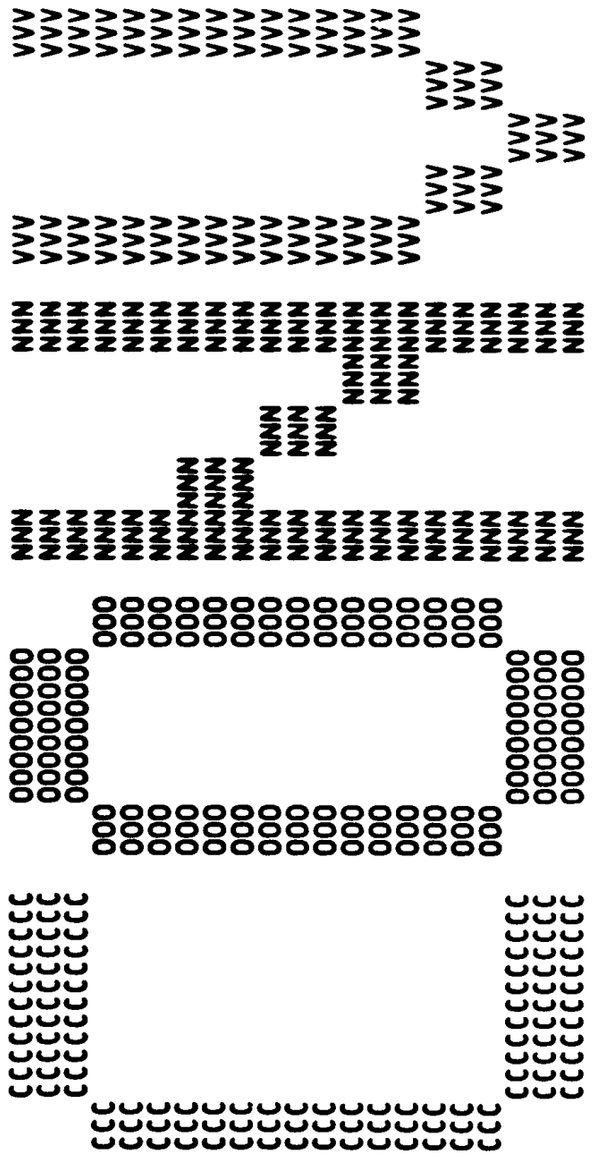

```
  CCCCCCCC    000000    NN      NN  VV      VV  FFFFFFFFFF    AAAAAA     SSSSSSSS  TTTTTTTTTT  MM      MM
  CCCCCCCC    000000    NN      NN  VV      VV  FFFFFFFFFF   AAAAAA     SSSSSSSS  TTTTTTTTTT  MM      MM
CC           00      00  NN      NN  VV      VV  FF          AA      AA  SS            TT      MMMM  MMMM
CC           00      00  NN      NN  VV      VV  FF          AA      AA  SS            TT      MMMM  MMMM
CC           00      00  NNNN    NN  VV      VV  FF          AA      AA  SS            TT      MM MM MM MM
CC           00      00  NNNN    NN  VV      VV  FF          AA      AA  SS            TT      MM MM MM MM
CC           00      00  NN NN   NN  VV      VV  FFFFFFFF    AA      AA   SSSSSS       TT      MM    MM
CC           00      00  NN NN   NN  VV      VV  FFFFFFFF    AA      AA   SSSSSS       TT      MM    MM
CC           00      00  NN  NNNN NN  VV      VV  FF          AAAAAAAAAA         SS   TT      MM    MM
CC           00      00  NN  NNNN NN  VV      VV  FF          AAAAAAAAAA         SS   TT      MM    MM
CC           00      00  NN    NN NN    VV  VV    FF          AA      AA         SS   TT      MM    MM
CC           00      00  NN    NN NN    VV  VV    FF          AA      AA         SS   TT      MM    MM
  CCCCCCCC    000000    NN      NN      VV      FF          AA      AA  SSSSSSSS  TT      MM    MM
  CCCCCCCC    000000    NN      NN      VV      FF          AA      AA  SSSSSSSS  TT      MM    MM

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
```

