IS–BASIC EXTENSIONS INSTRUCTION MANUAL.

IS–BASIC EXTENSIONS
Written by Andrew Richards.

Loading instructions:

The Basic Extensions have been divided into five parts:–

BASX_G.REL	(General functions)
BASX_C.REL	(Character functions)
BASX_M.REL	(Menu functions)
BASX_S.REL	(Sprite functions)
BASX_A.REL	(Assembler functions)

Included on the tape is an example Basic program:– BASX.BAS

Dividing them in sections, gives you the option of deciding how many functions you want present at one time.

To load an extension, enter Basic and then type LOAD "XXXXX", where 'XXXXX' is the name of the file to be loaded.

To load the example Basic program, load BASX_G.REL, BASX_C.REL, BASX_M.REL, BASX_S.REL in that order then load and run BASX.BAS.

IMPORTANT NOTE:

When you wish to load or run a program written using an IS–BASIC EXTENSION, you must first load the extensions in the same order as they were originally loaded when the program was written.

NOTE: Items enclosed in round brackets must be included. Items in square brackets are optional.

CHAN= CHANNEL
addr= ADDRESS

COMMAND/FUNCTION summary of BASX; A,C,G,M,S

BASX_C – Commands

CHRLET A$=B$
Copies the definition of character 'B$' to character 'A$'. eg. If A$="£" and B$="Z" then CHRLET A$=B$ would redefine '£' as a 'Z'. NOTE ALSO: CHRLET "£"=" Z" would give the same result.

CHRINV A$
Inverts the character in 'A$'.

CHRFLP A$
Flips the character in 'A$' upside–down.

CHRROR A$
Rotates the character in 'A$' right 90 degrees.

CHRROL A$
Rotates the character in 'A$' left 90 degrees.

STRINV, STRFLP, STRROR, STRROL
As CHRINV, CHRFLP etc, except that strings of more than one character may be given.

CHRDEF A$

Marks the start of the definition of the character in 'A$'.

CHRLIN LINE$

Defines a line of the character given by 'CHRDEF'. 'LINE$' must be 8 characters long and there are nine of them. Each character of line$ represents a bit in the definition of the character in 'A$'. A space is for a reset bit, a '.' is for a bit that should be left unchanged and any other character to set the bit. eg.

```
CHRDEF "\"
CHRLIN "        "
CHRLIN "XX....XX"
CHRLIN "        "
CHRLIN "........"
CHRLIN ". . . . "  etc nine times.
```

BASX_G – Commands

DOKE A,B

Stores 16 bit integer 'B' at address 'A' and 'A'+1. (Low byte first)

SDOKE S,A,B

As above except stores B, at address 'A' in segment 'S'.

CR [(CHAN)]

Prints a carriage return.

LF [(CHAN)]

Prints a line feed.

CLS [(CHAN)]

Clears channel. Sends an ASCII code 26 to specified channel.Default channel for 'CR', 'LF' and 'CLS' is 0. eg.

CLS(101) to clear a standard graphic screen.

VDU [£CHAN,] [8–bit number,] [16–bit number,] [string,] [&Hhexnum]

Sends a specified list of characters to a given channel. This can be an 8–bit, 16–bit or hexidecimal number or a character string. Default channel is £101. eg.

VDU £101,27,"s",27,"A",500;500,"Hello there!".

The '27' is the escape code, the small 's' will turn the beam off and the capital 'A' followed by two numbers 'X' and 'Y' move the beam to the position 'X' and 'Y' and print "Hello there!

FIND var

Finds the first occurence of 'var' and lists the line it is on.

FNEXT

Finds the next occurence of 'var' after the last FIND/FNEXT

DEFAULT device–string.

Sets the save/load default device name. No colon should be used. eg.

DEFAULT "TAPE" or if A$="TAPE" then DEFAULT A$

FILE [CHAN;] PTR=number

Sets the file pointer of the specified channel. Default channel is 106. This command will enable you to skip over a givennumber of characters in a file (tape or disk) and then set the file pointer to read the very next one. eg.

FILE £10;PTR=20

If a file consists of the characters 'ABCDEFGHIJKLM', and FILE PTR=10 is entered, then the next character to be read using a 'GET A$' command would read 'K'. NOTE: The 'FILE' command can only be used with disks.

VPOKE addr,8–bit value

Stores a byte at a specified video address which would first have been found by using 'VADDR'.

BASX_G – Functions

VADDR1 [(CHAN)]

Returns the address of the attribute data for the channel. Default channel is 101.

