The "Bit BLT" package is a group of subroutines that implements various operations on display windows. Among their several uses are displaying characters, scrolling, and "painting" with brushes. The operations which can be performed using Bit BLT are not necessarily confined to uses with the Alto display, but the description below focuses on simple cases with the display in mind. More complicated possibilities are left to the reader to invent as necessity and curiosity dictate. The binary files required are available as BitBLTB.Br, BitBLTA.Br, BitBLT.Mb in <ALTO>BitBLT.Dm. The sources, BitBLTB.Bepl, BitBLTA.Asm, BitBLT.Mu are available on <ALTOSOURCE>.

Definitions

A bit map is a region of memory that describes a collection of scan-lines which have a base core address (bca) and bit map width (bmw), the later being a word value. Our discussion will assume that scan-lines run horizontally from left to right (bits are addressed from \( x = 0 \) to \( x = \text{bmw} \times 16 - 1 \)). Scan-lines appear consecutively in a bit map. Thus the point \((0,0)\) is at the upper left of the display bit map -- \( y = 0 \) is the first scan-line, the next \( y = 1 \), etc. The core address of the first word of scan-line 0 is bca. The number of scan-lines is not relevant, for the purposes of Bit BLT. (Note that these conventions are similar to those for the Alto display bit map; if bca and bmw are both even, the bit map may be displayed 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 Leftx ("x offset")
- Block's Topy ("y offset")
- Block's Width
- Block's Height

Thus the block is defined by:

\[
\text{Leftx} \leq x \leq \text{Leftx} + \text{Width} - 1 \\
\text{Topy} \leq y \leq \text{Topy} + \text{Height} - 1.
\]

It is left to the caller to insure that ranges of \( x \) and \( y \) indeed fall within the bit map.
The Bcpl routine contains a structure declaration for managing this descriptor:

```c
structure BITRECTANGLE :
{
 bca
 bmw
 Leftx
 Topy
 Width
 Height
}
```

A block is sometimes used to designate a sequence of bits in memory, such as a 16 wide by 14 high region containing the bit pattern of a font character. In this case, bca points 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(lv destination, function, lv source, gray).
```

The destination is a pointer to a BITRECTANGLE structure.

The function is encoded as operation + source-type. The operation codes (2 low-order bits) are:

<table>
<thead>
<tr>
<th>Operation Code</th>
<th>Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>replacefunction:</td>
<td>Destination + Source</td>
</tr>
<tr>
<td>1</td>
<td>paintfunction:</td>
<td>Destination + Source xor Destination</td>
</tr>
<tr>
<td>2</td>
<td>invertfunction:</td>
<td>Destination + Source xor Destination</td>
</tr>
<tr>
<td>3</td>
<td>erasefunction:</td>
<td>Destination + not Source and Destination</td>
</tr>
</tbody>
</table>

These names and values are declares as manifests in the Bcpl routine.

The source-types (next 2 bits) are:

<table>
<thead>
<tr>
<th>Source Type Code</th>
<th>Source Type Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>blocksource: A block of a bit map</td>
</tr>
<tr>
<td>4</td>
<td>compblocksouce: The complement of a block of a bit map</td>
</tr>
<tr>
<td>8</td>
<td>brushsource: A block as a brush with a gray</td>
</tr>
<tr>
<td></td>
<td>A brush emits gray where the brush</td>
</tr>
<tr>
<td></td>
<td>is 1, and a copy of the destination (transparency)</td>
</tr>
<tr>
<td></td>
<td>where the brush is 0</td>
</tr>
<tr>
<td>12</td>
<td>constanctsouce: A gray</td>
</tr>
</tbody>
</table>

A gray is a one-word item of four 4-bit bytes that defines a 4-by-4 repeating bit pattern to be used as a source. A solid constant source is specified with a gray whose four fields are equal. (Note: The pattern always repeats so as to start at x and y positions in the destination that are even multiples of 4. Thus several adjoining blocks filled with the same gray pattern will not show a ragged half tone phase seam between them.)

