Beige Records

BEIGE stylez: GAME MODS. Reverse Engineering SUPER MARIO BROTHERS

2003

https://doi.org/10.25969/mediarep/17613

Veröffentlichungsversion / published version
Zeitschriftenartikel / journal article

Empfohlene Zitierung / Suggested Citation:
Beige Records: BEIGE stylez: GAME MODS. Reverse Engineering SUPER MARIO BROTHERS. In: *Dichtung Digital. Journal für Kunst und Kultur digitaler Medien*. Nr. 29, Jg. 5 (2003), Nr. 3, S. 1–11. DOI: https://doi.org/10.25969/mediarep/17613.

BEIGE stylez: GAME MODS. Reverse Engineering *Super Mario Brothers*

By Beige Records

No. 29 – 2003

Abstract

This page is a tutorial explaining how / why I recently hacked a Super Mario Brothers cartridge and erased everything but the clouds. This continues work being carried out by myself and other BEIGE representatives Paul Davis, Joe Beuckman, and Joe Bonn. I have chosen to present my motives behind the work by adding my thoughts about the project as comments in the source code. As a programmer [not a very good one, though a programmer none the less] my thoughts and motives are most easily exemplified by my code.

Introduction:

Recently I hacked a Mario Brothers Cartridge and erased everything but the clouds. This continues work being carried out by myself and your favorite BEIGE representatives Paul Davis, Joe Beuckman, and Joe Bonn. This page is a tutorial explaining how / why the work was made. I have chosen to present my motives behind the work by adding my thoughts about the project as comments in the source code. As a programmer [not a very good one, though a programmer none the less] my thoughts and motives are most easily exemplified by my code.

Thank you, Cory Arcangel / BEIGE

SOURCE CODE [with comments]:

Below I will go through the source code line by line. The idea here is to simply write a program that will take the clouds from Mario Brothers and scroll them across the screen. Here comes the first line of code:

```
;*****************************************************
;
; BEIGE 2002 - Cory Arcangel
; http://www.beigerecords.com
; http://www.beigerecords.com/cory/
;
; "you mess with the best, you die like the rest" - Anon
; "punks jump up to get beat down"  Brand Nubian
;
;*****************************************************
```

When computer code is made public it is common for programmers to put contact information for a variety of reasons. Here I have also included a few small phrases aimed at media artists who think they can step to my style. This is a trait I inherited from the early commodore64 cracking scene.

Before I get started with the code I would like to take this opportunity to state that I am not really a programmer. The first time I took a class in "computer science" was at a summer school when I was 8 or 9 years old and I remember crying and switching to the "storytelling" class. Years later in college I still didn't like computer science and got below 50% on most of my exams. I have since grown used to programming only because it is the mechanism that seems to make most of the world move. Believe me, if I could order Pizzas [dominos has a great online delivery

mechanism], by painting, I definitely would paint. So the first line in this program that actually does anything is:

```
PROCESSOR 6502
```

The above line, along with the rest of this program, is written in a language called assembly language. Assembly language is the lowest level someone can program. It is one step away from the ones and zeros, and in some cases involves actual ones and zeros. I tend to prefer assembly because it gives me control over the machine and assures me that aesthetic choices are based on the hardware of the machine and not, say, some dupe at Macromedia. The above line of code tells our assembler that the processor for with which we intend our program to run on is the 6502 chip. This is the chip that made the Apple II possible and thus revolutionized home computing. The Nintendo runs on a modified version of this chip. It should not be suprising that the Nintendo and Apple run on the same processor because video game systems are really just home computers with out disk drives.

```
DELAYSCROLL EQU #$01
NTShow EQU #$00
SCROLL EQU #$00

ORG $8000 ;32Kb PRG-ROM, 8Kb CHR-ROM
```