Table of contents

```
0000     1          .IDENT  /V04-000/
0000     2          .TITLE  CONV$FSTMR CONVERT macro routines
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     27 ;++
0000     28 ;
0000     29 ; Facility:     VAX-11 CONVERT
0000     30 ;
0000     31 ; Abstract:
0000     32 ;               CONVERT macro routines for data compression and key
0000     33 ;               conversion
0000     34 ;
0000     35 ; Environment:
0000     36 ;
0000     37 ;               VAX/VMS Operating System
0000     38 ;
0000     39 ;--
0000     40
0000     41 ;
0000     42 ; Authors:      Maria Nasr      Creation date:  1-May-81
0000     43 ;               P. S. Knibbe
0000     44 ;               Keith Thompson
0000     45 ;
0000     46 ; Modified by:
0000     47 ;
0000     48 ;       V03-002 KBT0107         Keith B. Thompson       5-Aug-1982
0000     49 ;               Don't do the conversion to ascii until rms desides what to do!
0000     50 ;
0000     51 ;       V03-001 KBT0011         Keith Thompson          16-Mar-1982
0000     52 ;               Make the modifications made to the corresponding rms module
0000     53 ;               which correct some problems in the data compression routine
0000     54 ;
0000     55 ;****
0000     56
00000000 57         .PSECT  CODE,EXE,PIC,NOWRT
0000     58
```

```
                     60 .SBTTL  CONV$$COMPRESS_DATA     - does random compression on data section
                     61 ;++
                     62 ;
                     63 ; Functional Description:
                     64 ;
                     65 ;       This routine is called to do compression on the data section of the
                     66 ;       record.  It searches for consecutive sequences of 5 or more
                     67 ;       repeating characters, and compresses them.  For each sequence that is
                     68 ;       not compressed, it allocates a word to count the number of characters
                     69 ;       in the data segment, and a byte to indicate the number of characters
                     70 ;       compressed from the end.
                     71 ;
                     72 ; Calling sequence:
                     73 ;
                     74 ;       CONV$$COMPRESS_DATA()
                     75 ;
                     76 ; Input Parameters:
                     77 ;
                     78 ;       R6 : Pointer to next field count
                     79 ;       R7 : Pointer to truncation count (end of record)
                     80 ;
                     81 ; Implicit Inputs:
                     82 ;       None
                     83 ;
                     84 ; Output Parameters:
                     85 ;
                     86 ; Implicit Outputs:
                     87 ;
                     88 ;       R3 points to one byte past end of record (byte after last
                     89 ;          truncation count)
                     90 ;
                     91 ; Routine Value:
                     92 ;       none
                     93 ;
                     94 ; Side Effects:
                     95 ;
                     96 ;       The data section is compressed
                     97 ;       Registers R1,R3,R6 and R7 are clobbered
                     98 ;
                     99 ; Working registers:
                    100 ;
                    101 ;       R3 : starting point of destination buffer
                    102 ;       R4 : starting point of non-compressed field
                    103 ;       R5 : starting point of possible compressed field
                    104 ;       R8 : index register thru search
                    105 ;       R9 : count of characters compressed
                    106 ;
                    107 ;--
                    108
                    109 CONV$$COMPRESS_DATA::
                    110
    0334 8F   BB    111         PUSHR   #^M<R2,R4,R5,R8,R9>     ; save registers
58  56   02   C1    112         ADDL3   #2,R6,R8                ; get pointer to start of data
     53   58   D0    113         MOVL    R8,R3                   ; save destination buffer start addr
     54   58   D0    114 10$:    MOVL    R8,R4                   ; reset start point
     58   57   D1    115 20$:    CMPL    R7,R8                   ; are we all done?
          03   1A    116         BGTRU   25$                     ; if no, branch
```

```
                 0070   31   0013   117          BRW     60$                     ; else, exit
           55    58   D0   0016   118 25$:      MOVL    R8,R5                   ; save start point of possible match
           88    88   B1   0019   119          CMPW    (R8)+,(R8)+             ; compare consecutive words
                 F0   12   001C   120          BNEQU   20$                     ; if no match, try next ones
      FE A8      FF A8   91   001E   121          CMPB    -1(R8),-2(R8)           ; compare characters in the word
                 E9   12   0023   122          BNEQU   20$                     ; if no match, try next ones
           58    57   D1   0025   123          CMPL    R7,R8                   ; if we have positioned past end
                 5C   1B   0028   124          BLEQU   60$                     ; then we don't have enough to compress
           59    03   D0   002A   125          MOVL    #3,R9                   ; a match of 4 found
           55    54   D1   002D   126          CMPL    R4,R5                   ; should we go back?
                 0C   13   0030   127          BEQLU   40$                     ; no, do not
                           0032   128
                           0032   129 ;
                           0032   130 ; Move backwards to search for any characters that might have been missed
                           0032   131 ;
           FF A5    65   91   0032   132 30$:      CMPB    (R5),-1(R5)             ; compare bytes
                    06   12   0036   133          BNEQU   40$                     ; no match
                    55   D7   0038   134          DECL    R5                      ; set new match point
           F4 59    06   F2   003A   135          AOBLSS  #6,R9,30$               ; indicate another match found, and if
