HRPrintFast.bas

The High Resolution printing routine allows standard sized characters to be positioned anywhere on the screen.

Usage

There is a an example program that uses this at the end of the page.

```
HRPrint(x,y,character,attribute,over)
```

Where * x is the x value in pixel co-ordinates * y is the y value in pixel co-ordinates * character is the memory address of the UDG style bytes for the character being printed. To print standard characters, use the address of the character from the ROM table. * attribute is the attribute byte value (As you'd get from the ATTR function) * over is 0 or 1, and chooses whether to print in OVER 0 or OVER 1 mode.

Prints the character to the screen at the given pixel co-ordinates.

> NOTE: The ZX Spectrum's attribute system is encoded into the hardware as a 32 character grid. HRPrint does its best, but changing the paper / bright / flash colour from the background is likely to look imperfect as the attribute blocks cannot line up well with the character printed unless it overlaps - as a result, the colour of attribute squares nearby is likely to be changed.

CODE

This code is too large to place in full, since it requires Britlion's Screen and Rotate Tables, and this wiki does not allow attachments. Please download HRPrintFast from the forum posting: http://boriel.com/mybb/showthread.php?tid=532&pid=3318#pid3318

```
SUB HRPrintFast (x AS UBYTE, y AS UBYTE, char AS UInteger, attribute AS UBYTE, overprint AS UBYTE)
REM High res Printing, based on code produced, with thanks, by Turkwel over on the WOS boards.
REM Brought to ZX Basic by Britlion, June 2010.

ASM
    ld a,(IX+13) ; Get overprint value
    AND a ; zero?
    JR Z,HRP_No_Over
    LD a,182
    JP HRP_Change_Code

HRP_No_Over:
    XOR A ; faster than LD a,0

HRP_Change_Code:
    LD (HRPOver1),a
    LD (HRPOver2),a
    ld b,(IX+7)
    ld c,(IX+5)
    push BC ; SAVE our co-ordinates.

;print_char:
    ld  d,(IX+09)
    inc d
    dec d
    jr z, HRPrint_From_Charset
    ld e,(IX+08)
    jp HR_Print

HRPrint_From_Charset:
    ld  de,(23606)
    ld  h,0
    ld  l,(IX+8) ; character
    add  hl,hl
    add  hl,hl
    add  hl,hl
    add  hl,de

HR_Print:
    ;call HRPat
    ;PUSH HL
    EX DE,HL  ; Save HL out in DE

    LD H, HRPScreenTables/256
    LD L,B
    LD A,(HL)

    INC H
    LD L,(HL)
    LD H,A

    LD A,C
    SRL A
    SRL A
    SRL A

    ADD A,L
    LD L,A
    EX DE,HL ; swap HL and DE Back

;convert the Y AND X pixel values TO the correct Screen Address  - Address in DE
    ld a,8

;set counter TO 8 - Bytes of Character Data TO put down
HRPrint0:
    push af

;save off Counter
    ;ld a,b
    ;cp 192
    ;jr c,HRprint1
    ;pop af
```

```
    ;jp HRPrintEnd

;don't print character if  > 191 - off the bottom of the screen - restore AF and exit Print routine
;[this can be removed IF you are keeping tight control of your Y values]
HRprint1:
    push hl
    push de
    push de

;save off Address of Character Data, Screen Address, Screen Address
    ld a,c
    AND 7
    ld d,a

;get lowest 3 bits of Screen address
    ;ld e,255

;set up E with the Mask TO use - 11111111b = All On
    ld a,(hl)
    jr nz,HRprint2

;get a BYTE of Character Data TO put down - but ignore the following Mask shifting
;if the the X value is on an actual Character boundary i.e. there's no need to shift anything
    ; rrca
    ; srl e
    ; dec d
    ; jp nz,HRprint2

    ld e,0
    jp HRprint3

HRprint2:
;Rotate the Character Data BYTE D times - AND Shift the Mask BYTE AS well, forcing Zeroes into the
;Left hand side. The Mask will be used TO split the Rotated Character Data OVER a Character boundary

; New version: Grab into DE, the split rotation values.
     LD E,A ; Put our working byte safe
     LD A,D ; Grab our number of rotates
     EX DE,HL ; Save HL
     DEC A ; decrease so 1->0
     SLA A ; Multiply by 2 because we have double tables.
     ADD A, RotateTables/256 ; Add in the base for rotate tables.
     LD H,A ; put it into our lookup.
     LD A,(HL) ; get high byte
     INC H
     LD L,(HL) ; get low byte
     LD H,A
     EX DE,HL ; put result in DE, and restore HL.


HRprint3:
    pop hl

;POP one of the Screen Addresses (formerly in DE) into HL
    ;ld d,a
    ;ld a,e
    ;AND d
    ld a,d ; get our first byte

HRPOver1:
    OR (hl)
    ld (hl),a

;take the Rotated Character Data, mask it with the Mask BYTE AND the OR it with what's already on the
;this takes care of the first part of the BYTE
;[remove the OR (HL) IF you just want a straight write rather than a merge]
    inc l
    ld a,l
    AND 31
    jr z, HRprint4

;Increment the Screen Address AND check TO see IF it's at the end of a line,
```

```
;if so THEN there's no need to put down the second part of the Byte
    ld a,e
    ;cpl
    ;AND d

HRPOver2:
    OR (hl)
    ld (hl),a

;Similar TO the first BYTE, we need TO Invert the mask with a CPL so we can put down the second part o
;in the NEXT Character location
;[again, remove the OR (HL) IF you just want a straight write rather than a merge]
HRprint4:
    pop de
    inc d
    inc b

;get the Screen Address back into DE, increment the MSB so it points the the Address immediately below
;it AND Increment the Y value in B AS well
    ld a,b
    AND 7

    ;call z,HRPat
    jr nz, HRPatSkip

    EX DE,HL  ; Save HL out in DE

    LD H, ScreenTables/256
    LD L,B
    LD A,(HL)

    INC H
    LD L,(HL)
    LD H,A

    LD A,C
    SRL A
    SRL A
    SRL A

    ADD A,L
    LD L,A
    EX DE,HL ; swap HL and DE Back

HRPatSkip:

;now check IF the Y value has gone OVER a Character Boundary i.e. we will need TO recalculate the Scre
;Address IF we've jumped from one Character Line to another - messy but necessary especially for lines
      pop hl
      inc hl

;get the Address of the Character Data back AND increment it ready FOR the NEXT BYTE of data
      pop af
      dec a
      jp nz,HRPrint0

;get the Counter value back, decrement it AND GO back FOR another write IF we haven't reached the end
      ; jp HRPrintAttributes (No need to jump around this now)

HRPrintAttributes:
      POP BC ; recover our X-Y co-ordinates.
      ld d,0
      ld a,(IX+11) ; attribute
      AND a
      jp z, HRPrintEnd  ; IF attribute=0, THEN we don't do attributes.
      ld e,a ; pass TO e

;transfer Attribute BYTE TO e FOR easier use
      ld a,b
      cp 192
      jp nc, HRPrintEnd
```

```
;check Y position AND EXIT IF off bottom of screen
      push bc

;save off Y AND X values FOR later
      AND 248
      ld h,22
      ld l,a
      add hl,hl
      add hl,hl
      srl c
      srl c
      srl c
      ld b,d
      add hl,bc

;calculate the correct Attribute Address FOR the Y\X values
      ld (hl),e

;set the Attribute - this is ALWAYS set no matter what the valid Y\X values used
      pop bc

;get the Y AND X values back into BC
      ;call print_attribute2

;call the subroutine TO see IF an adjacent Horizontal Attribute needs TO be set
print_attributes1:
      ld a,c
      cp 248
      jr nc,endPrintAttributes1

;check TO see IF we are AT Horizontal character 31 - IF so THEN no need TO set adjacent Horizontal Att
      AND 7
      jr z, endPrintAttributes1

;and don't set the adjacent Horizontal Attribute if there's no need to
      inc l
      ld (hl),e
      dec l

;increment the Attribute address - set the adjacent horizontal Attribute - THEN set the Attribute Addr
endPrintAttributes1:
      ld a,b
      cp 184
      jp nc, HRPrintEnd

;check TO see IF we are AT Vertical character 23 - IF so THEN no need TO set adjacent Vertical Attribu
      AND 7
      jp z, HRPrintEnd

;and don't set the adjacent Vertical Attribute if there's no need to & Exit routine
      ld a,l
      add a,32
      ld l,a
      ld a,d
      adc a,h
      ld h,a
      ld (hl),e

;set the Attribute address TO the line below  - AND set the adjacent Vertical Attribute
;drop through now into adjacent Horizontal Attribute subroutine - all RETs will now EXIT the routine
HRPrintAttribute2:
      ld a,c
      cp 248
      jp nc, HRPrintEnd

;check TO see IF we are AT Horizontal character 31 - IF so THEN no need TO set adjacent Horizontal Att
      AND 7
      jp z, HRPrintEnd

;and don't set the adjacent Horizontal Attribute if there's no need to
      inc l
      ld (hl),e
```

```
      dec l

;increment the Attribute address - set the adjacent horizontal Attribute - THEN set the Attribute Addr
      ;ret
      jp HRPrintEnd

#include once "Screentables.asm"
#include once "RotateTables.asm"

HRPrintEnd:

END ASM
END SUB
```