VADDR2 [(CHAN)]

Returns the address of the pixel data for the channel. Default channel is 101. NOTE: 'VADDR2' Would give the pixel at the top left of the channel.

CURX [(CHAN)]

Returns the x co–ordinate of the cursor on a video channel. Default channel is 102.

CURY [(CHAN)

Same again but for the y co–ordinate. NOTE: 'CURX' and 'CURY' will only work with text channels.

BEAMX [(CHAN)]

Returns the x co—ordinate of the beam on a video channel.

BEAMY [(CHAN)]

Same again but for y co—ordinate. NOTE: 'BEAMX' and 'BEAMY' will only work with graphics channels.

IF (condition,(if true),(if false) Returns the value of the first argument if 'condition' is true and the second if false. eg, The example PRINT IF (A>0,(B),(C)) will print the value of 'B' if 'A' is greater than 0 and if not, print the value of 'C'.

STRING$(length,string)

Returns a string of length 'length' made up of repeated occurences of 'string'. eg.

Print STRING$(10,"ABC") will print "CABCABCABC".

EOF [(CHAN)]

Returns the status of 'CHAN'. Defualt channel is the keyboard channel (105). This can be used to test if a key has been pressed without taking the value out of the keyboard buffer.

GET [(CHAN)]

Reads a byte from a channel and returns its ASCII value. Default channel is 105. Can be used to wait for a key to be pressed.

GET$ [(CHAN)]

As 'GET', except that 'GET$' returns the character as a string.

DEEK(addr)
Returns a 2–byte value at address 'addr' and 'addr'+1.

SDEEK(seg,addr)
As above but looks in segment 'seg'.

VPEEK(addr)
Returns a byte at video address 'addr'.

OPT(var–num)
Reads an exos variable and returns its value. Similar to 'ASK'.

BASX_M – Commands

MENU x–pos,width,title$,option$ [,option$]..
Draws a menu on the screen. 'title$' will be printed on the status line and the options will come underneath . All writing will be in ink 3. The menu will only work in 'TEXT 40' pages. The number of options which may be given will depend on the width of the menu opened as the command tends use a large amount of memory. A value of 11 for 'x–pos' will give column 1 on the screen. It is suggested that a value of 16 or greater is used as less than this will cause 'title$' to be truncated at the begining.

CLR_MENU
Restores the screen to how it was before 'MENU' was executed, assuming that no printing or scrolls have been done since.

MENU Lets the user select an option from the menu given by the 'MENU' command and returns its number. The top function is number 1, the second number 2 and so on. If 'ESC' is pressed, then 'MENU' will return –1.

Options are selected from the menu with the aid of the joystick. Move the joystick up or down until the desired selection is shown in inverse, then pressing the 'ENTER' key will return the number of that choice. You are advised to set colours 2&3 of the text page to something readable before executing 'MENU'. eg.

```
100 MENU 25,12,"Run again?","Yes","No"
110 SET £102:PALETTE BLACK,GREEN,BLACK,
WHITE
120 DO
130   OPT=MENU
140 LOOP WHILE OPT=-1
150 IF OPT=1 THEN
160   RUN
170 ELSE
180   CLR_MENU
190   END
200 END IF
```

The program above will display a menu with the title 'Play again?'. There are then two options 'Yes' and 'No'. If 'Yes' is selected the program will run again and display the same menu. If 'No', the program will then stop.

SPRX num

Sets the x co–ordinate of the sprite to be defined or printed. On the standard graphics page, the value for the x co–ordinate should be from 0 to 307. The sprite will only be printed to an accuracy of 4 pixels (one byte). This is not as bad as it sounds as many games do the same. (Sorcery for example).

SPRY num

Same as 'SPRX' but for the y–co–ordinate. The value is in pixels, and for the standard grahics page this will be 0 to 164. A zero value for 'SPRX' and 'SPRY' is the top left hand side of the screen.

SPRCHAN

Sets the graphics channel to display the sprites. Default channel is 101.

SPRNUM num

Sets the sprite number to be controlled by 'SPRDEF', 'MVLT','USESTICK' etc. The sprite's are 16 by 16 pixels on a hires page. Sprites on attribute and lores pages are twice the width. Upto 16 sprites can be defined and manipulated. They are numbered from 0 to 15.

SPDX num

Sets the distance to move in the x–direction for 'MVSTICK','MVRT' and 'MVLT'. Default value is 4.

SPDY num

Same as 'SPDX' but sets distance in the y–direction.

MVRT

Moves the present sprite right by 'SPDX' pixels. Uses XOR to print the and delete the sprite. The command assumes that the sprite has been printed using with XOR.

