Inter-Office Memorandum

To: Alto Users
From: Dan Ingalls
Subject: Bit BLT
Date: November 19, 1975
Location: Palo Alto
Organization: LRG

XEROX

Filed on: <INGALLS>BITBLT.BRAVO

This is a final description of the BIT BLT routines which grew out of discussions among Larry Tesler, Bob Sproull, Diana Merry and me. They are available as a BCPL driver with machine code inner loop and microcode which gets called if it is loaded. These routines assume that appropriate clipping or bounds checking has already been done on input. They are available as BITBLTB.BCPL, BITBLTA.ASM, BITBLT.MU in <INGALLS>BITBLT.DM.

Definitions

A bit map is a region of memory defined by bca and bmw, where bca is the base core address (starting location) and bmw is the bit map width in words; the number of scan lines is irrelevant for our purposes. (If both bmw and bca are even, then the bit map may be displayed on the screen using standard Alto facilities.)

A block is a rectangle within a bit map. It has four corners which need not fall on word boundaries. Any given block is described by a block descriptor whose contents are:

- Bit map's base core address (bca)
- Bit map's width in words (bmw)
- Block's Left x ("x offset")
- Block's Top y ("y offset")
- Block's Width
- Block's Height

A block is sometimes used to designate a sequence of bits in memory, such as the 16 wide 14 high region containing the bit pattern of a font character. In this case, bca point to the font character, bmw is 1, x and y are 0, width is 16, and height is 14.

Block Operations

From BCPL, one uses the call

FillBitMap (destination, function, source, gray) //in Caravan, or
FillBitMap (lv destination, function, lv source, gray) //in Alto OS.

The destination is a pointer to a block descriptor.

The function is encoded as operation + source-type. The operation codes are:

- 0 Replace: Destination \rightarrow Source
- 4 Paint: Destination \rightarrow Source xor Destination
- 8 Invert: Destination \rightarrow Source xor Destination
- 12 Erase: Destination \rightarrow not Source and Destination

The source-types are:
A gray is a one-word item specifying a 4-by-4 bit pattern to be used as a source. A solid constant source is specified with a gray whose four fields are equal.

The source depends on the source type:

0-4  A pointer to a block descriptor
2 8  A pointer to a block descriptor
and a gray
3 12  A gray

For source types 0-4, the source width and height are ignored and a simple transfer between equally-sized rectangles is performed.

The routine first considers the possibility of source-destination overlap and decides in which order to transfer words. It also generates masks and counts to be used in the transfer loops. Then a lower level routine is called which jumps into microcode if the RAM is loaded; otherwise it does the work in novacode.

Timing Details

The microcode has roughly the following speed characteristics:

- Horizontally, along one raster line (so to speak)
  - Store constant: 15 cycles/word
  - Move block (store): 36 cycles/word
  - Move block (OR): 42 cycles/word
- Vertical loop overhead (time to change raster lines): 25-30 cycles, depending on source/dest alignment
- Initial setup overhead (time to get going or resume from interrupt): approx 150 cycles

These are all in terms of Alto minor cycles and do include all memory wait time and do not include any degradation due to competing tasks, such as the display or disk.

Interim Details

The machine code is not presently reentrant, and consequently will crash if you try to use it in separate processes. We do not intend to alter this situation since, when the microcode is used, things behave reentrantly, and there is no problem. It is not hard to do, though, if someone has a good reason for it.

The microcode is not presently interruptable, and consequently large operations will cause significant delays in interrupt service (like 1/4 second to move most of the screen). This will be relieved shortly by allowing the microcode to save its state in the ACs and emerge to the nova emulator, later to resume where it left off.

