dtype in desc
6370 987
9DB7 31 6370 988 4$: BRW ERR_DATTYPERR ; unsupported dtype
6373 989 :+
6373 990 : Add has been in long. Determine destination type to convert to dest.
6373 991 :-
6373 992 DEST_CASE L:
05 06 55 02 5A DO 6373 993 34$:- MOVL R10, R5 ; save original pointer
06 02 A5 8F 6376 994 35$: CASEB DSC$B_DTYPE(R5), #DSC$K_DTYPE_B, #<DSC$K_DTYPE_D - DSC$K_DTYPE_B>
0106' 637B 995 1$: .WORD DEST_L_TO_B-1$ ; code for byte dtype
0215' 637D 996 .WORD DEST_L_TO_W-1$ ; code for word dtype
034E' 637F 997 .WORD STORE_LONG-1$ ; no conversion needed
9D7F' 6381 998 .WORD ERR_DATTYPERR-1$ ; quad not supported
0438' 6383 999 .WORD DEST_L_TO_F-1$ ; code for float dtype
0563' 6385 1000 .WORD DEST_L_TO_D-1$ ; code for double dtype
6387 1001
1B 02 A5 91 6387 1002 CMPB DSC$B_DTYPE(R5), #DSC$K_DTYPE_G
03 12 638B 1003 BNEQ 2$ ; dest not gfloat
06AD 31 638D 1004 BRW DEST_L_TO_G
6390 1005
1C 02 A5 91 6390 1006 2$: CMPB DSC$B_DTYPE(R5), #DSC$K_DTYPE_H
03 12 6394 1007 BNEQ 3$ ; dest not hfloat
07BC 31 6396 1008 BRW DEST_L_TO_H
6399 1009
18 02 A5 91 6399 1010 3$: CMPB DSC$B_DTYPE(R5), #DSC$K_DTYPE_DSC ; array of descriptors?
06 12 639D 1011 BNEQ 4$ ; branch if not
55 04 A5 DO 639F 1012 MOVL 4(R5), R5 ; move addr of desc to R5
D1 11 63A3 1013 BRB 35$ ; CASE again for dtype in desc
63A5 1014
9D52 31 63A5 1015 4$: BRW ERR_DATTYPERR ; unsupported dtype
63A8 1016 :+
63A8 1017 : Add has been in float. Determine destination type to convert to dest.

```

BASSMAT_ADD - Add 2 arrays giving a thi

```

        63A8 1018 :-
        63A8 1019
05 06 55 5A  DO 63A8 1020 DEST_CASE_F:
      02 A5 8F 63A8 1021 36$: MOVL R10, R5 ; save original pointer
      00D6 63AB 1022 37$: CASEB DSC$B_DTYPE(R5), #DSC$K_DTYPE_B, #<DSC$K_DTYPE_D - DSC$K_DTYPE_B>
      01E5 63B0 1023 1$: .WORD DEST_F_TO_B-1$ ; code for byte dtype
      02F4 63B2 1024 .WORD DEST_F_TO_W-1$ ; code for word dtype
      9D4A 63B4 1025 .WORD DEST_F_TO_L-1$ ; code for long dtype
      0428 63B6 1026 .WORD ERR_DATTYPERR-1$ ; quad not supported
      0541 63B8 1027 .WORD STORE_FLOAT-1$ ; no conversion needed
      63BA 1028 .WORD DEST_F_TO_D-1$ ; code for double dtype
      63BC 1029
1B 02 A5 91 63BC 1030 CMPB DSC$B_DTYPE(R5), #DSC$K_DTYPE_G
      03 12 63C0 1031 BNEQ 2$ ; dest not gfloat
      067E 31 63C2 1032 BRW DEST_F_TO_G
      63C5 1033
1C 02 A5 91 63C5 1034 2$: CMPB DSC$B_DTYPE(R5), #DSC$K_DTYPE_H
      03 12 63C9 1035 BNEQ 3$ ; dest not hfloat
      078D 31 63CB 1036 BRW DEST_F_TO_H
      63CE 1037
18 02 A5 91 63CE 1038 3$: CMPB DSC$B_DTYPE(R5), #DSC$K_DTYPE_DSC ; array of descriptors?
      06 12 63D2 1039 BNEQ 4$ ; branch if not
55 04 A5 D0 63D4 1040 MOVL 4(R5), R5 ; move addr of desc to R5
      D1 11 63D8 1041 BRB 37$ ; CASE again for dtype in desc
      63DA 1042
      9D1D 31 63DA 1043 4$: BRW ERR_DATTYPERR ; unsupported dtype
      63DD 1044 ;+
      63DD 1045 ; Add has been in double. Determine destination type to convert to dest.
      63DD 1046 :-
      63DD 1047
      63DD 1048 DEST_CASE_D:
05 06 55 5A  DO 63DD 1049 38$: MOVL R10, R5 ; save original pointer
      02 A5 8F 63E0 1050 39$: CASEB DSC$B_DTYPE(R5), #DSC$K_DTYPE_B, #<DSC$K_DTYPE_D - DSC$K_DTYPE_B>
      00A6 63E5 1051 1$: .WORD DEST_D_TO_B-1$ ; code for byte dtype
      01B5 63E7 1052 .WORD DEST_D_TO_W-1$ ; code for word dtype
      02C4 63E9 1053 .WORD DEST_D_TO_L-1$ ; code for long dtype
      9D15 63EB 1054 .WORD ERR_DATTYPERR-1$ ; quad not supported
      03D3 63ED 1055 .WORD DEST_D_TO_F-1$ ; code for float dtype
      056C 63EF 1056 .WORD STORE_DOUBLE-1$ ; no conversion needed
      63F1 1057
1B 02 A5 91 63F1 1058 CMPB DSC$B_DTYPE(R5), #DSC$K_DTYPE_G
      03 12 63F5 1059 BNEQ 3$
      064F 31 63F7 1060 BRW DEST_D_TO_G
      63FA 1061
1C 02 A5 91 63FA 1062 3$: CMPB DSC$B_DTYPE(R5), #DSC$K_DTYPE_H
      03 12 63FE 1063 BNEQ 2$ ; dest not hfloat
      075E 31 6400 1064 BRW DEST_D_TO_H
      6403 1065
18 02 A5 91 6403 1066 2$: CMPB DSC$B_DTYPE(R5), #DSC$K_DTYPE_DSC ; array of descriptors?
      06 12 6407 1067 BNEQ 4$ ; branch if not
55 04 A5 D0 6409 1068 MOVL 4(R5), R5 ; move addr of desc to R5
      D1 11 640D 1069 BRB 39$ ; CASE again for dtype in desc
      640F 1070
      9CE8 31 640F 1071 4$: BRW ERR_DATTYPERR ; unsupported dtype
      6412 1072 ; (or gfloat, which is not
      6412 1073 ; supported w/dbl)
      6412 1074 ;+

```

BASSMAT_ADD - Add 2 arrays giving a thi

