High-Res Astrocade Machine Language Subroutines

By MCM Design / Michael Matte

February, March 2020 and January 2021

Description

The following high-resolution subroutines for the Bally Arcade/Astrocade were created by MCM Design and sent to Adam Trionfo as photocopies and text documents via email in February and March 2020. Adam compiled this collection of subroutines from ten documents in January 2021. It contains Z80 machine language subroutines for use with a modified-for-hi-res Bally Arcade/Astrocade home videogame console.

Table of Contents

Michael Matte's compiled comments about this project:

You are getting my photocopied, handwritten pages of documentation. My library photocopier was set for dark prints. Some of these pages show erasure smudges. Looks like I found another good reason to make the time to learn how to program using the Zmac cross-assembler for easier editing.

Each subroutine is extensively commented on, including a note "This subroutine is similar to low-res sub #__" plus you might see a Nutting Manual reference page from where the hi-res sub was created. These ML sub docs are strictly for

someone who has access to a modified hi-res Astrocade, is experienced in ML/AL programming and is looking for a custom hi-speed subroutine application.

The doc's intent is to help a hi-res programmer get started with custom programming hi-res graphic patterns and moving patterns around the screen without the need to create this particular hi-res application from scratch.

These ML subroutines, except the custom subroutine example for MCM Design's hi-res multi-pager, function similar to their low-res equivalent. However, these hi-res subs must be called directly. There is no processing UPI (User Programmer Interface). Note, MCM Design's upcoming Hi-Res ROM will include sub's similar to these subs and will utilize a UPI. The ROM UPI and sub's will be well documented.

Perhaps someday in the future, someone with ML/AL experience may acquire or build a modified hi-res Astrocade and might find this info useful. Keep in mind that I will eventually submit to you my hi-res MLM and hi-res ROM projects.

Michael's Posting Wishes for these Subroutines

Post the scans and docs right below the posting on Ballyalley.com of my modified hi-res Astrocade with the static screen RAM, since that's what this user info was created for. Group together all of this info as "High-Res Astrocade Machine Language Subroutines" by MCM Design. Then, break down this grouping into 5 sections listing each of the 5 respective doc/scans. Maybe add a link to these postings in the Bally Alley ML section.

LOW AND HIGH-RES DATA BLOCK COMPARISONS
(Related To Graphic Patterns)
By MCM Design
Margins Left 0.9, Right 1.0

This posting only compares the necessary low and hi-res data blocks related to the writing and moving (vectoring) of graphic patterns on a TV/monitor screen. Refer to the attached diagrams detailing the data blocks and also their respective coordinate systems.

A low-res comparison of a hi-res vector block along with its limits table reveal some slight differences. In low-res, the X coordinate can normally vary from 0 to 159. This coordinate can be defined with just 1 byte (0–255). In hi-res, the X coordinate can normally vary from 0–319. So, the hi-res X coordinate must be defined using 2 bytes. Because of this difference, the hi-res vector block and hi-res limits table are longer compared to that of low-res.

The X and Y coordinates, plus the X and Y deltas within a vector block are all expanded with double digit precision using high and low designations. If you place an imaginary decimal point between the high and low designations, then it is easier to understand how these parameters are utilized. For example, the X coordinate can be viewed as:

X coordinate = XH.XL where,
XH is the actual X coordinate (to the left of the decimal point) that is plotted on the X axis of the coordinate system when writing graphic patterns.
XL is a double digit decimal breaking down the X coordinate further, down to 1/100 of a X coordinate unit.

In hi-res, the XH coordinate is defined using 2 bytes, so XH.XL requires 3 bytes. Similarly, in hi-res, deltaXH.deltaXL requires 3 bytes.

For low-res graphic applications, refer to other postings on the Bally Alley website such as:
The Nutting Manual
"An In-Depth Look At ..." tutorial series by MCM Design

The Better Bally Book website also provides info.

The Bally Alley also posts hi-res application docs. MCM Design will continue its effort to provide hi-res docs for users that have access to or desire to build a modified hi-res Astrocade.

End Of Posting
MCM Design
March 2020

This page intentionally left blank

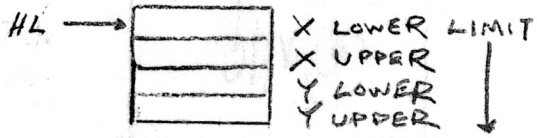
LOW AND HIGH-RES COMPARISONS — AN OVERVIEW FOR VECTOR (MOTION), COORDINATE LIMITS AND GRAPHIC PATTERN DATA BLOCKS

LOW-RES

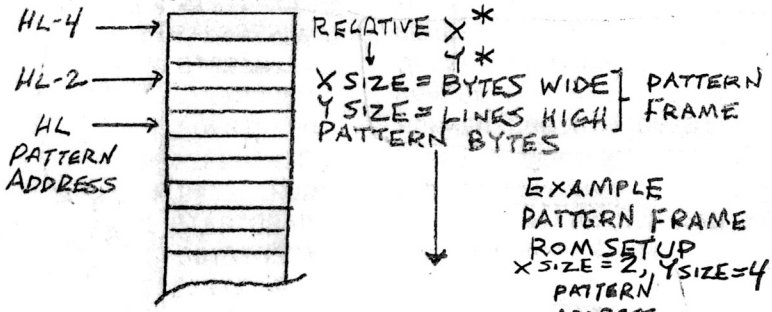
BALLY/ASTROCADE 15 BYTE VECTOR BLOCK

BYTE #		
0	MAGIC REGISTER VALUE	
1	VECTOR STATUS	
2	TIME BASE	
3	ΔX_L	DELTA X $\Delta X_H \cdot \Delta X_L$
4	ΔX_H	
5	X_L	X COORDINATE $X_H \cdot X_L$ $0\text{-}159_D$
6	X_H	
7	X CHECKS MASK	
8	ΔY_L	DELTA Y $\Delta Y_H \cdot \Delta Y_L$
9	ΔY_H	
10	Y_L	Y COORDINATE $Y_H \cdot Y_L$ $0\text{-}101_D$
11	Y_H	
12	Y CHECKS MASK	
13	OLD PATTERN MAGIC ADDRESS	OPTIONAL PATTERN BLANK (2 BYTES)
14		

