Commodore Plus/4, Mad Pascal i bitmapy

Tytułem wstępu

Ten artykuł jest małym uzupełnieniem artykułów **Carriona**, które ukazały się do tej pory na łamach portalu C64portal.pl, a mianowicie:

- Grafika na C+4 – Sztuka nietajemna. Część 1 – kolory.
- Grafika na C+4. Część 2 – Bitmap Multicolor

Polecam przeczytajć i obejrzeć dołączone do nich materiały wideo.

Narzędzia

W artykule posłużę się:

- Mad Pascal, dokumentacja
- Multipaint
- plus4emu lub inny emulator
- apultra
- Exomizer

No to jedziemy!

Impulsem do napisania tej notki było obdarowanie mnie przez Carriona i Bocianu grafiką, którą mam zamiar użyć zamykając swój testowy projekt na komputer Plus/4 (jakim jest gra Tron +4). Skoro stałem się szczęśliwym posiadaczem *obrazków* to pojawiło się pytanie jak je wyświetlić na komputerze z rodziny **C246**? Sprawa okazała się nader prosta :]

High-Resolution Bit Map

Z książki jaką polecam nakońcu artykułu wyjąłem następujące repetytorium:

> $FF06 Set bit 5 for bit-map mode.
>
> $FF12 Bit 2 determines RAM (0) or ROM (1). Bits 5-3 determine loca
> tion of bit map (bits 15-13).
>
> $FF14 Bits 7-3 determine location of luminance/color memory (bits
> 15-11).
>
> $FF19 Border luminance and color.
>
> The bit map displayed is determined by bit-map memory. The luminances of both colors are determined by color memory (normally $0800-$0BE7). Color for both colors is determined by screen memory (normally $0C00-$0FE7).

Przyda nam się też wyjątek z mapy pamięci dotyczący rejestrów **TED**:

```
R#   Bit 7       6       5       4       3       2       1       0
--------------------------------------------------------------------
6       Test    ECM     BMM     Blank   Rows    Y2      Y1      Y0
7       RVS off PAL     Freeze  MCM     Columns X2      X1      X0
12      -       -       BMB2    BMB1    BMB0    R-Bank  S1-9    S1-8
14      VM4     VM3     VM2     VM1     VM0     -       -       -
```

Otrzymaną od **Bocianu** bitmapę:

wczytałem do programu **Multipaint** i zapisałem jako `mp.prg` w formacie **Boticelli** czyli jako binarkę z dwubajtowym nagłówkiem `$0078` (co oznacza adres `$7800`) w której dane idą (w kolejności przy zachowaniu wielkości bloków) `1KB-1KB-8KB`:

- luminacja
- kolory
- grafika

Nagłówek będziemy pomijać bo sami zadecydujemy pod jakim adresem umieścić grafikę.

Program wyświetlający grafikę **HiRes** w języku **Mad Pascal** jest bardzo prosty i może wyglądać następująco:

- plik z zasobami `hgfx.rc`

`LOGO rcdata 'mp.prg' 2`

- program główny `hires.pas`

```
{$r hgfx.rc}

const
  LOGO = $5800;

var
  SETBITMAP                       : byte absolute $ff06;
```