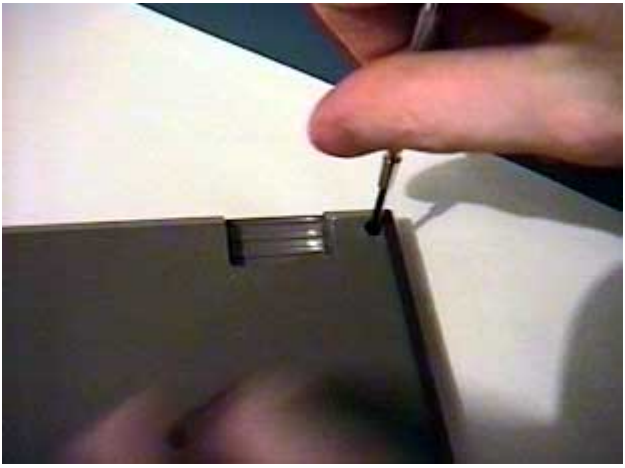
I like the idea of making things out of trash [one can easily find an NES in a dumpster these days], and I like the idea of actually having to break into something that I find in the trash even better. The only way to make work for the NES is to hack and solder a cartridge. To do this, I will clip off the program chip from an actual Mario cartridge, burn my new information [the output of this code compiled] to a chip, and solder it in the place of the old one. The above line tells the compiler that the chip we want this code to be burned on is 32k.

The next part of code tells the cartridge what to do when we press the reset button on the NES. This code is from a Canadian NES genius named Chris Covell, who apparently got it from Duck Hunt. Awesome. I learned to program in assembly language looking at examples of code posted by Chris Covell, and as with a lot of this 8-bit work, information comes mostly from a hobby scene. In my opinion these are the true hero's of contemporary computer art. Out of the hobby scene have come portable playstations, Dreamcasts that boot LINUX, and even hard drives that play music by spinning at different speeds.

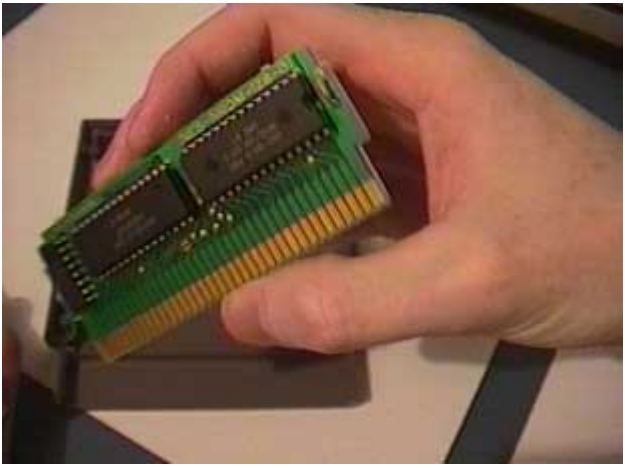
```
Reset_Routine SUBROUTINE
cld ;Clear decimal flag
sei ;Disable interrupts
.WaitV lda $2002
bpl .WaitV
ldx #$00
```

```
stx $2000
stx $2001
dex
txs

ldy #$06
sty $01
ldy #$00
sty $00
lda #$00

.Clear sta ($00),y
dey
bne .Clear

dec $01
bpl .Clear

lda #$20
sta $2006
lda #$00
sta $2006

ldx #$00
ldy #$10
.ClearPPU sta $2007
dex
bne .ClearPPU
dey
bne .ClearPPU

lda #$FF
sta $2003
tay
.ClearSpr sta $2004
dey

bne .ClearSpr

ldx #$3F
stx $2006
ldx #$00
stx $2006

ldx #$0D ;Colour Value (White)
ldy #$20 ;Clear BG & Sprite palettes.
.InitPal stx $2007
```

```
dey
bne .InitPal
```



Here I am unscrewing a Mario Brothers Cartridge.