We will probably provide an efficient entry for CONVRT-like operations; your suggestions in that direction are hereby solicited.
//FILE MAINTENANCE ---- \BBSCAN.SR

//April 20, 1976

//Notes and Code --- BitBlt and Scan Conversion --- D. MERRY

//The formal format for the table associated with BitBlt looks as follows:

//0  FUNCTION
//1  "GRAY"
//2  DESTINATION CORE BASE
//3  DESTINATION RASTER
//4  DESTINATION X
//5  DESTINATION Y
//6  WIDTH
//7  HEIGHT
//10 SOURCE CORE BASE
//11 SOURCE RASTER
//12 SOURCE X
//13 SOURCE Y
//14 SCRATCH GRAY
//15 SCRATCH GRAY
//16 SCRATCH GRAY
//17 SCRATCH GRAY

//The format of the "strike" font is the simple case of Cypress Glyphs as described
//in the FONT FORMAT memo of January 29, 1976. <MERRY>STRIKEFORMAT.BRAVO

//0  FORMAT
//1  MAXIMUM WIDTH
//2  ASCENT
//3  DESCENT
//4  OFFSET
//5  MIN
//6  MAX
//7  NSEGS
//10 SEGMENT WIDTH

FUNCTION

First address must be on even word boundary

"GRAY"

One-word constant defined by user

DESTINATION CORE BASE

Width in "Nova" words of destination rectangle

DESTINATION RASTER

DESTINATION X

DESTINATION Y

WIDTH

In bits

HEIGHT

In bits

SOURCE CORE BASE

SOURCE RASTER

SOURCE X

SOURCE Y

SCRATCH GRAY

Four locations for building gray words for microcode

SCRATCH GRAY

SCRATCH GRAY

SCRATCH GRAY

FORMAT

If high order bit on, it's a "strike" format font, otherwise it's in .AL format. For the simple case only the high order bit can be on in the strike format.

MAXIMUM WIDTH

Width of the widest character

ASCENT

Ascent + Descent + Segheight (Fontheight)

DESCENT

OFFSET

Negative for kerned font, 0 normally.

MIN

Character less than Min not used for some sort of control by the user will be displayed as illegal character

MAX

Largest legal Ascii in this font. Max + 1 will probably be the "Shazam" character which will be displayed whenever a character greater than Max is requested.

NSEGS

Must be 1 in the simple case

SEGMENT WIDTH

Total width of font in bits

This value + 15 and divided by 16 yields the raster value for BitBlt (simple case)
PINCH TOP 0 in simple case.
PINCH BOTTOM 0 in simple case.
CHARPOINTERS MIN thru MAX+2, indexed by Ascii. Value is left x of selected glyph. MAX+1 is the "illegal" character and MAX+2 the right x of "illegal" character.
SEGMENT ((Segwidth + 15)/16) * segheight

The following is code for scan conversion of characters using BitBlt and the "strike" font format. The table passed to BitBlt will look like this:

FUNCTION May vary with each character -- set by user. This location must be on even word boundary.
"GRAY" Only relevant if "painting" characters, i.e. Function > 7.
DESTINATION CORE BASE First word address of memory used for Alto display
DESTINATION RASTER Width of Display in "Nova" words
DESTINATION X Must be set by user for every "new line", updated by routine on each character.
DESTINATION Y Set by user -- will typically have "fontheight" added to it for a new line
WIDTH Computed -- Ascii+1's x - Ascii's x.
HEIGHT Fontheight
SOURCE CORE BASE Pointer to the bits of the font -- created by some setup routine
SOURCE RASTER (Segmentpointer + 15) / 16
SOURCE X Value in location Charpointers + Ascii
SOURCE Y 0
SCRATCH GRAY Place for Scratch Grays in case painting font.
SCRATCH GRAY
SCRATCH GRAY
SCRATCH GRAY
SAVE AC1 Place to save AC's.
SAVE AC2
SAVE AC3
BITBLT FONT Place to keep pointer to font whilst putting out a string of characters.
CHARACTER Place to hold character code, to facilitate exception checking.
CHANGE Zero means there has been no change in the font, the function, or the "gray" since the last time a string was scan converted. > 0 means to set up BBSTABLE according to information provided in SCANTABLE passed in AC1.

Pointer to subroutine which when called will return width of character passed in AC0 -- expects pointer to BBSTABLE in AC2.
HEIGHT SUBROUTINE
Pointer to subroutine which when called will return height of font -- expects pointer to BBSTABLE in AC2

SCAN SUBROUTINE
Pointer to subroutine which will make call on BitBlit -- depends on the kind of font being used.