LOW-RES COORDINATE LIMITS TABLE

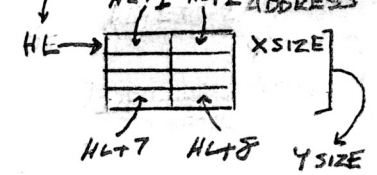


LOW-RES GRAPHIC PATTERN



*FOR OPTIONAL ANIMATED PATTERN AT X+RELATIVE X, Y+RELATIVE Y. IF NOT, ANIMATED PATTERN, SET RELATIVE X = RELATIVE Y = 0

EXAMPLE PATTERN FRAME ROM SETUP $X_{SIZE}=2$, $Y_{SIZE}=4$



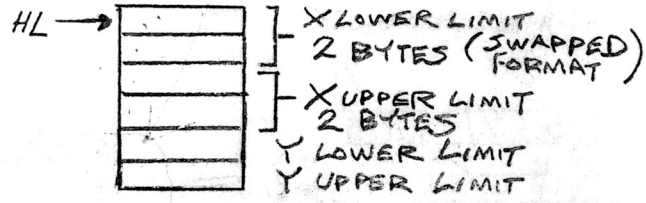
HI-RES

MCM DESIGN SUPPORTING SOFTWARE

MCM DESIGN SUPPORTED 17 BYTE VECTOR BLOCK

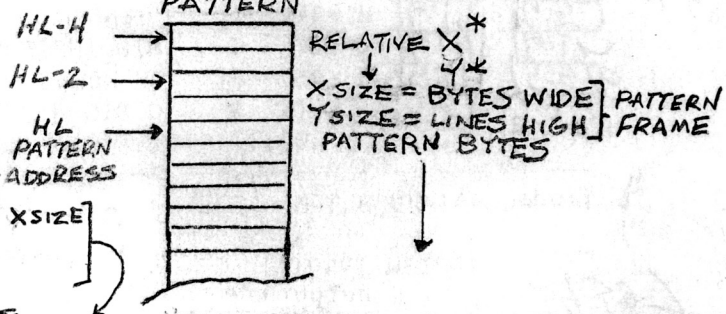
BYTE #		
0	MAGIC REGISTER VALUE	
1	VECTOR STATUS	
2	TIME BASE	
3	ΔX_L	DELTA X $\Delta X_H \cdot \Delta X_L$
4	ΔX_H 2 BYTES	
5		
6	X_L	X COORDINATE $X_H \cdot X_L$ $0\text{-}319_D$
7	X_H 2 BYTES	
8		
9	X CHECKS MASK	
10	ΔY_L	DELTA Y $\Delta Y_H \cdot \Delta Y_L$
11	ΔY_H	
12	Y_L	Y COORDINATE $Y_H \cdot Y_L$ $0\text{-}203_D$
13	Y_H	
14	Y CHECKS MASK	
15	OLD PATTERN MAGIC ADDRESS	OPTIONAL PATTERN BLANK (2 BYTES)
16		

HI-RES COORDINATE LIMITS TABLE



HI-RES GRAPHIC PATTERN

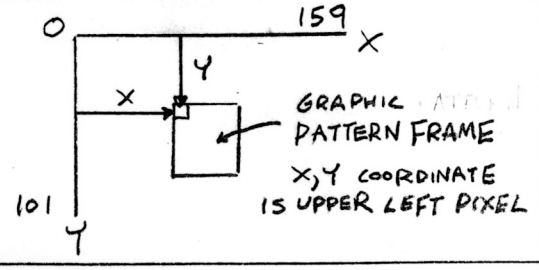
SAME SETUP AS LOW-RES



LOW AND HIGH-RES COMPARISONS - AN OVERVIEW FOR NORMAL AND FLOPPED COORDINATE SYSTEMS

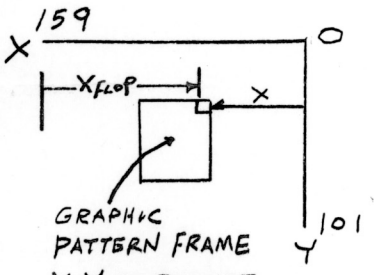
BALLY/ASTROCADE LOW-RES

NORMAL FRAME WRITE



FLOPPED MIRROR IMAGE FRAME WRITE (EXAMPLE, GUNFIGHT)

MAGIC REGISTER BIT 6=1



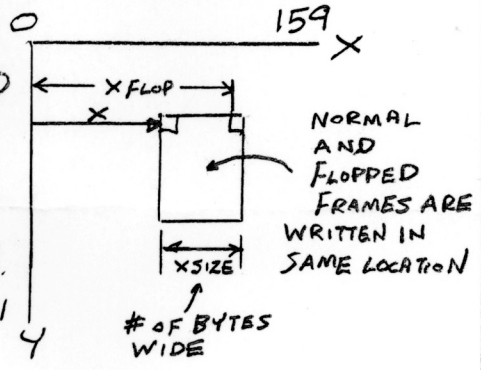
$X_{FLOP} = 159 - X$ ← NORMAL FRAME WRITE X COORDINATE

↑ CALCULATED (ADJUSTED) VALUE BY ON-BOARD LOW-RES GRAPHIC WRITE SUBROUTINES

USER OPTIONAL ADJUSTED FLOPPED FRAME WRITE

So, NORMAL AND FLOPPED FRAMES ARE WRITTEN IN THE SAME LOCATION.

MAGIC REGISTER 101 BIT 6=1



$X_{FLOP} = X + 4(X_{SIZE}) - 1$

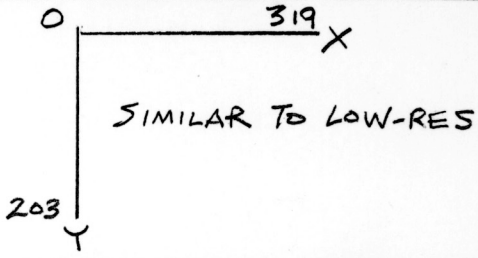
X_{SIZE} = NUMBER OF PATTERN FRAME BYTES WIDE

$X_{ADJUST} = 159 - X_{FLOP}$

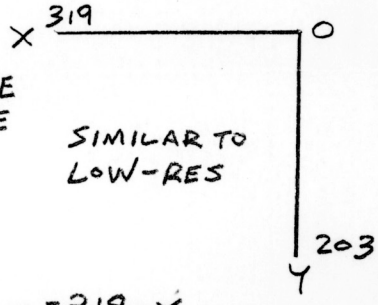
↑ THIS IS THE USER CALCULATED (ADJUSTED) X COORDINATE TO FLOP PATTERN FRAME IN THE SAME LOCATION AS THE NORMAL FRAME WRITE

MCM DESIGN SUPPORTING HIGH-RES (SIMILAR TO LOW-RES)

NORMAL FRAME WRITE



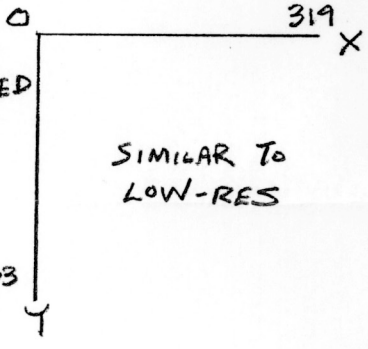
FLOPPED MIRROR IMAGE FRAME WRITE



$X_{FLOP} = 319 - X$ ← NORMAL FRAME WRITE X COORDINATE

↑ CALCULATED (ADJUSTED) VALUE BY HIGH-RES GRAPHIC WRITE SUBROUTINES (MCM DESIGN SUPPORTING SOFTWARE)

USER OPTIONAL ADJUSTED FLOPPED FRAME WRITE



X_{FLOP} = SAME AS LOW-RES

X_{SIZE} = SAME AS LOW-RES

$X_{ADJUST} = 319 - X_{FLOP}$

↑ USER CALCULATED (ADJUSTED) X COORDINATE TO USE FOR THE FLOPPED FRAME.

NOTE: IF THE PIXEL WIDTH OF THE ACTUAL PATTERN WITHIN ITS FRAME, IS LESS THAN THE FRAME'S PIXEL WIDTH, THEN TWEEK X_{ADJUST} SO THE NORMAL AND FLOPPED PIXEL PATTERNS ARE WRITTEN IN THE SAME LOCATION.

CONVERT HIGH-RES COORDINATES TO A MAGIC ADDRESS
For Use On A Modified Hi-Res Astrocade
Similar To The On-Board Low-Res Subroutine #56
(copied from MCM Design's hi-res Multipage Test Demo, in the hand written code listing, pages 55 and 69)
Margins Left 0.9, Right 1.0

This posting is for a ML/AL programmer who has access to a modified hi-res Astrocade. Rather than create an applicable subroutine from scratch, this tested hi-res subroutine from MCM Design can be used as a reference doc.