```
6412 1075 ; Add has been in gfloat. Determine destination type to convert to dest.
6412 1076 ; -
6412 1077
6412 1078 DEST_CASE G:
05 06 55 02 5A 00 8F 6412 1079 40$: MOVL R10, R5 ; save original pointer
0087' 641A 1081 41$: CASEB DSC$B_DTYPE(R5), #DSC$K_DTYPE_B, #<DSC$K_DTYPE_D - DSC$K_DTYPE_B>
0196' 641C 1082 1$: .WORD DEST_G_TO_B-1$ ; code for byte dtype
02A5' 641E 1083 .WORD DEST_G_TO_W-1$ ; code for word dtype
9CE0' 6420 1084 .WORD DEST_G_TO_L-1$ ; code for long dtype
03B4' 6422 1085 .WORD ERR_DATTYPERR-1$ ; quad not supported
04F0' 6424 1086 .WORD DEST_G_TO_F-1$ ; code for float dtype
6426 1087 .WORD DEST_G_TO_D-1$ ; code for double dtype
1B 02 A5 91 6426 1088 CMPB DSC$B_DTYPE(R5), #DSC$K_DTYPE_G
03 12 642A 1089 BNEQ 2$ ; dest not gfloat
0635 31 642C 1090 BRW STORE_GFLOAT
642F 1091
1C 02 A5 91 642F 1092 2$: CMPB DSC$B_DTYPE(R5), #DSC$K_DTYPE_H
03 12 6433 1093 BNEQ 3$
0740 31 6435 1094 BRW DEST_G_TO_H
6438 1095
18 02 A5 91 6438 1096 3$: CMPB DSC$B_DTYPE(R5), #DSC$K_DTYPE_DSC ; array of descriptors?
06 12 643C 1097 BNEQ 4$ ; branch if not
55 04 A5 00 643E 1098 MOVL 4(R5), R5 ; move addr of desc to R5
D1 11 6442 1099 BRB 41$ ; CASE again for dtype in desc
6444 1100
9CB3 31 6444 1101 4$: BRW ERR_DATTYPERR ; unsupported dtype
6447 1102 ; (note that dbl is unsupported)
6447 1103 ; with gfloat)
6447 1104
6447 1105 ;+
6447 1106 ; Add has been in hfloat. Determine destination type to convert to dest.
6447 1107 ;
6447 1108
6447 1109 DEST_CASE H:
05 06 55 02 5A 00 8F 6447 1110 42$: MOVL R10, R5 ; save original pointer
0058' 644A 1111 43$: CASEB DSC$B_DTYPE(R5), #DSC$K_DTYPE_B, #<DSC$K_DTYPE_D - DSC$K_DTYPE_B>
0167' 644F 1112 1$: .WORD DEST_H_TO_B-1$ ; code for byte dtype
0276' 6451 1113 .WORD DEST_H_TO_W-1$ ; code for word dtype
9CAB' 6453 1114 .WORD DEST_H_TO_L-1$ ; code for long dtype
0385' 6455 1115 .WORD ERR_DATTYPERR-1$ ; quad not supported
04EB' 6457 1116 .WORD DEST_H_TO_F-1$ ; code for float dtype
6459 1117 .WORD DEST_H_TO_D-1$ ; code for dbl dtype
645B 1118
1B 02 A5 91 645B 1119 CMPB DSC$B_DTYPE(R5), #DSC$K_DTYPE_G
03 12 645F 1120 BNEQ 2$ ; dest not gfloat
05FC 31 6461 1121 BRW DEST_H_TO_G
6464 1122
1C 02 A5 91 6464 1123 2$: CMPB DSC$B_DTYPE(R5), #DSC$K_DTYPE_H
03 12 6468 1124 BNEQ 3$ ; dest not hfloat
070F 31 646A 1125 BRW STORE_HFLOAT
646D 1126
18 02 A5 91 646D 1127 3$: CMPB DSC$B_DTYPE(R5), #DSC$K_DTYPE_DSC ; array of descriptors?
06 12 6471 1128 BNEQ 4$ ; branch if not
55 04 A5 00 6473 1129 MOVL 4(R5), R5 ; move addr of desc to R5
D1 11 6477 1130 BRB 43$ ; CASE again for dtype in desc
6479 1131
```

```

BASSMAT_ADD - Add 2 arrays giving a thi
9C7E 31 6479 1132 4$: BRW ERR_DATTYPERR ; unsupported dtype
        647C 1133
50 50 33 647C 1134 DEST_W_TO_B:
    2A 11 647C 1135 CVTWB RO, RO ; convert
        647F 1136 BRB STORE_BYTE ; go store
        6481 1137
50 50 F6 6481 1138 DEST_L_TO_B:
    25 11 6481 1139 CVTLB RO, RO ; convert
        6484 1140 BRB STORE_BYTE ; go store
        6486 1141
50 50 48 6486 1142 DEST_F_TO_B:
    20 11 6486 1143 CVTFB RO, RO ; convert
        6489 1144 BRB STORE_BYTE ; go store
        648B 1145
50 7E 50 70 648B 1146 DEST_D_TO_B:
    0C AD D0 648B 1147 MOVD RO, -(SP) ; save double
00000000 GF 16 648E 1148 MOVL SF$L_SAVE_FP(FP), RO ; pass FP to get scale
50 8E 50 67 6492 1149 JSB G^BAS$$$SCALE R1 ; get scale in R0 & R1
50 50 68 6498 1150 DIVD3 RO, (SP)+, RO ; descale for byte
    0A 11 649C 1151 CVTDB RO, RO ; convert to byte
        649F 1152 BRB STORE_BYTE
        64A1 1153
50 50 48FD 64A1 1154 DEST_G_TO_B:
    04 11 64A1 1155 CVTGB RO, RO ; convert
        64A5 1156 BRB STORE_BYTE ; go store
        64A7 1157
50 50 68FD 64A7 1158 DEST_H_TO_B:
        64A7 1159 CVTHB RO, RO ; convert
        64AB 1160 ; fall into store
        64AB 1161 STORE_BYTE:
51 5A D0 64AB 1162 MOVL R10, R1 ; pointer to dest descriptor
52 08 AE D0 64AE 1163 MOVL lower_bnd1+4(SP), R2 ; current row (extra longword
        64B2 1164 ; on top of stack for jsb)
53 5B D0 64B2 1165 MOVL R11, R3 ; current column
28 AE 50 90 64B5 1166 MOVB RO, DATA+4(SP)
        64B9 1167 ;+
        64B9 1168 ; Redefine the following offsets for the call to the STORE macro. The
        64B9 1169 ; BSBW to here added 4 to the stack.
        64B9 1170 ;-
        64B9 1171 ;-
00000020 64B9 1172 value_desc = 32
00000020 64B9 1173 str_len = 32
00000022 64B9 1174 dtype = 34
00000023 64B9 1175 class = 35
00000024 64B9 1176 pointer = 36
00000028 64B9 1177 data = 40
        64B9 1178
        64B9 1179 STORE B ; store
        658A 1180 ;+
        658A 1181 ; Restore the following offsets.
        658A 1182 ;-
        658A 1183 ;-
0000001C 658A 1184 value_desc = 28
0000001C 658A 1185 str_len = 28
0000001E 658A 1186 dtype = 30
0000001F 658A 1187 class = 31
00000020 658A 1188 pointer = 32

```

BASSMAT_ADD - Add 2 arrays giving a thi