SAVE GRAY (TEMP1)
Only necessary if going to have “gray” characters

GRAY COUNT (TEMP2)
Ditto

TRAIL CHARS
Needed only for justification in Smalltalk. Signals whether there have been non-space characters since the last space -- helps to deal with multiple spaces. Used in Smalltalk for window clipping

CROSSLEFT
Ditto

CROSSRIGHT
Ditto

RIGHTMARGIN
Ditto

MEASURE
Smalltalk switch -- so that PUTCCHARS code can be used both for measuring and scan converting

TEMP3
Needed only if .AL fonts expected

LASTVISIBLE
For returning last visible character when clipping occurs

//The format of the SCANTABLE passed in AC1 whenever there is a call for putting a string of characters is or setting up the BBSTABLE in preparation for a call for displaying a string of characters is as follows:

//0
FUNCTION
OR, STORE, etc.

//1
GRAY
Meaningful only if FUNCTION >7.

//2
FONT
Pointer to first word of font.

//3
DESTINATION X
Beginning X of first character of STRING.

//4
DESTINATION Y
“Top” Y of first character of STRING.

//5
STRING TABLE
Pointer to table with following format:

STRING POINTER
BEGINNING CHARACTER (BYTE)

LAST CHARACTER

//The following code will probably eventually check to see if the font has been set up and call setup code if necessary. In any event it assumes the following contain appropriate values:

//FUNCTION
//GRAY
//DESTINATION CORE BASE
//DESTINATION RASTER
//DESTINATION X
//DESTINATION Y
//SOURCE CORE BASE
//SOURCE RASTER
//SOURCE Y

//The Ascii value is received in AC0 and a pointer to the BitBlt table in AC1.

.TITL STRIKESCAN
.GETNOLIST "SMALLOPS"
.GETNOLIST "SMALLSYMS"
.GETNOLIST "SMDDISP.SYMS"
BEXTZ SMF      --DECLARED IN SMALL.SYMS
BEXT SETSCAN, PUTCHARS, DISPAD, DISPWD, DOIST
     ;///DECLARED IN SMDISP.SYMS

SREL
SETSCAN: SETSCANC
PUTCHARS: PUTCHARSC
DISPAD:  0
DISPWD:  0

EXCEPT: EXCEPTC
CLIP: CLIPC
STRIKESC: STRIKESCANC
ALSCAN: ALSCANC
DOIST: DOISTC

NREL

C7:  7
C13: 13
DISPAD: DISPAD
DISPWD: DISPWD

BITBLT = 60400

OFFSETS DEFINED IN SMDISP.SYMS

FUNC = 0
GRAY = 1
DBASE = 2
DRAST = 3
DESTX = 4
DESTY = 5
WIDTH = 6
HEIGHT = 7
SBASE = 10
SRAST = 11
SRCX = 12
SRCY = 13
GRAY1 = 14
GRAY2 = 15
GRAY3 = 16
GRAY4 = 17
SAV1 = 20
SAV2 = 21
SAV3 = 22
BBFONT = 23
CHAR = 24
CHANGE = 25
WIDTHSUBR = 26
HEIGHTSUBR = 27
SCANSUBR = 30
SAVGRAY = 31
GRAYCNT = 32
TEMP1 = 31
TEMP2 = 32
TRLCMR = 33
CROSSLEFT = 34
CROSSTIGHT = 35
RIGHTMARGIN = 36
MEASURE = 37
TEMP3 = 40
LASTVISIBLE = 41

FUNC = 0
GRAY = 1
FONT = 2
DX = 3
DY = 4
STRINGTAB = 5

FORMAT = 0
MAXWIDTH = 1
ASCENT = 2
DESCE N = 3
OFFSET = 4
MIN = 5

OFFSETS INTO BBSTABLE

FUNCTIONS INTO SCANTABLE

OFFSETS INTO FONT
LDA 0,@.STRIKESCAN
STA 0,SCANSUBR,2  ;//PASS BACK PTR TO SCAN CONVERTING SUBR IN
                      ;//BBSTABLE
MOV 2,3
LDA 2,SAV2,2       ;//RESTORE AC2 -- AC3 CONTAINS
                      ;//PTR TO BBSTABLE
                      ;//WHICH CALLER WILL WANT TO SQUIRREL AWAY
JMP @1,2

