Print64.bas

The 64 column printing routine allows text to be 4 pixels wide instead of 8. It is NOT proportional printing, but this is still useful for lining things up in columns.

This routine has been adopted as an included library - so you may include it with

```
#include <print64.bas>
```

Usage

```
printat64(y,x)
```

Moves the print64 system's print cursor to row Y, column X. Note that `0 <= x <= 63` - that is the range of values for X can be up to 63. The range of values for Y is the normal 0-23.

- Note that the print64 system's cursor position is independent from that of the ZX Basic Print routine, or any other, such as the print42 system.

```
printat64(STRING)
```

Prints the string to the screen at the current Print64 co-ordinates. It does so in the current permanent colours.

NOTE: The ZX Spectrum's attribute system is encoded into the hardware as a 32 character grid. Print64 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 pixel blocks.

CODE

- There is a version of this code included with the compiler (though the version listed here may or may not be more recent). Code converted to ZXBasic by Britlion, based on Andrew Owen's 64 Character code http://www.worldofspectrum.org/forums/showpost.php?p=167447&postcount=1

```
SUB printat64 (y as uByte, x as uByte)
   IF y<24 AND x<64 then
     POKE @p64coords,x
     POKE @p64coords+1,y
   ELSE
     asm
       rst     8               ; error "5   Out of screen"
       defb    4
     end asm
   END IF
END SUB


SUB print64 (characters$ as String)
ASM

; This frankencode created by Paul Fisher, Andrew Owen, Chris Born and Einar Saukas
; TODO:
; * Inverse
; * Bold (which will use a Second font)

LD L,(IX+4)
LD H,(IX+5) ; Get String address of characters$ into HL.

; Load BC with length of string, and move HL to point to first character.
        ld c, (hl)              ; 60020 78
        inc hl                  ; 60021 35
        ld b, (hl)              ; 60022 70
        inc hl                  ; 60023 35

; Test string length. If Zero, exit.
        ld a, c                 ; 60024 121
        or b                    ; 60025 176
        jp z, p64_END         ; 60026 200

examineChar:
        ld a, (hl)              ; Grab the character
        cp 128                  ; too high to print?
        jr nc, nextChar           ; then we go to next.

newLine:
        cp 13                   ; Is this a newline character? 60056 254 13
        jr nz, p64_isPrintable   ; If not, hop to testing to see if we can print this 60058 32  13
        push hl
        push bc
        ld b,0
        ld hl, p64_coords      ; Get coords 60060 237 91  68  235
        call BLp64_NEXT_ROW         ; Go to next line.     ; 60064 205 58  235
        pop bc
        pop hl

        ld (p64_coords), de    ; 60067 237 83  68  235
        jr nextChar             ; 60071 24  11

p64_isPrintable:
        cp 31                   ; Bigger than 31? 60073 254 31
        jr c, nextChar          ; If not, get the next one. 60075 56  7

        push hl                 ; Save position 60077 229
        push bc                 ; Save Count   60078 197
        call p64_PrintChar      ; Call Print SubRoutine



        pop bc                  ; Recover length count  60082 193
        pop hl                  ; Recover Position 60083 225

nextChar:
        inc hl                  ; Point to next character 60084 35
        dec bc                  ; Count off this character 60085 11
        ld a, b                 ; Did we run out? 60086 120
        or c                    ; 60087 177
```

```
        jr nz, examineChar       ; If not, examine the next one 60088 32  193
        jp p64_END               ; Otherwise hop to END. 60090 201

p64_PrintChar:
; Arrives with A as a byte to print.
    ld hl,p64_coords
    push    hl              ; save COL address for later
    ld      e, a            ; store character value in E
    ld      b,0
    ld      c, (hl)         ; store current column in BC

