Branching

This section contains the most technical aspects of event hacking. So far, the game has executed our event scripts in a straight line. If the beginning of the event was at address CA/2000, the game would execute the byte at CA/2000, then the byte at CA/2001, then the byte at CA/2002, and so on. But sometimes, we may want the event to “jump” or “branch” from CA/2000 to CA/4000 (or some other address). There are several reasons why you may want to do this:

- To reuse an event script several times. For example, when the party rests at an Inn, the game will always fade out the screen, play the “Nighty Night” song, restore the party's HP, and fade in the screen. It would be wasteful to write “96 F2 30 31 05 09 E0 06 27 FF F0 B8 B2 BD CF 00 FA F3 10 31 82 09 FF 94 B3 03 FE C7 00 FE” inside every cutscene triggered by speaking to an innkeeper. Instead, the developers placed that lengthy script at CA/00EA, and jumped to this address in every Inn cutscene.
- To skip over part of an event. If Cyan is in the party at the Imperial banquet, he will say an extra line of dialogue about General Leo. If he is not in the party, this part of the event must be skipped by the game.
- To use extra space. If you expand the size of your ROM using FF3usME, you will need to jump to the additional space to use it.
- To repeat part of an event many times (perhaps an infinite number of times).

Jumping to a Subroutine

The simplest way to jump to another address is to use the B2 command. Let's say that we want to jump to CA/4020. Following the instructions in the Event Commands Document, we know to start by subtracting CA0000 from CA4020, giving us 00 40 20. Then we need to reverse the order of the bytes. The full command is B2 20 40 00. You will need to follow this process many times when jumping or branching.

When the game jumps to CA/4020, it will execute the event script found there and then return when it reaches an $FE command. The important thing to note is that the $FE command does not end the event if it is executed inside a subroutine. Instead, it will return to the address at which you left off. If an event script is 4B 23 01 **B2 20 40 00** 4B 23 02, the game will display the first text box, execute the subroutine at B2 20 40 00, then display the second text box after returning from the subroutine. If you wanted the event to end directly after executing the subroutine, you could have written 4B 23 01 **B2 20 40 00** FE instead.

By itself, this command allows us to reuse parts of the event script, as in the innkeeper example above. It also lets us make use of extra space from expanding the ROM. Do not use this command to infinitely repeat part of an event; if you jump to subroutines 255 times without returning, the game will crash.

Event Bits

A way to branch based on certain conditions would also be very helpful. For example, after the first set of guards (in vanilla) in Narshe is defeated, the player should not be able to encounter them again. To accomplish this, we need an “event bit”. Every event bit contains a single value that can be

either 1 or 0. At the beginning of the game, the event bits are all 0. After you fight the first set of guards in Narshe, an event bit is changed to 1. This is how the game remembers that you have fought this set of guards.

Every time you step on that tile, the game will check to see if that particular event bit is 1 or 0. If it is 0, the game knows that the player has not fought the guards yet. If it is 1, the player must have fought the guards already. Even if the player saves and reloads the game, this event bit will revert to the state that it was in when the game was saved.

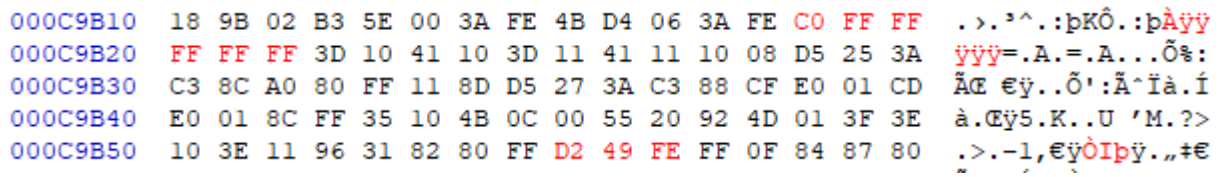
If we use event bits correctly, the game can answer questions like “Has the party met Shadow yet?” or “Has the party defeated my optional superboss?”. We have a lot more power than simply preventing a cutscene from repeating itself.

Example

Right now, players can trigger our new Narshe guard cutscene infinitely many times, summoning as many guards as they please. It's strange that Narshe has an endless supply of guards, and I think that players would be encouraged to grind guards early in the game. Let's change the cutscene so that activating it a second time will not spawn more guards.

First, we can look through the Event Bits Document to find a bit that is unused by the vanilla game. Arbitrarily, I choose to use bit 149 for our event. Our first goal is to set this bit at the end of the event so that we can check if this event has already been executed. According to the Event Commands Document, we need to write D2 49 to set this bit. We'll slip this in before the $FE command that ends the first cutscene.