SETAL:
INC 3,3
INC 3,3
STA 3,BBFONT,2
MKONE 0,0
STA 0,SRAST,2
MKZERO 0,0
STA 0,SRX2,2
LDA 0,@.GETWDAL
STA 0,WIDTHSUBR,2
LDA 0,@.GETHTAL
STA 0,HEIGHTSUBR,2
LDA 0,@.ASCAN
JMP SETTRN  ;//PASS BACK PTR TO SCAN CONVERTING SUBR IN
             ;//BBSTABLE

PUTCHARSC:
STA 3,1,2      ;STA 3, PUTCMT, 2
MOV 0,3
LDA 0,CHANGE,3
SIZ 0,0       ;//CHECK IF FONT, FUNCTION, OR GRAY CHANGED
JMP NOCHANGE
LDA 0,1,2     ;//IF CHANGE SAVE RETURN PTR FOR PUTCHE
STA 0,SAV3,3  ;//IN BBSTABLE AND CALL
              ;//SETSCAN WITH PTR
MOV 3,0       ;//TO BBSTABLE IN AC0 -- PTR IN SCANTABLE
JSR SETSCANNC
LDA 0,SAV3,3  ;//RESTORE RETURN PTR --
              ;//BOMBED BY SETSCAN
STA 0,1,2

NOCHANGE:
STA 2,SAV2,3   ;//SAVE PTR TO AC2 IN BBSTABLE --
MOV 3,2       ;//FOR BCPL
MOV 1,3       ;//BBSTABLE IN AC2
LDA 1,DX,3    ;//SCANTABLE IN AC3
STA 1,DEXTY,2 ;//SET UP DESTINATION X OF FIRST
              ;//CHAR IN STRING
LDA 1,DX,3    ;//AND TOP Y
STA 1,DESTRX,2
MKZERO 1,1    ;//WIDTH GOES INTO SCAN
              ;//CONVERSION AS ZERO
LDA 1,WIDTH2,2
LDA 1,STRINGTAB,3 ;//PTR TO STRING TABLE IN AC1 FOR SMF
LDA 0,SCANSUBR,2 ;//SCAN CONVERSION SUBR CALLED BY SMF

;SMF (SUBRUCTOR-MAP-FETCH) IS AN ENTRY IN A STRING PACKAGE WRITTEN BY
;LARRY TESLER --
;GOING IN:
;AC0 = PTR TO SUBROUTINE (MAP FUNCTION) TO BE CALLED
;WITH EACH CHARACTER
;AC1 = PTR TO STRING TABLE:
;STRING POINTER
;FIRST CHARACTER (BYTE PTR)
;LAST CHARACTER (BYTE PTR)
;AC2 = TRANSPARENT
;THE 1 FOLLOWING THE JSR MEANS WE'RE LOOKING AT A STRING AND
;PROCEEDING FROM FIRST TO LAST
;A = -1 WOULD MEAN TO PROCEED FROM LAST TO FIRST
;EACH TIME SMF CALLS THE DESIGNATED SUBROUTINE WITH AN
;ASCII VALUE IN AC0 AND THE CURRENT CHARACTER (BYTE PTR)
;IN AC1
;A NOSPRT RETURN MEANS THAT THE STRING HAS BEEN EXHAUSTED -- EITHER
;LAST CHARACTER REACHED OR BEYOND LENGTH OF STRING. A SKIP RETURN
;IS CAUSED BY A SKIP RETURN FROM THE USER'S SUBROUTINE. I ASSUME THERE
;IS A SIMILAR PACKAGE FOR PUMPING STRINGS IN BCPL

JSR @.SMF
1
SKIP         ;//NOSPRT - SUBSTRG EXHAUSTED
             ;//FOR MEASURING-----
             ;//AC0 = CHAR CODE
             ;//AC1 = BYTE PTR
JMP RETURN
LDA 3,DSPLGBS
NIL 0.0
STA NSPC,3
LDA 1,SSAV,2 // AND AC1 GETS BYTE PTR + 1 -- WHEN SUBSTRG
INC 1.1
// SMF DOES NOT RETURN BYTE PTR
RETURN:
LDA 2,SSAV,2
JMP @1,2