There are several manifests declared defining a collection of useful values as constant sources:

<table>
<thead>
<tr>
<th>Gray Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>black</td>
<td>-1</td>
</tr>
<tr>
<td>darkestgray</td>
<td>#76575</td>
</tr>
<tr>
<td>darkergray</td>
<td>#165727</td>
</tr>
</tbody>
</table>
The source depends on the source type:

- **0, 4**: A pointer to a block descriptor
- **8**: A pointer to a block descriptor and a gray
- **12**: A gray

For source types 0 and 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. The check for whether or not the microcode is present is performed every time the lowerlevel routine (FillBits) is called.

### 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 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.

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 the microcode modification cited above will permit reentrant operation.
IIFILE 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"    First address must be on even word boundary
//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 Gly'phs as described in the FONT FORMAT memo of January 29, 1976. <MERRY>STRIKEFORMAT.BRAVO

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

//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.

//Width of the widest character

//Ascent + Descent = Segheight (Fontheight)

//Negative for kerned font, 0 normally.

//Smallest legal Ascii in this font. Characters less than Min not used for some sort of control by the user will be displayed as illegal character

//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.

//Must be 1 in the simple case

//Total width of font in bits. This value + 15 and divided by 16 yields the raster value for BitBlit. (simple case)
PINCH TOP
PINCH BOTTOM
CHARPOINTERs
SEGMENT

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
"GRAY"
DESTINATION CORE BASE
DESTINATION RASTER
DESTINATION X
DESTINATION Y
WIDTH
HEIGHT
SOURCE CORE BASE
SOURCE RASTER
SOURCE X
SOURCE Y
SCRATCH GRAY
SCRATCH GRAY
SCRATCH GRAY
SCRATCH GRAY
SAVE AC1
SAVE AC2
SAVE AC3
BITBLT FONT
CHARACTER
CHANGE
WIDTH SUBROUTINE

May vary with each character -- set by user. This location must be on
even word boundary.
Only relevant if "painting" characters, i.e. Function ≠ 7.
First word address of memory used for Alto display
Width of Display in "Nova" words
Must be set by user for every
"new line", updated by routine on
each character.
Set by user -- will typically
have "fontheight" added to it for
a new line
Computed -- Ascii+1's x - Ascii's x.
Fontheight
Pointer to bits of the font --
created by some setup routine
(Segmentpointer + 15) / 16
Value in location Charpointers + Ascii
0
Place for Scratch Grays in case
painting font.
Place for Scratch Grays in case
painting font.
Place to save AC's.
Place to keep pointer to font whilst putting
out a string of characters.
Place to hold character code, to
facilitate exception checking.
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 BitBlt -- depends on the kind of font being used.

SAVE GRAY (TEMPI)

Only necessary if going to have "gray" characters

GRAY COUNT (TEMPI)

Ditto

TRAIL CHAR

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.

CROSSEFT

Ditto

CROSSRIGHT

Ditto

MEASURE

Smalltalk switch -- so that PUTCHARS 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:

FUNCTION OR, STORE, etc.

GRAY Meaningful only if FUNCTION >7.

FONT Pointer to first word of font.

DESTINATION X Beginning X of first character of STRING.

DESTINATION Y "Top" Y of first character of STRING.

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 Usually "or" or "store"

GRAY Usually not relevant

DESTINATION CORE BASE Usually upper left hand corner display word ad

DESTINATION RASTER "Display width"

DESTINATION X Set by user for first character in line --

DESTINATION Y updated by this routine for subsequent characters

SET by user line by line

HEIGHT Fontheight -- set once per font

SOURCE CORE BASE First word ad of font bits -- set once per font

SOURCE RASTER Width of font bits in words -- set once per font

SOURCE Y Always 0 in the simple case

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

.TTTL STRIKESCAN

.OUT

.GETNolist "SMALLOOPS"

.GETNolist "SMALLSYMS"

.GETNolist "SMDISP.SYMS"
//BEXTZ SMF --DECLARED IN SMALL.SYMS
//BEXT SETSCAN, PUTCHARS, DISPAD, DSPWDO,DOJST

.SREL
SETSCAN: SETSCANC
PUTCHARS: PUTCHARSC
DISPAD: 0
DSPWDO: 0

.EXCEPT: EXCEPTC
C7: 7
.DISPAD: DISPAD
.DSPWDO: DSPWDO

BITBLT = 60400

//OFFSETS DEFINED IN SMDISP.SYMS

//FUNC = 0 //OFFSETS INTO BBSTABLE
//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
//SAVGRA = 31
//GRAYCNT = 32
//TEMP1 = 31
//TEMP2 = 32
//TLCHR = 33
//CROSLEFT = 34
//CROSRIGHT = 35
//RIGHTMARGIN = 36
//MEASURE = 37
//LASTVIS = 40
//LASTVIS = 41

//FUNC = 0 //OFFSETS INTO SCANTABLE
//GRAY = 1
//FONT = 2
//DX = 3
//DY = 4
//STRINGTAB = 5

//FORMAT = 0
//MAXWIDTH = 1
//ASCENT = 2
//DESCENT = 3
//OFFSET = 4
//MIN = 5
// MAX = 6
// NSEGS = 7
// SEGWIDTH = 10
// PINCHTOP = 11
// PINCHBOT = 12
// CHARPTRS = 13

// BBSTABLEA: .BLK 41 // ALLOCATED IN DSPGLBS.SR

// SETSCAN SETS UP AS MUCH OF BITBLT'S TABLE AS POSSIBLE.