                           003E   136                                           ;  more left, try next
                           003E   137
                           003E   138 ;
                           003E   139 ; Look for the first character that does not match
                           003E   140 ;
                           003E   141
      52   57    58   C3   003E   142 40$:      SUBL3   R8,R7,R2                ; find characters left in record
 68   52     FF A8   3B   0042   143          SKPC    -1(R8),R2,(R8)          ; find first char that does not match
           52    50   C2   0047   144          SUBL2   R0,R2                   ; find how many matched
           58    51   D0   004A   145          MOVL    R1,R8                   ; set new starting addresses
FFBB 59    52    03   3D   004D   146          ACBW    #3,R2,R9,20$            ; increment number matched, and if less
                           0053   147                                           ;  than 5, no good
                           0053   148
                           0053   149 ;
                           0053   150 ; Make sure count is not bigger than 255 bytes
                           0053   151 ;
                           0053   152
000000FF 8F      59   D1   0053   153          CMPL    R9,#255
                 11   15   005A   154          BLEQ                            ; ok, if less
59   000000FF 8F      C2   005C   155          SUBL2   #255,R9                 ; find how many extra
        58       59   C2   0063   156          SUBL2   R9,R8                   ; back out that many
59   000000FF 8F      D0   0066   157          MOVL    #255,R9                 ; force 255 bytes
                           006D   158
                           006D   159 ;
                           006D   160 ; A sequence long enough has been found
                           006D   161 ;
                           006D   162
     51    55    54   C3   006D   163 50$:      SUBL3   R4,R5,R1                ; find length of non-compressed
                 51   D6   0071   164          INCL    R1                      ;  section
           66    51   B0   0073   165          MOVW    R1,(R6)                 ; store next field count
     63    64    51   28   0076   166          MOVC3   R1,(R4),(R3)            ; move to destination buffer
           83    59   90   007A   167          MOVB    R9,(R3)+                ; store truncation count
           56    53   D0   007D   168          MOVL    R3,R6                   ; reset new next field addr
           53    02   C0   0080   169          ADDL2   #2,R3                   ; new next field area
              FF85   31   0083   170          BRW     10$
                           0086   171
                           0086   172 ;
                           0086   173 ; Move the last field if not compressed
```

```
                        0086   174 ;
                        0086   175
     58   54   D1        0086   176 60$:    CMPL    R4,R8                   ; was last field compressed?
          11   13        0089   177         BEQLU   70$                     ; exit, if yes
51   57   54   C3        008B   178         SUBL3   R4,R7,R1                ; find length of section
     66   51   B0        008F   179         MOVW    R1,(R6)                 ; store next field count
63   64   51   28        0092   180         MOVC3   R1,(R4),(R3)            ; move to destination buffer
          63   94        0096   181         CLRB    (R3)                    ; truncation count is zero
          53   D6        0098   182         INCL    R3                      ; point to end of record
          03   11        009A   183         BRB     80$
     53   02   C2        009C   184 70$:    SUBL2   #2,R3                   ; point to end of record
     0334 8F   BA        009F   185 80$:    POPR    #^M<R2,R4,R5,R8,R9>     ; restore registers
               05        00A3   186         RSB                             ; return to caller
                        00A4   187
```

```
                        00A4    189 .SBTTL  CVT_TO_ASC      - Convert key to ASCII
                        00A4    190 ;++
                        00A4    191 ;
                        00A4    192 ; Functional Description:
                        00A4    193 ;
                        00A4    194 ;       This routine converts a string from any datatype to an
                        00A4    195 ;       equivalent string which collates like ASCII.
                        00A4    196 ;
                        00A4    197 ; Calling Sequence:
                        00A4    198 ;
                        00A4    199 ;       CONV$$CVT_TO_ASC();     (from Bliss)
                        00A4    200 ;
                        00A4    201 ; Input Parameters:
                        00A4    202 ;
                        00A4    203 ;       40(SP) - dest addr
                        00A4    204 ;       36(SP) - source addr
                        00A4    205 ;       32(SP) - length
                        00A4    206 ;       28(SP) - datatype
                        00A4    207 ;
                        00A4    208 ; Implicit Inputs:
                        00A4    209 ;       none
                        00A4    210 ;
                        00A4    211 ; Output Parameters:
                        00A4    212 ;
                        00A4    213 ;       Destination contains an ASCII equivalent string
                        00A4    214 ;
                        00A4    215 ; Implicit Outputs:
                        00A4    216 ;       none
                        00A4    217 ;
                        00A4    218 ; Routine Value:
                        00A4    219 ;       none
                        00A4    220 ;
                        00A4    221 ; Side Effects:
                        00A4    222 ;       none
                        00A4    223 ;
                        00A4    224 ; Working Registers:
                        00A4    225 ;       ?
                        00A4    226 ;
                        00A4    227 ;--
                        00A4    228
                        00A4    229 CONV$$CVT_TO_ASC::
