A2-3D2 Animation Package

subLOGIC

#010346

SUBLOGIC A2-3D2 EXTENDED GRAPHICS ANIMATION PACAKGE TECHNICAL MANUAL VERSION 1.1 - JUNE 22, 1981

by
Bruce Artwick

Sublogic Company
201 W. Springfield Avenue
Champaign, Illinois 61820 U.S.A.

(217) 359-8482

First Edition
First Printing

Printed in USA

TABLE OF CONTENTS

1 INTRODUCTION

1.1 General Program Information

The A2-3D2 Extended Graphics Animation Package is a high-performance upgrade to the A2-3D1 Animation Package. The A2-3D2 program supports all of the functions of the A2-3D1 package and thus maintains database compatibility with previously developed databases.

Four factors prompted the development of this package:

1. User feedback concerning desired features.
2. Competitive packages that offered some of these features.
3. Sublogic's internal need for high-performance drivers.
4. The low cost of 48K of memory.

The A2-3D2 package has more features, runs faster, and takes up more memory than A2-3D1.

New features include color lines (140 x 192 resolution), high resolution lines (280 x 192 resolution), independent object manipulation of as many objects as you like (as opposed to only 16 in competitive packages), range improvements (greater-than-32767-length lines are now acceptable), reference frame nesting, and the ability to call many functions with a simple subroutine call instead of in an interpretive mode. A new 3D-to-3D conversion feature allows you to generate an array of transformed 3D points in their transformed 3D reference frame, and 'Skip' and 'Pause' commands improve display file debugging and control.

1.2 A2-3D2's Relation to A2-3D1

You must own and be familiar with the A2-3D1 package to use the A2-3D2 upgrade. The concepts of 3-D databases, viewer location and rotation, and display file creation and interpretation are all described in the A2-3D1 technical manual.

This manual describes the extensions to the A2-3D1 package and explains new concepts needed to use them.

All old A2-3D1 commands remain the same, so old databases may be used with A2-3D2*. The call addresses remain the same.

Only the trig function variable address and multiply and divide calling addresses have changed. If you rely on called trig functions or multiply/divide patch-points, you must update the call addresses in your programs.

A2-3D2 is too large to fit in low memory (800 to 1fff hex). Only the high memory version (starting at 6000 hex) is available.

* NOTE: As long as the end-of-file mark is not a new command code. See the EOF command sheet (79 hex, 121 decimal) for details.

1.3 Program Features

The following chart presents the features of the A2-3D2 package and the corresponding A2-3D1 features where applicable.

FUNCTION	A2-3D1	A2-3D2
INTERPRETIVE FUNCTIONS:		
Pure Point (140 x 192)	0,x,y,z	0,x,y,z
Start Point (140 x 192)	1,x,y,z	1,x,y,z
Continue Point (140 x 192)	2,x,y,z	2,x,y,z
Ray Point (140 x 192)	3,x,y,z	3,x,y,z
Clipper Control	4,on/off	4,on/off
Viewer Position D.P.,Pseudodegrees	5,x,y,z,p,b,h	5,x,y,z,p,b,h
Draw 2D Line on Screen	6,x,y,x',y'	6,x,y,x',y'
Display Screen Select	7,code	7,code
Erase Screen/ Fill Screen	8,code	8,code
Write Screen Select	9,code	9,code
Plot a 2D White Point	10,x,y	10,x,y
Interpretive Jump	11,adrlsb,adrmsb	11,adrlsb,adrmsb
Set Line Drawing Mode	12,mode	12,mode
Turn on Output Array	13,adrlsb,adrmsb	13,adrlsb,adrmsb
Screen Size Select	14,w,h,cx,cy	14,w,h,cx,cy
Field of View Select	15,axr,ayr,azr	15,axr,ayr,azr
Easy Initialize	16	16
No Operation	17	17
Set Color Mode	-	18,col
Independent Object Call	-	19,stat,loc,addr
Set Resolution	-	20,res
Hi-Res (280x192) Line 2D	-	21,x1,xh,y,x1,xh,y
Set Hi-Res Bias	-	22,x1,xh,y
Hi-Res (x=256 Limited) Line 2D	-	23,x,y,x',y'
Hi-Res (280x192) Point Plot 2D	-	24,x1,xh,y
Hi-Res (x=256 Limited) Pnt. Plot 2D	-	25,x,y
Skip segment	-	26,size,status
Pause for n/5ths of a Second	-	27,time
Set 3D to 3D Array Gen. Address	-	28,adrlsb,adrmsb
Set 3D to 3D Array Gen. Status	-	29,status
End of File	undefined	121 (79 hex)
CALLABLE FUNCTIONS		
Sin/Cosine calls	yes	yes
Multiply (SP and DP)	-	yes
Divide (DP)	-	yes
Erase (hi/low page)	-	yes
Hi-Res Point Plot	-	yes
Color Point Plot	-	yes
Hi-Res Line Draw	-	yes
Color Line Draw	-	yes
Set Display Resolution	-	yes
GENERAL FEATURES		
Initialize Input Buffer Ptr.	-	yes
Line Length Limit	32767 max.	113508 max.*
World Movement	limited by overflow	unlimited **

* Distance from -32767,-32767,-32767 to 32767,32767,32767.
** As long as the value -32768 is avoided in databases and eye position.

1.4 GETTING A2-3D2 RUNNING

A2-3D2's program size and variables' locations are shown in the memory map that accompanies the package. A good starting point in getting familiar with the package is the built-in test program. See section 6.4 for details. The test program will assure you that the interpreter is properly installed in memory and performing its functions.

Instructions for installing A2-3D2 are supplied on the disk or with the cassette. You will need your old A2-3D1 program to generate the A2-3D2 enhancement.

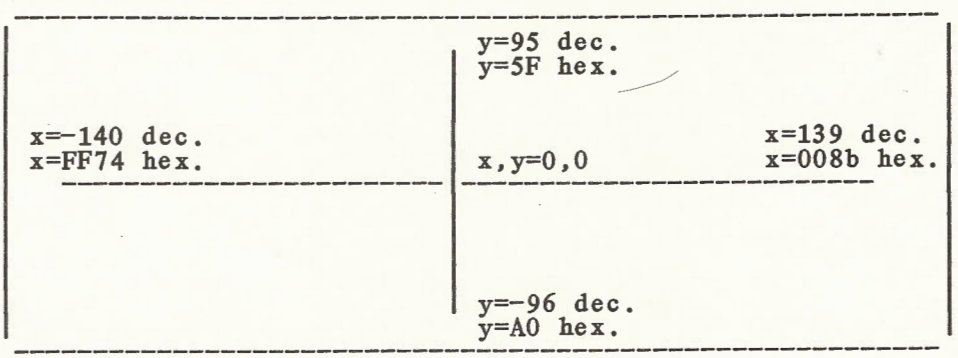
2 HI-RES GRAPHICS CONCEPTS

The hi-res screen's 280 x 192 non-color mode becomes useful in applications where fine detail is important. This can cause smeared, random colors if displayed on a color monitor. Good display results can be obtained by using a black and white monitor or color monitor in black and white mode.

This section describes the 280 x 192 line drawing feature.

2.1 Hi-Res Screen Mapping

The 280 x 192 mapping follows the Sublogic convention of putting x,y coordinate 0,0 at the screen's center:



Note that 2-bytes are required to represent the X range because the -140 to 139 range exceeds 1-byte's -128 to 127 range by 24 units.

2.2 Hi-Res Coordinate Specification Methods

2.2.1 3D Effects

When working in 3D coordinates, you are not concerned with the 2D screen because the 3D to 2D converter performs the mapping. The 3D coordinate representations thus are unaffected.

Line drawing mode can be switched from 140 x 192 color to 280 x 192 hi-res using the 14 (hex) 'Set Resolution' command (see command sheets). When hi-res is selected before 3D lines in the display file, the 3D lines are projected and drawn in the same way they would have been in color mode but with 280 x 192 resolution. In other words, the screen image looks the same (same size and shape) but now consists of very fine, accurate, non-colored lines.