Refer to the attached scanned hand written hi-res subroutine listing labeled as RELTA1.

This subroutine (or a variation of it) has been used in several hi-res demos created by MCM Design. It is usable for hi-res graphic pattern (or character) writes utilizing the various magic functions. For details and program examples for writing graphic patterns using magic functions, refer to MCM Design's "An In–Depth Look At ..." tutorial series posted on the Bally Alley.

https://ballyalley.com/ml/ml_docs/ml_docs.html

This tutorial series focuses on the low-res mode, but there is info related to writing patterns using magic functions that can be used as a guide for the hi-res mode. The only difference in hi-res, is that the screen RAM utilizes more pixels, more bytes.

This subroutine is called directly. There is no user programmer interface (UPI) required to process the calling of this subroutine. The Z80 CPU register entry requirements are specified at the beginning of the subroutine listing.

The subroutine is labeled as RELTA1 (Relative To Absolute) and is similar to the low-res sub#56. The low-res version of RELTA1 is listed in the Nutting Manual Z80 Cross Assembler listing, page 70. MCM Design used the low-res version and revised it for a hi-res application.

RELTA1 converts a graphic pattern's screen X,Y coordinates to their corresponding magic address. The X,Y coordinates of a pattern normally written to the screen display, point to the upper left pixel of the pattern frame. The coordinates of a flopped pattern point to the upper right pixel of the flopped pattern frame. The converted (calculated) magic address is passed on in the Z80 DE register at the exit of this subroutine for use with the actual graphic pattern magic write subroutine, which should follow RELTA1.

The ROM address of the graphic pattern to be written is specified in the Z80 register HL at entry of RELTA1. This pattern address in HL is saved and is also passed on in the Z80 HL register at the exit of RELTA1 for use with the following graphic pattern magic write subroutine.

RELTA1 examines bits 1 and 0 within the X coordinate to determine the necessary magic shift of 0,1,2 or 3 pixels. The magic register value at entry can specify any of the other legal magic functions, however, a graphic pattern magic write subroutine following RELTA1 must support the specified magic functions. The magic register value, with its adjusted shift amount in bits 1 and 0, is output to the Magic Register (port 0CH) at the end of RELTA1.

RELTA1 also supports mirror image flopped screen coordinates. RELTA1 calculates the mirror image flopped X coordinate as

XFLOP = 319 - X
where, X = the normal (unflopped) X coordinate.

The game Gunfight used this flopped request for the cowboy on the right. The intent of this flopped option has limited application. But, you can flop a pattern anywhere on the screen once you understand how the magic flop is written. See the separate Bally Alley posting by MCM Design detailing the hi-res normal and flopped coordinate systems. The posting is entitled:

Low And High-Res Data Block Comparisons

RELTA1 also has a 5 NOP future provision for a more usable flopped request. MCM Design has a plan to develop and test this new flopped request idea.

The hi-res X coordinate (0-319 range) must be represented in binary using 2 bytes. This coordinate must be in the Z80 register DE at entry of RELTA1. MCM Design chose to use the hi-res screen scratchpad address 7FF7H to specify the Y coordinate for this subroutine. This Y coordinate is labeled as REGY. The Y coordinate (REGY) must be in (7FF7H) at entry of RELTA1.

End Of Posting
MCM Design
March 2020

CONVERT COORDINATES TO MAGIC ADDRESS

ENTER WITH: HL = PATTERN ADDRESS (SAVED IN THIS SUB)
DE = X COORDINATE
A = MR VALUE TO OUTPUT TO MAGIC REGISTER (PORT 08_H)
($7FF7_H$) = REGY = Y COORD
EXIT WITH: DE = MAGIC ADDRESS CONVERSION
ADJUSTED MR VALUE OUTPUTED TO MAGIC REGISTER IN THIS SUBROUTINE.

```
RELTA1  2C00H  E5          PUSH HL          SAVE PATTERN ADDRESS
               E6 78       AND 78H          CLEAR SHIFT AMOUNT IN MR VALUE
               6F          LD L,A                 <- BITS 1 AND 0
               7B          LD A,E           ISOLATE SHIFT AMOUT
               E6 03       AND 03H          COMBINE IT WITH MR VALUE
               B5          OR L                   AT ENTRY.
        2C08H  F5          PUSH AF          SAVE MR VALUE
               E6 40       AND 40H          IF FLOP BIT IS SET,
               C2 4D 2F    JP NZ,FREQ       JUMP TO PROCESS IT
                  <- DOG

RELTAC  2C0EH  D5          PUSH DE          SAVE X COORD
               3A F7 7F    LD A,(REGY)      HL=Y
        2C12H  6F          LD L,A
               26 00       LD H,0
               29          ADD HL,HL        HL=COMPUTED MAGIC ADR
               29
               29
        2C18H  29
               54          LD D,H
               5D          LD E,L
               29          ADD HL,HL
               29          ADD HL,HL
               19          ADD HL,DE
               D1          POP DE
               CB 1A       RR D
        2C21H  CB 1B       RR E
               CB 3B       SRL H
               16 00       LD E,A
               19          ADD HL,DE
        2C28H  EB          EX DE,HL         DE=MAGIC ADR
               F1          POP AF           A=MR VALUE
               E1          POP HL           HL=PATTERN ADR
               D3 0C       OUT (MAGIC),A    OUT MAGIC REQUEST
        2C2DH  C9          RET
```

This page intentionally left blank

PROCESS MR FLOP REQUEST FOR "CONVERT COORDINATE TO MAGIC ADDRESS", P.55

```
FREQ 2F4D_H 00 00
            00 00 00
```

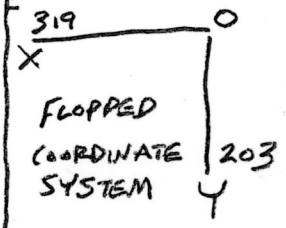
] FUTURE PROVISION FOR SPECIAL CASE FLOP PROCESSING (MR BIT 7=1) *

```
2F52  7B        LD A,E
      2F        CPL
      5F        LD E,A
      7A        LD A,D
      2F        CPL
      57        LD D,A
      13        INC DE
      21 3F 01  LD HL,319_D
      19        ADD HL,DE
      EB        EX DE,HL
2F5E_H C3 0E 2C JP RELTAC
```

] 2'S COMPLEMENT ON X COORD (NEGATE IT)

2'S COMPLEMENT NOTE YOU COULD DELETE THIS INSTRUCTION AND USE LDHL,320_D

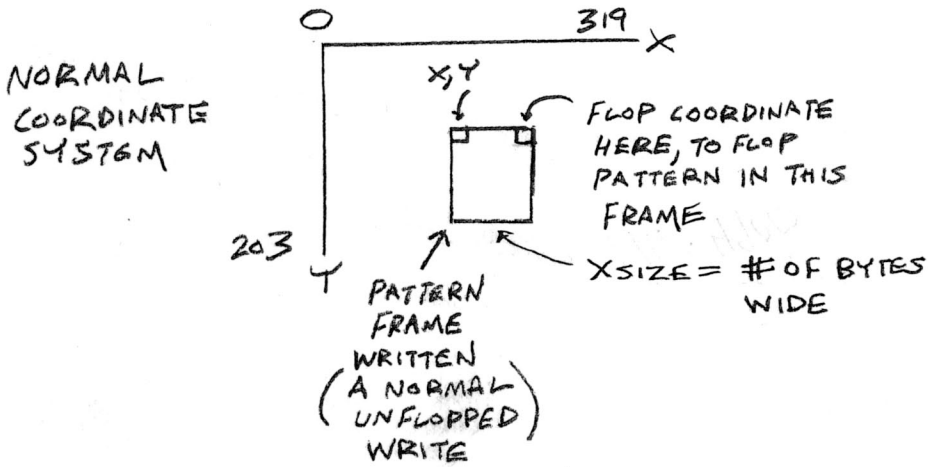
DE=XFLOP = 319-X



LIMITED APPLICATION

MCM DESIGN TEST IDEA (TEST IN HI-RES UPGRADED FISH DEMO)

* MCM DESIGN IDEA (UNTESTED AT THIS TIME, FEB 2020) SINCE BIT 7 IN MR IS NOT USED, USE BIT 7 IN MR VALUE AS A FLAG FOR A SPECIAL CASE FLOP, IE, FLOP A PATTERN WITHIN ITS NORMAL (UNFLOPPED) FRAME WRITE.



