C64 TÜRKİYE

SAYI : #$06 BİLGİ PAYLAŞTIKÇA ARTAR KASIM 2004

"Y" Ekseninde Simetrik Karakter Seti
90° Sola Yatık Karakter Seti
Hafızadaki Bir Byte'ı Ekrana Hex Olarak Yazdırma
Hex Sistem Tuş Kontrolü
1x1 Scroll ve RenkliRaster Şeritleri
Hades Basic #3
Sprite'lar ve Sprite Demo Program
Program Dökümleri

www.c64turkiye.com c64turkiye@yahoo.com hades@hi3s.org

PROGRAM BÖLÜMÜ

Bu sayının program bölümüne hoş geldiniz. İlk olarak C64 için kısa program-larla yeni karakter setleri üretmeye devam ediyoruz. Birinci program C64'ün standart karakterlerini Y ekseninde simetrik olacak şekilde değiştiriyor. (5. Sayıda ise X ekseninde -yatay- simetrik bir karakter seti elde etmiştik.) İkinci programımız yine karakter seti değiştirmeyle ilgili. Fakat ekrandaki yazıları okuyabilmek için monitörü sağa veya sola yatırmanız gerekmektedir. Karakter setleri programları önümüzdeki sayılarda da devam edecek. 1x1 olarak 2-3 tane kadar program kaldı. Daha sonra ise 1x1'lik seti 1x2, 2x1 ve 2x2 olarak değiştirmeye sıra gelecek. Bu arada belirtmekte fayda var. Bu programlar sadece standart karakter seti için kullanılabilir diye bir şart yok. Mesela elinizde 1x1'lik güzel bir karakter seti var. Önce bu seti $3000 adresinden itibaren yükleyin. Daha sonra aşağıdaki programı yükleyin fakat çalıştırmayın. RUN komutu yerine, programda "init" etiketli yere karşılık gelen adres için **SYS 2105** komutunu verin. Böylece başka bir yerden bulduğunuz 1x1'lik bir karakter setinden yeni karakter setleri üretmiş olursunuz. Son olarak 1-2 ufak rutin vererek bu sayıdaki program bölümünü bitiriyoruz. Herkese iyi çalışmalar...

DİKEY –Y EKSENİNDE- SİMETRİK KARAKTER SETİ PROGRAMI
KARAKTER SET NO:3

HAZIRLAYAN : İSMAİL ŞAHİN
İLETİŞİM : HADES@HI3S.ORG

(C64ASM V1.1 ile compile edilecek şekilde yazılmıştır.)

```
                *=$0801

                .word nextline
                .word 2004        ; 2003
                .byte $9e         ; SYS
                .text "2061"      ; 2061
                .byte 0           ; komutu
nextline        .word 0

                sei               ; İnterruptları engelle
                lda   $01         ; Karakter ROM'u
                and   #$fb        ; artık CPU tarafından
                sta   $01         ; okunabilir

                lda   #$d0        ; ROM adresi = $D000
                ldx   #$30        ; RAM adresi = $3000
                ldy   #$00        ; olacak şekilde ayarla.
                sty   $fb         ; $FB ve $FC adreslerinde
                sta   $fc         ; ROM adresi tutuluyor.
                sty   $fd         ; $FD ve $FE adreslerinde
                stx   $fe         ; RAM adresi tutuluyor.

                ldx   #$10        ; Blok sayacı = $10 (16)
transfer        lda   ($fb),y     ; ROM'dan o anki adresi
                sta   ($fd),y     ; oku ve RAM'e kopyala
                iny               ; byte sayacını 1 arttır
                bne   transfer    ; "0" değilse geri dön
                inc   $fc         ; ROM HIGH adresi 1 arttır
                inc   $fe         ; RAM HIGH adresi 1 arttır
                dex               ; Blok sayacını 1 azalt
                bne   transfer    ; "0" olmadıysa geri dön.
```

```
                lda   $01         ; I/O bölgesi eski haline
                ora   #$04        ; getiriliyor. CPU artık
                sta   $01         ; karakter roma erişemez
                cli               ; interruptlara izin ver
init            lda   #$1c        ; VIC'e yeni karakter
                sta   $d018       ; setinin yerini bildir.

new_fonts       lda   #$30        ; Karakter seti başlangıç
                ldy   #$00        ; adresi $3000 olarak
                sty   $fb         ; $FB ve $FC adreslerine
                sta   $fc         ; kaydediliyor

loop3           ldy   #$00        ; Y registeri byte sayacı=0
loop2           ldx   #$08        ; "bit" sayacı
                lda   #$00        ; Geçici adresi
                sta   $02         ; sıfırla
loop0           lda   ($fb),y     ; O anki adresin değerini oku
loop1           ror               ; bir bit sağa döndür - 7. biti carry'e koy
                rol   $02         ; bir bit sola döndür. Carry'i 7. bite koy
                dex               ; bit sayacını 1 azalt
                bne   loop1       ; sıfır olmadıysa tekrarla
                lda   $02         ; geçici adresi oku
                sta   ($fb),y     ; o anki adrese geri yaz
                iny               ; byte sayacını 1 arttır
                bne   loop2       ; sıfır olmadıysa bir byte daha oku

                inc   $fc         ; Karakter setinin HI adresini 1 arttır
next_char       lda   $fc         ; HI adresi al
                cmp   #$40        ; #$40 oldumu? Olmadıysa
                bne   loop3       ; işlemleri tekrarla
                rts               ; Olduysa programı bitir.

                .end
```

Yeni karakter setine ait ekran görüntüsü.

ARANIYOR

Dergi için C64 ile her türlü konuda (haberler, demo, oyun, utility tanıtımı, grafik ve müzik programlama, assembler ve basic, soru-cevap vs...) yardım edebilecek editörler aranıyor. Yardım etmek isteyenler kendilerini tanıtan ve ilgilendiği bölümleri belirten bir e-maili c64turkiye@yahoo.com veya hades@hi3s.org adreslerine atabilirler.

C64 İÇİN 90° SOLA YATIK KARAKTER SETİ PROGRAMI
KARAKTER SET NO:4

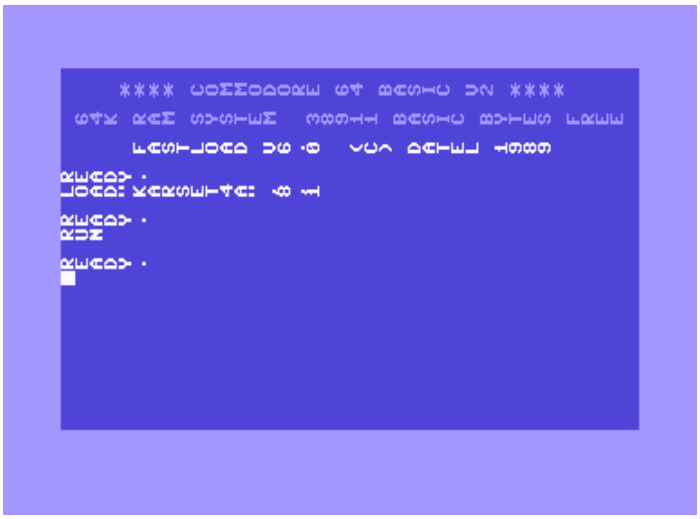
HAZIRLAYAN : İSMAİL ŞAHİN
İLETİŞİM : HADES@HI3S.ORG

(C64ASM V1.1 ile compile edilecek şekilde yazılmıştır.)

```
                *=$0801

                .word nextline
                .word 2004          ; 2004
                .byte $9e           ; SYS
                .text "2061"        ; 2061
                .byte 0             ; komutu
nextline        .word 0

                sei                 ; İnterruptları engelle
                lda   $01           ; Karakter ROM'u
                and   #$fb          ; artık CPU tarafından
                sta   $01           ; okunabilir
                lda   #$d0          ; ROM adresi = $D000
                ldx   #$30          ; RAM adresi = $3000
                ldy   #$00          ; olacak şekilde ayarla.
                sty   $fb           ; $FB ve $FC adreslerinde
                sta   $fc           ; ROM adresi tutuluyor.
                sty   $fd           ; $FD ve $FE adreslerinde
                stx   $fe           ; RAM adresi tutuluyor.
                ldx   #$10          ; Blok sayacı = $10 (16)
transfer        lda   ($fb),y       ; ROM'dan o anki adresi
                sta   ($fd),y       ; oku ve RAM'e kopyala
                iny                 ; byte sayacını 1 arttır
                bne   transfer      ; "0" değilse geri dön
                inc   $fc           ; ROM HIGH adresi 1 arttır
                inc   $fe           ; RAM HIGH adresi 1 arttır
                dex                 ; Blok sayacını 1 azalt
                bne   transfer      ; "0" olmadıysa geri dön.
                lda   $01           ; I/O bölgesi eski haline
                ora   #$04          ; getiriliyor. CPU artık
                sta   $01           ; karakter roma erişemez
                cli                 ; interruptlara izin ver
init            lda   #$1c          ; VIC'e yeni karakter
                sta   $d018         ; setinin yerini bildir.
new_fonts       lda   #$30          ; Karakter seti başlangıç
                ldy   #$00          ; adresi $3000 olarak
                sty   $fb           ; $FB ve $FC adreslerine
                sta   $fc           ; kaydediliyor
loop3           ldy   #$00          ; Y registeri byte sayacı=0
loop1           lda   ($fb),y       ; O anki adresin değerini oku
                ldx   #$00          ; X registeri "ror" işlem sayacı
loop0           clc                 ; carry bitini sıfırla
cfg0            ror                 ; aküdeki değeri 1 bit sağa döndür.
cfg1            rol   temp,x        ; "c" bitini "rol" ile temp'e yaz
                inx                 ; x registerini 1 arttır
                cpx   #$08          ; 8 oldumu?
                bne   loop0         ; olmadıysa tekrarla
                iny                 ; byte sayacını 1 arttır
                cpy   #$08          ; 8 oldumu?
                bne   loop1         ; olmadıysa işlemleri tekrarla.
```

```
                ldy   #$00        ; sayacı sıfırla
loop2           lda   temp,y      ; temp adresindeki byte'ları oku
                sta   ($fb),y     ; o anki karakterin adreslerine yaz
                iny               ; sayacı 1 arttır.
                cpy   #$08        ; 8 oldumu?
                bne   loop2       ; olmadıysa işlemleri tekrarla
                lda   $fb         ; Karakter setinin LO adresini al
                clc               ; Carry bitini sıfırla
                adc   #$08        ; 8 ile topla
                sta   $fb         ; LO adrese yaz
                bcc   next_char   ; elde biti yoksa git
                inc   $fc         ; Karakter setinin HI adresini 1 arttır
next_char       lda   $fc         ; HI adresi al
                cmp   #$40        ; #$40 oldumu? Olmadıysa
                bne   loop3       ; işlemleri tekrarla
                rts               ; Olduysa programı bitir.
temp            .byte 0,0,0,0,0,0,0,0

                .end
```