2.2.2 2D Effects

The old 'Draw a 2D Line on Screen' and 'Plot a 2D Point'commands, when preceded by a 'Set Resolution' (to high resolution) command, draw lines and plot points in the same way they would have in color mode but with 280 x 192 resolution. Again, the screen image looks the same (same size and shape) but now consists of very fine, accurate, non-colored lines. Note that the conversion to high-res is performed by the A2-3D2 driver by simply multiplying the user-specified coordinates by two. This works well, but all lines start and end on even X boundaries thus not fully utilizing the high resolution of the

screen.

Two new line specification methods (and corresponding commands for points) are available to make full use of the 280 X-resolution:

METHOD 1. Lines and points can be specified by submitting an X with -140 to +139 decimal range (this takes two bytes) and a Y with -96 to +95 range. The 'High Res (280 x 192) Point 2D' command is an example of this method. The format is: opcode, X lsb, X msb, Y. The X is submitted LSB-first in typical 6502 form. Although this method fully specifies lines and points, it is memory and computationally wasteful because double precision X storage and calculations are used. For more efficiency, method 2 may be used.

METHOD 2. X is treated as a -128 to +127 range field resulting in an X-resolution of 256 units. Twenty-four of the 280 X units are not used. This method allows points and lines to be submitted in single precision in the general form: opcode, X, Y. The position of the 256 possible X locations on the 280-wide screen is user selectable. The Set Hi-Res Bias command allows you to match the center of the 256 x 192 plotting window to any location on the 280 x 192 screen. You may choose to put the unused 24 pixels to the right or left side of the screen, or to split them in some way between right and left. By default, the 24 unused pixels are split as 12 left and 12 right.

Method 1 and 2 are independent of the selected screen resolution (280 x 192 or 140 x 192 color) and always draw hi-res lines. This is in contrast to the old 'Draw a 2D Line on Screen' command that draws color or high-res lines depending on selected screen resolution.

3 COLOR

When the original A2-3D1 package was written (in early 1979), the APPLE II had only 3 hi-res colors: green, purple, and white. With the introducion of the APPLE II Plus, 5-colors became available and more and more people began insisting on 140 x 192 color for their applications. The line drawers in the A2-3D2 driver thus were modified to generate color.

3.1 Color Modes

All 5-colors now can be drawn by A2-3D2 in 140 x 192 color mode. The choice of colors are: blue, purple, green, orange, and white. Keep in mind, however, that the APPLE II limits color mixing. There are actually two color sets or 'palettes' as they are sometimes called. You may chose between the green-purple-white palette, and the orange-blue-white palette. The palettes are selectable on a screen-byte basis. Therefore, every 4 (approximately) color pixels that are next to each other on a single line must all be of the same palette.

This can cause problems (if nice-looking colors are what you are after). For example, an orange line can be drawn horizontally across the screen. A diagonal purple line can then cross it. Because the palette at the intersection can not match orange and purple simultaneously (they are on different palettes), you end up with either a blue dot at the intersection (if you use the orange-blue-white palette), or about 4-horizontal pixels of green (if the green-purple-white palette is chosen).

As more lines of different palettes intersect, things get worse and worse. With the orange and purple line intersection above it is easy to see that the blue dot at the intersection is preferable to the big green 4-pixel blotch, but when 4-lines go through a single palette byte, what looks best is more difficult to determine (and can indeed be a matter of viewer preference).

The A2-3D2 driver makes no attempt to determine what looks best. Instead, it sets the palette to that of the latest color line that goes through the byte. Keep this in mind if you are using intersecting color lines extensively. The order in which you submit the lines will make a palette selection difference at intersections so you may want to switch the line submittal order to get a desired effect.

3.2 Color Specification Methods

The 'Set Color Mode' command (see command sheets) sets the driver to generate the desired color. All 3D lines and all 2D non-high-res lines that follow the 'Set Color Mode' command will be drawn in the desired color.

The 'Set Color Mode' command has no meaning when high-res (280 x 192) is selected by the 'Set Resolution' command. A 'Set Color Mode' command submitted while in hi-res mode will set the color line drawers to generate the specified color, but the hi-res line drawer is used to generate the hi-res line. Note that when you switch back to color mode, the color line drawer is ready to generate the color you last specified (even if it was specified within the hi-res drawing part of a display file).

4 INDEPENDENT OBJECT MANIPULATION

4.1 Manipulation Concepts

In some applications, individually movable and rotatable objects are desirable. The A2-3D2 package is able to manipulate as many such objects as desired.

It is first necessary to define an object. Simply build-up the object in its own coordinate system with 0,0,0 assumed to be the center of rotation. Once complete, put an EOF (end of file) at the end of it. This is now the object definition.

The object may now be projected by putting an 'Independent Object Call' (see commmand sheets) in your normal display file. This call specifies the address of the independent object definition and the current position and rotation of the object in space. The current position and rotation allows you to move an object around in space and to rotate it around its own axes. In formal graphics terms, 3D instancing is performed.

The independent object call is a very powerful feature because you can use a common independent object definition to represent many objects in space. An independent definition of a space ship, for example, may be called several times using multiple independent object calls - once for each ship in a whole fleet of space ships. Each space ship can navigate its own way around by simply modifying the positional and rotational information in the independent object calls.

See the command sheets for the interpretive 'Independent Object Call'.

4.2 Reference Frame Nesting

Independently manipulatable objects open a new world of dynamic object movement in space. With A2-3D1 you had to be content with viewing 'what was out there'. Users informed us that the competition's package allowed you to 'move things around'. A2-3D2 now allows you to do this also, but we have taken manipulation even one step further using the concept of nested reference frames. Nested reference frames can best be introduced with an example:

Assume that for an engineering demonstration you want to show the effects of road vibration loosening the valve cover on an automobile's tire (yes, it does sound absurd). You need to show the car moving around. You need to show the spinning tires on the car. Finally, you need to show the valve cover spinning loose. Doing this with A2-3D1 or any of the competitions' packages is horrendously difficult if not impossible. With A2-3D2 it is easy.

Simply define the car body as an independent object. Within the car body definition, insert an independent object call to the tire (another independent object). Because this independent object call is in the car body definition, it is in the car body's reference frame and moves as the car does.

Similarly, put an 'Independent Object Call' to the valve cover within the tire's definition. Now the valve is in the tire's reference frame, rotates with the tire, and moves along as the car moves. You may increase the heading angle of the valve cover to make it spin and increase its Y position to make it pop off (if you want to show that the cover does indeed vibrate loose).

This is a very powerful feature. The number of reference frames 'deep' you may nest is limited by stack size. Each level requires about 30 (decimal)-bytes of stack space. Assuming that the regular 3D portion of the driver requires about 40 (decimal) bytes, this allows for about 7-levels.

4.3 Manipulation Countdown and Recursion

Notice that there is a status flag in the 'Independent Object Call'. This can be used in two ways. When set to 00, the independent object will not be processed. Any nested objects within the 'turned off' independent object will, of course, also be turned off. You can therefore use the status byte as a convenient switch to turn independent objects on and off.

The status value ff (hex) turns an object on and leaves it on. A value less than ff is called a countdown value. Every time the object is called it is decremented by one until it reaches zero, at which time the object is turned off.

This feature can be used to simplify action and game-type programming. A 'photon-torpedo' for example can have its status byte set to the value 14. After 14 frames, the photon torpedo will turn itself off.

RECURSION - Independently manipulated objects can be nested. They can call other independent objects and put them in their own reference frames. Using the status flag properly, independently manipulated objects can call themselves recursively.

If an object is simply turned on all the time (status = ff), recursion is impossible because the object keeps nesting into itself indefinitely (actually until the stack blows-up). But by setting the status flag to the desired number of levels of recursion, the element can turn itself off after a given number of levels, thereby bypassing itself and running into its EOF thereby causing un-nesting.