```

00000024 658A 1189 data = 36
          658A 1190
          05 658A 1191 RSB
          658B 1192
          658B 1193 DEST_B_TO W:
50 50 99 658B 1194 CDTBW RO, RO ; convert
2A 11 658E 1195 BRB STORE_WORD ; go store
          6590 1196
          6590 1197 DEST_L_TO W:
50 50 F7 6590 1198 CDTLW RO, RO ; convert
25 11 6593 1199 BRB STORE_WORD ; go store
          6595 1200
          6595 1201 DEST_F_TO W:
50 50 49 6595 1202 CDTFW RO, RO ; convert
20 11 6598 1203 BRB STORE_WORD ; go store
          659A 1204
          659A 1205 DEST_D_TO W:
50 7E 50 70 659A 1206 MOVD RO, -(SP) ; save double
50 0C AD D0 659D 1207 MOVL SF$L SAVE FP(FP), RO ; pass FP to get scale
00000000 GF 16 65A1 1208 JSB G*BAS$$SCALE R1 ; get scale in R0 & R1
50 8E 50 67 65A7 1209 DIVD3 RO, (SP)+, RO ; descale for dest
          50 50 69 65AB 1210 CVDW RO, RO ; convert to word
          OA 11 65AE 1211 BRB STORE_WORD ; go store
          65B0 1212
          65B0 1213 DEST_G_TO W:
50 50 49FD 65B0 1214 CDTGW RO, RO ; convert
04 11 65B4 1215 BRB STORE_WORD ; go store
          65B6 1216
          65B6 1217 DEST_H_TO W:
50 50 69FD 65B6 1218 CDTHW RO, RO ; convert
          65BA 1219 ; fall into store
          65BA 1220
          65BA 1221 STORE_WORD:
50 51 5A D0 65BA 1222 MOVL R10, R1 ; pointer to dest descriptor
52 08 AE D0 65BD 1223 MOVL lower_bnd1+4(SP), R2 ; current row (extra longword
          65C1 1224 ; on top of stack for jsb)
          53 5B D0 65C1 1225 MOVL R11, R3 ; current column
28 AE 50 B0 65C4 1226 MOVW RO, DATA+4(SP)
          65C8 1227 ;+
          65C8 1228 ; Redefine the following offsets for the call to the STORE macro. The
          65C8 1229 ; BSBW to here added 4 to the stack.
          65C8 1230 ;-
          65C8 1231
00000020 65C8 1232 value_desc = 32
00000020 65C8 1233 str_len = 32
00000022 65C8 1234 dtype = 34
00000023 65C8 1235 class = 35
00000024 65C8 1236 pointer = 36
00000028 65C8 1237 data = 40
          65C8 1238
          65C8 1239 STORE W ; store
          6699 1240 ;+
          6699 1241 ; Restore the following offsets.
          6699 1242 ;-
          6699 1243
0000001C 6699 1244 value_desc = 28
0000001C 6699 1245 str_len = 28

```

BASSMAT_ADD - Add 2 arrays giving a thi

```

0000001E 6699 1246 dtype = 30
0000001F 6699 1247 class = 31
00000020 6699 1248 pointer = 32
00000024 6699 1249 data = 36
        6699 1250
        05 6699 1251          RSB          ; go continue loop
        669A 1252
50 50 98 669A 1253 DEST_B_TO L:
2A 11 669A 1254          CVTBL        RO, RO          ; convert
        669D 1255          BRB          STORE_LONG      ; go store
        669F 1256
50 50 32 669F 1257 DEST_W_TO L:
25 11 669F 1258          CVTWL        RO, RO          ; convert
        66A2 1259          BRB          STORE_LONG      ; go store
        66A4 1260
50 50 4A 66A4 1261 DEST_F_TO L:
20 11 66A4 1262          CVTFL        RO, RO          ; convert
        66A7 1263          BRB          STORE_LONG      ; go store
        66A9 1264
50 7E 50 70 66A9 1265 DEST_D_TO L:
OC AD DO 66A9 1266          MOVD        RO, -(SP)          ; save double
00000000 GF 16 66AC 1267          MOVL        SF$L SAVE FP(FP), RO ; pass FP to get scale
50 8E 50 67 66B0 1268          JSB          G*BAS$$$SCALE R1 ; get scale in R0 & R1
50 50 6A 66B6 1269          DIVD3       RO, (SP)+, RO ; descale for dest
OA 11 66BA 1270          CVTDL        RO, RO          ; convert
        66BD 1271          BRB          STORE_LONG      ; go store
        66BF 1272
50 50 4AFD 66BF 1273 DEST_G_TO L:
04 11 66BF 1274          CVTGL        RO, RO          ; convert
        66C3 1275          BRB          STORE_LONG      ; go store
        66C5 1276
50 50 6AFD 66C5 1277 DEST_H_TO L:
        66C9 1278          CVTHL        RO, RO          ; convert
        66C9 1279          ; fall into store
51 5A DO 66C9 1280 STORE_LONG:
08 AE DO 66CC 1281          MOVL        R10, R1          ; pointer to dest descriptor
53 5B DO 66D0 1282          MOVL        lower_bnd1+4(SP), R2 ; current row (extra longword
28 AE 50 DO 66D0 1283          ; on stack for jsb)
        66D0 1284          MOVL        R11, R3          ; current column
        66D3 1285          MOVL        RO, DATA+4(SP)
        66D7 1286          ;+
        66D7 1287          ; Redefine the following offsets for the call to the STORE macro. The
        66D7 1288          ; BSBW to here added 4 to the stack.
        66D7 1289          ;-
        66D7 1290
00000020 66D7 1291 value_desc = 32
00000020 66D7 1292 str_len = 32
00000022 66D7 1293 dtype = 34
00000023 66D7 1294 class = 35
00000024 66D7 1295 pointer = 36
00000028 66D7 1296 data = 40
        66D7 1297
        66D7 1298          STORE L          ; store
        67A8 1299          ;+
        67A8 1300          ; Restore the following offsets.
        67A8 1301          ;-
        67A8 1302

```

BASSMAT_ADD - Add 2 arrays giving a thi