                        00A4    230
   007E 8F    BB        00A4    231         PUSHR   #^M<R1,R2,R3,R4,R5,R6>
                        00A8    232
                        00A8    233 ;   if low bit of datatype is set, then this is a signed datatype.
                        00A8    234 ;
51    24 AE    D0      00A8    235         MOVL    36(SP),R1               ; Source
53    28 AE    D0      00AC    236         MOVL    40(SP),R3               ; Destination
         50    94      00B0    237         CLRB    R0                      ; assume unsigned
  04 1C AE    E9      00B2    238         BLBC    28(SP),10$              ; branch if we were right
50    50 8F    90      00B6    239         MOVB    #80,R0                  ; set up mask to flip sign bit
                        00BA    240
                        00BA    241 ;   Now actually convert it depending on datatype
                        00BA    242 ;
                        00BA    243 10$::   CASEB   28(SP),#0,#5
                        00BA    244 15$::   .WORD   20$-15$                 ; String
                        00BA    245 ;       .WORD   30$-15$                 ; 2 byte integer
```

```
                                00BA   246 ;       .WORD   40$-15$                 ; 2 byte binary
                                00BA   247 ;       .WORL   50$-15$                 ; 4 byte integer
                                00BA   248 ;       .WORQ   60$-15$                 ; 4 byte binary
                                00BA   249 ;       .WOFD   70$-15$                 ; Packed decimal
                                00BA   250 ;
                                00BA   251 ;   String datatype
                                00BA   252 ;
63   61   20 AE   28            00BA   253 20$:    MOVC3   32(SP),(R1),(R3)        ; just move the string
                    60   11     00BF   254         BRB     900$                    ; exit
                                00C1   255
                                00C1   256 ;   Two byte binary or integer
                                00C1   257 ;
                                00C1   258 30$:
83   01 A1   50     8D          00C1   259 40$:    XORB3   R0,1(R1),(R3)+          ; Set up the sign byte right
         63   61   90           00C6   260         MOVB    (R1),(R3)               ; move the other byte
              56   11           00C9   261         BRB     900$                    ; exit
                                00CB   262
                                00CB   263 ;   Four byte binary or integer
                                00CB   264 ;
                                00CB   265 50$:
83   03 A1   50     8D          00CB   266 60$:    XORB3   R0,3(R1),(R3)+          ; Fix up the sign byte
83     02 A1   90               00D0   267         MOVB    2(R1),(R3)+             ; Move the other bytes
83     01 A1   90               00D4   268         MOVB    1(R1),(R3)+             ;  in reverse order
         63   61   90           00D8   269         MOVB    (R1),(R3)               ;
              44   11           00DB   270         BRB     900$                    ; exit
                                00DD   271
                                00DD   272 ;   Packed decimal
                                00DD   273 ;
00   61   20 AE   FF 8F   F8    00DD   274 70$:    ASHP    #-1,32(SP),(R1),#0,-    ; Shift down 4 bits
         63   20 AE             00E4   275                 32(SP),(R3)             ;  to make room for sign
                                00E7   276
54   20 AE   FF 8F   78         00E7   277         ASHL    #-1,32(SP),R4           ; R4 <- disp to last byte
     55   54   51   C1          00ED   278         ADDL3   R1,R4,R5                ; R5 <- addr of last source byte
     56   54   53   C1          00F1   279         ADDL3   R3,R4,R6                ; R6 <- addr of last dest byte
                                00F5   280
50   65   04   04   EE          00F5   281         EXTV    #4,#4,(R5),R0           ; Restore last digit
66   04   00   50   F0          00FA   282         INSV    R0,#0,#4,(R6)
                                00FF   283
50   65   04   00   EF          00FF   284         EXTZV   #0,#4,(R5),R0           ; R0 <- Sign nibble
          15   50   E9          0104   285         BLBC    R0,100$                 ; All even's are plus
               50   0F   91     0107   286         CMPB    #^X0F,R0                ; 'F' is also plus
                    10   13     010A   287         BEQLU   100$
                                010C   288
                                010C   289 ; Negative number - we must flip each bit (except for sign nibble)
                                010C   290 ;  in order to preserve ordering
                                010C   291 ;
                    54   D6     010C   292         INCL    R4                      ; R4 <- number of bytes
               83   0F   8C     010E   293         XORB    #^X0F,(R3)+             ; Don't touch the sign nibble
                    04   11     0111   294         BRB     90$
          83   FF 8F   8C       0113   295 80$:    XORB    #^XFF,(R3)+             ; Switch every bit in string
               F9 54   F5       0117   296 90$:    SOBGTR  R4,80$                  ; Do this for every byte
                    05   11     011A   297         BRB     900$                    ; exit
                                011C   298
                                011C   299 ; Positive number - just set the sign nibble to something
                                011C   300 ;  higher that zero
                                011C   301 ;
63   04   04   08   F0          011C   302 100$:   INSV    #8,#4,#4,(R3)           ; Positive - set high bit in sign nibble
```

```
                          0121    303
                          0121    304 900$:
007E 8F    BA             0121    305             POPR    #^M<R1,R2,R3,R4,R5,R6>
           05             0125    306             RSB
                          0126    307
                          0126    308             .END
```

```
CONV$$COMPRESS_DATA             00000000 RG    01
CONV$$CVT_TO_ASC                000000A4 RG    01
```