EXAMPLE OF USE

```
CLS

DIM x,y,x2,y2 as uByte
DIM xd,yd as fixed
DIM counter as uInteger
DIM time,endtime as uLong

x=100
y=10
x2=x
y2=y
xd=2
yd=2


DO

x2=x+xd
y2=y+yd

' Timing loop to make the sprite update clean on the screen.
' Instead of a waste of time loop here, you could go off and do something for the equivalent time spa
' You could call subroutines to update numbers in memory, play some sound for a while, see if a key is
' The point is you need to do a halt, and then get to the sprite update bit a little while later - aft
' This version below clearly wastes over 130,000 T states per frame.

asm
halt
halt
halt
push hl ;11
push af ; 11

ld hl,5000 ;10
quickloop:
dec hl     ;6
ld a,l     ;4
or h       ;4

jr nz, quickloop   ; 12 (usually - one iteration of 7)
pop af     ; 10
pop hl     ;10
end asm

HRPrintFast(x,y,32,56,0)
HRPrintFast(x+8,y,32,56,0)
HRPrintFast(x,y+8,32,56,0)
HRPrintFast(x+8,y+8,32,56,0)


HRPrintFast(x2,y2,@gentle,76,0)
HRPrintFast(x2+8,y2,@gentle+8,76,1)
HRPrintFast(x2,y2+8,@gentle+16,76,0)
HRPrintFast(x2+8,y2+8,@gentle+24,76,1)

x=x2
y=y2


IF x<=0 OR x>=245 THEN xd=-xd : END IF
IF y<=0 OR y>=175 THEN yd=-yd : END IF

LOOP
end

gentle:
asm
    defb 15,15,15,15,15,15,13,15
    defb 240,144,208,208,240,240,176,240
    defb 15,14,63,0,0,12,26,30
    defb 176,112,252,0,0,24,104,120
end asm
```