Program listesinde **cfg0** ve **cfg1** etiketli satırlardaki komutların değiştirilmesiyle 4 farklı karakter seti elde edilebilir. Aşağıda bu komutları ve elde edilen karakter setlerini görebilirsiniz.



cfg0 **ror**
cfg1 **rol** **temp,x**

cfg0 **ror**
cfg1 **ror** **temp,x**

cfg0 **rol**
cfg1 **rol** **temp,x**

cfg0 **rol**
cfg1 **ror** **temp,x**

HAFIZADAN OKUNAN BİR BYTE'I HEX OLARAK EKRANA YAZDIRMA RUTİNİ (1.YÖNTEM)

```
                *=$0801

                .word nextline
                .word 2004        ; 2004
                .byte $9e         ; SYS
                .text "2061"      ; 2061
                .byte 0           ; komutu
nextline        .word 0

mem_read        lda   adres       ; adresteki değeri al
                pha               ; akünün değerini yığına at
                lsr               ; sağa kaydır
                lsr               ; sağa kaydır
                lsr               ; sağa kaydır
                lsr               ; sağa kaydır
                jsr   convert     ; byte çevirme var
                pla               ; yığındaki değeri geri al
                and   #$0f        ; alt 4 biti al
convert         cmp   #$0a        ; okunan değer 10'dan
                bcc   number      ; küçükse rakam demekmiş.
                clc               ; büyükse harf demekmiş.
                adc   #$07        ; önce 7 ile topla
number          clc               ; carry'i sıfırla
                adc   #$30        ; $30 (48) ile topla
print           jsr   $ffd2       ; ekrana yazdır.
                rts               ; Bu iş burada biter.
adres           .byte 179         ; Örnek sayı
```

Şimdi açıklama yapalım. Hex sistemde kullanılan karakterler 0-9 ve A-F arasındadır. Dolayısıyla bunların ASCII değerleri rakamlar için $30-$39 (48-57), harfler için $41-$46 (65-70) arasındadır. Hafızadan okunan bir değeri hex olarak ekrana yazdırmak demek ekrana iki karakter basmak demektir. Önce aküyü kullanarak hafızayı okuyoruz ve aynı değeri tekrar kullanabilmek için yığına atıyoruz. Sonra 4 defa LSR komutuyla 7-6-5-4 nolu bitleri 3-2-1-0 nolu bitlerin yerine taşıyoruz. Bunu yapmak zorundayız. Eğer yapmazsak ekrana istemediğimiz karakterler çıkabilir veya başka işler olabilir. Mesela ekran silinebilir. Devam ediyoruz. 4 LSR komutundan sonra JSR komutu ile çevirme işlemi alt rutinine gidiyoruz. Burada aslında alt rutin yok. Program tek parça ve bir kısmını alt rutinmiş gibi kullanıyoruz. Yani program kendi kendini çağırıyor.

Şimdi çevirme yapmaya başlayabiliriz. Önce aküdeki değeri 10 ile karşılaştırıyoruz. Eğer 10'dan küçükse ekrana rakam basmamız gerekiyor. Bunun için aküye $30 (48) sayısını ekliyoruz. Böylece ekrana basacağımız rakamın ASCII kodunu elde etmiş oluyoruz. JSR $FFD2 komutuyla ise ekrana bir adet rakam basıyoruz. Eğer karşılaştırma sonucu elimizdeki değer 10 ve daha büyükse (en fazla 15 olabilir) bu değeri 65-71 arasına tamamlamak için $37 (55) eklememiz gerekiyor. Önce 7 ile ve sonrasında hemen $30 ile toplayıp A ile F arasındaki bir harfin ASCII kodunu elde ederek ekrana bir adet harf basmış oluyoruz.

Daha sonra RTS komutuyla geri dönerek JSR CONVERT komutundan sonraki komuta geliyoruz. Yığındaki değeri geri alarak ve AND #$0F komutuyla sadece 7-6-5-4 nolu bitleri sıfırlayıp 3-2-1-0 nolu bitleri dikkate alıyoruz. Bundan sonraki işlemler ikinci paragraftaki anlattıklarımızın aynısı. Tek fark RTS komutuyla işimizi bitiriyoruz.

Örnek vererek anlattıklarımızı doğrulayalım. Adresteki sayımız 179 ($B3) olsun. Ekrana "B3" yazdırmamız gerekiyor. İlk olarak $B3 sayısını binary (ikili) sistemde yazalım. Bunun için önce $B (11) sayısının binary sistemdeki karşılığı olan **1011** sayısını ve hemen yanına ise 3 sayısının binary sistemdeki karşılığı olan **0011** yazalım. Demekki $B3=%10110011 oluyormuş. Devam edelim. İlk olarak aküye 4 LSR komutunu uygulayacağız.

```
   SAYI   LSR          LSR          LSR          LSR
10110011  → 01011001  →  00101100  → 00010110  →  00001011
```

Gördüğünüz gibi 4 LSR komutu ile sayının 7654 nolu bitleri birer sağa kayarak 3210 numaralı bitler haline geldi. Yani aküde artık **%00001011** sayısı var.

Sıradaki JSR komutu ile bir gezintiye çıkalım. Önce CMP #$0A ile karşılaştırma yapalım ve BCC NUMBER komutuna gelelim. BCC komutu ile daha önceki CMP komutunun sonucuna göre işlem yapıyoruz ve aküdeki değer, karşılaştırma yaptığımız değerden küçükse NUMBER kısmına gidiyoruz. Fakat burada aküdeki değer $B (11) olduğu için bir sonraki komuta geçiyoruz. Önce CLC ile "carry" bitini sıfırlayalım. Sonra ADC #$07 ile aküdeki değeri $7 ile toplayalım. Sonuç $12 (18) olur ve yine aküde tutulur. Devam ediyoruz. Bir CLC ve ADC #$30 ile aküdeki değer $12 + $30 = $42 olur. Daha sonra JSR $FFD2 komutu ile aküdeki değeri ekrana basıyoruz. Acaba gerçekten doğru mu? Eğer ASCII kodlara bakarsanız $42="B" demektir. Bu arada CLC kullanmazsak ne olur? Sonuç yanlış çıkabilir. ADC komutu şöyle çalışır.

AKÜDEKİ DEĞER + CARRY BİTİ + EKLENECEK SAYI = SONUÇ tekrar aküde tutulur. Buna göre CLC komutuyla Carry biti 0 yapıldıktan sonra toplama yapılırsa $0B + 0 + $07 = $12 ve devam edersek $12 + 0 + $30 = $42 olur. Eğer CLC kullanmazsak sonuç aşağıdaki gibi olur.

$0B + x + $7 = SONUÇ → Burada "x" 0 veya 1 olabilir ve sonuç $12 veya $13 olur. Bunun sonucunda programımız istediğimiz gibi çalışmaz.

Bu kısa bilgiden sonra RTS komutuyla kaldığımız yerden devam ediyoruz ve geri dönüş yaparak JSR'den sonraki komutu işletiyoruz. Aküye tekrar adresteki değeri yani $B3'ü yüklüyoruz. AND #$0F ile 7654 nolu bitleri sıfırlayarak sadece 3210 nolu bitleri dikkate alıyoruz. Bu arada AND komutuyla ilgili olarak bilgilerimizi tazeleyelim. AND komutu ile işleme sokulan bitlerin değerleri "1" ise sonuç "1", herhangi bir bit "0" ise sonuç "0" olur. Yani **0 AND 0 = 0, 0 AND 1 = 0, 1 AND 0 = 0** ve **1 AND 1 = 1** olur. Bu bilgilerin ışığında bizim AND #$0F ile nasıl bir sonuç elde ediyoruz inceleyelim.

AKÜ → $B3 → %10110011
SAYI → $0F → %00001111
SONUÇ → $03 → %00000011 → Sonuç aküde tutulur.

Devam ediyoruz ve tekrar CMP #$0A ve BCC NUMBER komutlarına geldik. BCC komutu programda karşılaştırma sonrası "**...den küçükse**" şartı için kullanılmaktadır ve burada $0A'dan küçükse NUMBER kısmına gidilecektir. Aküdeki sayımız $03 ve dolayısıyla $0A'dan küçüktür. NUMBER kısmında CLC ve ADC #$30 komutları ile aküyü $30 ile toplayıp $33 sayısını elde ediyoruz ve JSR $FFD2 ile ASCII kodu $33 sayısı olan "3" rakamını ekrana yazdırıyoruz. Son olarak RTS komutuyla işimizi bitiriyoruz.

Ön bilgi:

BCC → Küçükse;
BCS → Büyük veya eşitse;
BEQ → Eşitse;
BNE → Eşit değilse;

şartlarının kontrolü için kullanılır.

HAFIZADAN OKUNAN BİR BYTE'I HEX OLARAK EKRANA YAZDIRMA RUTİNİ (2.YÖNTEM)

Bazen bir problemin birden fazla çözümü olabilir. Bu nedenle aynı işi yapan bir rutin daha veriyoruz. Fakat bu rutinde, önceki gibi karşılaştırma, toplama vs.. gibi işlerle uğraşmayacağız.

```
                *=$0801

                .word nextline
                .word 2004        ; 2004
                .byte $9e         ; SYS
                .text "2061"      ; 2061
                .byte 0           ; komutu
nextline        .word 0

mem_read        lda   adres       ; hafızadan bir byte oku
                pha               ; yığına at
```

```
                lsr             ; sağa kaydır
                lsr             ; sağa kaydır
                lsr             ; sağa kaydır
                lsr             ; sağa kaydır
                jsr   convert   ; çevirme rutinine uğra
                pla             ; yığındaki değeri geri al
                and   #$0f      ; üst 4 biti sıfırla alt 4 biti kullan
convert         tax             ; aküdeki değeri x registerine kopyala
                lda   tablo,x   ; tablodan X registerinin gösterdiği byte'ı al
                jsr   $ffd2     ; ekrana yazdır
                rts             ; geri dön.

tablo           .text "0123456789ABCDEF"
adres           .byte 179
```

Bu arada 1. yöntemdeki rutin 29 byte, 2. yöntemdeki rutin 39 byte uzunluğundadır. Anlaşılabilirlik yönünden 2. yöntem daha iyidir. Bazı durumlarda tablo kullanmak kısa ve karışık bir program yazmaktan daha avantajlı olabilir.

