```
; FILE DM.text      Memory display utilities

;    Change Log
;
;           8-Sep-84 New Today
;          15-Sep-84 Cleanup, bug fixes (always fixbuf before printing!!)
;          16-Sep-84 More DLE's with count
;          22-Sep-84 No DBRA at top of loop!!!
;          23-Sep-84 Added DMbailout for printing
;          26-Sep-84 Added WIND/TERC templates
;

;-------------------------------------------------------------------------------------
--
; There are three debugger globals used in displaying memory.  The first is DMmemPtr,
which
; contains the current location of memory to display.  The second is DMmemEnd, which
points to
; the last memory location to display.  Finally there is DMcmdPtr, which points
; to the current memory template command to execute.
;
; A memory template is a set of interpreted byte commands.  The current list is
;
;   High  Short
;  Nybble Name      Description                         interpretation of low nybble
(extension nybble)
;
;    $0   HEX       Print data @MP as hex               Byte/Word/Long values
;    $1   ASCII     Print data @MP as ASCII             Length of ascii field
;    $2   MP        Print the MP as 6 hex digits
;    $3   STR       Print data @MP as pascal string
;    $4   TEXT      Display the following text string  Length of text string
;    $5   SP        Print spaces                        Number of spaces to print
;    $6   NL        Print a newLine
;    $7
;    $8   FWDMP     Advace the MP                       Amount to advance the MP by
;    $9   BWDMP     Decriment the MP                    Amount to decrement MP by
;    $A   PTRMP     MP -> stack, MP^ -> MP
;    $B
;    $C   REPEAT    Start a loop                        Number of times to loop
;    $D   END       Terminate loop/templete
;    $E   STACK     Push/pop from SP, acc. to ext.      (Push/pop),(Word/Long),(MP/Next
CMD value) (??? future)
;    $F
;
; NOTE : Where the low nybble is used as a numeric value, the value is always
interpreted
;         as one more than the actual nybble in the byte command, eg. $53 is the byte
command
;         to print 4 spaces.
;-------------------------------------------------------------------------------------
--

;-------------------------------------------------------------------------------------
--
; Routine Name      DisplayMem
;
; Registers    A0 (input)              ; ptr to memory template
;
; Function     Uses the memory template specified by A0 to display the memory specified
;              by DMmemPtr up to DMmemEnd
;-------------------------------------------------------------------------------------
```

```
--

DisplayMem
          MOVEQ     #0,D0                 ; set up for nil ptr test
          CMP.L     A0,D0                 ; any memory template ptr?
          BNE.S     @0                    ; yes, use it

          LEA       DMdefTemp,A0          ; no, use default template
@0        MOVE.L    A0,-(SP)              ; and save on stack

TemplateLoop
          MOVE.L    (SP),DMcmdPtr         ; set up command ptr
          CLR.L     -(SP)                 ; push nil command ptr on stack for end-of-DM
test
          CLR.W     -(SP)                 ; fake repeat count
          BSR       FixBuf                ; make sure IO buffer is ready to go

CmdByteLoop
          MOVEQ     #0,D0                 ; clear out extension nybble
          MOVEQ     #0,D1                 ; clear out command byte
          MOVE.L    DMcmdPtr,A1           ; get the command pointer
          MOVE.B    (A1)+,D0              ; get the command byte
          MOVE.L    A1,DMcmdPtr           ; set new command pointer
          MOVE.B    D0,D1                 ; set up cmd routine offset
          AND.B     #$0F,D0               ; D0 = extension nybble
          LSR.B     #4,D1                 ; D1 = command nybble
          ADD.W     D1,D1                 ; double it (word offset)
          LEA       DMdispT,A2            ; get command dispatch table
          ADD.W     0(A2,D1.W),A2         ; get address of code into A2
          MOVE.L    DMmemPtr,A0           ; get memory ptr into A0
          JSR       (A2)                  ; and JSR to it

          TST.L     DMcmdPtr              ; nil command ptr => we're done
          BNE.S     CmdByteLoop           ; no, keep looping

          MOVE.L    DMmemPtr,D0           ; get the mem ptr
          CMP.L     DMmemEnd,D0           ; are we past the limit yet?