$XFLOP = 4(XSIZE) - 1$

↑ 4 PIXELS IN A BYTE

MCM DESIGN NOTE! THE EQUATION MAY NEED TO BE REFINED. SEE YOUR FLOP ADDENDUM TO TEST THIS IDEA.

IF THIS IDEA WORKS, YOU CAN MAKE A GRAPHIC PATTERN FACE RIGHT OR FACE LEFT WHILE MOVING AROUND THE SCREEN. NO 2ND GRAPHIC PATTERN REQUIRED FACING THE OTHER DIRECTION.

This page intentionally left blank

STANDARD HI-RES STACKED GRAPHIC PATTERN WRITE SUBROUTINES
For Use On A Modified Hi-Res Astrocade
Similar To The On-Board Low-Res Subroutines #30 thru #38
(copied from MCM Design's hi-res Multipage Test Demo, in the hand written code listing, pages 64-67)
Margins Left 0.9, Right 1.0

This posting is for a ML/AL programmer who has access to a modified hi-res Astrocade. Rather than create an applicable subroutine from scratch, this tested hi-res subroutine from MCM Design can be used as a reference doc.

Refer to the attached scanned hand written hi-res subroutines listing labeled as VWRITR, WRITR, WRITP, WRIT and WPATHR.

These stacked subroutines are usable for the magic writing of hi-res graphic patterns.

This multi-entry stacked subroutine is called directly. There is no user programmer interface (UPI) required to process the calling of any of these 5 subroutines. Each of the 5 entry points has a specific purpose. The 5 entries are labeled below.

VWRITR
WRITR
WRITP
WRIT
WPATHR

The Z80 CPU register entry requirements are specified at the beginning of each of the 5 entries. This hi-res version was created from the low-res version listed in the Nutting Manual Z80 Cross Assembler listing, pages 49–51.

So, documentation related to the low-res sub#30 thru #38 can be used as a guide for this hi-res version. Refer also to the Nutting Manual system description and MCM Design's "An In-Depth Look At..." tutorial series, both posted on the Bally Alley, for info related to the magic RAM and magic write functions.

General Description Of 5 Entry Points

Entry 1
VWRITR
Write Relative From Vector Block

This entry uses the X,Y coordinates and the Magic Register value from a vector block in screen RAM to write a hi-res graphic pattern.

Refer to the Bally Alley posting: LOW AND HIGH-RES DATA BLOCK COMPARISONS.
This posting diagrams the required hi-res vector block and coordinate system.

++

Entry 2
WRITR
Write Relative

A relative X and Y is added to the entry X,Y coordinates of a graphic pattern frame for the writing of animated patterns such as a moving arm (Gunfight). Set relative

X and Y = 0 if you are just writing a regular pattern (not an animated pattern).

++

Entry 3
WRITP
Write With Pattern Size

The pattern's X size and Y size are loaded into the Z80 CPU register BC.

++

Entry 4
WRIT
Write With X,Y Coordinates Conversion

The Z80 CPU registers DE and A entry requirements are the same as the above WRITP. The Z80 register HL must now point at the actual graphic pattern and register BC = YSIZE XSIZE. Note that the Y coordinate must be in the screen RAM scratchpad address 7FF7H. WRIT calls subroutine RELTA1 to convert the X,Y coordinates to their corresponding magic address. For details on the subroutine RELTA1, refer to the Bally Alley posting:

CONVERT HIGH-RES COORDINATES TO A MAGIC ADDRESS

++

Entry 5
WPATHR
Write Pattern In Hi-Res

This is where the normal, expand and flop pattern writes are located. There is no write routine here to support the magic rotate function. The Magic Register value (MR) in the Z80 CPU register A now flags which magic write routine is to be executed.

End Of Posting
MCM Design
March 2020

This page intentionally left blank

WRITE RELATIVE FROM VECTOR BLOCK (SIMILAR TO SUB#30) 64

ENTER WITH: IX = VECTOR BLOCK (PACKET) ADDRESS

HL = PATTERN ADDRESS -4 (POINTING AT RELATIVE X)

VECTOR PACKET IS 15 BYTES LONG (X_H AND ΔX_H ARE 2 BYTES LONG)

VWRITR $2DB7_H$	DD 7E 00	LD A,(IX)	A = MAGIC REGISTER VALUE
	DD 46 0D	LD B,(IX+$0D_H$)	B = Y_H
	DD 5E 07	LD E,(IX+7)	DE = X_H
$2DC0_H$	DD 56 08	LD D,(IX+8)	
	DD CB 01 F6	SET 6,(IX+1)	SET BLANK BIT

WRITE RELATIVE (SIMILAR TO SUB#32)

ENTER WITH: HL = PATTERN ADDRESS -4 (POINTING AT RELATIVE X)

DE = X COORDINATE (X_H IN VECTOR PACKET)

B = Y COORDINATE (Y_H)

A = MR VALUE TO OUTPUT TO MAGIC REGISTER PORT 05_H

WRITR $2DC7_H$	F5	PUSH AF	SAVE MR VALUE
	7E	LD A,(HL)	A = RELATIVE X
	23	INC HL	POINT HL AT RELATIVE Y
	83	ADD A,E	DE = X_{COORD} + RELATIVE X OR X_H
	5F	LD E,A	
	7A	LD A,D	
	CE 00	ADC A,0	
	57	LD D,A	
$2DD0_H$	7E	LD A,(HL)	A = RELATIVE Y
	23	INC HL	POINT HL AT XSIZE
	80	ADD A,B	A = Y_{COORD} + RELATIVE Y OR Y_H
	32 F7 7F	LD (REGY),A	REGY = Y + RELATIVE Y
$2DD6_H$	F1	POP AF	A = MR VALUE

WRITE WITH PATTERN SIZE (SIMILAR TO SUB #34)

ENTER WITH: HL = PATTERN ADDRESS - 2
(POINTING AT XSIZE)

DE = X_{COORD} + RELATIVE X

($7FF7_H$) = REGY = Y_{COORD} + RELATIVE Y

A = MR VALUE TO OUTPUT TO MAGIC REG ($PORT08_H$)

WRITP $2DD7_H$	4E	LD C,(HL)	C = XSIZE
	23	INC HL	POINT HL AT Y SIZE
	46	LD B,(HL)	B = YSIZE
	23	INC HL	POINT HL AT PATTERN

WRITE WITH COORDINATE CONVERSION (SIMILAR TO SUB #36)

WRIT $2DDB_H$ CD 002C CALL RELTA1

WRITE PATTERN (SIMILAR TO SUB #38)

ENTER WITH: HL = PATTERN ADDRESS
DE = MAGIC ADDRESS TO WRITE TO
C = XSIZE OF PATTERN (# OF BYTES WIDE)
B = YSIZE ↓ (# OF LINES HIGH)

ALSO MR VALUE MUST BE IN Z80 REGISTER A AT ENTRY (APPROPRIATE TO FLAG WRITE ROUTINE)

NOTE: MAGIC REGISTER VALUE MUST BE OUTPUT TO THE MAGIC REGISTER (PORT 08H) PRIOR TO CALLING THIS SUB.