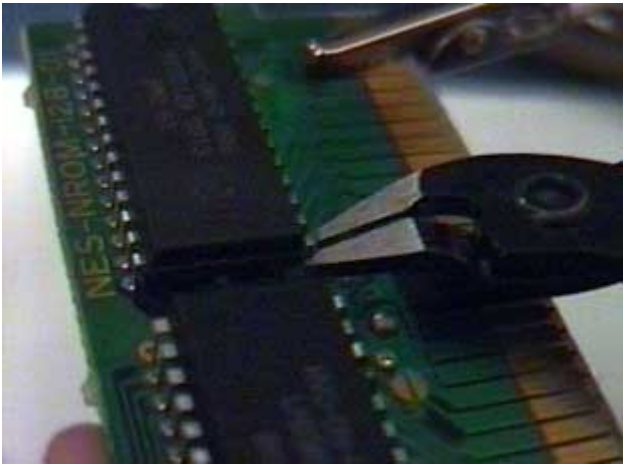


Inside one will find this. Small isn't it? The chip on the left contains the Mario graphics and the chip on the right contains the program code.

Ever wonder why Mario and Zelda were little squares? The Nintendo can only display graphics in 8 pixel by 8 pixel squares, and can only hold 8k of graphics in total therefore Mario and Zelda were simply adhering to the hardware limitations of the Nintendo System. These two hardware limitations defined the aesthetic of most early 80's video games on the Nintendo, and making "art" for this system is a study

of these limitations. Below I load up the color that this cartridge will use into the palette RAM. The Nintendo can only display 4 colors in any 8*8 square. For this cartridge I will use black, blue, light blue and white which translates to $od, $11, $21, $30 in the Nintendos palette.

```
lda #$3F
sta $2006
lda #$00
sta $2006

lda #$21 ;background [powder blue]
sta $2007
lda #$30 ;cloud inside [white]
sta $2007
lda #$11 ; highlight [blue]
sta $2007
lda #$0d ;outline [black]
sta $2007

lda #DELAYSCROLL
sta $21

lda #NTShow
sta $22

lda #SCROLL
sta $24
```

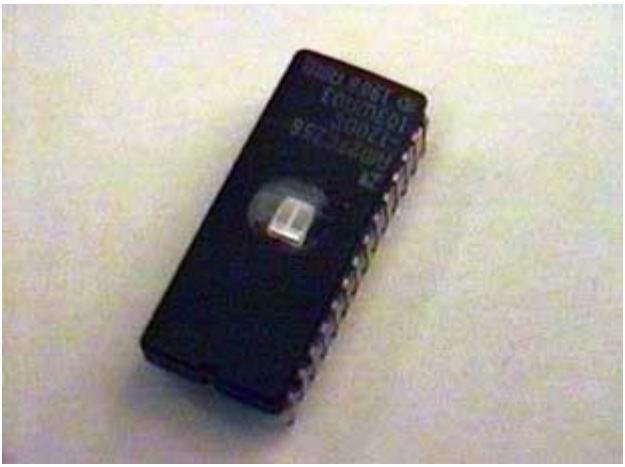


Here I am clipping off the program chip.

A typical NES Cartridge has two chips. One is a graphics chip, and the other is a program chip. Basically the program chip tells the graphics chip where to put the graphics, and thus if you do this in a interesting manner, you have a video game. When making a "Super Mario Clouds" cartridge, I only modify the program chip, and I leave the graphic chip from the original game intact. Therefore since I do not touch the graphics from the original cartridge, the clouds you see are the actual factory soldered clouds that come on the Mario cartridge. There is no generation loss, and no "copying" because I did not even have to make a copy. Wasss up.

The code below is where I load up the clouds.

```
lda #<.NAM_One
sta $00
lda #>.NAM_One
sta $01

lda #$20 ;Load up entire name & attribute table for screen 0.
sta $2006
lda #$00
sta $2006

ldx #$00
ldy #$08

.LoadNTOne
txa
pha

ldx #$00
lda ($00),X ;Load up NES image
sta $2007

pla
tax

inc $00
bne .NoNTOne
inc $01

.NoNTOne
dex
bne .LoadNTOne
dey
bne .LoadNTOne
```



These cartridges use 32k 28 pin 8-bit Eproms. Usually I get Am27C256. Try Jameco for these.