KLAVYEDEN HEX SİSTEM TUŞLARININ KONTROLU (1.YÖNTEM) (SADECE "0-9" VE "A-F" TUŞLARI)

```
                *=$0801

                .word nextline
                .word 2004      ; 2004
                .byte $9e       ; SYS
                .text "2061"    ; 2061
                .byte 0         ; komutu
nextline        .word 0

key_read        jsr   $ffe4     ; klavyeyi kontrol et
                cmp   #$30      ; "0" tuşundan
                bcc   key_read  ; küçükse tekrar dene
                cmp   #$3a      ; ":" tuşundan
                bcc   print     ; küçükse yazdır.
                cmp   #$41      ; "A" tuşundan
                bcc   key_read  ; küçükse tekrar dene
                cmp   #$47      ; "G" tuşuna
                bcs   key_read  ; eşitse veya büyükse tekrar dene
print           jsr   $ffd2     ; basılan tuşu ekrana yazdır.
                rts
```

Yukarıdaki kısa rutin ile klavyeden sadece hex sistemde kullanılan tuşlara basılıp basılmadığını kontrol edilmektedir. Önce JSR $FFE4 komutu ile klavyeyi kontrol ediyoruz. Sonra CMP #$30 BCC KEY_READ komutlarıyla ASCII kodu 0 ile $30 arasında olup olmadığına bakıyoruz. Eğer bu tuşlardan birine basıldıysa yani "**boşluk ! " # $ % & ' ( ) * + , - . /**" tuşlarından birine bastıysak hiçbir işlem yapmadan klavyeyi tekrar kontrol ediyoruz. Daha sonra CMP #$3A BCC PRİNT komutlarıyla basılan tuşların "**0 1 2 3 4 5 6 7 8 9**" tuşlarından biri olduğu anlıyoruz ve ekrana yazdırarak programdan çıkıyoruz. Burada $3A sayısı ":" tuşunun kodu olup "**9**" tuşunun kodundan sonra gelmektedir. Matematiksel bir anlatımla **$2F < TUŞ < $3A** şartını kontrol ediyoruz. Yani "/" < TUŞ < ":". Devam ediyoruz ve eğer yukarıdaki şart sağlanmadıysa bu sefer CMP #$41 BCC KEY_READ komutlarıyla **": ; < = > ? @"** tuşlarından birine basıldığını anlıyoruz ve işimize yaramadığı için tekrar klavyeyi kontrol ediyoruz. Eğer bu tuşlarada basılmadıysa son kez CMP #$47 BCS KEY_READ komutları ile **$40 < TUŞ ≥ $47 ("@" < TUŞ ≥ "G")** arasındaki tuşları kontrol ediliyor. Eğer bu şart gerçekleşmişse basılan tuşu ekrana yazdırıp programdan çıkıyoruz. Değilse klavye bir daha kontrol ediliyor.

KLAVYEDEN HEX SİSTEM TUŞLARININ KONTROLU (2.YÖNTEM) (SADECE "0-9" VE "A-F" TUŞLARI)

```
                *=$0801

                .word nextline
                .word 2004          ; 2004
                .byte $9e           ; SYS
                .text "2061"        ; 2061
                .byte 0             ; komutu
nextline        .word 0

key_read        jsr   $ffe4         ; klavyeyi kontrol et
                ldx   #$00          ; tablo sayacını sıfırla
control         cmp   tablo,x       ; aküdeki değeri tablodaki değerle karşılaştır.
                bne   next          ; aynı değilse sonraki iş için git
                jsr   $ffd2         ; aynı ise basılan tuşu ekrana yazdır.
                rts                 ; programdan çık

next            inx                 ; tablo sayacını 1 arttır
                cpx   #$10          ; tablonun uzunluğu ile karşılaştır.
                bne   control       ; eşit değilse karşılaştırmayı tekrarla
                jmp   key_read      ; Tablodaki olmayan bir tuşa basıldığı için tekrarla.

tablo           .text "0123456789ABCDEF"
```

Tuşların kontrolünü isterseniz her tuş için ayrı ayrı CMP #$TUSKODU yazarak ta yapabilirsiniz.

```
key_read        jsr   $ffe4
                cmp   #$30
                beq   print
                cmp   #$31
                beq   print
                ..............
                ..............
                jmp   key_read
print           jsr   $ffd2
                rts
```

Yukarıdaki rutin, daha anlaşılır olması avantajına rağmen daha uzundur. Bir tuş için 4 byte kullanılmaktadır ve sadece tuşlar için 64 byte harcanmaktadır. Geri kalan komutlar ise 10 byte tutmakta ve 74 byte ile işimiz görülmektedir. Bu rutinin bir avantajı daha vardır. Mesela programdan çıkış için "space" tuşunu kullanmak istiyorsunuz diyelim. Bunun için yapmanız gereken CMP #$20 BEQ ÇIKIŞ komutlarını diğer tuş kontrollerinin olduğu yere eklemek ve program içindeki bir RTS komutunu ÇIKIŞ etiketi ile göstermek. Daha genel bir ifadeyle yukarıdaki rutin istediğiniz tuşların kontrolü için kullanılabilir.

(NOT : EVLİLİK BİLGİSAYAR BAĞIMLILARI İÇİN TAVSİYE EDİLMEZ ☺☺)

İNTRO TEKNİKLERİ

1x1 SCROLL VE RENKLİ RASTER ŞERİTLERİ

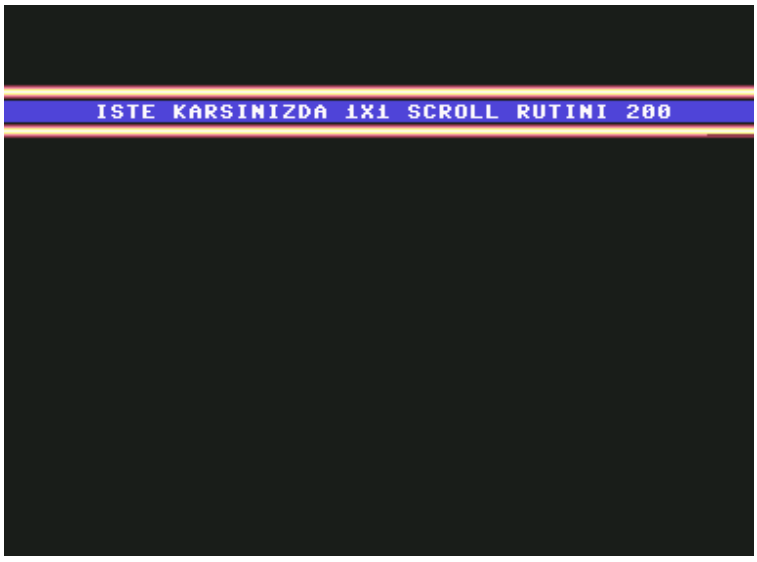
HAZIRLAYAN : İSMAİL ŞAHİN
İLETİŞİM : HADES@HI3S.ORG

Bu sayıdan itibaren intro programcısı olmak için çalışmalara başlıyoruz. İlk konumuz öncelikle neredeyse her introda karşımıza çıkan "text scroll" ve "color raster" efektleri. Bir intro yapabilmek için en başta iyi bir assembler bilgisi, C64'ün donanım adreslerini iyi bilmek ve birazda dizayn yeteneği gereklidir. Bunun yanında elinizin altında çeşitli karakter setleri, müzikler, sprite'lar, logolar ile çeşitli editörler (karakter, sprite, logo, grafik editörleri vs...) bulunmalıdır.

Programımız standart C64 karakter setini kullanmaktadır ve müzik yoktur. (C64ASM V1.1 ile compile edilecek şekilde yazılmıştır.)

```
                *=$0801
                .word nextline
                .word 2003          ; 2003
                .byte $9e           ; SYS
                .text "2061"        ; 2061
                .byte 0             ; komutu
nextline        .word 0
start           sei                 ; interruptları engelle
                lda   #$01          ; Yazı rengi beyaz olacak şekilde
                jsr   $e536         ; ekranı sil.
                lda   #$07          ; kayma miktarının ilk değerini
                sta   xpos          ; "xpos" adresine yaz
                lda   #<text0       ; Scroll için text'in hafıza adresini
                ldx   #>text0       ; LO HI şeklinde
                sta   tbase+1       ; programdaki karakter
                stx   tbase+2       ; okuma kısmına yaz.
                lda   #<int0        ; kendi interrupt rutinimizin
                ldx   #>int0        ; başlangıç adresini
                sta   $0314         ; interrupt vektörlerine
                stx   $0315         ; kopyala
                lda   #$00          ; Bu komutlar mutlaka
                ldx   #$01          ; olmalıdır ve
                ldy   #$7f          ; standarttır.
                sta   $dc0e         ; CIA1 zamanlayıcılarını sıfırla
                stx   $d01a         ; Raster interrupt enable olacak
                sty   $dc0d         ; CIA1 interrupt kontrol registeri
                cli                 ; interruptları serbest bırak
                jmp   *             ; normal kod sonsuz döngü olsun
int0            inc   $d019         ; Bir sonraki interrupta izin ver
                lda   xpos          ; Kayma miktarını
                sta   $d016         ; VIC kontrol (scroll) registerine ver.
                lda   #$38          ; interrupt yapılacak raster değerini
raster0         cmp   $d012         ; VIC'in raster registeriyle karşılaştır.
                bne   raster0       ; eşit değilse tekrar karşılaştır.
                ldx   #$00          ; Eşit olmuş ve sayıcıyı sıfırlayalım.
loop0           lda   colortab,x    ; Renk tablosundan o anki renk değerini al
                sta   $d020         ; VIC'in ekran dış renk registerine yaz.
                sta   $d021         ; VIC'in ekran iç renk registerine yaz.
                ldy   delaytab,x    ; Gecikme tablosundan o anki değeri al
delay0          dey                 ; değeri 1 azalt
                bne   delay0        ; "0" olmadıysa tekrar azaltma yap.
                inx                 ; Sayacı 1 arttır.
                cpx   #$13          ; $13 (19) olmuşmu?
                bne   loop0         ; olmadıysa işlemleri tekrarla.
```

```
                lda   #$08        ; VIC'in scroll registerine
                sta   $d016       ; normal değerini yaz.
wait            lda   $d012       ; Raster registerini oku.
                bne   wait        ; "0" oluncaya kadar bekle.
                jsr   scroll      ; Scroll rutinini git.
                jmp   $febc       ; İnterrupt rutininden çık.

scroll          dec   xpos        ; Kayma miktarını 1 azalt
                bpl   exit        ; Sonuç pozitif ise (0-127 arasındaysa) çık.
                lda   #$07        ; Negatif ise (128-255 arasındaysa)
                sta   xpos        ; kayma miktarını 7 yap
                ldx   #$00        ; Karakter sayacı=0
scr             lda   $0451,x     ; Ekrandaki karakterleri
                sta   $0450,x     ; sola kaydır
                inx               ; sayacı 1   arttır.
                cpx   #$27        ; 39 oldumu?
                bne   scr         ; olmadıysa tekrarla
tbase           lda   text0       ; hafızadan yeni karakter değerini oku
                bne   nextchar    ; Okunan değer 0 (yazı sonu) değilse git
                lda   #<text0     ; Hafızadaki yazının
                ldx   #>text0     ; başlangıç adresini
                sta   tbase+1     ; programdaki karakter okuma
                stx   tbase+2     ; kısmına yaz
                rts               ; ana rutine geri dön

nextchar        sta   $0477       ; yeni karakteri 39. Sütuna yaz.
                inc   tbase+1     ; karakter okuma adresinin LO byte'ını 1 arttır.
                bne   exit        ; 0 olmadıysa çık.
                inc   tbase+2     ; olduysa HI byte'ı 1 arttır.
exit            rts               ; Ana rutine geri dön.

text0           .scrl "iste karsinizda 1x1 scroll rutini 2003 hades.. "
                .byte 0           ; Bu sayı yazı sonunu gösterir ve mutlaka olmalıdır.

colortab        .byte 0,2,10,7,1,7,10,2,0,006,0,2,10,7,1,7,10,2,0    ;renk tablosu
delaytab        .byte 6,8,08,1,7,8,8,8,9,117,8,8,8,8,9,8,7,1,9,1     ;gecikme tablosu
xpos            .byte 7           ; kayma miktarı

                .end
```