Recursion may be useful when you need to put one object in an identical object's reference frame but want to avoid redundant definitions. A planetary gear system is a good example. Two identical gears may be required, but the planet gear must be put into the sun gear's reference frame. Two identical definitions could be setup (one for the sun, and one for the planet gear) and the sun gear definition could call the planet gear definition. To avoid the wastefulness of a redundant and probably complex gear definition, recursion could be used.

There are even more recursive tricks you can play with the countdown value. You can call many independent objects recursively within the definition, and each one can have a different countdown value. When one recursive nesting is done, it will internally start nesting the next. Thinking on this level gets quite confusing and if you come up with a good method of visualizing and notating this, please let us know.

4.4 Sense of Rotation and Translation

Independent objects' rotations and translations correspond directly with the viewer's sense of rotation and translation. Figure 14 in the A2-3D1 technical manual shows the change in view while looking out into space. Independent objects are translated and rotated as if you were inside their reference frame looking out into space.

If you are looking straight ahead with no pitch, bank or heading and an object (an aircraft for example) is in front of you and flying away from you, the following will happen as you manipulate the airplane's independent object call parameters.

+X MOVEMENT - Airplane moves to right on your screen.

+Y MOVEMENT - Airplane moves up on your screen.

+Z MOVEMENT - Airplane moves away on your screen.

+ HEADING - Airplane turns more to the right.

+ PITCH - Airplane drops its nose.

+ BANK - Airplane banks to left (as it moves away from you)

Note that the order of translation is:

1. X,Y,Z movement (object is first placed to its location in space.)

2. Heading. The object spins around parallel to the x-z plane.

3. Pitch. The object then pitches up or down, perpendicular to the direction (heading) it is pointed at.

4. Bank. The object rolls about the axis that has just been 'headed' and 'pitched'.

Remember that the rotations and translations correspond to normal 'EYE' rotations (Figure 14. in tech. manual). It is also important to note that this correspondence applies to a viewer looking out of the independent object's reference frame, not into it. The sense of movement and rotation will seem reversed if you think of the rotations and translations from the outside-in.

Movement in the X direction is a good example of reverse sense of translation. If you look at the airplane in front of you and then you move in the +X direction, the airplane will move to the left on the screen. If the airplane moves in the +X direction it will move to the right on the screen. In both cases X was increased, but the object moved to the left in one case and right in the other. But if you consider what would be seen out the window of the other airplane when he moves in a +X direction, objects in front of him would tend to move to the left of his viewing window.

5 OTHER IMPROVEMENTS OVER A2-3D1

Many small improvements that don't affect database structure or programming are incorporated in A2-3D2. They are outlined below:

5.1 Database Range

A2-3D1 had a number of database range limitations. Lines were limited to lengths less than 32767, and you had to make your database small enough so that you didn't 'fall off the edge of the world' (see Setting Up 3D Scenes in the A2-3D1 technical manual). These limitations now are removed. A 'spherical compression' method (first used on Sublogic's Z80 3D package) is incorporated to eliminate overflow and project close-in lines more accurately.

The only overflow that can still cause problems is that caused by database elements or viewer positions containing the value -32768. There is no positive equivalent to this value in 16-bit integer arithmetic so a sign error results in computations using it. Checking for this value and taking appropriate program action was considered, but we determined it was not worth slowing the display for. Simply avoiding this value in databases and viewer location is the preferred option.

Overflow also is a problem if offsets within nested reference frames accumulate to a value of greater than 65535 in any direction.

5.2 Speed Improvements

A2-3D2 has faster line generators than A2-3D1. A number of subroutines were replaced by inline code segments, and line overhead was reduced by converting generation tables to screen center coordinates (thus avoiding conversion of in-coming variables to corner-origin coordinates). A bit of speed was lost with the introduction of the spherical compressor, but 3D-line generation is still faster than A2-3D1, and 2-D line generation is faster than ever.

The erase command is a bit quicker than before. Separate erase routines for both screen pages make program self-modification (which used to take place prior to the erase) unnecesary.

5.3 Callable Graphic Functions

A2-3D1 had only one callable function (the TRIG) function. An appendix section later was added concerning multiplier and divider patch points. These directly callable routines proved to be quite popular, so A2-3D2 contains more of these functions. Direct-call functions now include graphic primitives such as erase, point plot, and line draw. See the 'Direct Call Function Sheets' for details.

NOTE: Before using direct call functions, read the caution note at the beginning of section 8.

5.4 Program Size and Eliminatable Code

A2-3D1 was about 4600 bytes long. Small program size was a prime concern because most users had only 16K of memory. A2-3D2 was not

optimized for size because memory costs have dropped dramatically (a user can now upgrade from 16K to 48K for about forty dollars). A2-3D2 is about 7900 bytes long. Memory space was traded-off for improved performance and features.

Although 7900 bytes doesn't constitute a huge program, it does have some consequences. The low memory version is no longer available (only 6144 bytes are available between 800h and 1FFFh).

Certain code segments may be eliminated from the high and low ends of the program if certain features are not used. Other segments that fall within the program may also be eliminated if not used, thereby leaving a usable memory 'hole' in the middle of the program. The segments appear below in memory address order. Note that if you want to free all the memory up to the routine you want, you must not use any of the functions whose code you eliminated.

Segment Name	Function	Condition to Eliminate
XENT1:	Entry vector 1	Use ENTRYS entry point instead.
XENT2:	Entry vector 2	Use ENTRYN entry point instead.
XSIN:	Sine computation	Use SINEX entry point instead.
XCOS:	Cosine computation	Use COSEX entry point instead.
ZPSAV:	Zero page save memory area	Don't use ENTRYS entry point
ENTRYS:	Page restoring program entry	Don't use ENTRYS entry point
ENTRYN:	Normal semi-restoring entry	Don't use ENTRYN entry point
COL2DT:	Color set code to array xfer	Don't use command 12 hex or 14 hex when in array generation mode.
SINEX:	Callable sine/cos processing	Don't use external sine/cos calls
ROWTBH:	APPLE screen drawing	Use output array only and no APPLE draw, erase, or any other screen functions.

NOTE: See memory map for addresses.

5.5 Skip Command

Turning elements off and eliminating them from the database used to be a matter of filling-in an area with NOPs or using a position-dependent interpretive jump. The Skip command now allows you to jump over sections of display files.

The skip command has both length and status arguments. The length tells how many bytes to skip-over (in addition to the skip command itself), and the status byte tells whether the skip should be performed.

By putting a skip command before an element (or a whole secion of elements), the element can be turned on or off by setting the status byte to ff (hex) to turn it on (no skip), or 00 to turn it off (skip). The sense (00 or ff hex) of the status byte may seem backwards. See the 'Skip' command sheet for details.

5.6 Pause Command

In many situations it is desirable to pause in the middle of a display file. A series of display frames with pauses between them, for example, makes a good A2-3D2 function test program. This is in fact how we test the A2-3D2 package.

The pause command is timed to allow you to pause from 0 to 255 fifths of a second.

5.7 3D to 3D Conversion

Many users need to know the position of a point in space after the 3D translation and rotation. Nested reference frames make this feature even more imperative. It is now possible to generate an output array of the points and lines after 3D translation and rotation (prior to clipping and projection).

The SET323 and GN323 commands allow you to set an array address and start generating an array of transformed 3D points. The 3D to 2D conversion and final projection can be suppressed (a GN323 command option) if no screen projection is desired.

Note that this array consists of points that are rotated into the eye's coordinate system and not points in absolute 3D space. All points, including those coming out of nested reference frames are in the eye's reference frame when dumped into this array. They can be used in relation to each other for functions such as 'hit detection' in games, but don't confuse these coordinates with absolute coordinates when performing such functions. If you want to compare a tank's projectile position with a target's position, you must rotate the target into the eye's coordinate system. Alternatively, you can generate the array with x,y,z, pitch, bank and heading all equal to zero (in this case, the eye and absolute reference frames are the same).

