Retro Jokers by rimsky82

Jokers are used all of the time in modern game hacking, for causing an effect with a button press. Even though retro hacking doesn't offer this kind of advanced code making, it can still be done by writing a routine.

This tutorial assumes you know how to RAM search, use a debugger, and have a good idea of how to hack and write assembly code.

We're going to create a joker for Contra (USA) for the NES. The first thing we need to know is what we want to change with our button press. Contra is one of those side-scrolling platformers that has different weapons, but those weapons have to be found in the game in order to use them, and could be lost when you lose a life. So, our joker will cycle through the different weapons, giving the cheater the ability to change his weapon on the fly without leaving the game.

According to the GameHacking.org website, our current weapon for player 1 is stored at $AA, so this is the address we manipulate in our routine, with values #00 - #04.

Finding a routine that reads the button state

The easiest way to do this is to RAM search while pressing different button combinations. It's never hard to find. It may be necessary to hold a button down and pause the emulation, since when you switch to the RAM search window, the emulator no longer has focus.

The button state address I found is at $F1. However, this is a non-latch state, which means that the state will stay if the button is held down, which is not what we want since the routine would execute more than once on a button push. Just a few bytes down is the address we want, at $00F5 with the following bitfield:

0x01	0x02	0x04	0x08	0x10	0x20	0x40	0x80
Right	Left	Down	Up	Start	Select	B	A

Since the select button doesn't have any use in the game, this will be our new weapon changing button.

Our next step is to find a routine that reads this address. Obviously, that's done by putting a read breakpoint on $F5. Of the two routines in Contra that read this address, we want $D040, since it's constantly executed.

```
07:D040:A5 F5     LDA $00F5 = #$00
07:D042:A4 25     LDY $0025 = #$00
07:D044:D0 0D     BNE $D053
07:D046:29 10     AND #$10
07:D048:F0 19     BEQ $D063
07:D04A:A9 01     LDA #$01
07:D04C:85 25     STA $0025 = #$00
```

ROMs have areas where there is no code, usually at the end. This is where we will write our new routine. So, in the hex editor, we just need to find an area where we can do this; with plenty of space. In Contra, you'll see beginning at $F612 there are plenty of 0xFF bytes that we can overwrite.

Writing our JMP

The next step is to write our absolute JMP, which will require overwriting 3 bytes. Let's write it starting at $D042, just before the branch. Make note of this code and the address where it branches, we'll need it in a second.

```
07:D040:A5 F5     LDA $00F5 = #$00
07:D042:4C 12 F6  JMP $F612
07:D045:0D 29 10  ORA $1029 = #$00
07:D048:F0 19     BEQ $D063
07:D04A:A9 01     LDA #$01
```

Now we overwrite our bytes with our jump. I like to use Genie codes right away, so we don't screw things up in other pages. With our first three codes, GGKIZAKX, ZPKILAIZ, and VYKIGAEI, we have our JMP at $D042. Make sure you don't run the game at this time, it will crash. Instead, go ahead and step through until we reach our routine extension at $F612.

```
07:F612:A4 25     LDY $0025 = #$00
07:F614:F0 03     BEQ $F619
07:F616:4C 53 D0  JMP $D053
07:F619:4C 46 D0  JMP $D046
07:F61C:FF        UNDEFINED
```

The first thing we need to do at this point, is write our code we previously overwrote, in order to make sure the game doesn't lose any of it's functionality or crash. It will almost be the same code, but we have to reverse the branch and write a jump after it, since we are no longer in range for a branch to $D053. I went ahead and put a JMP back to the original code at $F619 so we can test to make sure our "code cave" works- and it does. Go ahead put an execute breakpoint at $F612, and remove that JMP at $F619.

Writing our routine

Now for the fun part. We are going to start writing our own code. The state of the Accumulator is our button state so we just and with 0x20 to see if the select button is pushed. If not, we just branch to the end of our code. This is to check for the select button, so that the weapons don't constantly change when we don't want them to.

```
07:F612:A4 25     LDY $0025 = #$00
07:F614:F0 03     BEQ $F619
07:F616:4C 53 D0  JMP $D053
07:F619:29 20     AND #$20
07:F61B:F0 0E     BEQ $F62B
07:F61D:A5 AA     LDA $00AA = #$04
07:F61F:29 07     AND #$07
07:F621:C9 04     CMP #$04
07:F623:D0 02     BNE $F627
07:F625:A9 FF     LDA #$FF
07:F627:69 01     ADC #$01
07:F629:85 AA     STA $00AA = #$04
07:F62B:A5 F5     LDA $00F5 = #$00
07:F62D:4C 46 D0  JMP $D046
07:F630:FF        UNDEFINED
```

Now, to cycle the weapon for player 1, we need to read the current value at $AA and increment if less than 4, else set it to 0. Then we store the accumulator back into $AA, and we have our different weapon. All we have left to do is load the button state back into the accumulator, and jump back to the original code.

Conclusion

That's it! Now you can write jokers for retro games, and get to work on modifiying all of the old games. The only downside, is the amount of Game Genie codes this can require- this code takes 32. Perhaps it would be better to patch roms?...

GGKIZAKX
ZPKILAIZ
VYKIGAEI
KZOYZVNY
IZOYLVNY
EYOYGVNY
LAOYIVNY
GGOYTVNN
LIOYYVNY
EIONAVNY
PZONPVNN
AZONZVNY
EYONLVNY
APONGVNY
SZONIVNY
XZONTVNN
PZONYVNN
YAXYAVNY
OGXYPVNN
GAXYZVNY
EIXYLVNY
ZAXYGVNY
OZXYIVNN
PTXYYVNN
PAXNAVNY
SAXNPVNY
XZXNZVNN
SZXNLVNY
SYXNGVNY
GGXNIVNN
TGXNTVNY
EIXNYVNY