Ve işte karşınızda 1x1 "text scroll" ve "color raster" programının ekran görüntüsü.....

HADES BASIC #3

HAZIRLAYAN : İSMAİL ŞAHİN
İLETİŞİM : HADES@HI3S.ORG

HADES BASIC'in son bölümüne hoş geldiniz. Bu bölümde 6 adet yeni komut ekliyoruz. Aşağıda komutların neler olduğunu, nasıl kullanıldığını bulabilirsiniz.

1 – MIRROR : Ekran görüntüsünü yatay veya dikey olarak çevirebilirsiniz. Ekranda eğer yazı varsa çevirme işleminde sorun olmayacaktır. Eğer tuşlardaki grafikleri kullanarak bir şeyler çizmişseniz şekliniz bozuk çıkacaktır.
MIRROR H : Ekranı yatay olarak çevirir.
MIRROR V : Ekranı dikey olarak çevirir.

2 – INV : Bu komutla ekrana yazacağınız yazıları inverse (negatif) olarak yazdırabilirsiniz. Komutu INV : PRINT "....." şeklinde kullanırsanız yazdıklarınız ekrana negatif olarak çıkar. PRINT komutunu ikinci kez kullandığınızda yazılar normal olarak ekrana çıkar. Yani INV komutu sadece kendinden sonra gelen PRINT komutu için yazıları negatif hale getirir.

3 – FONT x : Hafızaya yüklediğiniz bir fontu aktif hale getirmek için kullanılır. X değeri 0 ila 7 arasında olmalıdır. POKE 53272 yerine kullanılır. Eğer "X" değeri kullanılmazsa yani sadece FONT olarak kullanılırsa default rom karakter seti seçilmiş olur. Aşağıda x değerlerine karşılık gelen font adresleri verilmiştir.

X=0 → $0000-$07FF
X=1 → $0800-$0FFF
X=2 → $1000-$17FF
X=3 → $1800-$1FFF
X=4 → $2000-$27FF
X=5 → $2800-$2FFF
X=6 → $3000-$37FF
X=7 → $3800-$3FFF

FONT 2 komutu default rom karakter setini seçer. Yani $1000-$1800 adreslerine bir karakter seti yüklerseniz FONT 2 komutu ile seçemezsiniz.

4 – SPEED x : Kursörün yanıp sönme hızını ayarlayabilirsiniz. X değeri 0 ila 255 arasında olmalıdır. X değeri verilmezse default yanıp sönme hızına geri dönülür. Default değer 33'tür. Space tuşuna basılı tutarak cursörün hızını kontrol edebilirsiniz.

5 – SCREEN x : Ekranın açık veya kapalı olmasını sağlar. X değeri 0 veya 1 olmalıdır. SCREEN 0 : Ekran kapalı SCREEN 1 : Ekran açık

6 – DELAY x : Program içinde x değeri kadar milisaniye gecikme sağlar. X=0 iken en uzun gecikme sağlanır (256 msn).

C64'ün esnek bir işletim sistemine sahip olması nedeniyle yeni komut eklemek çok kolay olmaktadır. Yeni bir komut eklemek için önce komutun yapacağı işi normal bir program olarak yazın. Programınızın çalıştığından emin olduktan sonra gerekli adresleri kullanarak parametreleri programa aktarın. Gerekiyorsa parametre sınır değerlerini kontrol edin. Bir kez alıştıktan sonra istediğiniz komutu ekleyebilirsiniz. Mesela aklımda Basic içinden interrupt kullanarak 1x1 text scroll yapan bir komut eklemek vardı. Program içinde SCROLL satır,hız,"text" komutunu kullanarak hiç makine dili bilmeden smooth scroll yapmak mümkün olacaktı.
Gevezeliği bir tarafa bırakıp aşağıdaki programı satır satır yazmaya başlayın. Kimbilir belki C64'e ekleyecek 1-2 yeni komut aklınıza gelebilir.

```
                *=$c000

                lda   #<yeni
                ldx   #>yeni
                sta   $0308
                stx   $0309
                rts
yeni            lda   $7a
                ldx   $7b
                sta   $fb
                stx   $fc
                lda   #<comset
                ldx   #>comset
                sta   $fd
                stx   $fe
                ldx   #$00
                ldy   #$00
loop0           jsr   $0073
                sta   $02
loop1           lda   ($fd),y
                cmp   $02
                bne   nextcom
                iny
                tya
                cmp   comlen,x
                bne   loop0
exec            txa
                asl
                tax
                lda   comexe+1,x
                pha
                lda   comexe,x
                pha
                rts
nextcom         lda   comlen,x
                clc
                adc   $fd
                sta   $fd
                bcc   next0
                inc   $fc
next0           inx
                cpx   commax
                bne   loop1
                lda   $fb
                ldx   $fc
                sta   $7a
                stx   $7b
                jmp   $a7e4
;-------------------------------------------------------------------------------------;
commax          .byte 6
comlen          .byte 5,3,3,5,6,5
comset          .text "mirr"            ; MIRR
                .byte $b0               ; OR
                .text "inv"             ; INV
                .byte $46,$91,$54       ; F "ON" T
                .text "speed"           ; SPEED
                .text "screen"          ; SCREEN
                .text "delay"           ; DELAY
comexe          .word mirror-1
                .word inv-1
                .word font-1
                .word speed-1
                .word screen-1
                .word delay-1
```

```
mirror          jsr   $73             ; parametre varmı kontrol et
                bne   mir0             ; varsa git.
mir_err         jmp   $af08            ; yoksa "syntax error" yazdır
mir0            cmp   #$56             ; parametre "V" mi?
                beq   vertical         ; evetse "vertical" kısmına git
                cmp   #$48             ; değilse kontrol et. "H" mi?
                bne   mir_err          ; değilse hata yazdır. Evet ise devam et.

horizontal      lda   #$00             ; Akü=0
                ldx   #$18             ; X=0
                sta   up               ; Aküdeki değer "UP" değişkenine
                sta   byte             ; Aküdeki değer "BYTE" değişkenine
                stx   down             ; X'teki değer "DOWN" değişkenine

one_line        ldy   byte             ; Y registerine "byte" değişkenini yükle
horiz           ldx   up               ; X registerine "up" değişkenini yükle
                jsr   scrcalc          ; ekran ve renk belleği adreslerini hesapla
                lda   $fb              ; $FB adresindeki değeri al
                sta   $fd              ; $FD adresine kopyala
                lda   $fc              ; $FC adresindeki değeri al
                sta   $fe              ; $FE adresine kopyala
                lda   $02              ; $02 adresindeki değeri al
                sta   $04              ; $04 adresine kopyala
                lda   $03              ; $03 adresindeki değeri al
                sta   $05              ; $05 adresine kopyala
                lda   ($fd),y          ; ekrandaki karakteri al
                pha                    ; yığına at
                lda   ($04),y          ; renk belleğindeki değeri al
                pha                    ; yığına at
                ldx   down             ; X registerine "down" değişkenini yükle
                jsr   scrcalc          ; ekran ve renk belleği adreslerini hesapla
                lda   ($02),y          ; renk belleğindeki değeri al
                sta   ($04),y          ; kopyala
                lda   ($fb),y          ; ekrandaki karakteri al
                sta   ($fd),y          ; kopyala
                pla                    ; yığındaki değeri al
                sta   ($02),y          ; renk belleğine yaz
                pla                    ; yığındaki değeri al
                sta   ($fb),y          ; ekrana yaz
                iny                    ; Y registerinin içeriğini 1 arttır
                cpy   #$28             ; $28 (40) oldu mu ?
                bne   horiz            ; olmadıysa tekrarla.
```