3D to 3D ARRAY FORMAT -

The transformed 3D points are put into the array pointed-at by ARR323+1 and ARR323+2 (lsb,msb) in the following sequence.

word	data	meaning
0	x lsb	transformed value of 1st point encountered
1	x msb	
2	y lsb	
3	y msb	
4	z lsb	
5	z msb	
6	x lsb	transformed value of 2nd point encountered
7	x msb	
.		
.		
.		

There are no opcodes to indicate where the points came from (start, continue, ray, or pure point). The user must manually keep track of what these points mean.

The end of the generated array may be found by observing the value of ARR323+1 and ARR323+2 (the lsb and msb of the array pointer). This pointer always points to the array append point where the next point is to be added.

CAUTION: It is important to note the limitations of this feature before using it. The A2-3D2 package has internal scaling routines that allow wide database and movement ranges. Eye location added to object location often exceeds the 65535 unit range of 16-bit integer arithmetic. When any sort of overflow occurs, the A2-3D2 routines perform nonstandard matrix rotation operations that bypass the normal 3D-to-3D conversion process. The desired 3D to 3D array can not be computed under these conditions.

Putting the program into 3D-to-3D array generation mode turns off the overflow bypass system to maintain correct results in the array. Note that you are restricted to world overflow limits as in A2-3D1 when you are generating the array.

This limitation only applies when 3D-to-3D array generation is turned on. You can still have free movement through a large database (a battlefield for example) with the array turned off while computing a relatively close-in object's position (a tank's projectile perhaps).

6 CONFIGURATION AND INITIALIZATION

6.1 Program Calling

There are many ways to call A2-3D2. It may be called three ways in its interpretive mode, and individual functions may be called by themselves. In all cases, certain parameters must be initialized (manually or automatically).

6.1.1. Interpretive Calls

There are three interpretive entry points. They provide a trade off of speed and ease of use. All three are easy to use, but you must go through more manual initialization and keep track of variables outside the A2-3D2 package if the fast entry points are used. The following heirarchy chart illustrates the three points:

Use a JSR to access these entry points.

XENT1: or ENTRYS:
1. saves %psw,%a,%x,%y
2. saves memory 60h to C2h
3. calls ENTRYN

    XENT2: or ENTRYN:
    1. clears decimal mode
    2. clears variables CLIPON to IBP
    3. initializes IBP
    4. initializes rotation matrix to 'head-on view'
    5. initializes screen bias and current screen select
    6. falls to NXTPT

        NXTPT:
        1. interprets instructions until end of file
        2. returns (to ENTRYS call or user call)

4. restores memory 60h to C2h
5. restores %y,%x,%a,%psw
6. returns in its identical calling state

NOTE: See memory map for addreses of entry points

Time may be saved by calling ENTRYN or NXTPT, but the proper initializations performed by the higher level functions must be done first. You may not even need to perform certain functions (XENT1 steps 2 and 4 for example) if you manually keep track of old variables and how they are modified by the program.

6.1.2. XENT1 and ENTRYS calling

XENT1 and ENTRYS are the easiest to use calls. They perform all necessary initialization and save zero page memory. The XENT1 calling point is located in the jump vector section of the program and simply jumps to ENTRYS. Time and memory may be saved by using ENTRYS instead.

If you use this entry point, things will be simple, but the program will run slowly, especially if a small database is used (as the initialization is in the program's overhead).

6.1.3. XENT2 and ENTRYN calling

This entry point initializes the variables to the states necessary to interpret the display file in accordance with the A2-3D1 definition. For example, unless otherwise stated with the EYE command, the viewing

angle is straight foreward. The ENTRYN routine initializes the matrix to straight foreward on each entry.

If you don't want automatic zero page variable saving and register saving, it is wise to use this entry point as it speeds things up. Note that variables 60-C2 may be destroyed by the program. There is no need to manually initialize any variables - just realize that these locations get destroyed if this entry point is used.

XENT2 is simply a jump from the jump vector section to the ENTRYN entry point. You can save time and memory by using the ENTRYN entry point instead.

6.1.4. NXTPT calling

This is the lowest level on which the interpreter may be called. It jumps right into the interpreter loop without initializing any variables. It is the fastest entry point.

You must manually initialize these variables or make sure that the variables are still good from the last pass through the program. The following must be initialized:

CLIPON:	0=clipper on not 0=clipper off
GENAR:	0=draw screen not 0=make array
M1: to M9:	valid xform matrix, or a call to matrix generator (manually or using EYE) before any projections are performed.
XV:	viewer's x location (or call EYE)
YV:	viewer's y location
ZV:	viewer's z location
ADDRS:	this byte must be 00
SCBIAS:	set to 00/20 hex for lo/hi screen pages
SCRLOW:	Lowest screen address. 20/40 hex for lo/hi pages
SCRHI:	High screen address. 40/60 hex for lo/hi pages
IBP:	Input buffer pointer. See section 6.2.

Low page variables from 60 to C2 may also be destroyed by this call and should be saved if desired.

6.2 Initializing input buffer starting point

The display file for A2-3D2 should start at address TDAT (see memory map). You also have the option to set the start address manually.

The IBP (input buffer pointer) is a parameter in the zero page that keeps track of where the interpreter is interpreting. It is similar to a computer's program counter. It is a 16-bit value (least sig. byte first) at IBP:.

The ENTRYN routine initializes the IBP to the address TDAT:. The two load-immediate instructions at IBPSET: and IBPS2: load IBP+1 and IBP respectively.

If you enter via XENT1, ENTRYS, XENT2, or ENTRYN and want to use your own initial IBP value, change the load-immediate instructions at IBPSET: and IBPS2:. If you enter at NXTPT, simply set the value at IBP: to the desired value. Be sure to note that the IBP value changes after you call the program (because it is the active buffer pointer, not an initializing value). On subsequent entries into the A23D2 program make sure to reinitialize it.

6.3 Eliminatable Code Segments

If you are tight on memory, the test database (starting at TDAT) may be eliminated. Various other program segements also may be eliminated. See section 5.4 for details.

6.4 Built-in Test Program

The complex interaction of A2-3D2 functions required extensive testing. A test database that exercises all functions was created and is being updated continuously. The test database starts at TDAT (see memory map).

A small test loop that calls the interpreter, updates eye and independent object call parameters, and tests callable functions resides at memory location 'TSTLUP'. You can test A2-3D2's operation by loading this program and running it.

The built-in test program, database, and the A2-3D2 driver itself come as 3 files on the disk or cassette. The driver will first have to be generated using your old A2-3D1 package. Instructions for putting these three pieces together accompany the cassette or disk. Once you have put the three files together, execute the built-in test by jumping to the start address at TSTLUP (see memory map). It is a continuous loop and will never return.

6.5 Increasing Projection Rate

The following actions will increase the projection rate:

1. Use the non-restoring entry point to call A2-3D2.
2. Use continue and ray points instead of start-cont, start-cont,...
3. Use the fastest lines:
   Fastest - White (140x192)
   Slower - Color (140x192)
   Slowest - Hi-Res (280x192)
4. Use non-clipped projection where appropriate.

7 INTERPRETIVE COMMAND SHEETS

The following set of command sheets describe the whole A2-3D2 command set. Although the A2-3D1 commands and databases developed around them will be interpreted properly, new extensions to the old commands exist. It is easier to use a complete set of command sheets than an old set and many addendum sheets, so all commands are presented.

PURE POINT (140x192 COLOR) COMMAND = 00 hex PNT
= 00 dec

OPERATION: This command specifies a point in space using X,Y,Z coordinates. The program converts this into a 2D point to be projected on the screen or sent to the output array. The point is not projected or sent if it falls off the screen. The point color is determined by the Set Color Mode command (default is white). The point will be plotted as a non-colored single pixel if the display mode has been set to Hi-Res (280x192).