          BLT.S     TemplateLoop          ; present < ending, not done yet

@0        ADDQ      #4,SP                 ; pop memory template ptr
          RTS                             ; and return

DMdispT
          .WORD     DMHex-DMdispT
          .WORD     DMAscii-DMdispT
          .WORD     DMMP-DMdispT
          .WORD     DMSTR-DMdispT
          .WORD     DMText-DMdispT
          .WORD     DMSP-DMdispT
          .WORD     DMNL-DMdispT
          .WORD     DMnull-DMdispT
          .WORD     DMFwdMP-DMdispT
          .WORD     DMBwdMP-DMdispT
          .WORD     DMPtrMP-DMdispT
          .WORD     DMnull-DMdispT
          .WORD     DMRepeat-DMdispT
          .WORD     DMEnd-DMdispT
          .WORD     DMSTACK-DMdispT
          .WORD     DMnull-DMdispT


; The various commands are dispatched to with A0 = DMmemPtr, A1 = DMcmdPtr,
; and D0 = extension nybble

; DMbailOut -- something went wrong in a command, terminate the display

DMbailOut
          CLR.L     DMcmdPtr              ; force DM to stop current template loop
          CLR.L     DMmemEnd              ; force DM to stop current display

          BSR       WriteLine             ; print everything up to here

          MOVE.L    (SP)+,A0              ; save return address
          ADDQ      #6,SP                 ; pop off dummy word count/cmd ptr
          MOVE.L    A0,(SP)+              ; push return address

; DMnull -- not an implemented command

DMnull
          RTS                             ; just return

;-----------------------------------------------------------------------------------------
--
; DMHEX ($0X) -- Print memory as hex, use low three bits of extension nybble to specify
byte,
; word or long.
;-----------------------------------------------------------------------------------------
--

DMHEX
          MOVEQ     #0,D2                 ; set up bit test counter
@0        MOVE.B    (A0)+,D1              ; get next byte
          BTST      D2,D0                 ; was it a match in format size?
(byte/word/long)
          BNE.S     @1                    ; yes, finish up
          ADDQ      #1,D2                 ; bump format tester
          LSL.L     #8,D1                 ; shift current value up one byte
          BRA.S     @0                    ; and keep looping

; D1 now has the value to print, A0 = new memory pointer, D2 contains format size

@1
          BSR.S     DMprintHex
          MOVE.B    #' ',(A6)+            ; do up a space
          RTS                             ; and return

DMprintHex
          MOVE.L    A0,DMmemPtr           ; set new memory pointer value
          MOVE.L    D1,D0                 ; set up for print call
          CMP.B     #1,D2                 ; what format
          BLT.S     @0                    ; D2 = 0, must be byte
          BEQ.S     @1                    ; D2 = 1, word
          BRA       PNT8HX                ; must be long, 8 hex chars
@0        BRA       PNT2HX                ; byte, two hex chars
@1        BRA       PNT4HX                ; word, four hex chars

;-----------------------------------------------------------------------------------------
--
; DMASCII ($1X) -- Display the next D0+1 bytes of memory as ascii
;-----------------------------------------------------------------------------------------
--

DMASCII
          MOVEQ     #0,D1                 ; clear out D1
          MOVE.B    D0,D1               ; save count in D1

@0        MOVE.B    (A0)+,D0            ; get next char to print
          BSR       Bin2Char            ; print it out
          DBRA      D1,@0

DMAscExit
          MOVE.L    A0,DMmemPtr         ; set new memory pointer
          RTS                           ; and return

;-----------------------------------------------------------------------------
--
; DMMP ($20) -- Print the current memory pointer
;-----------------------------------------------------------------------------
--

DMMP
          MOVE.L    DMmemPtr,D0         ; get the current value to display
          BRA       PNT6HX              ; and print it out

;-----------------------------------------------------------------------------
--
; DMSTR ($30) -- print @MP as a pascal string
;-----------------------------------------------------------------------------
--

DMSTR
          MOVE.L    (A0)+,D0            ; get ptr value
          AND.L     MaskBC,D0           ; mask off high byte
          MOVE.L    D0,A1               ; A1 = string ptr
          BEQ.S     DMAscExit           ; nil string ptr, bail out

          MOVEQ     #0,D0               ; clear out length
          MOVE.B    (A1)+,D0            ; get length
          AND.B     #$3F,D0             ; restrict to < 64
          SUBQ.L    #1,D0               ; for dbra loop