Yukarıdaki kısımda değişkenlere ilk değerleri yüklendikten sonra, ilk önce X registerine yüklenen "UP" değişkeninin değerine göre (ilk satır için) ekran ve renk belleklerinin adresi hesaplanıyor. Bu adresler yedekleniyor. Daha sonra "BYTE" değişkeninin değerine göre ekran belleğindeki ve renk belleğindeki değerler okunup yığına atılıyor. Sonra X registerine "DOWN" değişkeni yükleniyor ve (en alt satır için) ekran ve renk belleği adresleri elde ediliyor. Daha sonra ekran ve renk belleğindeki değerler okunuyor. Bu değerler daha önce yedeklediğimiz adreslere yazılıyor. Daha sonra ise yığındaki değerler geri alınıp o anki renk ve ekran belleğine geri yazılıyor. Böylece en üst ve en alttaki satırların ilk byte'ları yer değiştirmiş oluyor. Y registerinin değeri 1 arttırılıp 40 olup olmadığı kontrol ediliyor. 40 olmadıysa aynı satırların bir sonraki byte'ları için işlemler tekrarlanıyor.

```
                ldy   #$00             ; Y registerini sıfırla
                sty   byte             ; "BYTE" değişkenine kopyala
                inc   up               ; "UP" değişkenini 1 arttır
                dex                    ; X registerini 1 azalt.
                stx   down             ; "DOWN" değişkenine kopyala
                cpx   #$0c             ; X= $0c mi? (12-yani ekranın ortasındaki satır)
                bne   horiz            ; olmadıysa işlemleri yeni satır için tekrarla.
                jmp   $a7e4            ; olduysa kontrolü Basic Rom'a devret
```

```
scrcalc         lda   $ecf0,x          ; Rom'daki ekran belleği LOW adres değerini al
                sta   $fb              ; $FB adresine kopyala
                sta   $02              ; $02 adresine kopyala
                lda   scr_hi,x         ; Tablodan ekran belleği HIGH adres değerini al
                sta   $fc              ; $FC adresine kopyala
                clc                    ; "carry" bitini sıfırla
                adc   #$d4             ; #$D4 ile topla
                sta   $03              ; $03 adresine kopyala (Renk belleği HI byte)
                rts                    ; geri dön
;-------------------------------------------------------------------------------------;
vertical        ldx   #$00             ; X=0
                lda   #$04             ; AKÜ=4
                stx   $fb              ; X'in değerini $FB adresine yaz
                sta   $fc              ; Akünün değerini $FC adresine yaz
                stx   $fd              ; X'in değerini $FD adresine yaz
                clc                    ; elde bitini sıfırla
                adc   #$d4             ; #$D4 ile topla ve renk bellek adresini elde et
                sta   $fe              ; $FE adresine yaz
                ldx   #$00             ; X=0 → Satır sayacı olarak kullanılacak
vert1           lda   #$27             ; AKÜ=27
                sta   right            ; sağ değişkenine yaz
                ldy   #$00             ; Y=0
vert0           sty   left             ; sol değişkenine yaz
                ldy   right            ; sağ değişkenini al
                lda   ($fb),y          ; ekran belleği sağdaki adresi oku
                pha                    ; yığına at
                lda   ($fd),y          ; renk belleği sağdaki adresi oku
                pha                    ; yığına at
                ldy   left             ; sol değişkenini al
                lda   ($fd),y          ; renk belleği soldaki adresi oku
                ldy   right            ; sağ değişkenini al
                sta   ($fd),y          ; renk belleği sağdaki adrese yaz
                ldy   left             ; sol değişkenini al
                lda   ($fb),y          ; ekran belleği soldaki adresi oku
                ldy   right            ; sağ değişkenini al
                sta   ($fb),y          ; ekran belleği sağdaki adrese yaz
                ldy   left             ; sol değişkenini al
                pla                    ; yığından geri al
                sta   ($fd),y          ; renk belleği soldaki adrese yaz
                pla                    ; yığından geri al
                sta   ($fb),y          ; ekran belleği soldaki adrese yaz
                dec   right            ; sağ değişkenini 1 azalt
                iny                    ; y'yi (sol değişkeni) arttır
                cpy   #$14             ; 20 mi?
                bne   vert0            ; değilse tekrarla
                lda   $fb              ; ekran belleği adresi LOW byte'ı al
                clc                    ; elde bitini sıfırla
                adc   #$28             ; #$28 (40) ile topla
                sta   $fb              ; Adrese geri yaz
                sta   $fd              ; $FD adresine yaz
                bcc   no_hi            ; "Elde" yoksa git
                inc   $fc              ; $FC adresindeki değeri 1 arttır
                inc   $fe              ; $FE adresindeki değeri 1 arttır
no_hi           inx                    ; X registerini 1 arttır
                cpx   #$19             ; $19 (25) oldu mu? (25 satır oldu mu?)
                bne   vert1            ; olmadıysa yeni satır için işlemleri tekrarla
                jmp   $a7e4            ; Kontrolü Basic Rom'a devret.

scr_hi          .byte 4,4,4,4,4,4,4    ; Ekrandaki her satırın adresinin
                .byte 5,5,5,5,5,5      ; HIGH byte'larının bulunduğu tablo
                .byte 6,6,6,6,6,6,6
                .byte 7,7,7,7,7
up              .byte 0                ; Yukarı değişken sayaç adresi
down            .byte 0                ; Aşağı değişken sayaç adresi
```

```
left            .byte 0             ; Sol değişken sayaç adresi
right           .byte 0             ; Sağ değişken sayaç adresi
byte            .byte 0             ; Satırdan okunacak byte için sayaç
;-------------------------------------------------------------------------------;
inv             lda   #$12          ; "inverse" özelliğinin kodu
                jsr   $ffd2         ; Aktif hale getir
                jmp   $a7e4         ; Kontrolü Basic Rom'a devret
;-------------------------------------------------------------------------------;
font            jsr   $73           ; Komuttan sonra bir karakter var mı
                beq   font1         ; yoksa git
                jsr   $b79e         ; Evet var. Sayıyı veya değişkeni al
                cpx   #$08          ; sonucu 8 ile karşılaştır
                bcs   font_err      ; 8'e eşitse veya 8'den büyükse hata komutu
                txa                 ; X registerini Akü'ye transfer et
                asl                 ; 2 ile çarp
                clc                 ; Elde bitini sıfırla
                adc   #$10          ; #$10 (16) ile topla
                .byte $2c           ; Bilgisayarı kandırma sayısı ☺
font1           lda   #$14          ; Default Rom Karakter Set seçme değeri
                sta   $d018         ; $D018 kontrol registerine yaz
                jmp   $a7ae         ; Kontrolü Basic Rom'a devret
font_err        jmp   $b248         ; "illegal quantity error" mesajını yazdır.
;-------------------------------------------------------------------------------;
speed           jsr   $73           ; Komuttan sonra bir karakter var mı?
                beq   norm          ; yoksa git
                jsr   $b79e         ; Evet var. Sayıyı veya değişkeni al
                .byte $2c           ; Bilgisayarı kandır
norm            ldx   #$33          ; Normal değer.
                stx   $dc05         ; İlgili registere yaz.
                jmp   $a7ae         ; Kontrolü Basic Rom'a devret.
;-------------------------------------------------------------------------------;
screen          jsr   $b79b         ; Komuttan sonraki parametreyi al
                cpx   #$02          ; 2 ile karşılaştır
                bcs   scr_err       ; büyük veya eşitse hata mesajını yazdır
                lda   $d011         ; Ekran kontrol registerini oku
                cpx   #$01          ; X registerinin değeri 1
                beq   on            ; ise ekran açılacak.
off             and   #$ef          ; değilse ekranı kapat.
                .byte $2c           ; bilgisayarı kandır
on              ora   #$10          ; ekranı aç
                sta   $d011         ; ekran kontrol registerine yaz
                jmp   $a7ae         ; Kontrolü Basic Rom'a devret.
scr_err         jmp   $b248         ; "illegal quantity error" mesajını yazdır.
;-------------------------------------------------------------------------------;
delay           jsr   $b79b         ; Komuttan sonraki parametreyi al.
delay1          jsr   $eeb3         ; Romdaki 1ms gecikme rutinine git
                dex                 ; X registerini 1 azalt.
                bne   delay1        ; 0 olmadıysa tekrarla
                jmp   $a7ae         ; Kontrolü Basic Rom'a devret.
;-------------------------------------------------------------------------------;
                .end                ; Programın sonu.
```

C64'e yeni komutlar eklediğimiz bu yazı dizisi burada sona eriyor. Program döküm ekinde listeyi bulabilirsiniz. Bir sonraki sayının program döküm ekinde ise HADES BASIC'in tamamını vereceğim. Ayrıca yeni komutlarla ilgili örnek programlar yer alacak.

"SPRITE" LAR

SPRITE NEDİR ?

SPRITE ekrandaki yazılardan, şekillerden bağımsız donanım kaynaklı programlanabilir grafiklerdir. Kullanımı tamamen kullanıcıya kalmıştır. Yani ekranın istediğiniz yerinde gösterebilir, renklerini değiştirebilir, X ve Y eksenlerinde genişletebilir, istediğiniz şekile sokabilir, karakterlerin arkasında veya önünde gösterebilir ve animasyon yapabilirsiniz. Bütün bunları yapabilmeniz için ise bir sürü POKE komutuna ihtiyacınız olacaktır. Eğer BASIC kullanıyorsanız işiniz biraz kolay sayılır. Gerektiğinde içinde sprite komutları bulunan "basic extension" –mesela HADES BASIC- programlarını kullanabilirsiniz. Ama böyle bir programınız yoksa veya basic yerine assembler kullanıyorsanız bütün adresleri ezbere bilmeniz gerekecektir. C64'te fabrika çıkışı olarak 8 tane sprite olsa bile bazı demolarda onlarca hatta yüzlerce sprite'ı görebilirsiniz. Şu an için ekranda gösterilen en son sprite sayısı kaç bilmiyorum. Öylesine yazdığım bir code ile 96 tane sprite gösterdiğimi hatırlıyorum.

Bu arada not düşelim. Ekranda 8'den fazla sprite gösterme tekniğine MULTIPLEX denmektedir ve zamanı gelince "intro teknikleri" bölümünde açıklayacağım. Ayrıca sprite'ların "renkli raster şeritleri" efektiyle arası pek iyi değildir. Daha doğrusu ekranda raster şeritleri varken, şeritlerin üzerinde bir sprite göstermek isterseniz bütün zamanlamalarınız alt üst olur. Bu kısa bilgilerden sonra devam edelim.

SPRITE DATALARI NEREDE BULUNUR ?

Sprite'ların programlanabilir grafikler olduğunu biliyoruz. Dolayısıyla bu grafiklerin hafızada bir yerlerde bulunması gerekmektedir. Derginin 4. sayısında hatırlarsanız programlama bölümünde karakter setini anlatırken "BANK" lardan bahsetmiştik. Normalde C64 açıldığında $0000-$3FFF adresleri arasında bulunan BANK 0'ı kullanır. Bu nedenle spritelerin grafikleri de bu adresler arasında bulunması gerekir. Sprite'lar 24 pixel genişliğinde ve 21 pixel yüksekliğindedir. Yatay eksende 8 pixel hafızada 1 byte'a karşılık gelmektedir. Dolayısıyla 24 pixel=3 byte olmaktadır. Bir sprite hafızada 21 satır*3 byte yani 63 byte'lık bir bölge kullanmaktadır. Hesaplama yaparken 64 byte olarak alınır. Fakat siz bu 64 byte'lık bölgeleri kafanıza göre belirleyemezsiniz. Bu bölgeler belli adreslerden başlamaktadır.

Bir BANK 16384 byte'tan oluşmaktadır ve bir bank içinde en fazla 16384/64=256 adet sprite bulunabilir. Bu duruma göre siz sprite'larınızın datalarını 64'ün katları şeklinde olan adreslerde tutmanız gerekecektir. Her ne kadar durum böyle olsada BANK 0'da kullanamayacağınız adresler vardır ($0000-$03FF arası). Buralarda işletim sistemi tarafından kullanılan adresler, vektörler, yığın ve ekran belleği bulunmaktadır. Eğer çok fazla sprite kullanmayacaksanız veya sprite datalarınızın biraz dağınık olması önemli değilse bahsettiğimiz bölgelerde bazı boşluklar vardır ve buralara 4 adet sprite'a ait dataları yerleştirebilirisiniz.