Psect synopsis

```
PSECT name                      Allocation          PSECT No.  Attributes
----------                      ----------          ---------  ----------
. ABS .                         00000000  (    0.)  00 (  0.)  NOPIC   USR   CON   ABS   LCL NOSHR NOEXE NORD  NOWRT NOVEC BYTE
CODE                            00000126  (  294.)  01 (  1.)    PIC   USR   CON   REL   LCL NOSHR   EXE   RD  NOWRT NOVEC BYTE
```

Performance indicators

Phase	Page faults	CPU Time	Elapsed Time
Initialization	9	00:00:00.05	00:00:01.79
Command processing	66	00:00:00.41	00:00:03.99
Pass 1	51	00:00:00.73	00:00:04.22
Symbol table sort	0	00:00:00.00	00:00:00.01
Pass 2	59	00:00:00.61	00:00:03.03
Symbol table output	1	00:00:00.01	00:00:00.01
Psect synopsis output	1	00:00:00.01	00:00:00.01
Cross-reference output	0	00:00:00.00	00:00:00.00
Assembler run totals	187	00:00:01.82	00:00:13.06

```
The working set limit was 600 pages.
4887 bytes (10 pages) of virtual memory were used to buffer the intermediate code.
There were 10 pages of symbol table space allocated to hold 2 non-local and 21 local symbols.
308 source lines were read in Pass 1, producing 11 object records in Pass 2.
0 pages of virtual memory were used to define 0 macros.
```

Macro library statistics

Macro library name	Macros defined
_255DUA28:[SYSLIB]STARLET.MLB;2	0

```
0 GETS were required to define 0 macros.

There were no errors, warnings or information messages.

MACRO/LIS=LIS$:CONVFASTM/OBJ=OBJ$:CONVFASTM MSRC$:CONVFSTMR/UPDATE=(ENH$:CONVFSTMR)
```

CONVDATA
LIS

CONVFSTLD
LIS

CONVFSTIO
LIS

CONVFSTRC
LIS

CONVFILES
LIS

CONVERROR
LIS

CONVFASTM
LIS

CONVDCL
LIS