```

0000001C 67A8 1303 value_desc = 28
0000001C 67A8 1304 str_len = 28
0000001E 67A8 1305 dtype = 30
0000001F 67A8 1306 class = 31
00000020 67A8 1307 pointer = 32
00000024 67A8 1308 data = 36
        67A8 1309
        05 67A8 1310          RSB          ; go continue loop
        67A9 1311
50 50 4C 67A9 1312 DEST_B_TO F:
2A 11 67A9 1313          CVTBF      RO, RO          ; convert
        67AC 1314          BRB        STORE_FLOAT      ; go store
        67AE 1315
50 50 4D 67AE 1316 DEST_W_TO F:
25 11 67AE 1317          CVTWF      RO, RO          ; convert
        67B1 1318          BRB        STORE_FLOAT      ; go store
        67B3 1319
50 50 4E 67B3 1320 DEST_L_TO F:
20 11 67B3 1321          CVTLF      RO, RO          ; convert
        67B6 1322          BRB        STORE_FLOAT      ; go store
        67B8 1323
50 7E 50 70 67B8 1324 DEST_D_TO F:
00000000 50 0C AD D0 67B8 1325          MOVD      RO, -(SP)          ; save double
00000000 50 8E 50 67 67BB 1326          MOVL      SF$L SAVE_FP(FP), RO ; pass FP to get scale
50 8E 50 67 67BF 1327          JSB      G^BASS$SCALE R1 ; get scale in R0 & R1
        50 50 76 67C5 1328          DIVD3     RO, (SP)+, RO ; descale for dest
        50 50 76 67C9 1329          CVTDF      RO, RO          ; convert
        0A 11 67CC 1330          BRB        STORE_FLOAT      ; go store
        67CE 1331
50 50 33FD 67CE 1332 DEST_G_TO F:
04 11 67CE 1333          CVTGF      RO, RO          ; convert
        67D2 1334          BRB        STORE_FLOAT      ; go store
        67D4 1335
50 50 F6FD 67D4 1336 DEST_H_TO F:
        67D4 1337          CVTHF      RO, RO          ; convert
        67D8 1338          ; fall into store
        67D8 1339 STORE_FLOAT:
51 5A D0 67D8 1340          MOVL      R10, R1          ; pointer to dest descriptor
52 08 AE D0 67DB 1341          MOVL      lower_bnd1+4(SP), R2 ; current row (extra longword
        67DF 1342          ; on stack for jsb)
53 5B D0 67DF 1343          MOVL      R11, R3          ; current column
28 AE 50 50 67E2 1344          MOVF      RO, DATA+4(SP)
        67E6 1345          ;+
        67E6 1346          ; Redefine the following offsets for the call to the STORE macro. The
        67E6 1347          ; BSBW to here added 4 to the stack.
        67E6 1348          ; -
        67E6 1349
00000020 67E6 1350 value_desc = 32
00000020 67E6 1351 str_len = 32
00000022 67E6 1352 dtype = 34
00000023 67E6 1353 class = 35
00000024 67E6 1354 pointer = 36
00000028 67E6 1355 data = 40
        67E6 1356
        67E6 1357          STORE      F          ; store
        68B7 1358          ;+
        68B7 1359          ; Restore the following offsets.

```

BASSMAT_ADD - Add 2 arrays giving a thi

```

68B7 1360 :-
68B7 1361
0000001C 68B7 1362 value_desc = 28
0000001C 68B7 1363 str_len = 28
0000001E 68B7 1364 dtype = 30
0000001F 68B7 1365 class = 31
00000020 68B7 1366 pointer = 32
00000024 68B7 1367 data = 36
68B7 1368
05 68B7 1369 RSB ; go continue loop
68B8 1370
68B8 1371 DEST_B_TO D:
50 7E 50 6C 68B8 1372 CVTBD RO, -(SP) ; save double
50 0C AD D0 68B8 1373 MOVL SF$L_SAVE_FP(FP), RO ; pass FP to get scale
00000000'GF 16 68B8 1374 JSB G^BASS$SCALE_R1 ; get scale in R0 & R1
50 8E 64 68B8 1375 MULD2 (SP)+, RO ; scale for dest
0086 31 68C5 1376 BRW STORE_DOUBLE ; go store
68CB 1377
68CB 1378 DEST_W_TO D:
50 7E 50 6D 68CB 1379 CVTWD RO, -(SP) ; save double
50 0C AD D0 68CE 1380 MOVL SF$L_SAVE_FP(FP), RO ; pass FP to get scale
00000000'GF 16 68D2 1381 JSB G^BASS$SCALE_R1 ; get scale in R0 & R1
50 8E 64 68D8 1382 MULD2 (SP)+, RO ; scale for dest
0073 31 68DB 1383 BRW STORE_DOUBLE ; go store
68DE 1384
68DE 1385 DEST_L_TO D:
50 7E 50 6E 68DE 1386 CVTLD RO, -(SP) ; save double
50 0C AD D0 68E1 1387 MOVL SF$L_SAVE_FP(FP), RO ; pass FP to get scale
00000000'GF 16 68E5 1388 JSB G^BASS$SCALE_R1 ; get scale in R0 & R1
50 8E 64 68EB 1389 MULD2 (SP)+, RO ; scale for dest
0060 31 68EE 1390 BRW STORE_DOUBLE ; go store
68F1 1391
68F1 1392 DEST_F_TO D:
50 7E 50 56 68F1 1393 CVTFD RO, -(SP) ; save double
50 0C AD D0 68F4 1394 MOVL SF$L_SAVE_FP(FP), RO ; pass FP to get scale
00000000'GF 16 68F8 1395 JSB G^BASS$SCALE_R1 ; get scale in R0 & R1
50 8E 64 68FE 1396 MULD2 (SP)+, RO ; scale for dest
00000000'GF 16 6901 1397 JSB G^MTH$DINT_R4 ; integerize
0047 31 6907 1398 BRW STORE_DOUBLE ; go store
690A 1399
690A 1400 DEST_G_TO D:
690A 1401 ; Note the intermediate conversion to hfloat.
690A 1402
690A 1403
50 7E 52 D0 690A 1404 MOVL R2, -(SP) ; save regs which CVTGH
7E 53 D0 690D 1405 MOVL R3, -(SP) ; will destroy
50 50 56FD 6910 1406 CVTGH RO, RO ; cvt gfloat to hfloat
7E 50 F7FD 6914 1407 CVTHD RO, -(SP) ; cvt to desired double
53 8E D0 6918 1408 MOVL (SP)+, R3 ; restore regs
52 8E D0 691B 1409 MOVL (SP)+, R2
50 0C AD D0 691E 1410 MOVL SF$L_SAVE_FP(FP), RO ; pass FP to get scale
00000000'GF 16 6922 1411 JSB G^BASS$SCALE_R1 ; get scale in R0 & R1
50 8E 64 6928 1412 MULD2 (SP)+, RO ; scale for dest
7E 54 D0 692B 1413 MOVL R4, -(SP) ; save R4
00000000'GF 16 692E 1414 JSB G^MTH$DINT_R4 ; integerize
54 8E D0 6934 1415 MOVL (SP)+, R4 ; restore R4
0017 31 6937 1416 BRW STORE_DOUBLE

```


BASSMAT_ADD - Add 2 arrays giving a thi