STRIKESCANN:
STA 1,SSAV,2 // BBSTABLE IN AC2
LDA 3,BBFONT,2 // CHECK FOR 'LEGAL' ASCII
STA 0,CHAR,2 // EXPEDITED -- MAY NOT BE NEEDED
STA 1,SSAV,2 // DITTO -- BUT REQUIRED IN SMALLTALK
// CURRENT BYTE PTR FROM SMF HERE
LDA 1,MIN,3
SGE 0.1
JSR EXCEPTC
LDA 1,MAX,3
SLE 0.1
INC 1.0
STA 0,CHAR,2
// MAX + 1 IS 'ILLEGAL' CHARACTER
// EXCEPTION CODE; E.G. FOR CR, SPACE, ETC.
// COULD BE PUT HERE -- FOR SMALLTALK
// EXCEPTION CHARACTERS WILL BE KNOWN BY
// THEIR ZERO WIDTH
LDA 1,MIN,3 // SUB MIN FROM CHAR CODE -- SO INDEXING
SUB 1.0
// CORRECT
LDA 3,WIDTH,2 // UPDATE DESTX--WIDTH 0 FIRST TIME--SET IN
// PUTCHARS
LDA 1,DESTX,2
ADD 1.3
STA 1,DESTX,2
LDA 3,BBFONT,2
ADD 0.3
LDA 1,CHARPTRS,3
STA 1,SRXC,2
LDA 0,CHARPTRS+1,3
SUB 1.0
STA 0,WIDTH,2
SNZ 0.0
JSR EXCEPTC
JSR CLIPC
JMP NOSCAN
MOV 2.0
ZER 1.1
BITBLT
MOV 0.2
// BBSTABLE SENT IN AC0 FOR NOW
// AND AC1 IS FLAG********
// PUT TABLE BACK IN 2 CAUSE SMF BOMBS 3

NOSCAN:
JMP @SSAV,2

CLIPC:
MOV 3.1
// SUBR FOR WINDOW CLIPPING
// SAVE RETURN -- BBSTABLE COMES IN AC2
STA 1,TEMP,2
LDA 0,CHAR,2
LDA 1,SPACE
SNE 0.1
JMP SPCIT
ISZ TRLCR,2
// INDICATE NON-SPACE CHAR -- HELP DEAL
// WITH MULTIPLE SPACES IN JUSTIFICATION
// SOMETIMES USED AS NIL FLAG -- ARGHH!
NOP

SPCIT:
LDA 1,WIDTH,2
LDA 0,DESTX,2
LDA 3,CROSSLEFT,2
SLE 0.3
JMP CHKRT1
ADD 0.1
MOV 1.0
SGT 1.3
JMP RTN
SUB 3.1
// DIFF BETWEEN CROSSLEFT AND RIGHTX IS WIDTH
// WIDTH -- DIFF TO BE ADDED TO SRCX AND DESTX