MVLT

Same as 'MVRT' but moves sprite left.

MVUP

Moves sprite up.

MVDN

Moves sprite down.

USESTICK joystick–number

Sets the joystick to be used with 'MVSTICK'.

MVSTICK

Moves the sprite given by 'SPRNUM' in the direction indicated by the joystick by the amount given by 'SPDX' and 'SPDY'.

PTBL

Prints the sprite given by 'SPRNUM' at the position given by 'SPRX' and 'SPRY' on the channel given by 'SPRCHAN'.

PTXOR

Prints sprite by XORing it with the background.

PTAND

Prints sprite by using AND.

PTOR

Prints sprite by using OR.

SPRDEF

Defines the sprite given by 'SPRNUM' by copying the screen at position 'SPRX' and 'SPRY' into the sprite definition for 'SPRNUM'. To create a sprite, first draw the shape on the screen using 'PLOT' etc, then set 'SPRX' and 'SPRY' to the top right of the sprite which has been drawn, then use 'SPRDEF' to define the sprite.

SPRINV

Inverts the sprite given by 'SPRNUM'. (This is a colour inversion).

BASX_S – Functions

FRAME

Waits for the video frame flyback. This is already used by 'MVLT', 'MVRT' etc. This function will stop the flickering sometimes caused when a user Line Parameter Table has been set up and pages are being constantly displayed.

SPRX

Returns the x–position of the sprite.

SPRY

As above but for the y–position.

SPRADDR

Returns the start address of the sprite definition. This can be used to peek and poke with a sprites bit–map.

SPRHIT

Takes the part of the screen pointed to by 'SPRX' and 'SPRY' and 'ORs' it together with the sprite given by 'SPRNUM'. This can be used for collision detection. SPRHIT is not a true sprite collision Detection command so a little experimentation is needed to use it.

Here is an example program to move the sprite by the computer and the joystick.

```
100 PROGRAM "SPRITE.BAS"
110 ! Please load BASX_S.REL first.
120 GRAPHICS HIRES 4
130 SET COLOUR 1,MAGENTA
140 PLOT 32,688,ELLIPSE 23,23,PAINT
150 SPRX 0:SPRY 0 ! Top left of screen.
160 SPRNUM 0
170 SPRDEF  ! Define sprite 0
180 SPDX 4:SPDY 4
190 CLEAR GRAPHICS
200 !
210 ! Move sprite from left to right.
220 !
230 FOR REPEAT=1 TO 3
240   SPRX 0:SPRY 90 ! Middle of left side.
250   PTBL ! Print sprite.
260   DO
270     MVRT
280   LOOP UNTIL SPRX>300
290   WAIT 1
300   PTXOR  ! Remove sprite from the screen.
310 NEXT REPEAT
320 !
```

```
330 ! Move sprite from top left to right bottom of
screen.
340 !
350 FOR REPEAT=1 TO 3
360   SPRX 0:SPRY 0 ! Top left of screen.
370   PTBL
380   SPDY 2
390   DO
400     MVRT
410     MVDN
420   LOOP UNTIL SPRX>300
430   WAIT 1
440   PTXOR
450 NEXT REPEAT
460 !
470 ! Move the sprite using the joystick.
480 !
490 PRINT "Now use the internal joystick to move
the sprite."
500 PRINT "Press <ENTER> to finish."
510 USESTICK 0 ! Use the internal joystick.
520 SPRX 150:SPRY 90
530 SPDX 4
540 PTBL
550 DO
560   MVSTICK
570   LET EXIT$=INKEY$
580 LOOP UNTIL EXIT$=CHR$(13)
590 PTXOR
600 CLEAR TEXT
610 END
```

BASX_A – Commands.

KEY:
r = Register
rr = Register pair
n = 8 bit value
nn = 16 bit value

BASX_A – Mnemonics	Standard Z80 – Mnemonics
LDR A,B	LD A,B
LDN A,n	LD A,n
LDA nn	LD A,(nn)
TFR r,r	LD r,r
LDX rr,nn	LD rr,(nn)
STA nn	LD (nn),A
ADD r	ADD A,r
SUB r	SUB r
ADC r	ADC A,r
SBC r	SBC A,r
XOR r	XOR r
IOR r	OR r
AND r	AND r
CMP r	CP r
LXN rr,nn	LD rr,nn
LXA rr,nn	LD rr,(nn)
STX rr,nn	LD (nn),rr
ADX rr	ADD HL,rr
ACX rr	ADC HL,rr
SCX rr	SBC HL,rr
INC r	INC r
DEC r	DEC r
INX rr	INC rr