// IT WILL BE CALLED WHENEVER THERE IS A FONT CHANGE, A FUNCTION CHANGE,
// OR A "GRAY" CHANGE. IT MUST BE CALLED ONCE BY THE USER IN ORDER
// TO GET THE BBSTABLE ONTO AN EVEN WORD BOUNDARY. THEREAFTER,
// PUTCHARS WILL CHECK THE CHANGE LOCATION IN THE BBSTABLE TO
// DETERMINE IF IT NEED BE CALLED. THE CALL EXPECTS:
// AC0 -- POINTER TO BLOCK OF 31 (OCTAL) WORDS FOR BITBLT TABLE
// AC1 -- POINTER TO SCANTABLE:
// FUNCTION
// GRAY
// DESTINATION CORE BASE
// DESTINATION RASTER
// HEIGHT
// SOURCE CORE BASE
// SOURCE RASTER
// SOURCE Y
// BBFONT
// SCAN CONVERSION SUBROUTINE
//
// FOR A STRIKE FONT, THE ROUTINE WILL ESTABLISH:
// FUNCTION
// GRAY
// DESTINATION CORE BASE
// DESTINATION RASTER
// HEIGHT
// SOURCE CORE BASE
// SOURCE RASTER
// SOURCE Y
// BBFONT
// SCAN CONVERSION SUBROUTINE
//
// FOR AN AL FONT, THE ROUTINE WILL ESTABLISH:
// FUNCTION
// GRAY
// DESTINATION CORE BASE
// DESTINATION RASTER
// HEIGHT
// SOURCE RASTER
// SOURCE X
// SOURCE Y
// BBFONT
// SCAN CONVERSION SUBROUTINE
//

.GETWDSTRK: GETWDSTRK
.GETHTSTRK: GETHTSTRK
.STRIKESCAN: STRIKESCAN
.GETWDAL: GETWDAL
.GETHTAL: GETHTAL
.ALSCAN: ALSCAN

JMASK: 7417
CIS: 17
NIBMASK: 17

DOGRAY:
STA 3,SAV3,2 // BBSTABLE PASSED IN AC2
LDA 3,DESTY,2 // MOSTLY A COPY OF DAN INGALLS CODE
LDA 3,SR CY,2 // HERE BELOW
SGE 3,1
JMP NOINVERT
LDA 3,JMASK
AND 0,3 // INVERT NIBBLE ORDER
SUB 0,3 // A B C D BECOMES
SUBS 0,3
ADDS 3,0
NEG 1,1

NOINVERT:
LDA 3,HEIGHT,2
ADD 3,1
LDA 3,C3
AND 3,1 // SRCY - HEIGHT MOD 4
GRAYLOOP:
LDA 0,SAVGRAY,2
CYCLE 4
STA 0,GRAYCNT,2
AND 1,0
MOV 0,1
CYCLE 4
ADD 1,0
STA 0,GRAY1,3
INC 3,3
DSZ 3,GRAYCNT,2
JMP GRAYLOOP
JMP @SAV3,2
STA 3,1,2
MOV 0,3
SKEVEN #3,3 Szc
INC 3,3
STA 2,SAV2,3
MOV 3,2
MOV 1,3
LDA 0,FUNC,3
STA 1,GRAY,2
Sgt 0,1
JMP NOGRAY
LDA 0,GRAY,3
STA 0,GRAY,2
STA 3,SAV1,2
JSR DOGRAY
LDA 3,SAV1,2

SETSCANC:
STA 3,1,2
MOV 0,3
SKEVEN #3,3 Szc
INC 3,3
STA 2,SAV2,3
MOV 3,2
MOV 1,3
LDA 0,FUNC,3
STA 0,FUNC,2
Sgt 0,1
JMP NOGRAY
LDA 0,GRAY,3
STA 0,GRAY,2
STA 3,SAV1,2
JSR DOGRAY
LDA 3,SAV1,2

NOGRAY:
LDA 0,@DSPAD
STA 0,DBASE,2
LDA 0,@DSPWD
STA 0,DRAST,2
LDA 3,FONT,3
STA 3,BBFONT,2
MKZERO 0,0
STA 0,SRCY,2
STA 0,CHANGE,2
LDA 0,FORMAT,3
MOVZL #0,0 Snc
JMP SETAL
LDA 0,ASCENT,3