@0        MOVE.B    (A1)+,(A6)+         ; transfer the character
          DBRA      D0,@0

          BRA.S     DMAscExit           ; update memptr

;-----------------------------------------------------------------------------
--
; DMTEXT ($4X) -- Print a string of text.  D0+1 = # of bytes to print out
;-----------------------------------------------------------------------------
--

DMTEXT
          MOVE.B    (A1)+,(A6)+         ; transfer byte
          DBRA      D0,DMText           ; top of loop

          MOVE.L    A1,DMcmdPtr         ; set new command ptr
          RTS                           ; and return

;-----------------------------------------------------------------------------
--
; DMSP ($5X) -- Print out D0+1 spaces
;-----------------------------------------------------------------------------
--

DMSP
```

```
          MOVE.B     #' ',(A6)+               ; stuff a space
          DBRA       D0,DMSP                  ; loop
          RTS

;-----------------------------------------------------------------------------------
--
; DMNL ($60) -- Print out a newline
;-----------------------------------------------------------------------------------
--

DMNL
          BSR        WriteLine                ; print out the current I/O buffer line
          TST.B      AbortPrint               ; check if user bailed out
          BNE        DMbailout                ; yup, exit this stuff
          RTS                                 ; otherwise return

;-----------------------------------------------------------------------------------
--
; DMFWDMP ($8X) -- Adjust the memory pointer forward by D0+1
;-----------------------------------------------------------------------------------
--

DMFWDMP
          ADD.L      A0,D0                    ; bump ptr up
          ADDQ.L     #1,D0                    ; since amount is one less than needed

MoveMPExit
          MOVE.L     D0,DMmemPtr              ; and set new mem ptr
          RTS                                 ; return

;-----------------------------------------------------------------------------------
--
; MDBWDMP ($9X) -- Decriment the memory pointer by D0
;-----------------------------------------------------------------------------------
--

DMBWDMP
          SUB.L      D0,A0
          SUBQ.L     #1,A0                    ; since amount is one less than needed
          MOVE.L     A0,D0                    ; set up for exit checks
          BRA.S      MoveMPExit

;-----------------------------------------------------------------------------------
--
; DMPTRMP ($A0) -- Set MP to be the value pointed at by MP, save advanced underefed
memptr
; on stack.
;-----------------------------------------------------------------------------------
--

DMPTRMP
          MOVE.L     (A0)+,D0                 ; deref DMmemPtr
          MOVE.L     A0,-(SP)                 ; save advanced, underefed memptr on stack
          BRA.S      MoveMPExit               ; common exit

;-----------------------------------------------------------------------------------
--
; DMREPEAT ($CX) -- Repeat until the next END for D0+1 times
;-----------------------------------------------------------------------------------
--

DMREPEAT
```

```
          MOVE.L    (SP)+,A2              ; save off return address
          MOVE.L    A1,-(SP)              ; push current command ptr
          MOVE.W    D0,-(SP)              ; push count
          JMP       (A2)                  ; and return

;-------------------------------------------------------------------------------------
--
; DMEND ($D0) -- Handle end of repeat loop/end of memory template.  TOS is the repeat
count (word),
; then the current command ptr.
;-------------------------------------------------------------------------------------
--

DMEND
          MOVE.L    (SP)+,A2              ; save return address
          MOVE.W    (SP)+,D0              ; pop the current count
          MOVE.L    (SP)+,DMcmdPtr        ; get command ptr
          TST.W     D0                    ; is count = 0?
          BEQ.S     DMexitA2              ; all done, finished with this repeat loop

          SUBQ.W    #1,D0                 ; dec count
          MOVE.L    DMcmdPtr,-(SP)        ; restore command ptr to stack
          MOVE.W    D0,-(SP)              ; push new count

DMexitA2
          JMP       (A2)                  ; and return

;-------------------------------------------------------------------------------------
--
; DMSTACK ($EX) -- Manipulate the stack
;-------------------------------------------------------------------------------------
--

DMSTACK
          MOVE.L    (SP)+,A2              ; save return address
          BTST      #0,D0                 ; is it a pop?