DEX rr	DEC rr
PSH rr	PUSH rr
POP rr	POP rr
RST n	RST n
JSR nn	CALL nn
CEQ nn	CALL Z,nn
CNE	CALL NZ,nn
CCS	CALL C,nn
CCC	CALL NC,nn
CPL	CALL P,nn
CMI	CALL M,nn
CVS	CALL PO,nn
CVC	CALL PE,nn
JMP nn	JP nn
JEQ nn	JP Z,nn
JNE nn	JP NZ,nn
JCS nn	JP C,nn
JCC nn	JP NC,nn
JPL nn	JP P,nn
JMI nn	JP M,nn
JVS nn	JP PO,nn
JVC nn	JP PE,nn
BRA nn	JR nn
BEQ nn	JR Z,nn
BNE nn	JR NZ,nn
BCS nn	JR C,nn
BCC nn	JR NC,nn
DBR nn	DJNZ,nn
ROR r	RR r
ROL r	RL r
RRC r	RRC r
RLC r	RRC r
SRA r	SRA r

SLA r	SLA r
SRL r	SRL r
SLL r	SLL r
RET	RET
REQ	RET Z
RNE	RET NZ
RCS	RET C
RCC	RET NC
RPL	RET P
RMI	RET M
RVS	RET PO
RVC	RET PE
OPT n	OUT (n),A
INP n	IN A,(n)

This manual is not intended to teach the user how to program in assembly language, but to show the commands for BASX_A. If the user is not conversant with assembly language, it is then suggested that a book on learning assembly language for the Z80 chip is purchased from any good bookshop.

ASS [LISTING ON:OFF] [COMMAND name]

This will mark the start of the assembly code in the program listing. If 'LISTING' is 'ON', the object code and the addresses will be listed on the screen. The 'COMMAND' option enables the user to name the command used for executing the code.

ENT

Marks the entry point for the command. ie. In the example given below the variable 'HELLO' will be given the start address of the code pointed to by ENT.

ORG addr

Sets the start address where the code is to be assembled. This is normally 49152. When BASX_A has been loaded, it will allocate a segment where the assembled code is to be placed. Therefore the maximun size of the code allowed will be 16384 bytes. You can also allocate space for the code with the 'ALLOCATE' command and assemble the code into an 'ALLOCATE' address using ORG DEEK(540). (BASX_G must be loaded to use the 'DEEK' function.)

LAB var

Will define a label. 'var' will be given the value of the address of the location pointer.

EAS

Marks the end of the assembly language block. The assembler will execute the block twice (two–pass). On the first pass, any 'variable not initialised' errors will be ignored to allow for forward–referencing.

DFW word

Assembles a list of 16–bit numbers.

[LISTING ON:OFF]

Used with the 'ASS' command, it will decide wether or not to display the object code listing on the screen. Assembly is slower with the listing on. The listing would only probably be left on until the code has been fully debugged. eg.

```
100 PROGRAM "GREETING.BAS"
110 ASS LISTING ON COMMAND HELLO
120   ENT
130   LDN A,&HFF ! Channel to print text.
140   LXN DE,MESSAGE ! Address of text..
150   LXN BC,MESS_LEN ! Length of text.
160   RST 48 ! Call operating system.
170   DFB 8 ! Write block of text.
180   RST 24 ! Scan the extensions.
190   RET  ! Return to basic.
200   LET MESSAGE$="Hello there!"&CHR$(13)
210 LAB MESSAGE
220   DFB MESSAGE$
230   LET MESS_LEN=LEN(MESSAGE$)
240 EAS
```

Marks the start of a block for indentation. Only used for listing purposes. When the Editor sees this command, it will indent all the lines after this when a program is listed, until it sees a 'BND' command. This command has no run time purpose.

BND

Marks the end of a block for indentation. Only used for listing purposes. When the Editor sees this command it will stop the block indentation. This command has no run–time purpose.

The 'BEG' and 'BND' command can be used for formating the assembly blocks. Note:

Mnemonics use a different number–evaluation system for expressions. Hex numbers should be proceeded by '&H' ie. '&Hhhhh' – where 'hhhh' is the number in hex.

LXN HL,&HFFFF – is equivalent to LXN HL,–1.

Divide is not supported.

Functions or floating point numbers are not supported.

Binary operations supported are :–

NOT n
n AND n
n OR n
n XOR n

Basic variables and BASX_A variables are exactly the same.

POINTER

Returns the value of the location counter. Variables can be assigned to this value.

NOTE: Mnemonics cannot be used on multi–statement lines.