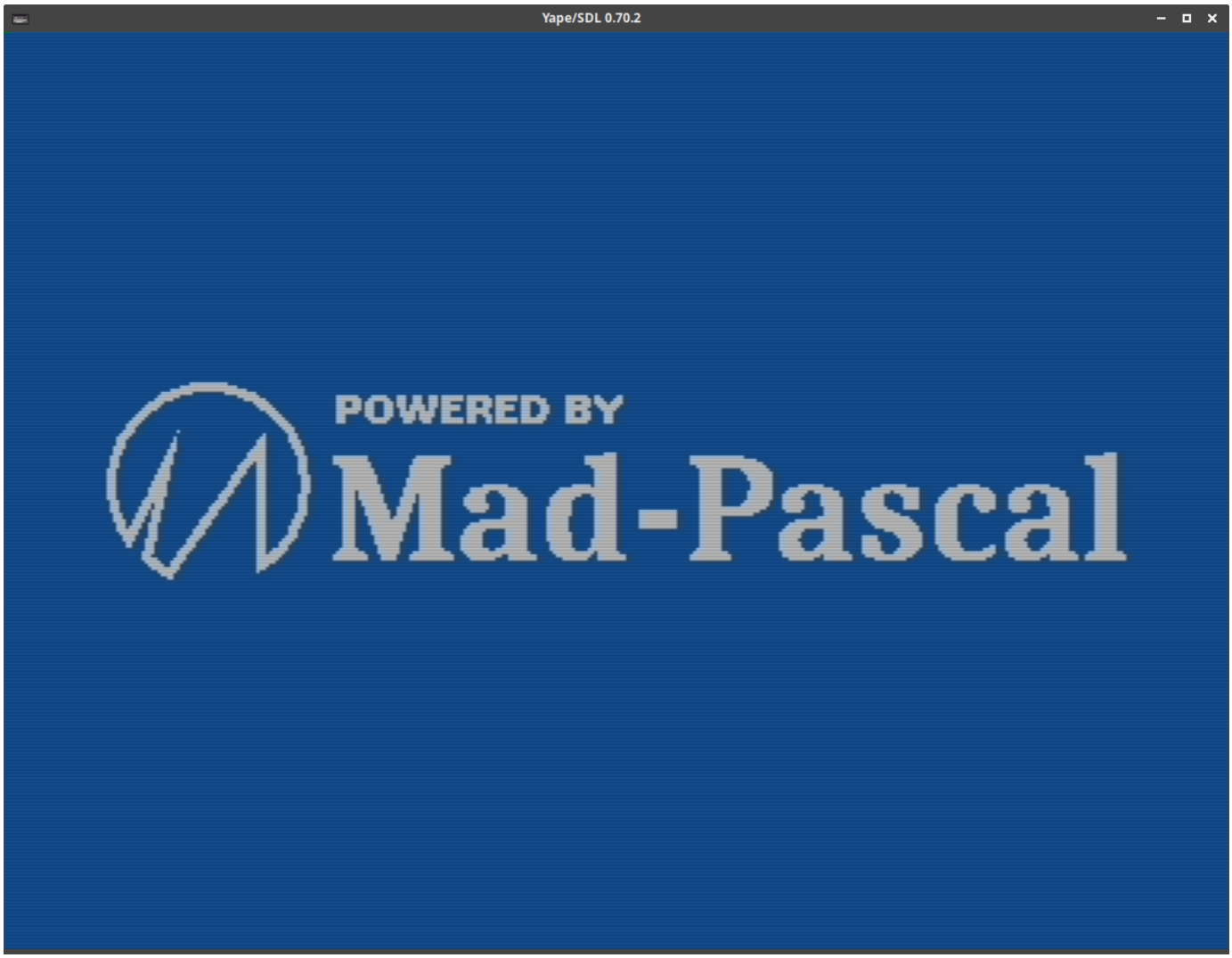
```
  SETMCOLOR                          : byte absolute $ff07;
  BITMAPADDR                         : byte absolute $ff12;
  VIDEOMATRIX                        : byte absolute $ff14;
  BORDER                             : byte absolute $ff19;

begin
  SETBITMAP := SETBITMAP or $20;
  SETMCOLOR := (SETMCOLOR and $40) or $8;

  // (01011xxx) $5800 = 11 * $800;
  VIDEOMATRIX := %01011000;
  // (xx011xxx) $6000 = 3 * $2000; bit 2 set to 0 means reading from RAM
  BITMAPADDR := %00011000 or (BITMAPADDR and %00000011);

  BORDER := $3d;

  repeat until false;
end.
```

Teraz krótkie objaśnienie co powyższy program robi:

- `LOGO rcdata 'mp.prg' 2` wczyta dane z pliku `mp.prg` z pminięciem `2` pierwszych bajtów pod adres przypisany etykiecie/stałej `LOGO`
- `{$r hgfx.rc}` dyrektywa dołączająca zadeklarowane zasoby
- `absolute` mapuję zmienne na potrzebne mi rejestry układu **TED**
- `SETBITMAP := SETBITMAP or $20` ustawiam `6` bit tego rejestru na `1` informując komputer, że chcę zmienić tryb wyświetlania z tekstowego na graficzny, resztę bitów pozostawiam bez zmian.
- `SETMCOLOR := (SETMCOLOR and $40) or $8` zapamiętuję tylko 6 bit rejestru informujący nas o aktualnym systeme telewizyjnym PAL/NTSC i zapalam `3` bit aby mieć ekran o normalnej szerokości 320 pikseli.
- `VIDEOMATRIX := %01011000` ustawiam adres dla luminacji, która zaczyna się od miejsca wczytania naszego pliku `mp.prg` czyli od adresu `$5800`. Luminacaja to blok o wielkości `1KB`, tyle samo zajmują kolory, razem `2KB`. Z tego tytułu grnulacja pamięci dla *mapy kolorów* naszej bitmapy to `2KB` czyli szesnastkowo `$800`. Dzieląc nasz adres `$5800` przez `$800` otrzymujemy dziesiętnie `11` i taką liczbę wpisujemy binarnie `%1011` do rejestru `$ff14` ale zaczynając dopiero od jego `3` bitu.
- `BITMAPADDR := %00011000 or (BITMAPADDR and %00000011)` jak już pisałem blok poprzedzający dane grafiki poprzedzają dane luminacji i koloru wielkości `2KB`, stąd adres bitmapy to `$5800 + $800 = $6000`. Granulacja dla bitmapy to `8KB` czyli możemy dane umieścić w pamięcy tylko co `$2000` zaczynając właśnie od tego adresu. Aby wyliczyć jaką wartość powinniśmy wpisać do rejestru `$ff12` musimy `$6000` podzielić przez `$2000` i zapisać wynik `%11` do rejestru także od `3` bitu. Bit `2` należy ustwić na `0` (zgasić) - to poinformuje układ **TED**, że dane