WPATHR $2DDE_H$	CB 77	
$2DE0_H$	20 2C	
	CB 5F	
	20 11	
WPLOP →	AF	
(NORMAL PLOP, XOR, OR)	C5	
	D5	
	47	
	ED B0 ← DOG	
	12	
DOG →	D1	
	EB	
	0E 50	LD C, BYTEPL HI-RES 80D BYTES/LINE
$2DF1_H$	09	
	EB	
	C1	
	10 F1	
$2DF6_H$	C9	

REFERENCE NUTTING MANUAL
Z80/ROM CODE BREAKDOWN P. 49-50

```
WEXPD  2DF7H  EB
(WRITE        C5
EXPANDED)     E5
              41
              1A
              13
              77
              23
              77
       2E00H  23
              10 F8
              70
              23
              70
              E1
              0E 50          LD C, BYTEPL     HI-RES 80D BYTES/LINE
              09
              C1
              10 EB
              C9
              CB 5F
              20 16
       2E10H  AF
WFLOP         C5
(WRITE        D5
FLOPPED)      47             DOG
WFLOP1 2E16H  ED A0
              1B
              1B
              EA 16 2E       JP PE, WFLOP1
              12
              D1
              EB
       2E20H  0E 50
              09
              EB             BOT
              C1
              10 EC
       2E27H  C9
```

REFERENCE NUTTING MANUAL
Z80/ROM CODE BREAKDOWN P. 50

REFERENCE NUTTING MANUAL
Z80/ROM CODE BREAKDOWN P. 50-51

WXFLOP (WRITE WITH EXPANDED FLOP)

$2E28_H$	EB
	C5
	E5
	41
	1A
	13
	77
	2B
$2E30_H$	77
	2B ← B.Y
	10 F8
	70
	2B
	70
	E1
	0E 50
	09
	C1
	10 EB
$2E3E_H$	C9

REFERENCE NUTTING MANUAL
Z80/ROM CODE BREAKDOWN P.51

01

LD C, BYTEPL HI-RES 80_D BYTES/LINE

This page intentionally left blank

CUSTOM HIGH-RES MOVE(VECTOR)SUBROUTINE
For Use On A Modified Hi-Res Astrocade
Similar To The On-Board Low-Res Subroutine #62
(copied from MCM Design's hi-res Multipage Test Demo, in the hand written code listing pages 84-89)
Margins Left 0.9, Right 1.0

This posting is for a ML/AL programmer who has access to a modified hi-res Astrocade. Rather than create an applicable subroutine from scratch, this tested hi-res subroutine from MCM Design can be used as a reference doc.

Refer to the attached scanned hand written hi-res subroutine listing labeled as MVECT.

MVECT can be used to move a hi-res graphic pattern around the screen in the X and Y directions.

The intent of this particular custom hi-res subroutine is to move and bounce a critter around the entire screen.

A hi-res vector block is nearly identical to the low-res vector block. Since the hi-res X coordinate can vary from 0–319, it must be defined using 2 bytes. So, the XH and DeltaXH components must each be 2 bytes long in the vector block.

The vector block X,Y coordinates and Delta X components are expanded with double digit precision in the low component allowing fine tuning of a vectoring subroutine that is normally executed many times per second. Vectoring (motion) updates are accomplished by adding the delta high/low components to the corresponding high/low coordinate components.

The vectoring subroutine also utilizes a "time base" within the vector block, which can be varied for motion speed. This speed increment works like a loop counter. The delta X (or Y) is added to its respective X (or Y) coordinate "time base" times. For example, if the time base = 02, then the delta is added twice.

A change in motion direction is referred to as a reverse delta. A reverse delta actually performs a twos complement on the delta components. The negated deltaX (or Y) components are added to the X (or Y) components.

Since this custom vector subroutine bounces the critter around the entire screen, there is no need to check either of the 2 checks mask bytes in the vector block for a limit check request or a reverse delta request.

MVECT updates the X coordinate first followed by the Y coordinate update. Lower and upper limits for each coordinate are checked automatically. A reverse delta (direction) is automatically performed when any coordinate limit is reached.

This subroutine is called directly. There is no user programmable interface (UPI) required to process the calling of this subroutine.

The Z80 CPU register entry requirements are specified at the beginning of the subroutine listing.

This hi-res version was created from the low-res version listed in the Nutting Manual's Z80 Cross Assembler listing, pages 43–46.

Since MVECT is similar to the low-res sub#62, documentation related to sub#62 can be used as a guide to examine how this hi-res version works.

Additional Notes (Deviations From Sub#62)

1. The vector status active bit 7 is NOT checked since this vector is always active in the multi-pager Critter Move demo.

2. The vector time base is NOT checked to see if it is zero. This subroutine does not zero the time base.

3. In Note 4 on page 84 of this MVECT subroutine listing, bits 0 and 1 in the vector checks mask are NOT checked by this custom subroutine. Limit checks and reverse deltas are automatically performed in this subroutine.

More Notes

4. You can revise this routine to update only the X coordinate by inserting two Z80 RET (return) instructions as indicated on page 87 in this subroutine MVECT listing. To update only the Y coordinate, CALL 3887H (see page 87).

5. The hi-res vector block and limits table diagrams are shown on the Bally Alley posting titled as:

Low And High-Res Data Block Comparisons

6. The vector block in screen RAM must be initialized for the program application prior to calling the MVECT subroutine. Depending on the application, it might take fewer bytes to initialize/copy the vector block from ROM to RAM by using a Z80 LDIR (opcode ED BO) instruction.

7. For general application of vector subroutines and writing graphic patterns, consult the following reference guides on the Bally Alley:

MCM Design supporting software documentation related to hi-res graphics
Nutting Manual software and system descriptions, page 1–107.
"An In–Depth Look At..." tutorial series, a supplement to the NM, by MCM Design.