```

693A 1417
693A 1418
693A 1419 DEST_H_TO D:
50 7E 50 F7FD 693A 1420 CVTHD R0, -(SP) ; save double
50 0C AD D0 693E 1421 MOVL SF$L_SAVE_FP(FP), R0 ; pass FP to get scale
00000000*GF 16 6942 1422 JSB G^BASS$SCALE_R1 ; get scale in R0 & R1
50 8E 64 6948 1423 MULD2 (SP)+ R0 ; scale for dest
00000000*GF 16 694B 1424 JSB G^MTH$DINT_R4 ; integerize
6951 1425 ; fall into store
6951 1426
6951 1427 STORE_DOUBLE:
53 52 5A D0 6951 1428 MOVL R10, R2 ; pointer to dest descriptor
53 08 AE D0 6954 1429 MOVL lower_bnd1+4(SP), R3 ; current row (extra longword
6958 1430 ; on stack for jsb)
54 5B D0 6958 1431 MOVL R11, R4 ; current column
28 AE 50 70 695B 1432 MOVD R0, DATA+4(SP)
695F 1433 ;+
695F 1434 ; Redefine the following offsets for the call to the STORE macro. The
695F 1435 ; BSBW to here added 4 to the stack.
695F 1436 ; -
695F 1437
00000020 695F 1438 value_desc = 32
00000020 695F 1439 str_len = 32
00000022 695F 1440 dtype = 34
00000023 695F 1441 class = 35
00000024 695F 1442 pointer = 36
00000028 695F 1443 data = 40
695F 1444
695F 1445 STORE D ; store
6A30 1446 ;+
6A30 1447 ; Restore the following offsets.
6A30 1448 ; -
6A30 1449
0000001C 6A30 1450 value_desc = 28
0000001C 6A30 1451 str_len = 28
0000001E 6A30 1452 dtype = 30
0000001F 6A30 1453 class = 31
00000020 6A30 1454 pointer = 32
00000024 6A30 1455 data = 36
6A30 1456
05 6A30 1457 RSB ; go continue loop
6A31 1458
50 50 4CFD 6A31 1459 DEST_B_TO G:
2D 11 6A31 1460 CVTBG R0, R0 ; convert
6A35 1461 BRB STORE_GFLOAT ; go store
6A37 1462
6A37 1463 DEST_W_TO G:
50 50 4DFD 6A37 1464 CVTWG R0, R0 ; convert
27 11 6A3B 1465 BRB STORE_GFLOAT ; go store
6A3D 1466
50 50 4EFD 6A3D 1467 DEST_L_TO G:
21 11 6A3D 1468 CVTLG R0, R0 ; convert
6A41 1469 BRB STORE_GFLOAT ; go store
6A43 1470
50 50 99FD 6A43 1471 DEST_F_TO G:
1B 11 6A43 1472 CVTFG R0, R0 ; convert
6A47 1473 BRB STORE_GFLOAT ; go store

```

```

6A49 1474
6A49 1475 DEST_D_TO_G:
6A49 1476
6A49 1477
6A49 1478
7E 52 DO 6A49 1479
7E 53 DO 6A4C 1480
50 50 32FD 6A4F 1481
50 50 76FD 6A53 1482
53 8E DO 6A57 1483
52 8E DO 6A5A 1484
0004 31 6A5D 1485
6A60 1486
6A60 1487 DEST_H_TO_G:
50 50 76FD 6A60 1488
6A64 1489
6A64 1490
6A64 1491 STORE_GFLOAT:
53 52 5A DO 6A64 1492
08 AE DO 6A67 1493
6A6B 1494
54 5B DO 6A6B 1495
28 AE 50 50FD 6A6E 1496
6A73 1497
6A73 1498
6A73 1499
6A73 1500
6A73 1501
00000020 6A73 1502 value_desc = 32
00000020 6A73 1503 str_len = 32
00000022 6A73 1504 dtype = 34
00000023 6A73 1505 class = 35
00000024 6A73 1506 pointer = 36
00000028 6A73 1507 data = 40
6A73 1508
6A73 1509 STORE G
6B48 1510
6B48 1511
6B48 1512
6B48 1513
0000001C 6B48 1514 value_desc = 28
0000001C 6B48 1515 str_len = 28
0000001E 6B48 1516 dtype = 30
0000001F 6B48 1517 class = 31
00000020 6B48 1518 pointer = 32
00000024 6B48 1519 data = 36
6B48 1520
05 6B48 1521 RSB
6B49 1522
6B49 1523 DEST_B_TO_H:
50 50 6CFD 6B49 1524
2D 11 6B4D 1525
6B4F 1526
6B4F 1527 DEST_W_TO_H:
50 50 6DFD 6B4F 1528
27 11 6B53 1529
6B55 1530

```

Note the intermediate conversion to hfloat.

; save regs which CVTDH
; will destroy
; cvt dbl to hfloat
; cvt to desired gfloat
; restore regs

; convert
; fall into store

; pointer to dest descriptor
; current row (extra longword
; on stack for jsb)
; current column

;+
; Redefine the following offsets for the call to the STORE macro. The
; BSBW to here added 4 to the stack.

STORE G

;+
; Restore the following offsets.

; convert
; go store

; convert
; go store

