Fast Font Rendering for the Apple II using Transposed Fonts

Normal font rendering: glyphs stored sequentially

```
    ldx #0
    ldy column_save
draw_loop
draw_loop_smc
    lda $ffff,x
    sta (hgr_ptr),y
    clc
    lda #4
    adc hgr_ptr+1
    sta hgr_ptr+1
    inx
    cpx #8
    bcc draw_loop
```

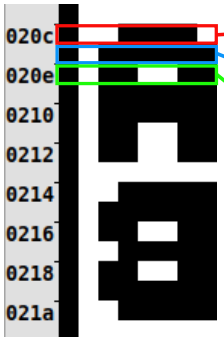
* indexed addressing to read glyph; address set using self-modifying code

* indirect indexed addressing to write to screen; must increment by $400 each line

* works with any font in memory

* can write to either hi-res page

Compiled font rendering: glyph data embedded in code



```
COMPILEDFONT_H1_64
    lda #$9e
    sta (hgr_ptr),y
    clc
    lda #4
    adc hgr_ptr+1
    sta hgr_ptr+1
    lda #$b3
    sta (hgr_ptr),y
    clc
    lda #4
    adc hgr_ptr+1
    sta hgr_ptr+1
    lda #$b3
    sta (hgr_ptr),y
    clc
```

* jump table to find definition of glyph, column in Y register

* one entry in jump table for each glyph; each entry can write glyph to any column or row

* immediate mode data for each scan line of glyph

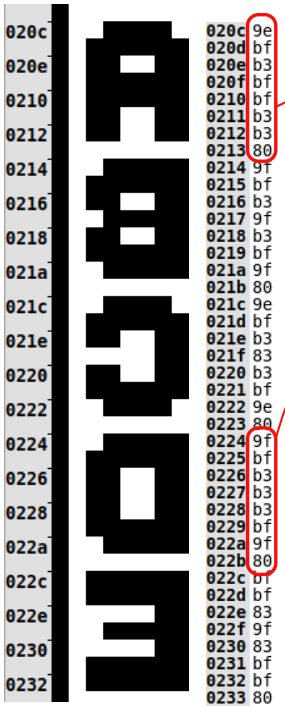
* indirect indexed addressing to store on screen, must increment address by $400 to move to next scan line

* requires tool to generate code

* each font requires separate jump table and glyph definitions

* can write to either hi-res page

Transposed font rendering: glyph bytes reordered



```
TransposedFontRow0   .byte  ... , $9e, $9f, $9e, $9f, $bf, ...
TransposedFontRow1   .byte  ... , $bf, $bf, $bf, $bf, $bf, ...
TransposedFontRow2   .byte  ... , $b3, $b3, $b3, $b3, $83, ...
TransposedFontRow3   .byte  ... , $bf, $9f, $83, $b3, $9f, ...
TransposedFontRow4   .byte  ... , $bf, $b3, $b3, $b3, $83, ...
TransposedFontRow5   .byte  ... , $b3, $bf, $bf, $bf, $bf, ...
TransposedFontRow6   .byte  ... , $b3, $9f, $9e, $9f, $bf, ...
TransposedFontRow7   .byte  ... , $80, $80, $80, $80, $80, ...
```

```
FASTFONT_H1_0
    lda TransposedFontRow0,y
    sta $2000,x
    lda TransposedFontRow1,y
    sta $2400,x
    lda TransposedFontRow2,y
    sta $2800,x
    lda TransposedFontRow3,y
    sta $2c00,x
    lda TransposedFontRow4,y
    sta $3000,x
    lda TransposedFontRow5,y
    sta $3400,x
    lda TransposedFontRow6,y
    sta $3800,x
    lda TransposedFontRow7,y
    sta $3c00,x
    ldy scratch_0
    rts
```

* jump table to find row, column in X register, glyph index in Y register

* one entry in jump table for each row; each entry can write any glyph to any column in **one** row

* **the big win**: no pointer addition or index incrementing necessary to write entire glyph

* requires tool to generate code and transpose font: e.g. 128 glyph font: 8 rows of 128 bytes each. First row of 128 bytes contains topmost byte of each glyph, 2nd row contains row below that, etc.

* each font requires new jump table and row definition

* need second jump table and new entries for each row to write to second hi-res page.

Speed comparison:

Fill entire screen with characters (i.e. calls each routine 960 times)	Cycles	Frames	Time (ms)	Code Size (bytes)
Baseline code (not shown)	375,000	22.0	367	154
Self-modifying code	320,000	18.8	313	155
Compiled font	211,000	12.4	207	12018*
Transposed font	129,000	7.6	126	2462*

* includes built-in 1024 byte font; others can use any font in memory

Code generator now available! See:

https://github.com/robmcmullen/asmgen

Requires Python 3.6 to generate code; target assembler is configurable.