    ; Check if character font must be rotated, self-modifying the code accordingly

        xor     c               ; compare BIT 0 from character value and column
        rra
        ld      a, 256-(BLp64_END_LOOP-BLp64_SKIP_RLC) ; instruction DJNZ skipping rotation
        jr      nc, BLp64_NOT_RLC             ; decide based on BIT 0 comparison
        ld      a, 256-(BLp64_END_LOOP-BLp64_INIT_RLC) ; instruction DJNZ using rotation

BLp64_NOT_RLC:
        ld      (BLp64_END_LOOP - 1), a       ; modify DJNZ instruction directly

; Check the half screen byte to be changed, self-modifying the code accordingly
        srl     c               ; check BIT 0 from current column
        ld      a, %00001111    ; mask to change left half of the screen byte
        jr      nc, BLp64_SCR_LEFT    ; decide based on odd or even column
        cpl                     ; mask to change right half of the screen byte

BLp64_SCR_LEFT:
        ld      (BLp64_SCR_MASK + 1), a   ; modify screen mask value directly
        cpl
        ld      (BLp64_FONT_MASK + 1), a  ; modify font mask value directly

; Calculate location of the first byte to be changed on screen
; The row value is a 5 bits value (0-23), here represented as %000RRrrr
; The column value is a 6 bits value (0-63), here represented as %00CCCCCc
; Formula: 0x4000 + ((row & 0x18) << 8) + ((row & 0x07) << 5) + (col >> 1)

        inc     hl              ; now HL references ROW address
        ld      a, (hl)         ; now A = %000RRrrr
        call    0e9eh           ; now HL = %010RR000rrr00000
        add     hl, bc          ; now HL = %010RR000rrrCCCCC
        ex      de, hl          ; now DE = %010RR000rrrCCCCC
                                ; and e=char -> l=char

; Calculate location of the character font data in p64_charset
; Formula: p64_charset + 7 * INT ((char-32)/2) - 1

        ld      h, b            ; now HL = char (because b=0)
        srl     l               ; now HL = INT (char/2)
        ld      c, l            ; now BC = INT (char/2)
        add     hl, hl          ; now HL = 2 * INT (char/2)
        add     hl, hl          ; now HL = 4 * INT (char/2)
        add     hl, hl          ; now HL = 8 * INT (char/2)
        sbc     hl, bc          ; now HL = 7 * INT (char/2)
        ld      bc, p64_charset - 71h
        add     hl, bc          ; now HL = p64_charset + 7 * INT (char/2) - 0x71


; Main loop to copy 8 font bytes into screen (1 blank + 7 from font data)
        xor     a               ; first font byte is always blank
        ld      b, 8            ; execute loop 8 times

BLp64_INIT_RLC:
        rlca                    ; switch position between bits 0-3 and bits 4-7
        rlca
        rlca
        rlca

BLp64_SKIP_RLC:

; ------------------------------------------------------------------------
```

```
; STANDARD OR INVERSE
;
BLp64_INV_C:  nop                     ; either 'NOP' or 'CPL' (modified)
; ------------------------------------------------------------------------------

BLp64_FONT_MASK:
        and     %11110000       ; mask half of the font byte
        ld      c, a            ; store half of the font byte in C
        ld      a, (de)         ; get screen byte

BLp64_SCR_MASK:
        and     %00001111       ; mask half of the screen byte
        or      c               ; combine half screen and half font
        ld      (de), a         ; write result back to screen
        inc     d               ; next screen location
        inc     hl              ; next font data location
        ld      a, (hl)         ; store next font byte in A
        djnz    BLp64_INIT_RLC        ; repeat loop 8 times (this instruction gets modified)

BLp64_END_LOOP:
        ; attributes
        ld de,(p64_coords)       ; grab coords
        and a                    ; clear carry
        rr e                     ; divide x by 2 to get bytes instead of nybbles
        ld a, d                  ; Get Y coord
        sra a                    ;
        sra a                    ;
        sra a                    ; Multiply by 8 60155 203 47