Aslında $0000-$03FF arasını dilediğiniz gibi kullanabilirsiniz. $0000-$03FF arasındaki bölge 16 sprite'lık bir yer demektir (1024/64=16) ve biz bunun 13 sprite'lık bölümünü rahatça kullanabiliriz. Şimdi bu iş nasıl olacak açıklayalım. Demo veya intro yapacağımız için dolayısıyla assembler –ve interrupt- kullanacağımız için sistem değişkenlerinin çoğu işimize yaramayacaktır. Ama siz yinede bu adresleri kullanmamaya özen gösterin.

$0000-$003F : Kullanmayın, çünkü 1 numaralı adres hafızayı düzenlemek için kullanılıyor. Fakat 2 ve daha sonraki adresleri başka işlemler için rahatça kullanabilirsiniz.

$0040-$00FF : Kasete/diskete erişim yapmayacaksanız veya kursör kullanan işlemler yapmayacaksanız buraları sizindir. Üç sprite için kullanılabilir ama tavsiye etmem.

$0100-$01BF : Burası yığına ait olduğu için kullanılamaz gibi gözükse bile üç sprite için kullanabilirsiniz. Eğer programınızda arka arkaya çok fazla JSR –20 kadar- veya PHA –60 kadar- komutları kullanırsanız bu bölgeye girmeye başlarsınız. Fakat arka arkaya JSR komutu kullanmak programın anlaşılmasını zorlaştırır. Gerekirse kullanabilirsiniz.

$01C0-$01FF : Kesinlikle kullanmayın. Aksi takdirde C64'ünüz kilitlenebilir.

$0200-$02BF : Kullanabilirsiniz. Üç sprite için kullanabilirsiniz.

$02C0-$0300 : Burası her zaman boştur. Güle güle kullanın. Bir sprite içindir.

$0300-$033F : Kullanmayın. Çünkü burada işletim sistemi tarafından kullanılan vektörler vardır. Eğer kullanırsanız istenmeyen sonuçlar çıkacaktır.
$0340-$03FF : Kullanabilirsiniz. Üç sprite için yeterlidir.
$0400-$07FF : Ekran belleği ve sprite göstergeçleri vardır. Kullanamazsınız.
$0800-$3FFF : İstediğiniz gibi kullanın. Kod, müzik, grafik, text, sprite data vs..

Yukarıdaki adreslerden $02C0-$0300 ve $0340-$03FF aralarını her zaman kullanabilirsiniz.

SPRITE OLUŞTURMA

Bir sprite için 63 byte gerekli demiştik. Bir byte 8 bitten oluştuğu için 63*8=504 tane bitten hangilerinin açık hangilerinin kapalı olacağını belirleyerek sprite'ımıza istediğimiz şekli verebiliriz. Daha sonra bu bitlerin durumuna göre her byte'ın değerini hafızadaki ilgili adreslere yazdığımızda sprite'ımız kullanılabilir hale gelmiş olacaktır

Çizdiğiniz sprite'ın datalarını hesaplamak için işaretlediğiniz noktalar için "1", işaretlemedikleriniz için ise "0" değerini, tablodaki "değer" satırındaki değerlerle çarpmanız ve sonuşları toplamanız gerekmektedir. Yani bir byte'ın değerini bulmak için (bit numarası * değer) çarpımlarını toplayacaksınız. Aslında yaptığınız işlem ikiliden onlu sayıya çevirme işlemidir. Bu çevirme işlemi derginin 1. sayısında anlatılmıştır. Bir sprite için 63 adet sayı elde etmeniz lazımdır.

Bit-ler	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Değer	128	64	32	16	8	4	2	1	128	64	32	16	8	4	2	1	128	64	32	16	8	4	2	1
Satır 0																								
Satır 1																								
Satır 2																								
Satır 3																								
Satır 4																								
Satır 5																								
Satır 6																								
Satır 7																								
Satır 8																								
Satır 9																								
Satır 10																								
Satır 11																								
Satır 12																								
Satır 13																								
Satır 14																								
Satır 15																								
Satır 16																								
Satır 17																								
Satır 18																								
Satır 19																								
Satır 20																								

Sprite tasarlama tablosu. Fotokopi çektirin ve kullanmaya başlayın.

SPRITE GÖSTERGEÇLERİ

Sprite'larımızın hafızada hazır olduğunu varsayıyoruz. Bir sprite'ı ekranda göstermeden önce VIC'e sprite'ın hafızada bulunduğu yeri bildirmemiz gerekiyor. Bunun için ekran belleğinin ($0400-$07FF) son sekiz adresini (**$07F8-$07FF**) kullanacağız. Her sprite için bir adres ayrılmıştır. 8 tane sprite'ımız var ve bunları 0,1,2,3,4,5,6,7 olarak numaralandıracağız. Bu durumda 0. sprite için $07F8, 1. sprite için $07F9 ..... 7. sprite için $07FF adresini kullanacağız. Eğer VIC bank'ını ve/veya ekran belleğinin yerini değiştirirseniz bu pointer adresleride değişecektir.

Daha önce sprite datalarının hafızada 64'ün katları şeklinde olan adreslerde bulunması gerektiğini yazmıştık. Bir sprite datasının bulunduğu adresi 64'e bölüp sonucu o sprite'a ait pointer adresine yazarsak datayı VIC'e göstermiş oluruz.

SPRITE'LARI GÖRÜNTÜLEMEK

VIC'in **$D015** (53269) numaralı registeri sprite'ların ekranda görünmesini veya kaybolmasını sağlar. Registerin her biti bir sprite için kullanılır. Bir sprite'a ait bit "1" yapılırsa ve koordinat değerleri görülebilen ekran sınırları içindeyse sprite ekranda gözükecektir. Eğer ilgili biti "0" yaparsanız sprite görünmez hale gelecektir. Hangi sprite'ı göstermek istiyorsanız o sprite ait bitin değerini $D015'e yerleştirilmelidir.

Bit	7	6	5	4	3	2	1	0
Değer	128	64	32	16	8	4	2	1

Diğer sprite'lara zarar vermeden istenilen bir sprite'ı göstermek için, göstermek istediğiniz sprite'ın bit değerini EOR komutu ile kullanın ve $D015 adresine yazın.

```
lda   $d015
eor   #$değer
sta   $d015
```

"EOR #$DEĞER" komutu bir tür anahtar gibi çalışır. Sprite açıksa kapatır, kapalı ise açar. Bu yöntem açma için "ora" kapatma için "and" komutu kullanmaktan çok daha iyidir, çünkü "ora" ve "and" komutları için kullanılan parametreler birbirinden farklıdır. Mesela 5. Sprite'ı açmak için ORA #$20, kapatmak için "AND #$DF" komutunu kullanmanız gerekir.

Kontrol etmek istediğiniz sprite birden fazlaysa, bit değerlerinin toplamını EOR komutu için kullanırsınız.

SPRITE RENKLERİ

Her sprite için bir renk registeri ayrılmıştır ve bu adresler **$D027-$D02E**(dahil) (53287-53294) arasındadır. Sprite'ı tasarlarken işaretlenen bitler sprite renginde, işaretlenmeyen bitler ise saydam olup arkasında ne varsa o görüntülenecektir.

ÇOK RENKLİ SPRITE'LAR

Çok renkli modunda bir sprite kullanmak istiyorsanız **$D01C** (53276) adresini kullanabilirsiniz. Kullanılışı aynı $D015 adresi gibidir. Her sprite için bir bit ayrılmıştır. Çok renkli modda bir sprite göstermek istiyorsanız ilgili biti "1" yapmalısınız.

```
lda   $d01c
eor   #$değer
sta   $d01c
```

ÇOK RENKLİ MOD

Çok renkli modda bir sprite için 4 renk kullanma imkanı vardır. Fakat çok renkli moda geçtiğinizde yatay çözünürlük yarıya düşer. Artık yatayda 24 bit değil 12 bit çifti kullanabilirsiniz. Sprite'ı renklendirirken bu bit çiftlerinin durumuna göre renkler belirlenir.

BİT ÇİFTİ		AÇIKLAMA
0	0	Saydam, ekran rengi
0	1	0 numaralı çok renkli mod sprite registeri ($D025-53285)
1	0	Sprite renk registeri
1	1	1 numaralı çok renkli mod sprite registeri ($D026-53286)

SPRİTE'LARIN GENİŞLEMESİ

Sprite'ların X ve Y eksenlerinde 2 kat büyüme özelliği vardır. Bir sprite'ı X ekseninde 2 kat büyütmek için **$D01D** (53277), Y ekseninde 2 kat büyütmek için ise **$D017** (53271) adresleri kullanılır. Her sprite için bir bit ayrılmıştır. Eğer bit "1" ise o sprite geniş olarak görüntülenir.

```
lda   $d01d
eor   #$değer
sta   $d01d

lda   $d017
eor   #$değer
sta   $d017
```

SPRITE KOORDİNATLARI

Sprite'ları ekranda göstermek için iki koordinata ihtiyacımız vardır. X ve Y koordinatları. VIC'in ilk 16 registeri X,Y sırasında olmak üzere 8 sprite'ın x ve Y koordinat değerlerini tutar. X koordinatlarıyla ilgili olarak bir registerimiz daha vardır. Bu registerin her biti bir sprite için ayrılmıştır ve "EXB" (Extra X Bit) olarak bilinir.