          BNE.S     DMPop                 ; yup

          MOVE.L    A0,-(SP)              ; push the memory pointer
          BRA.S     DMexitA2

DMPop
          MOVE.L    (SP)+,DMmemPtr        ; pop into memory ptr
          BRA.S     DMexitA2

;-------------------------------------------------------------------------------------
--
; Memory templates
;-------------------------------------------------------------------------------------
--

DMdefTemp
          .BYTE     $20                   ; MP
          .BYTE     $53                   ; SP(4)
          .BYTE     $02                   ; HEX(Word)
          .BYTE     $02                   ; HEX(Word)
          .BYTE     $02                   ; HEX(Word)
          .BYTE     $02                   ; HEX(Word)
          .BYTE     $50                   ; SP(1)
          .BYTE     $02                   ; HEX(Word)
          .BYTE     $02                   ; HEX(Word)
          .BYTE     $02                   ; HEX(Word)
          .BYTE     $02                     ; HEX(Word)
          .BYTE     $52                     ; SP(3)
          .BYTE     $9F                     ; BWDMP(16)
          .BYTE     $1F                     ; ASCII(16)
          .BYTE     $60                     ; NL
          .BYTE     $D0                     ; END

          .IF       0
NotSimpleMindedTestCaseForDMdefTemp
          .BYTE     $20                     ; MP
          .BYTE     $53                     ; SP(4)
          .BYTE     $C1                     ; REPEAT(2)
          .BYTE     $C3                     ;   REPEAT(4)
          .BYTE     $02                     ;     HEX(Word)
          .BYTE     $50                     ;     SP(1)
          .BYTE     $D0                     ;     END
          .BYTE     $51                     ;   SP(2)
          .BYTE     $D0                     ;   END
          .BYTE     $53                     ; SP(4)
          .BYTE     $9F                     ; BWDMP(16)
          .BYTE     $1F                     ; ASCII(16)
          .BYTE     $60                     ; NL
          .BYTE     $D0                     ; END
          .ENDC

DMiopb
          .BYTE     $8B                     ; FWDMP(12)
          .BYTE     $4C                     ; TEXT(13)
          .ASCII    'IOCOMPLETION '         ;
          .BYTE     $08                     ; HEX(Long)
          .BYTE     $48                     ; TEXT(9)
          .ASCII    'IORESULT '             ;
          .BYTE     $02                     ; HEX(Word)
          .BYTE     $60                     ; NL

          .BYTE     $49                     ; TEXT(10)
          .ASCII    'IONAMEPTR '            ;
          .BYTE     $30                     ; STR
          .BYTE     $60                     ; NL

          .BYTE     $49                     ; TEXT(10)
          .ASCII    'IOVREFNUM '            ;
          .BYTE     $02                     ; HEX(Word)
          .BYTE     $48                     ; TEXT(9)
          .ASCII    'IOREFNUM '             ;
          .BYTE     $02                     ; HEX(Word)
          .BYTE     $80                     ; FWDMP(1)
          .BYTE     $49                     ; TEXT(10)
          .ASCII    'IOPERMSSN '            ;
          .BYTE     $01                     ; HEX(Byte)
          .BYTE     $46                     ; TEXT(7)
          .ASCII    'IOMISC '               ;
          .BYTE     $08                     ; HEX(Long)
          .BYTE     $60                     ; NL

          .BYTE     $48                     ; TEXT(9)
          .ASCII    'IOBUFFER '             ;
          .BYTE     $08                     ; HEX(Long)
          .BYTE     $4A                     ; TEXT(11)
          .ASCII    'IOREQCOUNT '           ;
          .BYTE     $08                     ; HEX(Long)
          .BYTE     $60                     ; NL
          .BYTE     $4A                 ; TEXT(11)
          .ASCII    'IOACTCOUNT '       ;
          .BYTE     $08                 ; HEX(Long)
          .BYTE     $49                 ; TEXT(10)
          .ASCII    'IOPOSMODE '        ;
          .BYTE     $02                 ; HEX(Word)
          .BYTE     $60                 ; NL

          .BYTE     $4B                 ; TEXT(12)
          .ASCII    'IOPOSOFFSET '      ;
          .BYTE     $08                 ; HEX(Long)
          .BYTE     $60                 ; NL
          .BYTE     $D0                 ; END

DMwind
          .BYTE     $8F                 ; FWD(16)             * skip the grafport record
($6C or &108)
          .BYTE     $8F                 ; FWD(16)
          .BYTE     $8F                 ; FWD(16)
          .BYTE     $8F                 ; FWD(16)
          .BYTE     $8F                 ; FWD(16)
          .BYTE     $8F                 ; FWD(16)
          .BYTE     $8B                 ; FWD(12)

          .BYTE     $4A                 ; TEXT(11)
          .ASCII    'WINDOWKIND '       ;
          .BYTE     $02                 ; HEX(Word)
          .BYTE     $47                 ; TEXT(8)
          .ASCII    'VISIBLE '          ;