BYTES: 7

FORMAT: 00, X lsb, X msb, Y lsb, Y msb, Z lsb, Z msb in sequential memory locations. X,Y, and Z are double precision, 2's complement, byte swapped values.

EXAMPLE: Input:

Address	Data	Meaning
1b39	00	PNT opcode
1b3a,3b	34,12	X coord. (1234 hex)
1b3c,3d	79,19	Y coord. (1979 hex)
1b3e,3f	00,01	Z coord. (0100 hex)

Resulting Output: The 3-D point X,Y,Z would be projected onto the 2D screen or put into the output array at the current output array address. The output array pointer is advanced to the start of the next output element.

USES: Points are useful where single small dots are require (stars for example).

RULES: Any number of points may be used in any location in the input array.

ARRAY GEN: The 2D projection of this PNT is put in the output array if array generation is turned on.

START POINT (140x192 COLOR)

COMMAND = 01 hex SPNT
= 01 dec

OPERATION: SPNT specifies the beginning of a line in space using X,Y,Z coords. The program converts the start point and the following continue point and projects a 2D line on the screen. The line is eliminated if off the screen and clipped if partially on the screen (if the clipper is turned on). If the clipper is off and the line is partially off the screen, the line is eliminated. A line entry is made in the output array if array generation is on. The line's color is controlled by the Set Color Mode command. A non-color line of 280x192 resolution will be drawn if hi-res mode was previously selected.

BYTES: 7

FORMAT: 01, x lsb, x msb, y lsb, y msb, z lsb, z msb in sequential order, with x,y,z double precision byte-swapped.

EXAMPLE: Input:

Address	Data	Meaning
1co8	01	SPNT opcode
1c09,0A	00,01	X coord. 0100 hex
1c0b,0c	77,00	Y coord. 0077 hex
1c0d,0e	21,43	Z coord. 4321 hex
1c0f	xx,xx	Continue point code and coords.

Result: The line defined in 3D space by the start point and the following continue point is projected as a 2D line on the screen or is entered in the output array, and the output array pointer is advanced to the beginning of the next array entry.

USES: The line is the most-used element in 3D projections. Every line starts with a start point (directly or indirectly). Wherever lines are needed, the start point is useful.

RULES: A continue point MUST follow a start point. No other command or type of point will suffice. If a continue point doesn't follow, the following command will be assumed to be a continue point causing a bad line at best, and an out-of-command synchronization problem and subsequent error exit at worst.

Note: A STCOL command can be used in the middle of a line string (between continue and ray points) to change colors in the middle of the string.

ARRAY GEN: The 2D projection of the line specified by the start point and following continue point is placed in the output array if array generation is turned on.

SET LINE DRAWING MODE

COMMAND = 0c hex LMODE
= 12 dec

OPERATION: LMODE selects between normal ('or') line drawing and exclusive 'or' line drawing, and turns off output array generation.

BYTES: 2

DEFAULT: Upon program loading, normal line mode is in effect. Once changed to the exclusive or line mode, the program remains in the mode until specified otherwise - - even from call-to-call.

FORMAT: 0c, n in sequence causes either normal or exclusive-or line drawing modes to be entered.

n=00 : normal mode
n=01 : exclusive 'or' mode

EXAMPLE: Input:

Address	Data	Meaning
1b80,81	0c,01	draw the following in exclusive 'or' mode
1b82	xxxx	lines to be drawn in exclusive 'or' mode
1c56,57	0c,00	switch back to regular line mode
1c58	xxxx	lines to be drawn in normal mode

USES: Exclusive 'or' lines can be used to draw black on white, white on black, or they can selectively erase lines by drawing the same line twice.

RULES: This command can be used anywhere in an input array. Remember that the program will remain in the line drawing mode it was last in until changed - even from call-to-call.

CAUTION: Never use the LMODE command when any color other than white is in effect. Other colors are generated using an 'AND-OR' mask rathar than a simple 'OR' mask. Selective erasures are thus not possible. IN THE CURRENT PROGRAM, A CRASH CAN RESULT IF YOU USE LMODE WHEN A NON-WHITE COLOR IS IN EFFECT.

ARRAY GEN: This command turns array generation off and puts an end-of-file 79(hex) at the end of the array.

TURN ON OUTPUT ARRAY

COMMAND = 0d hex ARRAY
= 13 dec

OPERATION: ARRAY causes an output array to be generated instead of a screen projection. The array address is also specified.

BYTES: 3

DEFAULT: Upon input array entry, and every time the program is called, the output array generator is turned off.

FORMAT: 0d, a lsb, a msb in sequence causes an output array to be generated starting at address 'a'. The value 'a' is byte-swapped. Array generation ends when an end of array command or 0c (set line drawing mode) command is encountered. The end of file is denoted by an end of array mark (a 79 hex).
The output array is identical in construction to the input array and consists of 0a (points) and 06 (2D lines). The output array may be interpreted as an input array by the program.

EXAMPLE: Input:

Address	Data	Meaning
1cd0	xxxx	lines being projected on screen
1d78	0d	ARRAY opcode
1d79,7a	00,1f	create array at 1f00 hex
1d7b	xxxx	lines to be placed in output array

Result: The lines starting at 1d7b are put into the output array (after 3D-to-2D conversion).

USES: The array feature is useful on non-APPLE machines where seperate line drawing routines are being used. The output array can be used to sve images for later display, and to hold-off projection of 2D lines until all the 3D transformations have been sone, resulting in smoother displays.

RULES: Care should be taken in setting the output array's address. It should not overlap a program or screen display area. APPLE II screen control commands should not be used while creating an output array. They create no array entries but still affect the APPLE II screen functions and memory.

ARRAY GEN: Array generation is turned on and the address of array generation is specified.

SCREEN SIZE SELECT

COMMAND = 0e hex SCRSZ
= 14 dec

OPERATION: SCRSZ privides the program with screen bit ratio and screen centering information. Arguments in the command specify screen bit height and width and x,y centers.

BYTES: 5

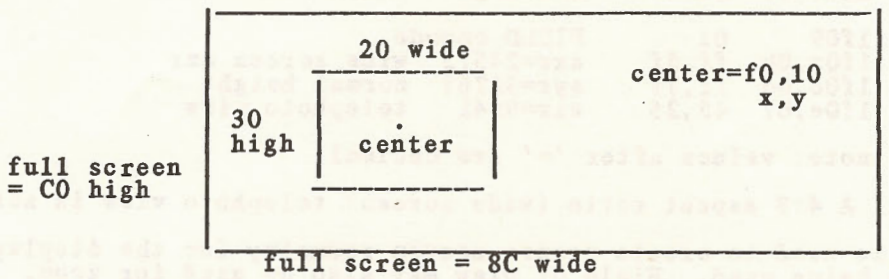
DEFAULT: Screen size on program load-up is preset to 140-wide, 192-high (decimal). Screen center is at 0,0.

FORMAT: 0e, scr. width, scr. height, scr. x center, scr. y center in sequential order sets the bit ratio and center. Height and width can reach a maximum of 256 x 256, and screen center can be anywhere within +/- 127.

EXAMPLE: Input:

Address	Data	Meaning
1f19	0e	SCRSZ opcode
1f1a,1b	20,30	bit ratio is 20 hex wide, 30 hex high
1f1c,1d	f0,10	screen center is x=f0 hex y=10 hex

Result: The images are projected onto the following screen area:



USES: SCRSZ is useful for setting screen size on non-APPLE devices, and for creating multiple and custom windows on APPLE displays.

RULES: Don't center the screen in such a way as to cause the image to fall outside of the display device's plotting boundary.

ARRAY GEN: No entry is made in the output array.

EASY INITIALIZE

COMMAND = 10 hex INIT
= 16 dec

OPERATION: This command puts the APPLE II screen into the high-res, split graphics/text, page 1 viewing mode.