C64'ün ekran çözünürlüğü 320*200 olup, bir sprite'ın X koordinat değeri 255'ten büyük olabilir. Oysa X ve Y koordinatlarını tutan registerler en fazla 255 olabilir. 255'ten büyük X koordinat değerleri için **$D010** adresinde her sprite için bir bit ayrılmıştır. Böylece X koordinat değeri 0-511 arasında olabilir. Y koordinat değeri ise 0-255 arasında olabilir. Aslında genişletilmemiş sprite'lar için ekranda tam olarak görüntülenebildiği koordinatlar minimum X,Y ($18-24,$32-50), maksimum X,Y ($140-320,$E5-229) dur. Genişletilmiş sprite'lar için minimum X,Y ($18,$32) maksimum X,Y ($128-296,$D0-208) dir.

```
$D000 - 0. Sprite için X koordinatı
$D001 - 0. Sprite için Y koordinatı
$D002 - 1. Sprite için X koordinatı
$D003 - 1. Sprite için Y koordinatı
$D004 - 2. Sprite için X koordinatı
$D005 - 2. Sprite için Y koordinatı
$D006 - 3. Sprite için X koordinatı
$D007 - 3. Sprite için Y koordinatı
$D008 - 4. Sprite için X koordinatı
$D009 - 4. Sprite için Y koordinatı
$D00A - 5. Sprite için X koordinatı
$D00B - 5. Sprite için Y koordinatı
$D00C - 6. Sprite için X koordinatı
$D00D - 6. Sprite için Y koordinatı
$D00E - 7. Sprite için X koordinatı
$D00F - 7. Sprite için Y koordinatı
 $D010 - Sprite'ler için EXB adresi
```

SPRITE ÖNCELİKLERİ

Sprite'ların birbirlerine göre öncelikleri vardır. Aynı koordinatlarda bulunan iki sprite'dan numarası küçük olan önde gözükecektir. Yani mesela 0 ve 4 numaralı spritelar aynı koordinatlarda bulunuyorsa 0 numaralı sprite önde gözükecektir.

Aynı şekilde sprite'ların arka plandaki karakterlerle, grafiklerle olan önceliği **$D01B** adresindeki sprite-background öncelik registeri tarafından belirlenir. Adresin her biti bir sprite için ayrılmıştır. Eğer bir bitin değeri "0" ise sprite arka plandaki karakterlerin, grafiklerin önünde gözükecektir. Eğer "1" ise sprite karakterlerin, grafiklerin arkasında gözükecektir.

SPRİTE'LARIN ÇARPIŞMASI

VIC'in en ilginç özelliklerinden biriside sprite-sprite ve sprite-background çarpışmalarını kontrol edebilmesidir. Bir çarpışmanın olabilmesi için bir sprite'ın "1" olan kısmı ile başka bir sprite'ın veya karakterin "1" olan kısmının çakışması gerekir.

VIC'in $D01E adresi sprite-sprite çarpışmasını, $D01F adresi sprite-background çarpışmasını kontrol eder. Eğer bir bit "1" ise çarpışma olmuş demektir.

SPRİTE DEMO PROGRAM

Buraya kadar teorik olarak spritelar hakkında bilgi sahibi olduk. Şimdi bir program ile bu öğrendiklerimizi pratik olarak gerçekleştireceğiz. Fakat bu programda çarpışma kontrolünü ve çok renkli spritelar için renk değiştirme işlemlerini yapmayacağız. Sprite'larımız içi kapalı kare şeklinde olacaktır. Ekrandaki sprite'lar 1'den 8'e kadar numaralandırılmıştır. Önce seçmek istediğiniz sprite'ın numarasına basın. Sonra sprite özelliklerini değiştirin. Sprite background önceliğini sprite'ı hareket ettirmeden değiştirin. "D" tuşu ile sprite eski yerine yerleştirilir. Özellikler eski değerlerine gelir.

TUŞ	ÖZELLİK
X	X yönünde genişletir/normale döndürür.
Y	Y yönünde genişletir/normale döndürür.
C	Sprite rengini değiştirir.
P	Sprite-background önceliğini belirler.
D	Default değerleri geri yükler.
O	Sprite on/off kontrolü.
E	EXB'yi açar/kapatır.
CURSOR SAĞ	X ekseninde koordinat değerini 1 arttırır. Sprite sağa hareket eder.
CURSOR SOL	X ekseninde koordinat değerini 1 azaltır. Sprite sola hareket eder.
CURSOR AŞAĞI	Y ekseninde koordinat değerini 1 arttırır. Sprite aşağı hareket eder.
CURSOR YUKARI	Y ekseninde koordinat değerini 1 arttırır. Sprite yukarı hareket eder.

```
                *=$0801

                .word nextline
                .word 2004        ; 2004
                .byte $9e         ; SYS
                .text "2061"      ; 2061
                .byte 0           ; komutu
nextline        .word 0

                lda   #$0c
                jsr   $e536
                ldx   #$00
                stx   $d020
                stx   $d021
                lda   #$ff
ldata           sta   sdata,x
                inx
                cpx   #$3f
                bne   ldata
                ldx   #$00
lkoor           lda   koor,x
                sta   $d000,x
                inx
                cpx   #$10
                bne   lkoor
                ldx   #$00
                ldy   #$01
lcolor          lda   select,x
                sta   $d027,x
                sta   $0400,y
```

```
                sta   $0450,y
                lda   #(sdata/$40)
                sta   $07f8,x
                iny
                iny
                iny
                inx
                cpx   #$08
                bne   lcolor
                lda   #$ff
                sta   $d015

key             jsr   $ffe4
                ldx   #$00
compare         cmp   select,x
                beq   change
                inx
                cpx   #$08
                bne   compare
                jmp   key

change          stx   sprnum
                sta   $d020
set             jsr   $ffe4

on_off          ldx   sprnum
                cmp   #$4f        ;"O" tuşu
                bne   colch
                lda   $d015
                eor   eorcode,x
                sta   $d015
                jmp   set
colch           cmp   #$43        ;"C" tuşu
                bne   x_pand
                inc   $d027,x
                jmp   set
x_pand          cmp   #$58        ;"X" tuşu
                bne   y_pand
                lda   $d01d
                eor   eorcode,x
                sta   $d01d
                jmp   set
y_pand          cmp   #$59        ;"Y" tuşu
                bne   prior
                lda   $d017
                eor   eorcode,x
                sta   $d017
                jmp   set
prior           cmp   #$50        ;"P" tuşu
                bne   default
                lda   $d01b
                eor   eorcode,x
                sta   $d01b
                jmp   set
default         cmp   #$44        ;"D" tuşu
                bne   inc_x
                lda   select,x
                sta   $d027,x
                lda   $d015
                ora   eorcode,x
                sta   $d015
                lda   $d01d
                and   off,x
                sta   $d01d
```

```
                sta   $d017
                sta   $d010
                sta   $d01b
                lda   #$00
                sta   $d020
                txa
                asl
                tax
                lda   koor,x
                sta   $d000,x
                lda   koor+1,x
                sta   $d000+1,x
                jmp   key
inc_x           cmp   #$1d        ;"RIGHT" tuşu
                bne   dec_x
                txa
                asl
                tax
                inc   $d000,x
                bne   noexb
setx            ldx   sprnum
                lda   $d010
                eor   eorcode,x
                sta   $d010
noexb           jmp   set
dec_x           cmp   #$9d        ;"LEFT" tuşu
                bne   inc_y
                txa
                asl
                tax
                dec   $d000,x
                bne   noexb
                jmp   setx
inc_y           cmp   #$11        ;"DOWN" tuşu
                bne   dec_y
                txa
                asl
                tax
                inc   $d001,x
                jmp   set
dec_y           cmp   #$91        ;"UP" tuşu
                bne   exb
                txa
                asl
                tax
                dec   $d001,x
                jmp   set
exb             cmp   #$45        ;"E" tuşu
                bne   tekrar
                lda   $d010
                eor   eorcode,x
                sta   $d010
tekrar          jmp   set
                rts

koor            .byte $18,$3a,$30,$3a,$48,$3a,$60,$3a
                .byte $78,$3a,$90,$3a,$a8,$3a,$c0,$3a
select          .byte $31,$32,$33,$34,$35,$36,$37,$38
sprnum          .byte 0
eorcode         .byte 1,2,4,8,16,32,64,128
off             .byte 254,253,251,247,239,223,191,127
sdata           =$02c0
                .end
```

PROGRAM DÖKÜMLERİ

PROGRAM ADI: KARSET3 0801 0866

```
0801: 0B 08 D4 07 9E 32 30 36  - 43A8
0809: 31 00 00 00 78 A5 01 29  - 333E
0811: FB 85 01 A9 D0 A2 30 A0  - 8AF0
0819: 00 84 FB 85 FC 84 FD 86  - A869
0821: FE A2 10 B1 FB 91 FD C8  - BC58
0829: D0 F9 E6 FC E6 FE CA D0  - E25B
0831: F2 A5 01 09 04 85 01 58  - 4593
0839: A9 1C 8D 18 D0 A9 30 A0  - 7ADD
0841: 00 84 FB 85 FC A0 00 A2  - 8A64
0849: 08 A9 00 85 02 B1 FB 6A  - 7698
0851: 26 02 CA D0 FA A5 02 91  - 843E
0859: FB C8 D0 EB E6 FC A5 FC  - E199
0861: C9 40 D0 E1 60 00 00 00  - 49E6
```

PROGRAM ADI: KARSET4A 0801 0881

```
0801: 0B 08 D4 07 9E 32 30 36  - 43A8
0809: 31 00 00 00 78 A5 01 29  - 333E
0811: FB 85 01 A9 D0 A2 30 A0  - 8AF0
0819: 00 84 FB 85 FC 84 FD 86  - A869
0821: FE A2 10 B1 FB 91 FD C8  - BC58
0829: D0 F9 E6 FC E6 FE CA D0  - E25B
0831: F2 A5 01 09 04 85 01 58  - 4593
0839: A9 1C 8D 18 D0 A9 30 A0  - 7ADD
0841: 00 84 FB 85 FC A0 00 B1  - 8DD9
0849: FB A2 00 18 6A 3E 79 08  - 4A6E
0851: E8 E0 08 D0 F6 C8 C0 08  - 939A
0859: D0 ED A0 00 B9 79 08 91  - 785E
0861: FB C8 C0 08 D0 F6 A5 FB  - C2AF
0869: 18 69 08 85 FB 90 02 E6  - 81B5
0871: FC A5 FC C9 40 D0 CE 60  - A752
0879: 00 00 00 00 00 00 00 00  - 0000
```

PROGRAM ADI: KARSET4B 0801 0881

```
0801: 0B 08 D4 07 9E 32 30 36  - 43A8
0809: 31 00 00 00 78 A5 01 29  - 333E
0811: FB 85 01 A9 D0 A2 30 A0  - 8AF0
0819: 00 84 FB 85 FC 84 FD 86  - A869
0821: FE A2 10 B1 FB 91 FD C8  - BC58
0829: D0 F9 E6 FC E6 FE CA D0  - E25B
0831: F2 A5 01 09 04 85 01 58  - 4593
0839: A9 1C 8D 18 D0 A9 30 A0  - 7ADD
0841: 00 84 FB 85 FC A0 00 B1  - 8DD9
0849: FB A2 00 18 6A 7E 79 08  - 53AE
0851: E8 E0 08 D0 F6 C8 C0 08  - 939A
0859: D0 ED A0 00 B9 79 08 91  - 785E
0861: FB C8 C0 08 D0 F6 A5 FB  - C2AF
0869: 18 69 08 85 FB 90 02 E6  - 81B5
0871: FC A5 FC C9 40 D0 CE 60  - A752
0879: 00 00 00 00 00 00 00 00  - 0000
```