For this cartridge I simply draw everything and then turn the screen on. The NES can not just draw a picture to the screen like a modern computer [in fact to make something that looks similar on a modern computer would take about 3 minutes in Photoshop]. It is too slow for that, so in order to change backgrounds one has to turn the screen off, draw a new picture, and turn it back on again. This is why the screen goes black for a split second between worlds on most early NES games. I turn the screen on by placing a sequence of ones and zeros into a specific Nintendo memory location. One highlight of working in assembly language is being able to actually use ones and zeros.

```
lda #%10011100
sta $2000
lda #%00011110
sta $2001

lda #$01
sta DELAYSCROLL

lda #$00
sta NTShow

lda #$00
sta $24 ;scroll

.Loop
jmp .Loop
```

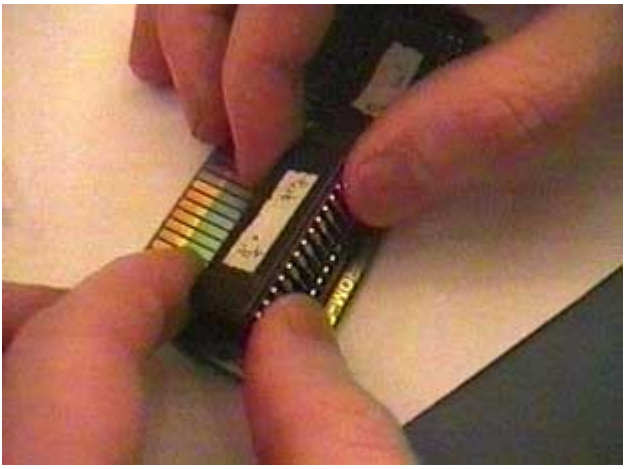
Below is the file which sets the pattern of the clouds. I typed this by hand. I really didn't need to do it by hand, but it makes me feel better. Not that anyone could ever see the difference anyway, so you will just have to trust me. It is written in a format called HEX which is shorthand for binary.

.NAM_One

INCLUDE Clouds.NAM -< click HERE

That's it for the intro code. Now below I make it scroll

NMI_Routine SUBROUTINE



Once you solder on a 28pin low profile socket and solder it onto the board where you clipped off the program chip you sawp these chips in and out of the cartridge.

A TV works by drawing a picture faster than your eye can see every 1/60th of a second. [In Europe it is 1/50th] Every time this scan line gets to the bottom of the screen it has to jump back to the top. This is called a vertical blanking interrupt. The NES can only draw graphics to the screen when this line is jumping from the bottom of the screen to the top. Below is the code which scrolls the clouds during this period where the electron beam is jumping. A highlight of working on early game systems is this intimate access to the display machanism of the television.

```
dec DELAYSCROLL
bne .end_no_scroll

lda #$20
sta DELAYSCROLL
```

```
lda #$ff
cmp $24 ;scroll
beq .NT_adj
jmp .end

.NT_adj

lda #$00
cmp NTShow
beq .Show_Zero

lda #%10011100
sta $2000
lda #%00011110
sta $2001

lda #$00
sta NTShow

jmp .end

.Show_Zero
lda #%10011101
sta $2000
lda #%00011110
sta $2001

lda #$01
sta NTShow

.end

inc $24 ;scroll
lda $24 ;scroll
sta $2005
lda #$00
sta $2005

.end_no_scroll

rti

IRQ_Routine
rti
```

That's all the code. Now we just need to set the vector table appropriately. This tells the cartridge what to do when you put it in.

```
ORG $FFFA,0
dc.w NMI_Routine
dc.w Reset_Routine
dc.w IRQ_Routine ;Not used, just points to RTI
```

The end. The cartridge is complete. Not that bad. Actually programming for the NES is pretty simple. Now I would compile this program, and then burn it to a chip. Check the pictures along the side of the page.

Download the cartridge here:
SuperMarioClouds.nes

The end cartridge work in an standard Nintendo. As my friend DRX in Boden-standig would say "kick the balistixx"

Another example of plug and play technbology 1984 style.