znajdują się w **RAM** a nie **ROM**. Ja jeszcze w powyższym przykładzie staram się zachować dwa pierwsze bity rejestru, które nie są związane z grafiką.

Teraz kompilujemy program (jak to się robi można samodzielnie doszukać w dokumentacji **Mad Pascala**, lub zapytać mnie na forum o szczegóły). Po uruchumieniu naszego programu powinniśmy zobaczyć:



Multicolor Bit Map

Teraz czas na grafikę **Carriona**, którą od razu otrzymałem w formacie **Boticelli**.

- plik z zasobami `mgfx.rc`

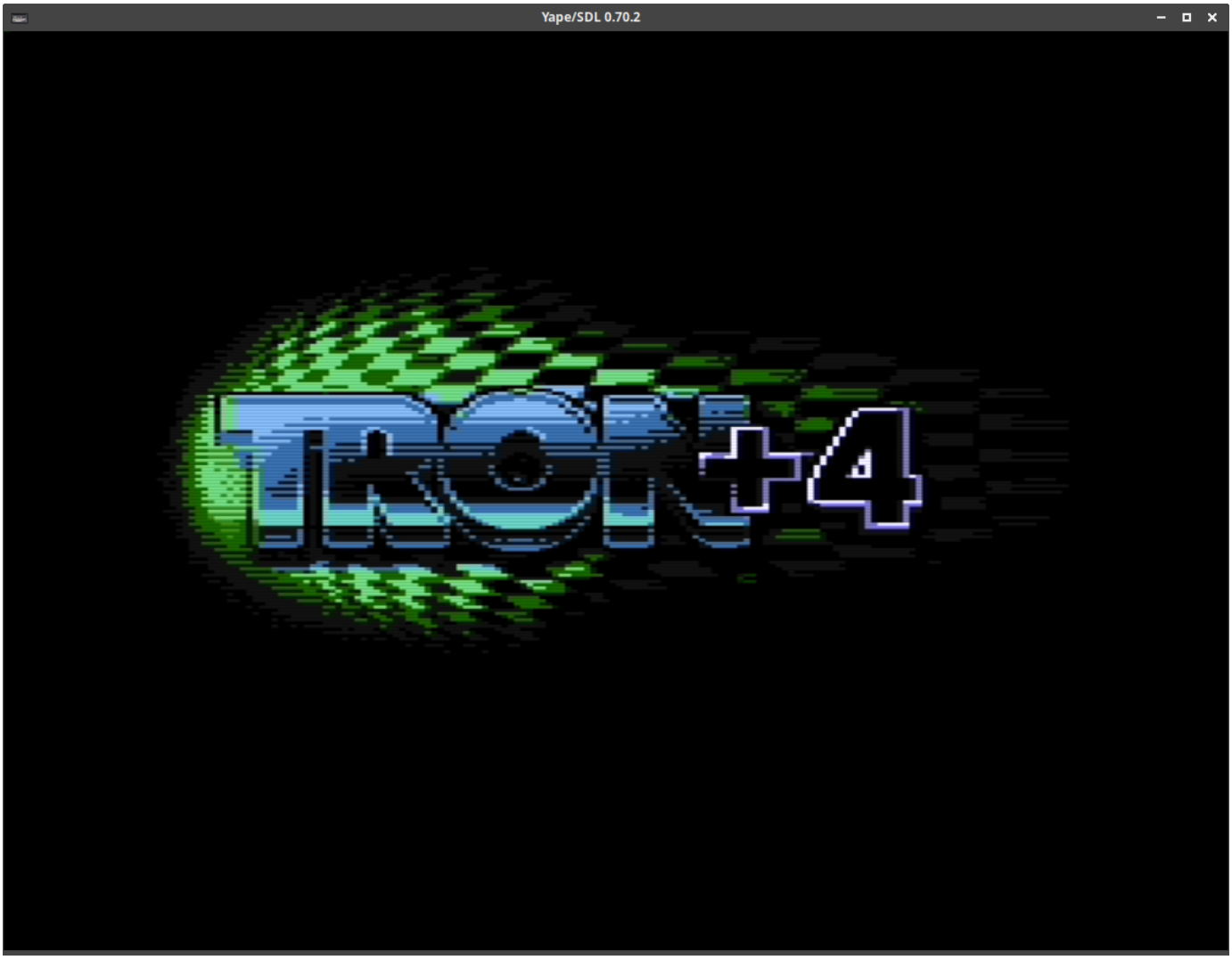
```
TRON_TITLE_SCREEN rcdata 'tron.prg' 2
```