```

BASSMAT_ADD - Add 2 arrays giving a thi
50 50 6EFD 6B55 1531 DEST_L_TO_H:
21 11 6B55 1532 CCTLH RO, RO ; convert
6B59 1533 BRB STORE_HFLOAT ; go store
6B5B 1534
6B5B 1535 DEST_F_TO_H:
50 50 98FD 6B5B 1536 CTFH RO, RO ; convert
1B 11 6B5F 1537 BRB STORE_HFLOAT ; go store
6B61 1538
6B61 1539 DEST_D_TO_H:
7E 50 70 6B61 1540 MOVD RO, -(SP) ; save double
50 0C AD D0 6B64 1541 MOVL SF$L_SAVE_FP(FP), R0 ; pass FP to get scale
00000000 GF 16 6B68 1542 JSB G*BASS$SCALE R1 ; get scale in R0 & R1
50 8E 50 67 6B6E 1543 DIVD3 RO, (SP)+, R0 ; descale for dest
50 50 32FD 6B72 1544 CVTDH RO, RO ; convert
04 11 6B76 1545 BRB STORE_HFLOAT ; go store
6B78 1546
6B78 1547 DEST_G_TO_H:
50 50 56FD 6B78 1548 CVTGH RO, RO ; convert
6B7C 1549 ; fall into store
6B7C 1550 STORE_HFLOAT:
54 5A D0 6B7C 1551 MOVL R10, R4 ; pointer to dest descriptor
55 0B AE D0 6B7F 1552 MOVL lower_bnd1+4(SP), R5 ; current row (extra longword
6B83 1553 ; on stack for jsb)
56 5B D0 6B83 1554 MOVL R11, R6 ; current column
28 AE 50 70FD 6B86 1555 MOVH RO, DATA+4(SP)
6B8B 1556 ;+
6B8B 1557 ; Redefine the following offsets for the call to the STORE macro. The
6B8B 1558 ; BSBW to here added 4 to the stack.
6B8B 1559 ;-
6B8B 1560
00000020 6B8B 1561 value_desc = 32
00000020 6B8B 1562 str_len = 32
00000022 6B8B 1563 dtype = 34
00000023 6B8B 1564 class = 35
00000024 6B8B 1565 pointer = 36
00000028 6B8B 1566 data = 40
6B8B 1567
6B8B 1568 STORE H ; go continue loop
6C60 1569 ;+
6C60 1570 ; Restore the following offsets.
6C60 1571 ;-
6C60 1572
0000001C 6C60 1573 value_desc = 28
0000001C 6C60 1574 str_len = 28
0000001E 6C60 1575 dtype = 30
0000001F 6C60 1576 class = 31
00000020 6C60 1577 pointer = 32
00000024 6C60 1578 data = 36
6C60 1579
05 6C60 1580 RSB
6C61 1581
6C61 1582 .END

```

BASSMAT_ADD
Symbol Table

E 5

15-SEP-1984 23:39:02 VAX/VMS Macro V04-00
6-SEP-1984 10:28:41 [BASRTL.SRC]BASMATADD.MAR;1

BASSSCALE_R1	*****	X	00	DEST_F_TO_W	00006595	R	02
BASSSTOP	*****	X	00	DEST_G_TO_B	000064A1	R	02
BASSFETCH_BFA	*****	X	00	DEST_G_TO_D	0000690A	R	02
BASSFETCH_DESC	*****	X	00	DEST_G_TO_F	000067CE	R	02
BASSFET_FA_B_RB	*****	X	00	DEST_G_TO_H	00006878	R	02
BASSFET_FA_D_RB	*****	X	00	DEST_G_TO_L	000066BF	R	02
BASSFET_FA_F_RB	*****	X	00	DEST_G_TO_W	000065B0	R	02
BASSFET_FA_G_RB	*****	X	00	DEST_H_TO_B	000064A7	R	02
BASSFET_FA_H_RB	*****	X	00	DEST_H_TO_D	0000693A	R	02
BASSFET_FA_L_RB	*****	X	00	DEST_H_TO_F	000067D4	R	02
BASSFET_FA_W_RB	*****	X	00	DEST_H_TO_G	00006A60	R	02
BASSK_ARGDONMAT	*****	X	00	DEST_H_TO_L	000066C5	R	02
BASSK_ARRMUSSAM	*****	X	00	DEST_H_TO_W	000065B6	R	02
BASSK_DATTYPERR	*****	X	00	DEST_L_TO_B	00006481	R	02
BASSK_MATDIMERR	*****	X	00	DEST_L_TO_D	000068DE	R	02
BASSMAT_ADD	00000000	RG	02	DEST_L_TO_F	000067B3	R	02
BASSMAT_REDIM	*****	X	00	DEST_L_TO_G	00006A3D	R	02
BASSSTORE_BFA	*****	X	00	DEST_L_TO_H	00006855	R	02
BASSSTO_FA_B_RB	*****	X	00	DEST_L_TO_W	00006590	R	02
BASSSTO_FA_D_RB	*****	X	00	DEST_MATRIX	= 0000000C		
BASSSTO_FA_F_RB	*****	X	00	DEST_W_TO_B	0000647C	R	02
BASSSTO_FA_G_RB	*****	X	00	DEST_W_TO_D	000068CB	R	02
BASSSTO_FA_H_RB	*****	X	00	DEST_W_TO_F	000067AE	R	02
BASSSTO_FA_L_RB	*****	X	00	DEST_W_TO_G	00006A37	R	02
BASSSTO_FA_W_RB	*****	X	00	DEST_W_TO_H	0000684F	R	02
BYTE	00000107	R	02	DEST_W_TO_L	0000669F	R	02
BYTE_TO_BYTE	00000139	R	02	DOUBLE	000038BF	R	02
BYTE_TO_DOUBLE	000008FA	R	02	DOUBLE_TO_BYTE	000038F1	R	02
BYTE_TO_FLOAT	00000709	R	02	DOUBLE_TO_DOUBL	00004101	R	02
BYTE_TO_GFLOAT	00000B03	R	02	DOUBLE_TO_FLOAT	00003EFD	R	02
BYTE_TO_HFLOAT	00000CFC	R	02	DOUBLE_TO_GFLOA	000042EF	R	02
BYTE_TO_LONG	00000518	R	02	DOUBLE_TO_HFLOA	000044EC	R	02
BYTE_TO_WORD	00000327	R	02	DOUBLE_TO_LONG	00003CF9	R	02
DEST_B_TO_D	000068B8	R	02	DOUBLE_TO_WORD	00003AF5	R	02