Reference also The Better Bally Book website

End Of Posting
MCM Design
March 2020

This page intentionally left blank

UPDATE X AND Y COORDINATES IN VECTOR PACKET

ENTER WITH: IX = VECTOR PACKET ADDRESS

HL = LIMITS TABLE ADDRESS
(POINTING TO LOWER X LIMIT)

INITIALIZE VECTOR PACKET IN RAM TO SUIT PROGRAM APPLICATION

NOTES

(1) THIS SUB IS SIMILAR TO ON-BOARD LOW-RES SUB #62 (REVISED FOR HI-RES)

(2) VECTOR STATUS (IX+1)
BIT 7, ACTIVE BIT IS NOT SET
BIT 5 1 = NO MOTION OCCURED
0 = MOTION OCCURED (X_H OR Y_H CHANGED)

(3) TIME BASE (IX+2) THE TIME BASE IS NOT ZEROED (INITIALIZE MOVE SPEED)

(4) X OR Y CHECKS MASK (IX+9) OR (IX+0E)
BIT 0 1 = DO LIMITS CHECK
0 = NO
BIT 1 1 = REVERSE DELTA AT LIMIT
0 = NO REVERSE DELTA AT LIMIT
BIT 3 1 = X (OR Y) LIMIT WAS ATTAINED
0 = LIMIT NOT ATTAINED

(5) AN UPI IS NOT UTILIZED WITH THIS SUBROUTINE. THERE IS NO PASSING OF DATA OR SETTING BITS WITHIN A CONTEXT BLOCK.

```
MVECT 32FD_H  DD 4E 02     LD C, (IX+2)      C = TIME BASE
      3300_H  DD CB 01 EE  SET 5, (IX+01)    USE BIT 5 IN VECTOR STATUS
                                             AS MOTION BIT
                                             1 = NO MOTION
                                             0 = MOTION OCCURED (X_H OR Y_H CHANGED)
      DOGS -> 11 03 00     LD DE, 3          ] POINT IX AT ΔX_L
              DD 19        ADD IX, DE        ]
```

UPDATE X COORDINATE

```
VECTX         E5           PUSH HL           SAVE LIMITS TABLE POINTER
              DD 56 02     LD D, (IX+2)      ] DE = ΔX_H
              DD 5E 01     LD E, (IX+1)      ]
      3310_H  DD 66 05     LD H, (IX+5)      ] HL = OLD X_H
              DD 6E 04     LD L, (IX+4)      ]
      3316_H  E5           PUSH HL           SAVE OLD X_H
```

```
                                                    B= TIME BASE
                                                     = LOOP CTR
3317H  41          LD B,C
       DD 4E 00    LD C,(IX)      IX → ΔXL
                                  C= ΔXL
       DD 7E 03    LD A,(IX+3)    A= XL
VCTX1  81          ADD A,C        A= XL + ΔXL        RESULT
       ED 5A       ADC HL,DE      HL= OLD XH + ΔXH + CARRY
                                      H      DE
3321H  10 FB       DJNZ VCTX1
       DD 77 03    LD (IX+3),A    LOAD UPDATED (NEW) XL
                                  INTO PACKET NOW
       C1          POP BC         BC= OLD XH
       5D  ← DOG   LD E,L
       54          LD D,H         DE = UPDATED (NEW) XH
       A7  ← DOG   AND A          CARRY = 0
       ED 42       SBC HL,BC      HL= NEW XH - OLD XH - CARRY
                                     HL      BC
       28 04       JRZ, VTX1A
       DD CB FE AE RES 5,(IX-2)   RESET BIT 5 IN
                                  VECTOR STATUS
                                  0= MOTION OCCURED

VTX1A  3332H  E1   POP HL         POINT HL AT X LOWER LIMIT

   GET X LOWER LIMIT
       4E          LD C,(HL)
       23          INC HL
       46          LD B,(HL)      BC= X LOWER LIMIT
       23          INC HL         POINT HL AT X UPPER LIMIT
       C5          PUSH BC        SAVE LOWER LIMIT

   CHECK IF X REACHED LOWER LIMIT
       01 6F 01    LD BC, 367D    ← -48  0 → 319 +48
                                     (SIMILAR TO LOW-RES
                                      VECTOR ROUTINE)
       CD DE 33    CALL LCHK
       C1          POP BC         BC= LOWER LIMIT
       30 0E       JR NC, VCTX2
3341H  CD DE 33    CALL LCHK
3344H  38 09       JRC, VCTX2
```

ADD DELTA TO COORDINATE "TIME BASE TIMES"

IF MOTION OCCURED (XH HAS CHANGED), RESET BIT 5 IN VECTOR STATUS

HANDLE < 0 CASE. ANY X > 367D IS CONSIDERED NEGATIVE

IF NEW XH < LOWER LIMIT JMP TO VCTX2

```
                    GET X UPPER LIMIT
3346H 4E            LD C,(HL)
      23            INC HL
      46            LD B,(HL)

                    CHECK X UPPER LIMIT                    ] IF NEW XH < UPPER LIMIT JMP TO VCTX3
      CD DE33       CALL LCHK
      38 2E         JRC,VCTX3
      2B ← BOY      DEC HL
      23            INC HL
VCTX2
      23            INC HL                                 HL POINTS AT Y LOWER LIMIT
3350H 23            INC HL
      DD 71 04      LD (IX+4),C                            ] XH IN VECTOR PACKET = X LIMIT
      DD 70 05      LD (IX+5),B                            ]
      DD 36 03 00   LD (IX+3),0                            XL IN VECTOR PACKET = 0
      DD CB 06 DE   SET 3,(IX+6)  ← BOY                    SET X LIMIT ATTAINED BIT (IN VP'S X CHECKS MASK)

                    PROCEED TO REVERSE THE DELTA (2'S COMPLEMENT)
      E5            PUSH HL                                SAVE LIMIT POINTER (POINTS AT Y LOWER LIMIT)
3360H DDE5          PUSH IX                                ] POINT DE AT ΔXL
DOG → D1            POP DE                                 ]
      1A            LD A,(DE)   A=ΔXL                      ] 2'S COMPLEMENT ΔXL
      2F            CPL                                    ]
      C6 01         ADD A,1                                ]
      12            LD (DE),A                              ]
      13            INC DE                                 ] HL = COMPLEMENT ΔXH
      1A            LD A,(DE)                              ]
      2F            CPL                                    ]
      6F            LD L,A                                 ]
      13            INC DE                                 ]
      1A            LD A,(DE)                              ]
      2F            CPL                                    ]
336FH 67            LD H,A                                 ]
```

```
3370H 01 00 00      LD BC,0
      ED 4A         ADC HL,BC     HL = HL + BC + CARRY
      EB  <- BOY    EX DE,HL      DE = ΔXH + CARRY
                                  HL = POINTS TO HIGH ORDER XH
UPDATE X ONLY,
INSERT OPTIONAL
RETURE HERE
      72            LD (HL),D  ]
      2B            DEC HL     ] LOAD 2'S COMPLEMENT OF XH INTO VECTOR PACKET
      73            LD (HL),E  ]
      E1            POP HL        HL POINTS AT Y LOWER LIMIT
   -> 337AH 18 0B   JR VECTY      JUMP TO VECTY

NEW XH WHERE:  LOWER LIMIT < NEW XH < UPPER LIMIT
LOAD NEW XH IN VECTOR PACKET
                    INC HL        POINT HL NOW AT Y LOWER LIMIT
VCTX3         23 <- 2 DOGS
              DD 73 04  LD (IX+4),E  ]
UPDATE        3380H DD 72 05 LD (IX+5),D  ] SET XH IN VECTOR PACKET TO UPDATED (NEW) XH
X ONLY,       DD CB 06 9E   RES 3,(IX+6)   CLEAR X LIMIT ATTAINED BIT
INSERT                                     (IN VP X CHECKS MASK)
OPTIONAL
RETURN HERE ->
```