LDA 1,SRCX,2
ADD 3,1
STA 1,SRCX,2
LDA 1,DESTX,2
ADD 3,1
STA 1,DESTX,2
JMP CHKRT2

; // AND DESTX

CHKRT1:
LDA 3,CROSSRIGHT,2
SLE 0,3
JMP RTN

; // CHECK HERE IS INITIAL DESTX OFF RIGHT

ADD 1,0

; // ADD IN WIDTH FOR NEXT CHECK

CHKRT2:
LDA 3,RIGHTMARGIN,2
LDA 1,CROSSRIGHT,2
SNNIL 1,1
JMP CHKEDGE

; // NOW CHECK FOR SPANNING CROSSRIGHT

SET 1,1
JMP CHKEDGE

; // IF CROSSRIGHT NIL -- WE'VE ALREADY CROSSED IT

JMP CHKEDGE

; // IF CROSSRIGHT LESS THAN RIGHTMARGIN

SGT 0,1
JMP CHKCHAR
NIL 3,3
STA 3,CROSSRIGHT,2

; // IF WITHIN CROSSRIGHT SCAN IN

; // OTHERWISE SET CROSSRIGHT TO NIL AND

; // PRUNE

PRUNE:
LDA 3,SAVI,2
STA 3,LASTVISIBLE,2
SUB 1,0
LDA 1,WIDTH,2
SUB 0,1
STA 1,WIDTH,2

; // PRUNED CHARACTER WILL BE LAST VISIBLE

; // MAINLY FOR SMALLTALK PURPOSES

; // SPANNING CROSSRIGHT OR RIGHT MARGIN

; // PRUNING WIDTH WILL GET PARTIAL CHAR

CHKCHAR:
LDA 0,CHAR,2
LDA 1,SPACE
LDA 3,TAB
SEQ 0,1
SNE 0,3
JMP RTN
JMP SKPRTN

; // SKPRTN WILL CALL BITBLT

CHKEDGE:
MOV 3,1
LDA 3,CROSSRIGHT,2
SNNIL 3,3
JMP RTN
SGT 0,1
JMP CHKCHAR
NIL 3,3
STA 3,TRLC,2
JMP PRUNE

; // IF ALREADY CROSSED RIGHT DON'T SCAN

; // IF SPANNING RIGHT MARGIN THEN PRUNE

; // NIL TRAIL CHAR COUNT AS FLAG FOR STOPPING

; // AT MEASURING TIME

; // AND GO PRUNE

SKPRTN:
LDA 1,MEASURE,2
SZE 1,1
JMP MRTN
LDA 3,TEMP1,2
JMP 1,3

; // AC0 PRESERVED HERE FOR MSE FINDING ROUTINES

; // FIRST SEE IF WE'RE MEASURING -- DIFFERENT RTN

; // SKPRTN MEAN SCAN IT IN

RTN:
LDA 1,MEASURE,2
SZE 1,1
JMP MRTN
JMP @TEMPl,2

; // AC0 PRESERVED HERE FOR MSE FINDING ROUTINES

; // FIRST SEE IF WE'RE MEASURING -- DIFFERENT RTN

; // NOSKP MEANS NO SCAN

; // EXCEPTION CODE -- HANDLES CR, TAB, AND SPACE

EXCEPTC:
MOV 3,1
STA 1,TEMPl,2
LDA 3,DISPLBS
LDA 0,CHAR,2
LDA 1,SPACE
SNE 0,1
JMP DOSPACE
LDA 1,TAB
SNE 0,1
JMP DOTAB
LDA 1,LCR
SNE 0,1
JMP DOCR
ZER 0,0

; // BBSTABLE COMES IN AC2

; // FOR SMALLTALK
STA 0,WIDTH,2
LDA 0,C257
LDA 3,SCANSUBR,2
JMP 1,3

//RECALL SCAN SUBR
//WITH ILLEGAL CHAR

DOSPACE:
LDA 0,MEASURE,2
SZE 0,0
JMP MSPACE
LDA 0,THISLINE,3
SZE 0,0
JMP NOJST

//SEE IF IN MEASURE MODE
//IF ACTUALLY SCAN CONVERTING THEN
//SEE IF JUSTIFYING

DOJSTC:
DSZ CNT1,3
//ENTRY USED BY SMALLTALK FNDMS ROUTINE
//IF SO, SEE IF WE'VE
//COUNTED DOWN COUNTERS
JMP JUSTIT
LDA 1,CNT2,3
SZE 1,1
JMP LASTJST //IF CNT2 ALREADY ZERO OR COUNTED DOWN?
STA 1,CNT1,3
ISZ LEAD,3
ISZ NWID,3
MKZERO 0,0
STA 0,CNT2,3
JMP JUSTIT