BYTES: 1

FORMAT: 10(hex) in an input array initializes the screen.

EXAMPLE: Input:

Address	Data	Meaning
1b00	10	initialize the screen

Result: The screen goes into hi-res, split, page 1 viewing mode

USES: INIT is a good way to start most page 1 input arrays. It puts the screeen in an easy to use mode because 4-lines of text are provided on the bottom of the screen. This command avoids having to use 3 seperate 07 (display select) commands.

RULES: It is good to use this at the beginning of input arrays. There are no location restrictions however.

ARRAY GEN: No entry is made in the output array.

NO OPERATION

COMMAND = 11 hex NOP
= 17 dec

OPERATION: NOP performs no operation and is skipped-over by the program.

BYTES: 1

FORMAT: An 11 (hex) in the input array is ignored and skipped over.

EXAMPLE: Input:

Address	Data	Meaning
1e05	11	NOP code
1e06	11	NOP code
1e07	xxxx	valid commands

Result: The two NOPs are skipped and the valid commands at 1e07 are processed.

USES: The NOP is good for filling space that might be used later (especially at the beginning of an input array where initializes, erases, and location information may be added). It is also good for elimination unwanted commands without compressing the entire array to fill-in the gap left by the command's removal. If you are using many sequential NOPs, the 'SKIP' command might be more appropriate. See command 1a (hex).

RULES: NOPs may be used anywhere.

ARRAY GEN: No entry is made in the output array.

SET COLOR MODE

COMMAND = 12 hex STCOL
= 18 dec

OPERATION: STCOL sets the color mode that will be in effect for the following lines. Five colors are available. If array generation is turned on the color command is copied into the output array.

BYTES: 2

FORMAT: 12(hex),n in sequence specifies the color mode. The value 'n' corresponds to:

n	color
00	white
01	green
02	purple
03	blue
04	orange

EXAMPLE: Input:

address	data	meaning
1352	12	STCOL opcode
1353	02	set color to purple
1354	xxxx	lines and points to be plotted as purple

Result: Lines and points starting at 1354 are plotted as purple. If the array generator is on, the STCOL command is copied into the output array.

USES: This command is used wherever color points and lines are needed.

RULES: This command is instruction-modifying and stays in effect from call-to-call. Changing colors takes a lot of overhead time. It is wise to group all lines of the same color together to avoid excessive switching.

CAUTION: Never use the LMODE command when any color other than white is in effect. Other colors are generated using an 'AND-OR' mask rathar than a simple 'OR' mask. Selective erasures are thus not possible. IN THE CURRENT PROGRAM, A CRASH CAN RESULT IF YOU USE LMODE WHEN A NON-WHITE COLOR IS IN EFFECT.

RATE: 1500 color changes/second.

SPEED: 650 usec.

ARRAY GEN: The STCOL command is copied directly into the output array if array generation is turned on.

INDEPENDENT OBJECT CALL

COMMAND = 13 hex ICALL
= 19 dec

OPERATION: ICALL projects an object that was defined earlier in its own reference frame. The object is projected onto the screen or into an output array (if selected). ICALL also passes object positional and rotational information.

BYTES: 13

FORMAT:

```
.byte 13 hex  -independent object call opcode
.byte 0       -status flag   00=inactive  ff=active  nn=countdown
.byte 0,0     -x offset (double precision lsb,msb)
.byte 0,0     -y offset
.byte 0,0     -z offset
.byte 0       -pitch (single precision)
.byte 0       -bank (single precision)
.byte 0       -heading (single precision)
.byte 0,0     -definition address
```

Status flag - If set to 0, the object is not projected. If set to ff hex the object is projected. If not 00 or ff, then the value is a countdown value that is decremented every frame (from fe down to 00 for instance). The object is projected until 00 count is reached.

Offset X,Y,Z- This is used to move the object about in space. This is the displacement of the object's rotation-center from the absolute 0,0,0 center of space.

Pitch,Bank,Heading - These are single precision pseudodegrees. They define the rotation of the object about its own center.
See section 4.4 for rotation and translation sense of direction.

Definition Address- This double precision byte-swapped address value points to the object definition.

EXAMPLE: Input:

address	data	meaning
1000	13	ICALL opcode
1001	ff	status = object turned on
1002,1003	32 12	object's X center in space (1234 hex)
1004,1005	00 fe	object's Y center in space (fe00 hex)
1006,1007	81 19	object's Z center in space (1981 hex)
1008	34	object's pitch about its own center (34 hex)
1009	c7	object's bank about its center
100a	01	object's heading about its center
100b,100c	00 20	address of object definition (2000 hex)

Result:
The object is 'moved' to position X=1234, Y=fe00, Z=1981 and is rotated about its F axis by heading, its transverse axis by pitch, and finally about its lateral axis by bank. The 'moved' object is then projected as viewed by the viewer's eye.

ARRAY GEN: No entry is made in the output array.

SET RESOLUTION

COMMAND = 14 hex SRES
= 20 dec

OPERATION: SRES sets the line drawer mode to either hi-res (280 x 192) or low res (140x192 color) mode. 3D lines as well as LIN2D and PNT2D lines are affected. If output array generation is enabled, this command is transferred directly to the output array to control display resolution at draw-time.

BYTES: 2

FORMAT: 14(hex),m where m=0 for low res
where m=1 for hi-res

EXAMPLE: Input:

address	data	meaning
1000	14	SRES opcode
1001	01	hi-res mode
1002	xxxx	lines to be drawn in hi-res mode

Result: Lines starting at address 1002 are plotted in hi-res mode. If the output array is turned on, the SRES command is transfered to it.

USES: Hi-res lines are used where very accurate but non-colored drawings are needed.

DEFAULT: Low-res.

RULES: SRES sets line jumpout traps by dynamically modifying the program. The mode is not initialized by entry into A2-3D2 and will stay the same from call-to call unless explicitly changed.

ARRAY GEN: The SRES command is copied directly into the output array if array generation is turned on.

HI RES (280 x 192) LINE 2D

COMMAND = 15 hex HLIN
= 21 dec

OPERATION: HLIN feeds line drawing information directly to the display screen. It takes a start and end point (both with double precision X values to accomodate 280 resolution) specified in the command and draws it on the screen. The line has no specific color characteristics.

BYTES: 7

FORMAT: 15 hex, Xlo, Xhi, Y, X'lo, X'hi, Y' in sequential memory locations plots the line X,Y to X'Y' on the screen. X values may range from -140 to 139 (decimal) and Y may range from -96 to 95. Screen center is 0,0 - positive x is right, positive y is up.

EXAMPLE: Input:

Address	Data	Meaning		
1b00	15	HLIN opcode		
1b01,2,3	07,01,22	line start point	x=0107,	y=22 hex
1b04,5,6	29,ff,e4	line end point	x=ff29,	y=e4 hex

Result: The line from 0107,22 to ff29,e4 is drawn.

USES: HLIN is used for drawing additional 2D lines where accuracy as well as a full 280 x range is required.

RULES: Set Resolution and Set Color commands have no effect on HLIN lines. They are always hi-res.

ARRAY GEN: No entry is made in the output array.

SET HI-RES BIAS

COMMAND = 16 hex SHRB
= 22 dec

OPERATION: SHRB sets the center point of a 256 x 192 drawing field on the 280 x 192 screen. This bias is used with HLIN2 lines and HPNT2 points.

BYTES: 4

FORMAT: 16 hex, Xlo, Xhi, Y, where Xhi,Xlo = 16-bit center X
where Y = center Y

EXAMPLE: Input:

Address	Data	Meaning
9000	16	SHRB opcode
9001,02	F4,FF	Shift 256 wide X field FFF4 units
9003	00	Y bias = screen center

Result: The 256x192 drawing field will be shifted -12 (FFF4) units. This corresponds to a left shift of the field by 12 units. The -128 to 127 range drawing field will map to screen locations -140 to 115. The screen values of 116 to 139 are inaccessable by HLIN2 and HPNT2.