LDA 1,DESCENT,3
ADD 0,1
STA 1,HEIGHT,2
LDA 0,MIN,3
LDA 1,MAX,3
INC 1,1
INC 1,1
INC 1,1
SUB 0,1
LDA 0,C13
ADD 1,0
ADD 3,0
STA 0,BASE,2
LDA 0,SEGWIDTH,3
LDA 1,C15
ADD 1,0
MOVZR 0,0
MOVZR 0,0
MOVZR 0,0
MOVZR 0,0
STA 0,SRAST,2
LDA 0,@GETWDSTRK
STA 0,WIDTHSUBR,2
LDA 0,@GETHTSTRK
STA 0,HEIGHTSUBR,2

ADDZL 1,1
CYCLE 0
STA 0,SAVGRAY,2
LDA 1,C4
STA 1,GRAYCNT,2
MOV 2,3

://* 4
://IS HOW MUCH TO ROTATE GRAY
://SAVE IT IN BBSTABLE TEMP
://SET UP COUNTER FOR FILLING SCRATCH WORDS
://GRAY PASSED IN AC3
://GET BBSTABLE INTO AC3 SO COUNT DOWN IN
://GRAYLOOP WILL WORK

LDA 0,SAVGRAY,2
CYCLE 4
STA 0,GRAYCNT,2
ADD 1,0
MOV 0,1
CYCLE 4
ADD 1,0
STA 0,GRAY1,3
INC 3,3
DSZ 3,GRAYCNT,2
JMP GRAYLOOP
JMP @SAV3,2
STA 3,1,2
MOV 0,3
SKEVEN #3,3 Szc
INC 3,3
STA 2,SAV2,3
MOV 3,2
MOV 1,3
LDA 0,FUNC,3
STA 1,GRAY,2
Sgt 0,1
JMP NOGRAY
LDA 0,GRAY,3
STA 0,GRAY,2
STA 3,SAV1,2
JSR DOGRAY
LDA 3,SAV1,2

://INSURE ON EVEN WORD BOUNDARY
://SAVE AC2 -- MOSTLY FOR BCPL
://GET BBSTABLE IN AC2
://GET SCANTABLE IN AC3
://SET UP FUNCTION
://IF FUNCTION >7 DO GRAY SHUFFLE

://SAVE PTR TO SCANTABLE WHILE DOING GRAY FIX
://SAVE1 HAPPENS TO BE AVAILABLE

://RESOTRE SCANTABLE INTO AC3

://BEGINNING AD OF DISPLAY -- MOST LIKELY
://A GLOBAL IN SREL OR ZREL
://THAT'S THE CORE BASE FOR BB'S DESTINATION
://LIKELYWISE DISPLAY WIDTH IS RASTER FOR
://BB'S DESTINATION
://BBSTABLE NEEDS ITS OWN PTR TO THE FONT

://SOURCE Y IS ZERO

://TURN OFF CHANGE SWITCH
://HIGH ORDER BIT OF FIRST WORD OF FONT TELLS
://US IF STRIKE OR .AL FORMAT. 1=STRIKE 0=.AL

://IF STRIKE COMPUTE HEIGHT OF FONT

://SOURCE BASE ADDRESS FIRST
://WORD OF GLYPHS

://MAX+2 - MIN + TOP OF XTABLE GET TO GLYPHS

://(SEGMENT WIDTH + 15) / 16 = SOURCE RASTER

://SET UP WIDTH GETTING ROUTINE

://SET UP HEIGHT GETTING ROUTINE
SETRTN:
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, BBFONIN, 2
MKONE 0, 0
STA 0, SRAST, 2
MKZERO 0, 0
STA 0, SRCX, 2
LDA 0, @GETWDAL   ; //SET UP WIDTH GETTING ROUTINE
STA 0, WIDTHSUBR, 2
LDA 0, @GETHTAL   ; //SET UP HEIGHT GETTING ROUTINE
STA 0, HEIGHTSUBR, 2
LDA 0, @ALSCAN
JMP SETRTN       ; //PASS BACK PTR TO SCAN CONVERTING SUBR IN
                   ; //BBSTABLE

PUTCHARSC:
STA 3, 1, 2
MOV 3, 0
LDA 0, CHANGE, 3
SGZ 0, 0