UPDATE Y COORDINATE

ENTER WITH: IX POINTING TO ΔX_L IN VECTOR PACKET
HL = LOWER Y LIMIT (IN LIMITS TABLE)

```
VECTY 3387H DD 4E FF   LD C,(IX-1)     C = TIME BASE
      11 07 00         LD DE,7         ] POINT IX AT ΔYL
      DD 19            ADD IX,DE       ]
2 DOGS -> E5           PUSH HL         SAVE Y LOWER LIMIT POINTER
      3390H DD 56 01   LD D,(IX+1)     ]
      DD 5E 00         LD E,(IX+0)     ] DE = ΔY (ΔYH, ΔYL)
      DD 66 03         LD H,(IX+3)     ]
      DD 6E 02         LD L,(IX+2)     ] HL = YH YL
      7C               LD A,H          A = YH, SAVE YH
      41               LD B,C          B = TIME BASE CTR
VECT1 19               ADD HL,DE       ] ADD DELTA TO COORDINATE "TIME BASE TIMES"
      10 FD <- DOG     DJNZ VECT1      ]
      33A1H BOY -> BC  CP H            ] IF NO CHANGE IN Y, JMP TO VCT1A
      28 04            JR Z,VCT1A      ]
      33A4H DD CB F7 AE RES 5,(IX-9)   MOTION OCCURED
                                       RESET MOTION BITS IN VECTOR STATUS
```

```
                    GET Y LOWER LIMIT
VCT1A 33A8H 7C      LD A,H          A=YH
            E3      EX (SP),HL      TOP OF STACK = Y COORDINATE
                                    HL = Y LIMITS POINTER
            46      LD B,(HL)       B = Y LOWER LIMIT
            23      INC HL          POINT HL AT Y UPPER LIMIT

                    CHECK IF Y REACHED LOWER LIMIT
            FE FA   CP 250D         -48 0 → 203D +48   ] HANDLE <0 CASE.
            30 07   JR NC,VECT2                        ] ANY X>250 IS
                                                       ] CONSIDERED NEG
            B8      CP B            ] IF YH < LOWER LIMIT,
33B0H       38 04   JR C,VECT2      ] JMP TO VECT2 (LIMIT WAS REACHED)
(BOY → B8)

                    GET Y UPPER LIMIT, CHECK IF IT REACHED UPPER LIMIT
            46      LD B,(HL)       B = Y UPPER LIMIT   ] IF YH < UPPER LIMIT,
                                                        ] JMP VECT3
            B8      CP B
(BOY → B8)
            38 1A   JR C,VECT3

                    LOAD UPPER LIMIT INTO VECTOR PACKET
VECT2       DD 70 03    LD (IX+3),B     YH IN VP = NEW YH
            DD 36 02 00 LD (IX+2),0     YL ↓ = 0
            DD CB 04 DE SET 3,(IX+4)    SET Y LIMIT ATTAINED BIT 3 IN
                                        Y CHECKS MASK.
33C2H       F1          POP AF          CLEAN UP STACK

                    LIMIT ATTAINED, REVERSE THE DELTA (ΔYH ΔYL), DE = ΔYH ΔYL
            7A      LD A,D      ]
            2F      CPL         ] D = ΔȲH
            57      LD D,A      ]
            7B      LD A,E      ]
            2F      CPL         ] E = ΔȲL
            5F      LD E,A      ]
            13      INC DE                  ] REVERSE DELTA
                                            ] (2'S COMPLEMENT)
            DD 73 00    LD (IX+0),E     ]
            DD 72 01    LD (IX+1),D     ] LOAD REVERSE DELTA
                                        ] IN VECTOR PACKET
            C9          RET
33D0H
VECT3       E3          EX (SP),HL      HL = YH YL = NEW COORDINATE
            DD 75 02    LD (IX+2),L     ] LOAD COORDINATE
            DD 74 03    LD (IX+3),H     ] IN VECTOR PACKET
            E1          POP HL          CLEAN UP STACK
            DD CB 04 9E RES 3,(IX+4)    RESET LIMIT ATTAINED BIT 3
                                        IN Y CHECKS MASK
33DDH C9    RET
```

LIMIT CHECK

ENTER WITH: DE = UPDATED (NEW) X_H
BC = X LIMIT (LOWER OF UPPER)

LCHK	$33DE_H$	E5	PUSH HL	SAVE LIMIT POINTER
		6B ← BY	LD L, E	HL = UPDATED (NEW) X_H
			LD H, D	
	$33E0_H$	62		
		A7	AND A	CARRY = 0
		ED 42	SBC HL, BC	HL = NEW X_H (HL) − LIMIT (BC) − CARRY
		E1	POP HL	HL = LIMIT POINTER
	$33E5_H$	C9	RET	

COMPARE NEW X_H WITH LIMIT

This page intentionally left blank

CUSTOM HI-RES MULTI-PAGER GRAPHIC PATTERN WRITE SUBROUTINE
For Use On A Modified Hi-Res Astrocade
Utilizing MCM Design's Hi-Res Static Screen RAM Multi-pager
Similar To The On-Board Low-Res Subroutines #30 thru #38
(copied from MCM Design's hi-res Multipage Test Demo, in the hand written code listing, pages 96-98)
Margins Left 0.9, Right 1.0

This posting is for a ML/AL programmer who has access to a modified hi-res Astrocade with MCM Design's multi-pager. Rather than create an applicable subroutine from scratch, this tested hi-res subroutine can be used as a reference doc.

SUBROUTINE PROGRAM NOTE
This subroutine was used as a test in MCM Design's Multi-pager Test/Demonstration Demo to write a critter in each of the 8 pages of screen RAM while a main program was being executed in the page 7 scratchpad area. So, the Z80 stack area pointer (register SP) was being switched (pointed) to the page that the critter was written to. After the critter write, the stack area was then pointed back to page 7 to continue execution of the main program. The main program is only 98 bytes, but the program calls 5 subroutines in cartridge ROM 2000–3FFFH.

Refer to the attached scanned hand written listing with stacked hi-res subroutines labeled as CVWRIT, CWRITR, CWRITP, CWRIT and CMWRIT.

These stacked subroutines are usable for the magic writing of the specific hi-res graphic pattern write subroutine labeled as CWRT.

This multi-entry stacked subroutine is called directly. There is no user programmer interface (UPI) required to process the calling of any of these 5 subroutines. Each of the 5 entry points has a specific purpose. The 5 entries are labeled below.

CVWRIT
CWRITR
CWRITP
CWRIT
CMWRIT

The Z80 CPU register entry requirements are only specified at the beginning of CVWRIT. This hi-res version was created from the low-res version listed in the Nutting Manual Z80/ROM Cross Assembler listing, pages 49–51.

So, documentation related to the low-res sub#30 thru #38 can be used as a guide for this hi-res version. Refer also to the Nutting Manual system description and MCM Design's "An In-Depth Look At..." series, both posted on the Bally Alley, for info related to the magic RAM and magic write functions.

General Description Of 5 Entry Points

Entry 1
CVWRIT
Write Relative From Vector Block

This entry uses the X,Y coordinates and the Magic Register value from a vector block in screen RAM to write a hi-res graphic pattern.

Refer to the Bally Alley posting: LOW AND HIGH-RES DATA BLOCK COMPARISONS.
This posting diagrams the required hi-res vector block and coordinate system.

++

Entry 2
CWRITR
Write Relative

A relative X and Y is added to the entry X,Y coordinates of a graphic pattern frame for the writing of animated patterns such as a moving arm (Gunfight). Set relative X and Y = 0 if you are just writing a regular pattern (not an animated pattern).

++

Entry 3
CWRITP
Write With Pattern Size

The pattern's X size and Y size are loaded into the Z80 CPU register BC.