USES: Used to set screen bias prior to HLIN2 and HPNT2 command.

DEFAULT: The Hi-Res bias is 0,0 on program load. These values are not reinitialized upon program entry, so they stay the same from frame to frame unless explicitly changed.

ARRAY GEN: No entry is made in the output array.

HI-RES (X=256 limited) LINE 2D

COMMAND = 17 hex HLIN2
= 23 dec

OPERATION: HLIN2 draws lines in hi-res (256 x 192) mode. There are 24 unused X points out of the available 280 on the APPLE II screen. The Set Hi-Res Bias command determines where in the 280 X field the 256 X values will be mapped.

BYTES: 5

FORMAT: 17 hex, x, y, x', y' in sequential memory locations plots the line x,y to x',y' on the screen. X values may range from -128 to 127 and Y values may range from -96 to 95.

EXAMPLE: Input:

Address	Data	Meaning		
1b0f	17	HLIN2 opcode		
1b10,11	73,20	line start point	x=73 hex,	y=20 hex
1b12,13	84,f3	line end point	x=84 hex,	y=f3 hex

Result: The line 73,20 to 84,f3 is drawn in hi-res mode. Actual screen position of the line depends on the bias values set by the 'Set Hi-Res Bias' command.

USES: Used where hi-res lines are desired, but memory storage and computation must be limited to 8-bit x values.

ARRAY GEN: No entry is made in the output array.

HI-RES (280 x 192) POINT PLOT 2D COMMAND = 18 hex HPNT
= 24 dec

OPERATION: HPNT plots a point in hi-res (280 x 192) coordinates. A double precision X is needed to fully specify the 280 horizontal resolution.

BYTES: 4

FORMAT: 18 hex, Xlo, Xhi, Y, in sequential memory locations plots the point X,Y on the 280 x 192 resolution screen.

EXAMPLE: Input:

Address	Data	Meaning
1000	18	HPNT opcode
1001	07,01	X value = 107 hex
1003	22	Y value = 22 hex

Result: The point 107,22 is plotted on the hi-res screen.

USES: Used where hi-res points and full 280 X range points are required.

ARRAY GEN: No entry is made in the output array.

HI-RES (X=256 limited) POINT PLOT 2D

COMMAND = 19 hex HPNT2
= 25 dec

OPERATION: HPNT2 plots a point in hi-res (256 x 192) coordinates. The screen bias values set by the Set Hi-Res Bias command determine how the point is mapped onto the 280 x 192 screen.

BYTES: 3

FORMAT: 19 hex, X, Y, in sequential memory locations plot the point X,Y on the screen. Actual screen location depends on the Hi-Res Bias in effect at the time of command execution.

EXAMPLE: Input:

Address	Data	Meaning
1000	19	HPNT2 command
1001	74,e7	X=74 Y=e7 hex

Result: The point 74,e7 is plotted on the hi-res screen. Actual position is biased by the values set in the SHRB command.

USES: Used where hi-res points are needed but where computation and memory must be limited to 8-bit precision.

ARRAY GEN: No entry is made in the output array.

SKIP SEGMENT

COMMAND = 1a hex SKIP
= 26 dec

OPERATION: SKIP conditionally branches display file interpretation around a given display segment (specified by a byte count). A status byte within the SKIP command allows the command to be turned on or off to enable or disable branching.

BYTES: 3

FORMAT: 1a(hex), size, status in sequential memory locations cause the next 'size' bytes to be skipped if status=00 hex. If status=ff hex, the SKIP is inactive and interpretation resumes immediately following the SKIP instruction. Note that when 'size'=0 no skipping takes place (display file interpretation resumes after the SKIP command).

Note: Size is a nonsigned positive integer. Range = 0 to 252. Don't use 253, 254 or 255.

EXAMPLE: Input:

Address	Data	Meaning
1000	1a	SKIP opcode
1001,1002	3,0	size=3 bytes, skip=0=active
1003	xx	skipped-over
1004	xx	skipped-over
1005	xx	skipped-over
1006	xxxx	interpretation resumes here.

Result: The bytes at 1003,1004 and 1005 are skipped over.

USES: This command is used when commands are to be replaced (replacement by skipping-over versus NOP-filling), and when commands and display segments are to be switched on and off (by toggling the status between 0 and ff).

NOTE: The 'sense' of the status byte may seem backwards (0=skip, ff=no skip), but this 'sense' was chosen for a reason. Think of the skip command as a 'turn object on or off' command (0=object off, ff=object on).

ARRAY GEN: No entry is made in the output array.

PAUSE

COMMAND = 1b hex PAUS
= 27 dec

OPERATION: A pause of 'n-fifths' of a second occurs when the PAUS command is encountered in a display file.

BYTES: 2

FORMAT: 1b(hex), time in sequential memory locations cause a pause of time/5 seconds.

EXAMPLE: Input:

Address	Data	
1000	1b	PAUS opcode
1001	05	time = 5 units

Result: Display interpretation pauses for 5/5ths or one second.

USES: Used where a series of display frames with time gaps between them are desired.

RULES: The time value 0 results in no pause. This can be used as a convenient switch to turn the pause off.

ARRAY GEN: No entry is made in the output array.

SET 3D-to-3D CONVERSION ARRAY ADDRESS COMMAND = 1c hex SET323
= 28 dec

OPERATION: The 3D-to-3D conversion output array pointer is set to the specified location in memory. Array generation begins at this point.

BYTES: 3

FORMAT: 1c(hex), address lsb, address msb in sequential memory locations sets the output array pointer to the specified address.

EXAMPLE: Input:

Address	Data	Meaning
1300	1c	SET323 opcode
1301,02	23,14	address = 1423 hex

Results: Array generation is set to start at 1423 hex

RULES: The address is byte swapped (lsb,msb).

DEFAULT: Array generation's default address is 'TDAT' (see memory map) if not specified by this command.

ARRAY GEN: No entry is made in the regular output array. Note that the 3D-to-3D conversion array is a seperate array and is not associated with the regular output array.

SET 3D-to-3D CONVERSION ARRAY STATUS COMMAND = 1d hex GN323
= 29 dec

OPERATION: The GN323 command turns 3D to 3D conversion array generation on and off. It can also turn projection on or off.

BYTES: 2

FORMAT: 1d(hex), status in sequential memory locations affects the 3D to 3D conversion array generation system as follows:

status	result
00	Turn array generation off
01	Turn array generation on
02	Turn array generation on and suppress final projection.

EXAMPLE: Input:

Address	Data	Meaning
1500	1d	GN323 opcode
1501	02	status = supress screen projection.

Results: From 1503 on, all 3D to 3D conversions are submitted to an output array. Screen projection is suppressed.

RULES: The array is built at the address specified by the SET323 command.
The array pointer is continually updated as the array is built. The array pointer is at arrset+1 (lsb) and arrset+2 (msb). See the memory map for the addresses.

DEFAULT: 3D to 3D conversion array generation is initially off. The default array generation address is 'TDAT' (see memory map) until respecified by SET323.

ARRAY GEN: No entry is made in the regular output array. Note that the 3D-to-3D conversion array is a seperate array and is not associated with the regular output array.

END OF FILE

COMMAND = 79 hex EOF
= 121 dec

OPERATION: EOF defines the end of a display file. Non-command codes also cause end of file action, but today's non-command codes may be tomorrow's new command codes. The value 79 hex will always remain the official EOF marker.

BYTES: 1

FORMAT: 79 hex at the end of a file indicates an end of file. Program data, another display file, or anything else may follow the EOF and will be ignored by the A2-3D2 software.

NOTE: It is possible to 'SKIP' or 'JUMP' over an EOF, so be careful when you use these instructions.

ARRAY GEN: An EOF is appended to the output array.

8 DIRECT CALL FUNCTION SHEETS

The following 'DIRECT CALL' functions can be used when display file interpretation is not desired.