- program główny `hires.pas`

```
{$r mgfx.rc}

const
  TRON_TITLE_SCREEN = $5800;

var
  SETBITMAP                       : byte absolute $ff06;
```

```
  SETMCOLOR                           : byte absolute $ff07;
  BITMAPADDR                          : byte absolute $ff12;
  VIDEOMATRIX                         : byte absolute $ff14;
  BACKGROUND                          : byte absolute $ff15;
  COLOUR1                             : byte absolute $ff16;
  BORDER                              : byte absolute $ff19;

begin
  SETBITMAP := SETBITMAP or $20;
  SETMCOLOR := (SETMCOLOR and $40) or $18;

  // (01011xxx) $5800 = 11 * $800;
  VIDEOMATRIX := %01011000;
  // (xx011xxx) $6000 = 3 * $2000; bit 2 set to 0 means reading from RAM
  BITMAPADDR := %00011000 or (BITMAPADDR and %00000011);

  BORDER := 0;
  BACKGROUND := 0;
  COLOUR1 := 1;

  repeat until false;
end.
```

Jedyna istotna zmiana to zapalenie w rejestrze `$ff07` bitu `4` który poinformuje układ **TED**, że ma interpretować dostarczone dane jako multikolor.



Packer APL

Jak można sobie łatwo policzyć bitmapy zajmują `10KB` co dla komputera wyposarzonego a `64KB` jest liczbą znaczącą, jeżeli byśmy w naszym programiw jednocześnie chcielibyśmy trzymać kilka grafik to w postaci *surowej* moglimyśmy sobie pozwolić na ich niewielką licznę.

W takiej sytuacji przychodzą nam w sukurs znane algorytmy pakujące dane jak: **deflate**, **lz4** czy **apl**. **Mad Pascal** dostarcza biblioteki do wszystkich jakie wymieniłem. My skorzystam z **apl**.

Logo **Mad Pascala** w postaci niespakowanej to jak już dokładnie już wiesz `10KB`, tymczasem po spakowaniu grafiki programem **apultra** otrzymujemy plik wielkości lekko przekraczającej `1KB`! :]

Jak się do tego zabrać?

Napier pozbądźmy się nagłówka z pliku **prg**, możesz do zrobić dowolnym edytorem, ja posłużę sie programem **dd**:

```
dd if=logo.prg of=logo.bin bs=1 skip=2
```

teraz możemy spakować czyste dane graficzne:

```
apultra.exe logo.bin logo.apl
```

Dla tak *przygotowanych* grafik program w **Mad Pascalu** może wyglądać następująco:

- plik z zasobami `hgfx_apl.rc`

`APLBIN rcdata 'mp.apl'`

- program główny `hires_apl.pas`

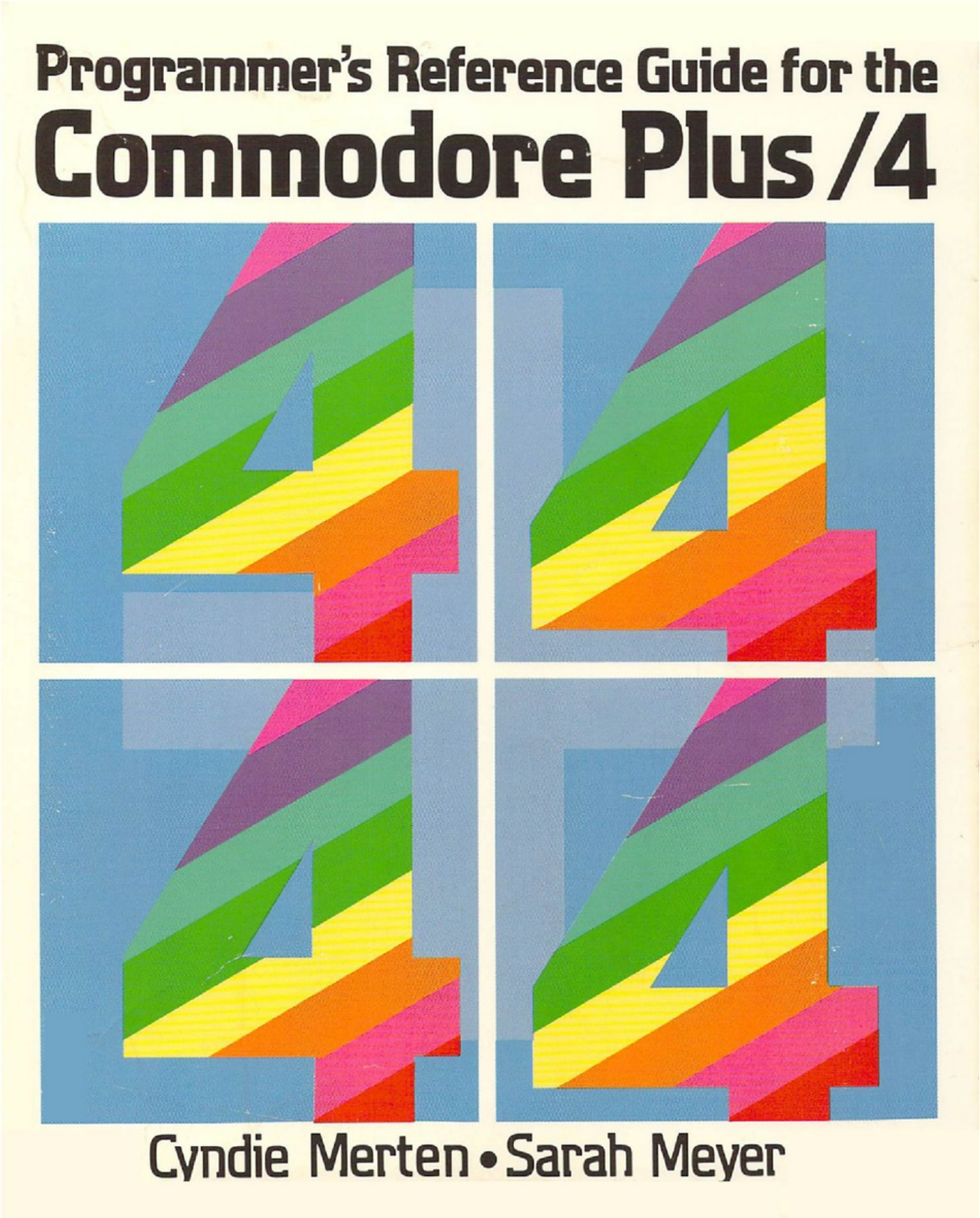
```
{$r hgfx_apl.rc}

uses aplib;

const
  APLBIN = $3000;
  LOGO   = $5800;

var
  SETBITMAP                          : byte absolute $ff06;
  SETMCOLOR                          : byte absolute $ff07;
  BITMAPADDR                         : byte absolute $ff12;
  VIDEOMATRIX                        : byte absolute $ff14;
  BORDER                             : byte absolute $ff19;

begin

  unapl(pointer(APLBIN), pointer(LOGO));

  SETBITMAP := SETBITMAP or $20;
  SETMCOLOR := (SETMCOLOR and $40) or $8;

  // (01011xxx) $5800 = 11 * $800;
  VIDEOMATRIX := %01011000;
  // (xx011xxx) $6000 = 3 * $2000; bit 2 set to 0 means reading from RAM
  BITMAPADDR := %00011000 or (BITMAPADDR and %00000011);

  BORDER := $3d;

  repeat until false;
end.
```

Na koniec może nam się nie podobać, że wynikowy plik ma wciaż około `9KB`, jest tak między innym dlatego że spakowane pliki dołączam od adresu `$3000` a program główny jest krótki i zaczyna się około adrsu `$1000`. Kompilator przestrzeń między tymi adresami wypełni zerami. Dla treningu możemy spakować nasz **prg** jeszcze raz, tym razem **Exomizerem**:

`exomizer sfx sys -n -t 4 -o hires_apl_exo.prg hires_apl.prg`

i otrzymać plik wielkości `1,7KB`.

Literatura



Aby w pełni zrozumieć ninejszy artykuł nieodzowne jest zapoznanie się z **rozdziałem 4** książki [Programmer's Reference Guide for the Commodore Plus 4]