JMP NOCHANGE
LDA 0, 1, 2
STA 0, SAV3, 3    ; //SETUP WIDTH GETTING ROUTINE
                   ; //IN BBSTABLE AND CALL
MOV 3, 0
MOV 3, 0
JMP @I, 2

NOCHANGE:
STA 0, 1, 2
LDA 0, CHANCE, 3
SGZ 0, 0
JMP NOCHANGE
LDA 0, 1, 2
STA 0, SAV3, 3    ; //IF CHANGE SAVE RETURN PTR FOR PUTCHAR
                   ; //IN BBSTABLE AND CALL
MOV 3, 0
JMP @I, 2

JSR SETSCAN       ; //RESTORE RETURN PTR --
                   ; //BOMBED BY SETSCAN

JSR @SMF          ; //SMF (SUBSTRACTOR-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 NOSKIP 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
                   ; // FOR MEASURING-----
                   ; //AC0 = CHAR CODE
                   ; //AC1 = BYTE PTR
JMP RETURN
LDA 3,DISPGLBS
NIL 0.0
STA 0,NSPC,3
LDA 1,SAV1,2 ;//AND ACI GETS BYTE PTR + 1 -- WHEN SUBSTR
INC 1,1
LDA 2,SAV2,2
JMP @I,2
RETURN:
STRA 3,SAV3,2
;IBBSTABLE IN AC2
LDA 3,BBFONT,2 ;//CHECK FOR 'LEGAL' ASCII
STA 0,CHAR,2
STA 1,SAV1,2 ;IDITTO BUT REQUIRED IN SMALLTALK
SLE 0.1
INC 1,0
STA 0,CHAR,2
;MAX + 1 IS 'ILLEGAL' CHARACTER
LDA 1,MIN,3
SLE 0.1
JSR EXCEPTC
LDA 1,MAX,3
SLE 0.1
INC 1,0
STA 0,CHAR,2
;MAX + 1 IS 'ILLEGAL' CHARACTER
LDA 1,MIN,3
SUB 1,0
;SUB MIN FROM CHAR_CODE -- SO INDEXING
LDA 3,MIN,3
ADD 1,3
STA 1,MIN,3
ADD 0.3
STA 1,DESTX,2
LDA 3,BBFONT,2
SET UP SOURCE X
LDA 1,CHARPTRS,3
STA 1,CHAR,2
SET UP WIDTH
LDA 0,CHARPTRS+1,3
SUB 1,0
STA 0,WIDTH,2
SNZ 0.0
JSR EXCEPTC
JSR CLIPC
JMP NOSCAN
MOV 2,0
;BBSTABLE SENT IN ACO FOR NOW
ZER 1,1
BITBLT
MOV 0,2
NOSCAN:
JMP @SAV3,2
CLIPC:
MOV 3,1
;SUBR FOR WINDOW CLIPPING
STA 1,ITEMP1,2
LDA 0,CHAR,2
LDA 1,SPACE
SNE 0.1
JMP SPCIT
ISZ TRLCHR,2
NOC:
NOP
SPCIT:
LDA 1,WIDTH,2
LDA 0,DESTX,2
LDA 3,CROSSLEFT,2
SLT 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
LDA 3,WIDTH,2
STA 1,WIDTH,2
SUB 1,3
;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

CHKRT2:
LDA 3,CROSSRIGHT,2
SLE 0,3
JMP RTN
ADD 1,0

LDA 3,RIGHTMARGIN,2
LDA 1,CROSSRIGHT,2
SNNIL 1,1
JMP CHKEDGE
///NOW CHECK FOR SPANNING CROSSRIGHT
///IF CROSSRIGHT NIL-- WE'VE ALREADY CROSSED IT
///SO CHECK RIGHTMARGIN -- SO MEASURING WORKS
///IF CROSSRIGHT LESS THAN RIGHTMARGIN
JMP CHKEDGE
///IF EQUAL CHECK SPANNING OF 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