++

Entry 4
CWRIT
Write With X,Y Coordinates Conversion

The Z80 CPU registers DE and A entry requirements are the same as the above CWRITP. The Z80 register HL must now point at the actual graphic pattern and register BC = YSIZE XSIZE. Note that the Y coordinate must be in the screen RAM scratchpad address 7FF7H. CWRIT calls subroutine RELTA1 to convert the X,Y coordinates to their corresponding magic address. For details on the subroutine RELTA1, refer to the Bally Alley posting:

CONVERT HIGH-RES COORDINATES TO A MAGIC ADDRESS

++

Entry 5
CMWRIT
Write Pattern In Hi-Res

This is where only a custom but normal plop pattern write is located. There are no write routines here to support the magic expand, flop or rotate functions. See the above SUBROUTINE PROGRAM NOTE.

End Of Posting
MCM Design
March 2020

This page intentionally left blank

CUSTOM MULTI-PAGER WRITE ROUTINE 96.

USED TO PLOP WRITE A CRITTER IN ANY OF 8 PAGES WHILE THE Z80 R/Ws ARE WORKING THE STACK AND VARIABLE(S) WITHIN SCREEN RAM PAGE 7 FOR THE MAIN PROGRAM.

WRITE THE CRITTER USING VECTOR BLOCK (SIMILAR TO LOW-RES SUB#30)
VECTOR BLOCK IS 15 BYTES USING 2 BYTES EACH FOR X_H AND ΔX_H.
ENTER WITH: IX = VECTOR BLOCK (PACKET) ADDRESS
HL = PATTERN ADDRESS - 4 (POINTING AT RELATIVE X)
($7FF9_H$) = PAGE NUMBER 0-7 TO WRITE (VIEW) CRITTER.

NOTE: ($7FF7_H$) = REGY = Y COORD SAVED FOR WRITING CRITTER.

```
CVWRIT  3583H  DD 7E 00     LD A,(IX)        A = MAGIC REGISTER VALUE
               DD 46 0D     LD B,(IX+0DH)    B = YH
               DD 5E 07     LD E,(IX+7)      DE = XH
               DD 56 08     LD D,(IX+8)
               DD CB 01 F6  SET 6,(IX+1)     SET BLANK BIT
```

WRITE RELATIVE (SIMILAR TO LOW-RES SUB#32)

```
CWRITR  3593H  F5           PUSH AF          SAVE MR VALUE
               7E           LD A,(HL)        A = RELATIVE X
               23           INC HL           POINT HL AT RELATIVE Y
               83           ADD A,E
               5F           LD E,A           DE = XH
               7A           LD A,D                OR
               CE 00        ADC A,0                = XCOORD + RELATIVE X
               57           LD D,A
               7E           LD A,(HL)        A = RELATIVE Y
               23           INC HL           POINT HL AT XSIZE
               80           ADD A,B          A = YH OR YCOORD + RELATIVE Y
               32 F7 7F     LD (REGY),A      SAVE Y = REGY
        35A2H  F1           POP AF           A = MR VALUE
```

WRITE WITH PATTERN SIZE (SIMILAR TO LOW-RES SUB#34)

```
CWRITP         4E           LD C,(HL)        C = XSIZE
        35A4   23           INC HL           POINT HL AT Y SIZE
```

$35A5_H$ 46 LD B,(HL) B=YSIZE
23 INC HL POINT HL AT PATTERN TO WRITE

WRITE WITH COORDINATES CONVERSION (SIMILAR TO LOW-RES SUB#36)

CWRIT $35A7_H$ CD002C CALL RELTA1

WRITE THE PATTERN

THIS SUBROUTINE IS AN EXAMPLE OF HOW TO UTILIZE McM DESIGN'S MULTI-PAGER HARDWARE TO MAGIC WRITE A PATTERN TO ONE PAGE WHILE THE Z80 IS (NORMALLY) WORKING THE STACK AND SCRATCHPAD VARIABLE(S) IN ANOTHER PAGE SPECIFIED BY THE MULTI-PAGER OUTPUT PORT 75_H. THIS SUBROUTINE IS SIMILAR TO THE LOW-RES NORMAL MAGIC WRITE SUB MWRT. ENTER THIS SUBROUTINE WITH THE Z80 R/W'S POINTING TO PAGE 7 AND WITH ($7FF9_H$) = THE PAGE NUMBER (0-7) TO WRITE CRITTER PATTERN INTO,

POINT THE Z80 TO R/W A PAGE USING OUTPUT PORT 75_H = 0XXX 0XXX
(0XXX: Z80 WRITE; 0XXX: Z80 READ, READ MAGIC HARDWARE FOR XOR, OR WRITE)

WHERE XXX = THE PAGE NUMBER (0-7).

POINTING THE Z80 IN THIS MANNER ALLOWS THE Z80 TO WORK IN THIS PAGE, THE STACK AND ANY SCRATCHPAD VARIABLES, PLUS ALLOWS THE MAGIC HARDWARE TO READ RAM BYTES IN THE PAGE FOR MAGIC XOR, OR LOGICAL WRITES.

POINT Z80 R/W AT PAGE CRITTER IS BEING WRITTEN INTO

CMWRIT $35AA_H$ C5 PUSH BC SAVE YSIZE, XSIZE
3AF97F LD A,($7FF9_H$) A=PAGE NUMBER (0-7) TO WRITE PATTERN INTO
47 LD B,A
CB00 RLC B
$35B1_H$ CB00 RLC B
CB00 RLC B
CB00 RLC B
(SHIFT LEFT PAGE NUMBER INTO BITS 4-7 IN B)
80 ADD A,B A=0XXX 0XXX = PAGE R/W
C1 POP BC BC=YSIZE, XSIZE
FE77 CP 77
2809 JR Z,CWRT
(IS THIS R/W PAGE 7? IF SO, JUMP AHEAD. MAIN PROGRAM STACK IS ALREADY SETUP.)
ED73FA7F LD ($7FFA_H$),SP SAVE PAGE 7 STACK POINTER IN PAGE 7 SCRATCHPAD.
$35C1_H$ D375 OUT (75_H),A POINT Z80 R/W AT THIS PAGE (0-6)
$35C3_H$ 31C07F LD SP,$7FC0_H$ INITIALIZE THE STACK POINTER IN THIS PAGE (FOR CRITTER WRITE, STACK PUSH/POP).

(PLOP) WRITE CRITTER INTO PAGE (SIMILAR TO LOW-RES NORMAL PLOP CRIT*)

CWRT	$35C6_H$	AF	XOR AF	SIMILAR TO LOW-RES MWRT. SEE NUTTING MANUAL Z80/ROM CODE BREAKDOWN, PAGE 50.
		C5	PUSH BC	
		D5	PUSH DE	
		47	LD B,A	
		ED B0	LDIR	
		12	LD (DE),A	
		D1	POP DE	
		EB	EX DE,HL	
		0E 50	LD C,80_D	HI-RES 80 BYTES/LINE
	$35D1_H$	09	ADD HL,BC	
		EB	EX DE,HL	
		C1	POP BC	
		10 F1	DJNZ CWRT	
		3A F9 7F	LD A,(7FF9) A=PAGE #	IS CRITTER BEING WRITTEN IN PAGE 7? IF SO, RETURN NOW. NO STACK RESTORATION REQ'D.
		FE 07	CP 7	
		C8	RET Z	
		3E 77	LD A,77_H	RESTORE Z80 R/W TO POINT BACK TO PAGE 7
		D3 75	OUT (75_H),A	
	$35E0_H$	ED 7B FA 7F	LD SP,($7FFA_H$)	RESTORE PAGE 7 STACK POINTER
	$35E4_H$	C9	RET	

* SEE NUTTING MANUAL Z80/ROM CODE LISTING, PAGE 50