DEST_B_TO_F	000067A9	R	02	DSCSA_A0	= 00000010		
DEST_B_TO_G	00006A31	R	02	DSCSB_AFLAGS	= 0000000A		
DEST_B_TO_H	00006849	R	02	DSCSB_CLASS	= 00000003		
DEST_B_TO_L	0000669A	R	02	DSCSB_DIMCT	= 0000000B		
DEST_B_TO_W	0000658B	R	02	DSCSB_DTYPE	= 00000002		
DEST_CASE_B	00006309	R	02	DSCSK_CLASS_BFA	= 000000BF		
DEST_CASE_D	000063DD	R	02	DSCSK_DTYPE_B	= 00000006		
DEST_CASE_F	000063A8	R	02	DSCSK_DTYPE_D	= 0000000B		
DEST_CASE_G	00006412	R	02	DSCSK_DTYPE_DSC	= 00000018		
DEST_CASE_H	00006447	R	02	DSCSK_DTYPE_G	= 0000001B		
DEST_CASE_L	00006373	R	02	DSCSK_DTYPE_H	= 0000001C		
DEST_CASE_W	0000633E	R	02	DSCSL_L1_1	= 00000018		
DEST_D_TO_B	0000648B	R	02	DSCSL_L1_2	= 0000001C		
DEST_D_TO_F	000067B8	R	02	DSCSL_L2_2	= 00000024		
DEST_D_TO_G	00006A49	R	02	DSCSL_M1	= 00000014		
DEST_D_TO_H	00006861	R	02	DSCSL_M2	= 00000018		
DEST_D_TO_L	000066A9	R	02	DSCSL_U1_1	= 0000001C		
DEST_D_TO_W	0000659A	R	02	DSCSL_U1_2	= 00000020		
DEST_F_TO_B	00006486	R	02	DSCSL_U2_2	= 00000028		
DEST_F_TO_D	000068F1	R	02	DSCSV_FL_BOUNDS	= 00000007		
DEST_F_TO_G	00006A43	R	02	DSCSW_LENGTH	= 00000000		
DEST_F_TO_H	00006B5B	R	02	ERR_ARGDONMAT	0000002A	R	02
DEST_F_TO_L	000066A4	R	02	ERR_ARRMUSSAM	00000076	R	02

BASMAT ADD
Symbol Table

F 5

15-SEP-1984 23:39:02 VAX/VMS Macro V04-00
6-SEP-1984 10:28:41 [BASRTL.SRC]BASMATADD.MAR;1

Page 82
(7)

ERR_DATTYPERR	000000FA	R	02	LOOP_1ST_SUBGB	00004717	R	02
ERR_MATDIMERR	00000069	R	02	LOOP_1ST_SUBGD	00004EFF	R	02
FLOAT	00002AD1	R	02	LOOP_1ST_SUBGF	00004D05	R	02
FLOAT_TO_BYTE	00002B03	R	02	LOOP_1ST_SUBGG	000050FD	R	02
FLOAT_TO_DOUBLE	000032C4	R	02	LOOP_1ST_SUBGH	000052F9	R	02
FLOAT_TO_FLOAT	000030D6	R	02	LOOP_1ST_SUBGL	00004B0B	R	02
FLOAT_TO_GFLOAT	000034CD	R	02	LOOP_1ST_SUBGW	00004911	R	02
FLOAT_TO_HFLOAT	000036C6	R	02	LOOP_1ST_SUBHB	0000552B	R	02
FLOAT_TO_LONG	00002EE5	R	02	LOOP_1ST_SUBHD	00005D13	R	02
FLOAT_TO_WORD	00002CF4	R	02	LOOP_1ST_SUBHF	00005B19	R	02
GFLOAT	000046E5	R	02	LOOP_1ST_SUBHG	00005F0D	R	02
GFLOAT_TO_BYTE	00004717	R	02	LOOP_1ST_SUBHH	0000610D	R	02
GFLOAT_TO_DOUBL	00004EFF	R	02	LOOP_1ST_SUBHL	0000591F	R	02
GFLOAT_TO_FLOAT	00004D05	R	02	LOOP_1ST_SUBHW	00005725	R	02
GFLOAT_TO_GFLOA	000050FD	R	02	LOOP_1ST_SUBLB	00001D15	R	02
GFLOAT_TO_HFLOA	000052F9	R	02	LOOP_1ST_SUBLD	000024D6	R	02
GFLOAT_TO_LONG	00004B0B	R	02	LOOP_1ST_SUBLF	000022E5	R	02
GFLOAT_TO_WORD	00004911	R	02	LOOP_1ST_SUBLG	000026DF	R	02
HFLOAT	000054F9	R	02	LOOP_1ST_SUBLH	000028D8	R	02
HFLOAT_TO_BYTE	0000552B	R	02	LOOP_1ST_SUBLI	000020F7	R	02
HFLOAT_TO_DOUBL	00005D13	R	02	LOOP_1ST_SUBLJ	00001F06	R	02
HFLOAT_TO_FLOAT	00005B19	R	02	LOOP_1ST_SUBLK	00000F27	R	02
HFLOAT_TO_GFLOA	00005F0D	R	02	LOOP_1ST_SUBLL	000016E8	R	02
HFLOAT_TO_HFLOA	0000610D	R	02	LOOP_1ST_SUBLM	000014F7	R	02
HFLOAT_TO_LONG	0000591F	R	02	LOOP_1ST_SUBLN	000018F1	R	02
HFLOAT_TO_WORD	00005725	R	02	LOOP_1ST_SUBLP	00001AEA	R	02
INIT_ONE_SUB	00000037	R	02	LOOP_1ST_SUBLQ	00001306	R	02
INIT_TWO_SUBS	00000083	R	02	LOOP_1ST_SUBLR	00001118	R	02
LONG	00001CE3	R	02	LOOP_2ND_SUBBS	0000013C	R	02
LONG_TO_BYTE	00001D15	R	02	LOOP_2ND_SUBBT	000008FD	R	02
LONG_TO_DOUBLE	000024D6	R	02	LOOP_2ND_SUBBU	0000070C	R	02
LONG_TO_FLOAT	000022E5	R	02	LOOP_2ND_SUBBV	00000B06	R	02
LONG_TO_GFLOAT	000026DF	R	02	LOOP_2ND_SUBBW	00000CFF	R	02
LONG_TO_HFLOAT	000028D8	R	02	LOOP_2ND_SUBBX	0000051B	R	02
LONG_TO_LONG	000020F7	R	02	LOOP_2ND_SUBBY	0000032A	R	02
LONG_TO_WORD	00001F06	R	02	LOOP_2ND_SUBBZ	000038F4	R	02
LOOP_1ST_SUBBB	00000139	R	02	LOOP_2ND_SUBCA	00004104	R	02
LOOP_1ST_SUBBD	000008FA	R	02	LOOP_2ND_SUBCB	00003F00	R	02
LOOP_1ST_SUBBF	00000709	R	02	LOOP_2ND_SUBCC	000042F2	R	02
LOOP_1ST_SUBBG	00000B03	R	02	LOOP_2ND_SUBCD	000044EF	R	02
LOOP_1ST_SUBBH	00000CFC	R	02	LOOP_2ND_SUBCE	00003CFC	R	02