LDA 3,SAV1,2
///PRUNED CHARACTER WILL BE LAST VISIBLE
STA 3,LASTVISIBLE,2
///MAINLY FOR SMALLTALK PURPOSES
SUB 1,0
///IF SPANNING CROSSRIGHT OR RIGHT MARGIN
LDA 1,WIDTH,2
///PRUNING WIDTH WILL GET PARTIAL CHAR
SUB 0,1
///IF SPANNING CROSSRIGHT OR RIGHT MARGIN
STA 1,WIDTH,2

LDA 0,CHAR,2
LDA 1,SPACE
LDA 3,TAB
SEQ 0,1
SNE 0,3
JMP RTN
LDA 3,CROSSRIGHT,2
SNNIL 3,3
JMP RTN
LDA 3,CROSSR,2
SNNIL 3,3
JMP RTN
LDA 3,TAB
SNE 0,1
JMP DOTAB
LDA 3,CR
SNE 0,1
JMP DOCR
ZER 0,0

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

LDA 1,MEASURE,2
SZE 1,1
JMP MRTN
LDA 3,TEMP1,2
JMP @TEMP1,2

CHECKRT:
LDA 3,CROSSRIGHT,2
SLE 0,3
JMP RTN
///AND CHECK RIGHT BOUNDARIES
///AND ADD IN WIDTH FOR NEXT CHECK

PRUNE:
LDA 3,SAV1,2
///PRUNED CHARACTER WILL BE LAST VISIBLE
STA 3,LASTVISIBLE,2
///MAINLY FOR SMALLTALK PURPOSES
SUB 1,0
///IF SPANNING CROSSRIGHT OR RIGHT MARGIN
LDA 1,WIDTH,2
///PRUNING WIDTH WILL GET PARTIAL CHAR
SUB 0,1
///IF SPANNING CROSSRIGHT OR RIGHT MARGIN
STA 1,WIDTH,2

LDA 0,CHAR,2
LDA 1,SPACE
LDA 3,TAB
SEQ 0,1
SNE 0,3
JMP RTN
LDA 3,CROSSRIGHT,2
SNNIL 3,3
JMP RTN
LDA 3,CROSSR,2
SNNIL 3,3
JMP RTN
LDA 3,TAB
SNE 0,1
JMP DOTAB
LDA 3,CR
SNE 0,1
JMP DOCR
ZER 0,0

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

LDA 1,MEASURE,2
SZE 1,1
JMP MRTN
LDA 3,TEMP1,2
JMP @TEMP1,2

RTN:
LDA 1,MEASURE,2
SZE 1,1
JMP MRTN
LDA 3,TEMP1,2
JMP @TEMP1,2

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

EXCEPTC:
MOV 3,1
STA 1,TEMP1,2
LDA 3,DSPGLBS
LDA 0,CHAR,2
LDA 1,SPACE
SNE 0,1
JMP DSPACE
LDA 1,TAB
SNE 0,1
JMP DOTAB
LDA 1,CR
SNE 0,1
JMP DOCR
ZER 0,0

MOV 3,1
///BBSTABLE Comes in AC2
STA 1,TEMP1,2
LDA 3,DSPGLBS
LDA 0,CHAR,2
LDA 1,SPACE
SNE 0,1
JMP DSPACE
LDA 1,TAB
SNE 0,1
JMP DOTAB
LDA 1,CR
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

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

DOJSTC:
DSZ CNT1,3
JMP JUSTIT
LDA I,CNT2,3
JMP LASTJST

LASTJST:
STA 1,THISLINE,3

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

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

STA 0,JSTCR,3
LDA 1,SAV1,2
LDA 3,SAV3,2
JMP 1,3

DOCR:
LDA 0,TRLCHR,2
SZE 0,0
JMP MULSPC
ZER 0,0
STA 0,TRLCHR,2
STA 0,NSPCC,3

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