PROGRAM ADI: KARSET4C 0801 0881

```
0801: 0B 08 D4 07 9E 32 30 36  - 43A8
0809: 31 00 00 00 78 A5 01 29  - 333E
0811: FB 85 01 A9 D0 A2 30 A0  - 8AF0
0819: 00 84 FB 85 FC 84 FD 86  - A869
0821: FE A2 10 B1 FB 91 FD C8  - BC58
0829: D0 F9 E6 FC E6 FE CA D0  - E25B
0831: F2 A5 01 09 04 85 01 58  - 4593
0839: A9 1C 8D 18 D0 A9 30 A0  - 7ADD
0841: 00 84 FB 85 FC A0 00 B1  - 8DD9
0849: FB A2 00 18 2A 3E 79 08  - 42AE
0851: E8 E0 08 D0 F6 C8 C0 08  - 939A
0859: D0 ED A0 00 B9 79 08 91  - 785E
0861: FB C8 C0 08 D0 F6 A5 FB  - C2AF
0869: 18 69 08 85 FB 90 02 E6  - 81B5
0871: FC A5 FC C9 40 D0 CE 60  - A752
0879: 00 00 00 00 00 00 00 00  - 0000
```

PROGRAM ADI: KARSET4D 0801 0881

```
0801: 0B 08 D4 07 9E 32 30 36  - 43A8
0809: 31 00 00 00 78 A5 01 29  - 333E
0811: FB 85 01 A9 D0 A2 30 A0  - 8AF0
0819: 00 84 FB 85 FC 84 FD 86  - A869
0821: FE A2 10 B1 FB 91 FD C8  - BC58
0829: D0 F9 E6 FC E6 FE CA D0  - E25B
0831: F2 A5 01 09 04 85 01 58  - 4593
0839: A9 1C 8D 18 D0 A9 30 A0  - 7ADD
0841: 00 84 FB 85 FC A0 00 B1  - 8DD9
0849: FB A2 00 18 2A 7E 79 08  - 4BEE
0851: E8 E0 08 D0 F6 C8 C0 08  - 939A
0859: D0 ED A0 00 B9 79 08 91  - 785E
0861: FB C8 C0 08 D0 F6 A5 FB  - C2AF
0869: 18 69 08 85 FB 90 02 E6  - 81B5
0871: FC A5 FC C9 40 D0 CE 60  - A752
0879: 00 00 00 00 00 00 00 00  - 0000
```

PROGRAM ADI: HADES BASIC C000 C1DE

(NOT : Programı yükleyin, NEW komutu verin, sonra SYS 49152 komutu ile yeni komutları aktif hale getirin.)

```
C000: A9 0B A2 C0 8D 08 03 8E  - 63C8
C008: 09 03 60 A5 7A A6 7B 85  - 7541
C010: FB 86 FC A9 61 A2 C0 85  - A4F6
C018: FD 86 FE A2 00 A0 00 20  - 626B
C020: 73 00 85 02 B1 FD C5 02  - 6DD3
C028: D0 13 C8 98 DD 5B C0 D0  - A903
C030: EE 8A 0A AA BD 7D C0 48  - 8680
C038: BD 7C C0 48 60 BD 5B C0  - 90ED
C040: 18 65 FD 85 FD 90 02 E6  - 97AA
C048: FC E8 EC 5A C0 D0 D5 A5  - BECE
C050: FB A6 FC 85 7A 86 7B 4C  - 8A0D
C058: E4 A7 06 05 03 03 05 06  - 1F9F
C060: 05 4D 49 52 52 B0 49 4E  - 56EE
C068: 56 46 91 54 53 50 45 45  - 5204
C070: 44 53 43 52 45 45 4E 44  - 487A
C078: 45 4C 41 59 87 C0 7F C1  - 8710
C080: 87 C1 A4 C1 B5 C1 D1 C1  - BBA7
C088: 20 73 00 D0 03 4C 08 AF  - 572F
C090: C9 56 F0 6E C9 48 D0 F5  - B247
C098: A9 00 A2 18 8D 7B C1 8D  - 7EC1
C0A0: 7F C1 8E 7C C1 AC 7F C1  - A29D
C0A8: AE 7B C1 20 F0 C0 A5 FB  - B6BC
C0B0: 85 FD A5 FC 85 FE A5 02  - 96AF
C0B8: 85 04 A5 03 85 05 B1 FD  - 7FC7
C0C0: 48 B1 04 48 AE 7C C1 20  - 67B6
C0C8: F0 C0 B1 02 91 04 B1 FB  - 96A6
C0D0: 91 FD 68 91 02 68 91 FB  - 9685
C0D8: C8 C0 28 D0 CB A0 00 8C  - 86A9
C0E0: 7F C1 EE 7B C1 CA 8E 7C  - A1F6
C0E8: C1 E0 0C D0 BB 4C E4 A7  - A2B7
C0F0: BD F0 EC 85 FB 85 02 BD  - A021
C0F8: 62 C1 85 FC 18 69 D4 85  - 9004
C100: 03 60 A2 00 A9 04 86 FB  - 7A3B
C108: 85 FC 86 FD 18 69 D4 85  - 96EC
C110: FE A2 00 A9 27 8D 7E C1  - 89CC
C118: A0 00 8C 7D C1 AC 7E C1  - 9641
C120: B1 FB 48 B1 FD 48 AC 7D  - 9A4D
C128: C1 B1 FD AC 7E C1 91 FD  - C0DA
C130: AC 7D C1 B1 FB AC 7E C1  - B201
C138: 91 FB AC 7D C1 68 91 FD  - B1CE
C140: 68 91 FB CE 7E C1 C8 C0  - B6FD
C148: 14 D0 CF A5 FB 18 69 28  - 71F4
C150: 85 FB 85 FD 90 04 E6 FC  - B500
C158: E6 FE E8 E0 19 D0 B4 4C  - 9FC7
C160: E4 A7 04 04 04 04 04 04  - 1EF9
C168: 04 05 05 05 05 05 05 06  - 052A
C170: 06 06 06 06 06 06 07 07  - 0664
C178: 07 07 07 00 00 00 00 00  - 019D
C180: A9 12 20 D2 FF 4C E4 A7  - 9C17
C188: 20 73 00 F0 0D 20 9E B7  - 6F87
C190: E0 08 B0 0E 8A 0A 18 69  - 4F11
C198: 10 2C A9 14 8D 18 D0 4C  - 5D26
C1A0: AE A7 4C 48 B2 20 73 00  - 5388
C1A8: F0 04 20 9E B7 2C A2 33  - 673A
C1B0: 8E 05 DC 4C AE A7 20 9B  - 7C3B
C1B8: B7 E0 02 B0 12 AD 11 D0  - 7EBD
C1C0: E0 01 F0 03 29 EF 2C 09  - 557B
C1C8: 10 8D 11 D0 4C AE A7 4C  - 733B
C1D0: 48 B2 20 9B B7 20 B3 EE  - 94BB
C1D8: CA D0 FA 4C AE A7 00 00  - 6921
```

PROGRAM ADI: SPRITE DEMO 0801 0975

```
0801: 0B 08 D4 07 9E 32 30 36  - 43A8
0809: 31 00 00 00 A9 0C 20 36  - 2B06
0811: E5 A2 00 8E 20 D0 8E 21  - 6B9A
0819: D0 A9 FF 9D C0 02 E8 E0  - B35F
0821: 3F D0 F8 A2 00 BD 4C 09  - 65D1
0829: 9D 00 D0 E8 E0 10 D0 F5  - AE9C
0831: A2 00 A0 01 BD 5C 09 9D  - 630E
0839: 27 D0 99 00 04 99 50 04  - 441B
0841: A9 0B 9D F8 07 C8 C8 C8  - A21E
0849: E8 E0 08 D0 E7 A9 FF 8D  - B60C
0851: 15 D0 20 E4 FF A2 00 DD  - 96C3
0859: 5C 09 F0 08 E8 E0 08 D0  - 8AEF
0861: F6 4C 53 08 8E 64 09 8D  - 5FED
0869: 20 D0 20 E4 FF AE 64 09  - 7862
0871: C9 4F D0 0C AD 15 D0 5D  - 75FD
0879: 65 09 8D 15 D0 4C 6B 08  - 4D93
0881: C9 43 D0 06 FE 27 D0 4C  - 7CE9
0889: 6B 08 C9 58 D0 0C AD 1D  - 610A
0891: D0 5D 65 09 8D 1D D0 4C  - 66E7
0899: 6B 08 C9 59 D0 0C AD 17  - 5FC5
08A1: D0 5D 65 09 8D 17 D0 4C  - 6609
08A9: 6B 08 C9 50 D0 0C AD 1B  - 5FAC
08B1: D0 5D 65 09 8D 1B D0 4C  - 669D
08B9: 6B 08 C9 44 D0 38 BD 5C  - 7637
08C1: 09 9D 27 D0 AD 15 D0 1D  - 674C
08C9: 65 09 8D 15 D0 AD 1D D0  - 7D32
08D1: 3D 6D 09 8D 1D D0 8D 17  - 5A61
08D9: D0 8D 10 D0 8D 1B D0 A9  - 8E84
08E1: 00 8D 20 D0 8A 0A AA BD  - 7DD8
08E9: 4C 09 9D 00 D0 BD 4D 09  - 56BB
08F1: 9D 01 D0 4C 53 08 C9 1D  - 57E1
08F9: D0 17 8A 0A AA FE 00 D0  - 8649
0901: D0 0C AE 64 09 AD 10 D0  - 7642
0909: 5D 65 09 8D 10 D0 4C 6B  - 6349
0911: 08 C9 9D D0 0B 8A 0A AA  - 732D
0919: DE 00 D0 D0 F1 4C 03 09  - 63B7
0921: C9 11 D0 09 8A 0A AA FE  - 8A3D
0929: 01 D0 4C 6B 08 C9 91 D0  - 87A2
0931: 09 8A 0A AA DE 01 D0 4C  - 6CDA
0939: 6B 08 C9 45 D0 09 AD 10  - 5B75
0941: D0 5D 65 09 8D 10 D0 4C  - 6506
0949: 6B 08 60 18 3A 30 3A 48  - 3AE3
0951: 3A 60 3A 78 3A 90 3A A8  - 69A0
0959: 3A C0 3A 31 32 33 34 35  - 3ED5
0961: 36 37 38 00 01 02 04 08  - 0F98
0969: 10 20 40 80 FE FD FB F7  - BC23
0971: EF DF BF 7F 00 00 00 00  - 3FF8
```