        add a, 88                ; Add to attrbute base address
        ld h, a                  ; Put high byte value for attribute into H.
        ld a, d                  ; get y value again
        and 7                    ; set within third
        rrca                     ;
        rrca                     ;
        rrca                     ;
        add a, e                 ; add in x value
        ld l, a                  ; Put low byte for attribute into l
        ld a, (23693)            ; Get permanent Colours from System Variable
        ld (hl), a               ; Write new attribute

        pop     hl              ; restore AT_COL address
        inc     (hl)            ; next column
        bit     6, (hl)         ; column lower than 64?
        ret     z               ; return if so

BLp64_NEXT_ROW:
        ld      (hl), b         ; reset AT_COL
        inc     hl              ; store AT_ROW address in HL
        inc     (hl)            ; next row
        ld      a, (hl)
        cp      24              ; row lower than 23?
        ret     c               ; return if so
        ld      (hl), b         ; reset AT_ROW
        ret                     ; done!


end asm
p64coords:
asm
p64_coords:
       defb 0;  X Coordinate store
       defb 0;  Y Coordinate Store

p64_charset:         ; 60230
        DEFB 2,2,2,2,0,2,0                  ; Space !
        DEFB 80,82,7,2,7,2,0                ; " #
        DEFB 37,113,66,114,20,117,32        ; $ %
        DEFB 34,84,32,96,80,96,0            ; & '
        DEFB 36,66,66,66,66,36,0            ; ( )
        DEFB 0,82,34,119,34,82,0            ; * +
        DEFB 0,0,0,7,32,32,64               ; , -
        DEFB 1,1,2,2,100,100,0              ; . /
```

```
        DEFB 34,86,82,82,82,39,0            ; 0 1
        DEFB 34,85,18,33,69,114,0           ; 2 3
        DEFB 87,84,118,17,21,18,0           ; 4 5
        DEFB 55,65,97,82,84,36,0            ; 6 7
        DEFB 34,85,37,83,85,34,0            ; 8 9
        DEFB 0,2,32,0,34,2,4                ; : ;
        DEFB 0,16,39,64,39,16,0             ; < =
        DEFB 2,69,33,18,32,66,0             ; > ?
        DEFB 98,149,183,181,133,101,0       ; @ A
        DEFB 98,85,100,84,85,98,0           ; B C
        DEFB 103,84,86,84,84,103,0          ; D E
        DEFB 114,69,116,71,69,66,0          ; F G
        DEFB 87,82,114,82,82,87,0           ; H I
        DEFB 53,21,22,21,85,37,0            ; J K
        DEFB 69,71,71,69,69,117,0           ; L M
        DEFB 82,85,117,117,85,82,0          ; N O
        DEFB 98,85,85,103,71,67,0           ; P Q
        DEFB 98,85,82,97,85,82,0            ; R S
        DEFB 117,37,37,37,37,34,0           ; T U
        DEFB 85,85,85,87,39,37,0            ; V W
        DEFB 85,85,37,82,82,82,0            ; X Y
        DEFB 119,20,36,36,68,119,0          ; Z [
        DEFB 71,65,33,33,17,23,0            ; \ ]
        DEFB 32,112,32,32,32,47,0           ; ^ _
        DEFB 32,86,65,99,69,115,0           ; £ a
        DEFB 64,66,101,84,85,98,0           ; b c
        DEFB 16,18,53,86,84,35,0            ; d e
        DEFB 32,82,69,101,67,69,2           ; f g
        DEFB 66,64,102,82,82,87,0           ; h i
        DEFB 20,4,53,22,21,85,32            ; j k
        DEFB 64,69,71,71,85,37,0            ; l m
        DEFB 0,98,85,85,85,82,0             ; n o
        DEFB 0,99,85,85,99,65,65            ; p q
        DEFB 0,99,84,66,65,70,0             ; r s
        DEFB 64,117,69,69,85,34,0           ; t u
        DEFB 0,85,85,87,39,37,0             ; v w
        DEFB 0,85,85,35,81,85,2             ; x y
        DEFB 0,113,18,38,66,113,0           ; z {
        DEFB 32,36,34,35,34,36,0            ; | {
        DEFB 6,169,86,12,6,9,6              ; ~ (c)

p64_END:
End Asm
End Sub
```

There's an example of usage here:

```
REM Example

DIM n,x,y as uInteger
CLS

FOR n=1 to 1000
   y=rnd*23
   x=rnd*62

   ink rnd*8

   printat64(y, x)
   print64 ("ABCDEFGHIJKLMNOPQRSTUVWXYZ"(n MOD 26 TO n MOD 26))
NEXT n
END
```