CAUTION: MAKE SURE TO CALL THE INTERPRETER ONCE IN INTERPRETIVE MODE BEFORE USING THESE COMMANDS. DISPLAY VARIABLES ARE INITIALIZED BY THIS ACTION.

TRIG FUNCTION CALL

```
JSR  XSIN
JSR  XCOS
```

OPERATION: An angle in 0-255 range pseudodegrees poked into the TDATA byte in memory (see memory map) generates a sine or cosine funcion at location TDATA (lsb) and TDATA+1 (msb).

INPUT: TDATA = angle in pseudodegrees

CALL: JSR XSIN or JSR XCOS depending on desired result.

OUTPUT: TDATA,TDATA+1 = 16-bit fractional 2's comp., byte-swapped result.

DOUBLE PRECISION MULTIPLY CALL JSR MULT

OPERATION: The fractional product of MPYER * MCAND yield a fractional result in RESULT. The result is an approximation that is accurate to about 2 or 3 units error. The MPYER, MCAND and RESULT are all 16-bit fractional.

INPUT:
MPYER = multiplier lsb
MPYER+1 = multiplier msb
MCAND = multiplicand lsb
MCAND+1 = multiplicand msb

CALL: JSR MULT

OUTPUT:
%A = lsb result
%X = msb result

SINGLE PRECISION MULTIPLY CALL JSR FSTMUL

OPERATION: The fractional product of %A * %X yields a fractional result in %A. The results are exact. NOTE: No negative %X is allowed

INPUT: %A = 8-bit multiplier
%X = 8-bit multiplicand

CALL: JSR FSTMUL

OUTPUT: %A = 8-bit fractional product

LIMITS: No negative multiplicands (%X) are allowed.

DOUBLE PRECISION DIVIDE CALL JSR DIVIDE

OPERATION: The 16-bit fractional quotient of %A,%X/MCAND is stored in the MPYER word.

INPUT:
%X = top of fraction (lsb)
%A = bottom of fraction (msb)
MCAND = bottom of fraction (lsb)
MCAND+1= bottom of fraction (msb)

CALL: JSR DIVIDE

OUTPUT:
MPYER = quotient (lsb)
MPYER+1 = quotient (msb)

LIMITS: The fraction top must be less than the fraction bottom or overflow will result.

SCREEN ERASE JSR ERA

OPERATION: The high or low screen is erased (or filled with white).

INPUT: %A = 00 for black erase
= FF for white fill

CALL: JSR ERASE2 for low screen (2000-3fff) erase
JSR ERASE4 for high screen (4000-5fff) erase

OUTPUT: Screen erased or filled. %X destroyed.

HI-RES POINT PLOT JSR DPOINT

OPERATION: A hi-res point is plotted on the hi-res screen.

INPUT: %X= 8 most significant bits of X (280 range)
STRTXL= bit 7 =lsb of X bits 6-0 = 0
%Y= Y

CALL: JSR DPOINT

LIMITS: X range from -140 to 139 decimal.
Y range from -96 to 95 decimal.
The screen (page 1 or 2) is the last screen selected by an interpretive call.

COLOR POINT PLOT JSR LPOINT

OPERATION: A color point is plotted on the hi-res screen.

INPUT: %X= X
%Y= Y

CALL: JSR LPOINT

LIMITS: X range from -70 to +69
Y range from -96 to +95
Screen page and color are those last selected by interpretive call.

HI-RES LINE DRAW JSR DLINE

OPERATION: Draws a hi-res (280 x 192) line on screen.

INPUT:
```
startx = 8 msb of start x
strtx1 = bit 7 = lsb of start x   bits 6-0 = 0
starty = start y
endx   = 8 msb of start x
endx1  = bit 7 = lsb of end x     bits 6-0 = 0
endy   = end y
```

CALL: JSR DLINE

OUTPUT: Line is drawn on screen last selected by interpretive call.

LIMITS:
```
X range = -140 to 139
Y range = -96 to 95
```

COLOR LINE DRAW JSR LLINE

OPERATION: Draws a color line (140 x 192) on screen.

INPUT:
```
startx = start x
starty = start y
endx   = end point x
endy   = end point y
```

CALL: JSR LLINE

OUTPUT: Line is drawn on screen.

LIMITS:
```
X range = -70 to 69
Y range = -96 to 95
```

SET DISPLAY RESOLUTION

JSR SRHI
JSR SRLO

OPERATION: Sets line drawer resolution to HI (280 x 192) or LO (140 x 192 color).

INPUT: None

CALL:
```
JSR  SRHI   to set HI resolution
JSR  SRLO   to set LO resolution
```

OUTPUT: None

```
TDL Z80 CP/M DISK ASSEMBLER VERSION 2.21
.MAIN. - A2-3D2
A2-3D2 Memory Map

                        @; ***********  subLOGIC A2-3D2 MEMORY MAP  ***********
                        @
                        @; INTERPRETER CALL ADDRESSES
                        @
    6000                @adrs = xent1    ; normal restoring entry
    6003                @adrs = xent2    ; fast nonrestoring entry
    606C                @adrs = entrys   ; same function as xent1
    6090                @adrs = entryn   ; same function as xent2
    6118                @adrs = nxtpt    ; display loop entry
                        @
                        @; CALLABLE FUNCTION CALL ADDRESSES
                        @
    6006                @adrs = xsin     ; sine
    61F6                @adrs = sinex    ; same function as sine
    6009                @adrs = xcos     ; cosine
    620F                @adrs = cosex    ; same function as xcos
    6480                @adrs = mult     ; D.P. multiply
    659D                @adrs = fstmul   ; S.P. multiply
    65DE                @adrs = divide   ; D.P. divide
    7629                @adrs = erase2   ; low page erase
    76A8                @adrs = erase4   ; hi-page erase
    7714                @adrs = dpoint   ; hi-res point
    771F                @adrs = lpoint   ; color point
    7E42                @adrs = dline    ; hi-res line
    7748                @adrs = lline    ; color line
    8051                @adrs = srhi     ; set resolution to hi
    805F                @adrs = srlo     ; set resolution to lo
                        @
                        @; IMPORTANT VARIABLES
                        @
    00B3                @adrs = startx  ;2-d start point
    00A4                @adrs = strtx1
    00B4                @adrs = starty
    00B5                @adrs = endx    ;2-d end point
    00A3                @adrs = endx1
    00B6                @adrs = endy
    007C                @adrs = clipon  ;clipper switch  0=clip 1=no clip
    007D                @adrs = genar   ;generate array switch 0=screen 1=array
    007E                @adrs = m1      ;9-element rotation matrix m1-m9
    0080                @adrs = m2
    0082                @adrs = m3      ;    |m1 m2 m3|
    0084                @adrs = m4      ;    |m4 m5 m6|
    0086                @adrs = m5      ;    |m7 m8 m9|
    0088                @adrs = m6
    008A                @adrs = m7
    008C                @adrs = m8
    008E                @adrs = m9
    0090                @adrs = xv      ;viewer x,y,z
    0092                @adrs = yv
    0094                @adrs = zv
    0099                @adrs = addrs   ;line gen. variable
    009B                @adrs = ibp     ;input buffer pointer
    613E                @adrs = tdata   ;trig data
    0078                @adrs = mpyer   ;multiply/divide data
    007A                @adrs = mcand
    609A                @adrs = ibpset  ;input buffer immediate load inst. 1
    609E                @adrs = ibps2   ;input buffer immediate load inst. 2
    00BB                @adrs = scbias  ;screen address bias (byte)
    00BC                @adrs = scrlow  ;screen address low (byte)
    00BD                @adrs = scrhi   ;screen address high (byte)
    71F5                @adrs = arr323+1 ;3D-to-3D array pointer (lsb)
    71F6                @adrs = arr323+2 ;3D-to-3D array pointer (msb)
```

3D2 ENHANCEMENT