MRTN:
LDA 0,TRLCHR,2
SNNIL 1,1
JMP NOSPCC

CR: 15
TAB: 11
SPACE: 40
.TABWD: TABWD

DOTAB: JSR II .TABWD

DOCR: STA 0,JSTCR,3
LDA 1,SAV1,2
LDA 3,SAV3,2
JMP 1,3

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

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

MRTN:
LDA 0,TRLCHR,2
SNNIL 1,1
JMP NOSPCC

//RECALL SCAN SUBR
//WITH ILLEGAL CHAR
//+1 TO KEEP SCANSUBR'S RETURN CORRECT
//SEE IF IN MEASURE MODE
//IF ACTUALLY SCAN CONVERTING THEN
//SEE IF JUSTIFYING
//ENTRY USED BY SMALLTALK FNDMS ROUTINE
//IF SO, SEE IF WE'VE
//COUNTED DOWN COUNTERS
//CNT2 ALREADY ZERO OR COUNTED DOWN?
//IF SO ONE MORE TIME AND STOP
//UPDATE LEAD SO TABS WORK RIGHT
//NOW+1 NEW JUST SPACE
//ZERO CNT2
//SHUT OFF JUSTIFICATION FOR NEXT TIME
//AND UPDATE EX
//BUMP TAB DELTA -- LEAD SET IN LNOT,SR
//SAVE PROPER RIGHTX
//FOR JUSTIFICATION
//FOR CHECKING IN JUSTIFICATION
//GET CURRENT BYTE PTR INTO ACI
//POP OUT OF SMF IN PUTCHARS
//SPACE EXCEPTION CODE WHEN MEASURING
//FIX MULTIPLE SPACES IF NECESSARY
//SAVE PTR TO THIS SPACE
//BUMP SPACE COUNTERS
//IF NOT GO GET MORE CHARS
//NIL TRLCHR TELLS US PAST RIGHTMARGIN
//IF NOT GET MORE CHARS
//GET LAST CHAR INTO AC0 FOR PSTRG
//NIL LSTSP MEANS NO SPACES IN LINE
LDA 1,SPCX,3       //MAKE LAST DEXTX
STA 1,DESTX,2
LDA 1,LSTSP,3
INC 1,1
MOUT:
LDA 3,SAV3,2       //SEND BYTE PTR BACK IN AC1 AND CHAR CODE IN
JMP 1,3
NOSPC:
LDA 1,SAV1,2       //NO SPACES IN LINE MEANS
JMP MOUT
LBYTEMSK: 177400
RBYTEMSK: 377
C20: 20
C257: 401
ALSCAN:
STA 3,SAV3,2       //BBSTABLE COMES IN AC2
STA 1,SAV1,2       //SAVE SMFS BYTE PTR
EXTENTION:
STA 0,CHAR,2       //SAVE CHAR CODE FOR EXCEPTION CHECKING
LDA 1,WIDTH,2
LDA 3,DESTX,2
ADD 1,3
STA 3,DESTX,2
LDA 3,BFONT,2
ADD 0,3
LDA 0,0,3
ADD 0,3
LDA 0,0,3
STA 0,TEMP2,2
STA 0,TEMP3,2
LDA 1,C20
MOVZR 0,0 SNC
MOV 1,0
STA 0,WIDTH,2
LDA 1,1,LBYTEMSK
ANDS 0,1
LDA 0,DESTY,2
STA 0,TEMP3,2
ADD 1,0
STA 0,DESTY,2
LDA 0,1,3
LDA 1,1,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:
JSRH .EXCEPT
JSRH CLIP
JMP NOSCANAL
MOV 2,0
ZER 1,1
BITBLT
MOV 0,2
NOSCANAL:
LDA 0,TEMP3,2
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 EXTENTION
JMP @SAV3,2      //AND GO FOR ANOTHER CHARACTER
.EXCEPT: EXCEPT
.END