          .BYTE     $01                 ; HEX(Byte)
          .BYTE     $60                 ; NL

          .BYTE     $47                 ; TEXT(8)
          .ASCII    'HILITED '          ;
          .BYTE     $01                 ; Hex(Byte)
          .BYTE     $81                 ; FWDMP(2)
          .BYTE     $48                 ; TEXT(9)
          .ASCII    'STRUCRGN '         ;
          .BYTE     $08                 ; HEX(Long)
          .BYTE     $47                 ; TEXT(8)
          .ASCII    'CONTRGN '         ;
          .BYTE     $08                 ; HEX(Long)
          .BYTE     $60                 ; NL

          .BYTE     $49                 ; TEXT(10)
          .ASCII    'UPDATERGN '        ;
          .BYTE     $08                 ; HEX(Long)
          .BYTE     $87                 ; FWDMP(8)
          .BYTE     $4B                 ; TEXT(12)
          .ASCII    'TITLEHANDLE '      ;
          .BYTE     $08                 ; HEX(Long)
          .BYTE     $60                 ; NL

          .BYTE     $81                 ; FWDMP(2)
          .BYTE     $4B                 ; TEXT(12)
          .ASCII    'CONTROLLIST '      ;
          .BYTE     $08                 ; HEX(Long)
          .BYTE     $4A                 ; TEXT(11)
          .ASCII    'NEXTWINDOW '       ;
          .BYTE     $08                 ; HEX(Long)
          .BYTE     $60                 ; NL
          .BYTE     $00                  ; END

DMterc
          .BYTE     $48                  ; TEXT(9)
          .ASCII    'DESTRECT '          ;
          .BYTE     $02                  ; HEX(Word)
          .BYTE     $02                  ; HEX(Word)
          .BYTE     $02                  ; HEX(Word)
          .BYTE     $02                  ; HEX(Word)

          .BYTE     $48                  ; TEXT(9)
          .ASCII    'VIEWRECT '          ;
          .BYTE     $02                  ; HEX(Word)
          .BYTE     $02                  ; HEX(Word)
          .BYTE     $02                  ; HEX(Word)
          .BYTE     $02                  ; HEX(Word)
          .BYTE     $60                  ; NL

          .BYTE     $87                  ; FWDMP(8)        * skip the selRect
          .BYTE     $4A                  ; TEXT(11)
          .ASCII    'LINEHEIGHT '        ;
          .BYTE     $02                  ; HEX(Word)

          .BYTE     $47                  ; TEXT(8)
          .ASCII    'FIRSTBL '           ;
          .BYTE     $02                  ; HEX(Word)

          .BYTE     $48                  ; TEXT(9)
          .ASCII    'SELPOINT '          ;
          .BYTE     $02                  ; HEX(Word)
          .BYTE     $02                  ; HEX(Word)
          .BYTE     $60                  ; NL

          .BYTE     $48                  ; TEXT(9)
          .ASCII    'SELSTART '          ;
          .BYTE     $02                  ; HEX(Word)

          .BYTE     $46                  ; TEXT(7)
          .ASCII    'SELEND '            ;
          .BYTE     $02                  ; HEX(Word)

          .BYTE     $8F                  ; FWDMP(16)       * skip active...caretState
          .BYTE     $85                  ; FWDMP(6)

          .BYTE     $44                  ; TEXT(5)
          .ASCII    'JUST '              ;
          .BYTE     $02                  ; HEX(Word)
          .BYTE     $60                  ; NL

          .BYTE     $46                  ; TEXT(7)
          .ASCII    'LENGTH '            ;
          .BYTE     $02                  ; HEX(Word)

          .BYTE     $45                  ; TEXT(6)
          .ASCII    'HTEXT '             ;
          .BYTE     $08                  ; HEX(Long)

          .BYTE     $85                  ; FWDMP(6)        * skip recalBack...clikStuff

          .BYTE     $46                  ; TEXT(7)
          .ASCII    'CRONLY '            ;
          .BYTE     $02                  ; HEX(Word)
```

```
         .BYTE     $60               ; NL

         .BYTE     $46               ; TEXT(7)
         .ASCII    'TXFONT '         ;
         .BYTE     $02               ; HEX(Word)

         .BYTE     $46               ; TEXT(7)
         .ASCII    'TXFACE '         ;
         .BYTE     $02               ; HEX(Word)

         .BYTE     $46               ; TEXT(7)
         .ASCII    'TXMODE '         ;
         .BYTE     $02               ; HEX(Word)

         .BYTE     $46               ; TEXT(7)
         .ASCII    'TXSIZE '         ;
         .BYTE     $02               ; HEX(Word)
         .BYTE     $60               ; NL

         .BYTE     $46               ; TEXT(7)
         .ASCII    'INPORT '         ;
         .BYTE     $08               ; HEX(Long)

         .BYTE     $87               ; FWDMP(8)          * skip highHook/caretHook

         .BYTE     $46               ; TEXT(7)
         .ASCII    'NLINES '         ;
         .BYTE     $02               ; HEX(Word)

         .BYTE     $D0               ; END


DMdlog
         .BYTE     $60               ; NL
         .BYTE     $D0               ; END

         .ALIGN    2                 ; make sure we line up on a word boundary
```