LOOP_1ST_SUBBI	00000518	R	02	LOOP_2ND_SUBCF	00003AF8	R	02
LOOP_1ST_SUBBJ	00000327	R	02	LOOP_2ND_SUBCG	00002B06	R	02
LOOP_1ST_SUBBK	000038F1	R	02	LOOP_2ND_SUBCH	000032C7	R	02
LOOP_1ST_SUBLB	00004101	R	02	LOOP_2ND_SUBCI	000030D9	R	02
LOOP_1ST_SUBLD	00003EFD	R	02	LOOP_2ND_SUBCJ	000034D0	R	02
LOOP_1ST_SUBLF	000042EF	R	02	LOOP_2ND_SUBCK	000036C9	R	02
LOOP_1ST_SUBLG	000044EC	R	02	LOOP_2ND_SUBCL	00002EE8	R	02
LOOP_1ST_SUBLH	00003CF9	R	02	LOOP_2ND_SUBCM	00002CF7	R	02
LOOP_1ST_SUBLI	00003AF5	R	02	LOOP_2ND_SUBCN	0000471A	R	02
LOOP_1ST_SUBLJ	00002B03	R	02	LOOP_2ND_SUBCO	00004F02	R	02
LOOP_1ST_SUBLK	000032C4	R	02	LOOP_2ND_SUBCP	00004D08	R	02
LOOP_1ST_SUBLL	000030D6	R	02	LOOP_2ND_SUBCQ	00005100	R	02
LOOP_1ST_SUBLM	000034CD	R	02	LOOP_2ND_SUBCR	000052FC	R	02
LOOP_1ST_SUBLN	000036C6	R	02	LOOP_2ND_SUBCS	00004B0E	R	02
LOOP_1ST_SUBLP	00002EE5	R	02	LOOP_2ND_SUBCT	00004914	R	02
LOOP_1ST_SUBLQ	00002CF4	R	02	LOOP_2ND_SUBCU	0000552E	R	02

LOOP_2ND_SUBHD	00005D16	R	02
LOOP_2ND_SUBHF	00005B1C	R	02
LOOP_2ND_SUBHG	00005F10	R	02
LOOP_2ND_SUBHH	00006110	R	02
LOOP_2ND_SUBHL	00005922	R	02
LOOP_2ND_SUBHW	00005728	R	02
LOOP_2ND_SUBLB	00001D18	R	02
LOOP_2ND_SUBLD	000024D9	R	02
LOOP_2ND_SUBLF	000022E8	R	02
LOOP_2ND_SUBLG	000026E2	R	02
LOOP_2ND_SUBLH	000028DB	R	02
LOOP_2ND_SUBLL	000020FA	R	02
LOOP_2ND_SUBLW	00001F09	R	02
LOOP_2ND_SUBWB	00000F2A	R	02
LOOP_2ND_SUBWD	000016EB	R	02
LOOP_2ND_SUBWF	000014FA	R	02
LOOP_2ND_SUBWG	000018F4	R	02
LOOP_2ND_SUBWH	00001AED	R	02
LOOP_2ND_SUBWL	00001309	R	02
LOOP_2ND_SUBWW	0000111B	R	02
LOWER_BND1	= 00000004		
LOWER_BND2	= 00000000		
MTHSDINT R4	*****	X	00
SAVE_SRCT	= 0000000C		
SEPARATE_DTYPES	000000CB	R	02
SF\$L_SAVE_FP	= 0000000C		
SRC1_MATRIX	= 00000004		
SRC2_MATRIX	= 00000008		
STORE_BYTE	000064AB	R	02
STORE_DOUBLE	00006951	R	02
STORE_FLOAT	000067D8	R	02
STORE_GFLOAT	00006A64	R	02
STORE_HFLOAT	0000687C	R	02
STORE_LONG	000066C9	R	02
STORE_WORD	000065BA	R	02
UPPER_BND1	= 00000008		
WORD	00000EF5	R	02
WORD_TO_BYTE	00000F27	R	02
WORD_TO_DOUBLE	000016E8	R	02
WORD_TO_FLOAT	000014F7	R	02
WORD_TO_GFLOAT	000018F1	R	02
WORD_TO_HFLOAT	00001AEA	R	02
WORD_TO_LONG	00001306	R	02
WORD_TO_WORD	00001118	R	02

+-----+
! Psect synopsis !
+-----+

<u>PSECT name</u>	<u>Allocation</u>	<u>PSECT No.</u>	<u>Attributes</u>										
. 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
_BASSCODE	00006C61 (27745.)	02 (2.)	PIC	USR	CON	REL	LCL	SHR	EXE	RD	NOWRT	NOVEC	LONG

! Performance indicators !

Phase	Page faults	CPU Time	Elapsed Time
-----	-----	-----	-----
Initialization	29	00:00:00.10	00:00:00.39
Command processing	116	00:00:00.66	00:00:04.89
Pass 1	826	00:00:43.90	00:01:48.08
Symbol table sort	0	00:00:02.24	00:00:05.11
Pass 2	375	00:00:10.83	00:00:24.60
Symbol table output	31	00:00:00.23	00:00:00.35
Psect synopsis output	2	00:00:00.01	00:00:00.20
Cross-reference output	0	00:00:00.00	00:00:00.00
Assembler run totals	1381	00:00:57.97	00:02:23.67

The working set limit was 2000 pages.
319544 bytes (625 pages) of virtual memory were used to buffer the intermediate code.
There were 60 pages of symbol table space allocated to hold 422 non-local and 909 local symbols.
1582 source lines were read in Pass 1, producing 85 object records in Pass 2.
36 pages of virtual memory were used to define 11 macros.

! Macro library statistics !

Macro library name	Macros defined
-----	-----
_\$255\$DUA28:[BASRTL.OBJ]BASRTL.MLB;1	2
-\$255\$DUA28:[SYSLIB]STARLET.MLB;2	5
TOTALS (all libraries)	7

493 GETS were required to define 7 macros.

There were no errors, warnings or information messages.

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

0025 AH-BT13A-SE
VAX/VMS V4.0

DIGITAL EQUIPMENT CORPORATION
CONFIDENTIAL AND PROPRIETARY