//COUNT TIME AND STOP
//UPDATE LEAD SO TABS WORK RIGHT
//NWID+1 NEW JUST SPACE
//ZERO CNT2

LASTJST:
STA 1,THISLINE,3
//SHUT OFF JUSTIFICATION FOR NEXT TIME
//AND UPDATEX

JUSTIT:
LDA 0,LEAD,3
LDA 1,DELTA,3
ADD 0,1
STA 1,DELTA,3
LDA 0,NWID,3

//BUMP TAB DELTA -- LEAD SET IN LOUT,SR

UPDATEWIDTH:
STA 0,WIDTH,2
LDA 1,DESTX,2
ADD 0,1
STA 1,SPCX,3

//SAVE PROPER RIGHTX
//FOR JUSTIFICATION
JMP RTN

NOJST:
LDA 0,FSPACE,3
JMP UPDATEWIDTH

CR:
TAB:
SPACE:
15
11
40

.TABWD:
TABWD
//IN DSPUTIILS,SR

.DOTAB:
JSR H,TABWD
//RETURNS TABWIDTH IN AC0

DOCR:
STA 0,ISTCR,3
LDA 1,SAV1,2 //GET CURRENT BYTE PTR INTO AC1
LDA 3,SAV3,2
JMP 1,3

//SAVE FOR CHECKING IN JUSTIFICATION
//POP OUT OF SMF IN PUTCHARS

MSPACE:
LDA 0,TRLCHR,2
SZE 0,0
JMP MULSPC
ZER 0,0
STA 0,TRLCHR,2
STA 0,NSPC2,3

//SPACE EXCEPTION CODE WHEN MEASURING
//FIX MULTIPLE SPACES IF NECESSARY

MULSPC:
LDA 1,SAV1,2
STA 1,LSTSP,3
ISZ NSPC,3
ISZ NSPC2,3
JMP NOJST

//SAVE PTR TO THIS SPACE
//BUMP SPACE COUNTERS
//IF NOT GO GET MORE CHAR

MRTN:
LDA 0,TRLCHR,2
SNIL 0,0
JMP @TEMP1,2
LDA 3,DSPLBS
LDA 0,CHAR,2
LDA 1,LSTSP,3
SNNIL 1,1
JMP NOSPC

//NIL TRLCHR TELLS US PAST RIGHTMARGIN
//IF NOT GET MORE CHAR
//GET LAST CHAR INTO AC0 FOR PSTRG
//NIL LSTSP MEANS NO SPACES IN LINE
LDA 1,SPCX,3 //MAKE LAST DEXTX
STA 1,DEXTX,2 //CORRECT FOR JUSTIFICATION
LDA 1,LISTP,3 INC 1,1
STA 1,DEXTX,2 //SEND BYTE PTR BACK IN AC1 AND CHAR CODE IN
INC 1,1 //AC0 -- EXPECTED IN PSTRG SR
MOUT:
LDA 3,SAV3,2 //SKPRTN OUT OF SMF IN PUTCHARS
JMP 1,3
NOSPC:
LDA 1,SAV1,2 //NO SPACES IN LINE MEANS
JMP MOUT //SAV1 HAS CORRECT
//BYTE PTR FOR PSTRG SR

LBYTEMSK: 17400
RBYTEMSK: 377
C20: 20
C257: 401
.CLP: CLIP

ALSCANC:
STA 3,SAV3,2 //BBSTABLE COMES IN AC2
STA 1,SAV1,2 //SAVE SMF'S BYTE PTR

EXTENTION:
STA 0,CHAR,2 //SAVE CHAR CODE FOR EXCEPTION CHECKING
LDA 1,WIDTH,2
LDA 1,DEXTX,2
ADD 1,3
STA 3,DEXTX,2
LDA 3,BBFONT,2
ADD 0,3
LDA 0,0,3
ADD 0,3
LDA 0,0,3
STA 0,TEMP2,2
LDA 1,C20
MOVZR 0,0 SNC
MOV 1,0
STA 0,WIDTH,2
LDA 0,1,3
LDA 1,LBYTEMSK
ANDS 0,1
LDA 0,DESTY,2
STA 0,TEMP3,2
ADD 1,0
STA 0,DESTY,2
LDA 0,1,3
LDA 1,RBYTEMSK
AND 0,1
STA 1,HEIGHT,2
SUB 1,3
STA 3,SBASE,2
LDA 0,CHAR,2
LDA 1,SPACE
SNE 0,1
JMP EXCEPTION
LDA 1,TAB
SNE 0,1
JMP EXCEPTION
LDA 1,CR
SNE 0,1

EXCEPTION:
JSRR .EXCEPT
JSRR .CLIP
JMP NOSCANAL
MOV 2,0
ZER 1,1
BITFLT
MOV 0,2

NOSCANAL:
LDA 0,TEMP3,2 //FIX DESTY
STA 0,DESTY,2
MKZERO 0,0 //FIX SRCX -- LEFTSIDE CLIPPING MAY ALTER
STA 0,SRCX,2
LDA 0,TEMP2,2
MOVZR 0,0 SNC
JMP EXCEPTION
JMP @SAV3,2 //AND GO FOR ANOTHER CHARACTER

.EXCEPT:

.END