Now we need to revisit the beginning of the cutscene to add the command that branches based on our event bit. Looking in the Event Commands Document, we can see that the $C0 command allows us to do this. We need to specify five parameters for this command, so we will need to copy the existing cutscene and paste it six bytes ahead. For now, let's replace the parameters of the $C0 command with the placeholder “FF”. This is how our cutscene looks now, with the most recent changes highlighted in red:



The first two parameters of the $C0 command allow us to choose which bit to branch on. We want to branch if bit 149 is set, so we invert 01 49 and add $80 to the second parameter. Therefore, the first two parameters are 49 81.

We can use the last three parameters to specify the address to branch to. In this case, we want to branch to the address just after our original cutscene. Looking at the image above, we can see that this address is $CC/9B5B. We subtract CA0000 and invert the bytes to get 5B 9B 02. Therefore, the entire command is C0 49 81 5B 9B 02.

Finally, we need to write an alternate cutscene that is shown if the guards have been defeated. I want

to show the player a textbox containing the message “There may be more where that came from. Let's just go!”. Then the party will move a space upwards. We will replace the text at index 95 in FF3usME because it is clearly unused. Then we can use 4B 60 00 to display the textbox and 31 82 80 FF to move the party. The alternate cutscene will be placed at CC/9B5B, which is the address that we jumped to with the $C0 command. Here is the final cutscene:

```
000C9B10  18 9B 02 B3 5E 00 3A FE 4B D4 06 3A FE C0 49 81  .›.³^.:þKÔ.:þÀI.
000C9B20  5B 9B 02 3D 10 41 10 3D 11 41 11 10 08 D5 25 3A  [›.=.A.=.A...Õ%:
000C9B30  C3 8C A0 80 FF 11 8D D5 27 3A C3 88 CF E0 01 CD  ÃŒ €ÿ..Õ':Ã^Ïà.Í
000C9B40  E0 01 8C FF 35 10 4B 0C 00 55 20 92 4D 01 3F 3E  à.Œÿ5.K..U ’M.?>
000C9B50  10 3E 11 96 31 82 80 FF D2 49 FE 4B 60 00 31 82  .>.–1‚€ÿÒIþK`.1‚
000C9B60  80 FF FE 88 CD E0 02 CC E0 06 80 FF 36 0E 36 0F  €ÿþ^Íà.Ìà.€ÿ6.6.
```

Conveniently, the entire cutscene fits in the space provided by the vanilla cutscene (the next cutscene start at CC/9B71, as shown by the Event Script Dump), so we don't have to jump to extra space to make everything fit. If this was necessary, it could have been accomplished with the B2 command.

Reading the Event Script Dump

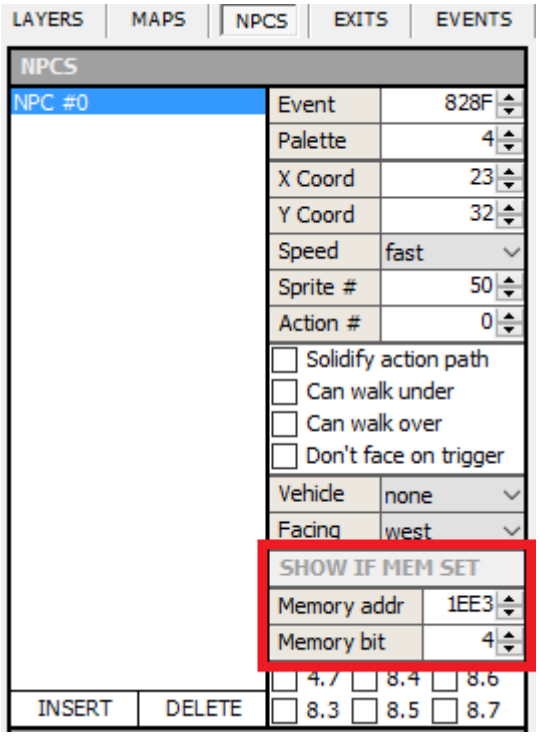
Event bits are represented in different ways depending on what document or tool you are using to view them. In the Event Bits Document, each Event Bit is represented by a number between $000 and $2FF, as you have seen. However, if you look in the Event Script Dump, you may see a line like the following:

```
Set event bit $1E80($1CC) [$1EB9, bit 4]
```

This line only refers to a single event bit ($1CC), which has been bolded and underlined. Knowing this will help you to cross-reference the Event Bits Document and the Event Script Dump.

Event Bits Higher Than $2FF

You may have noticed that event bits higher than $2FF are not listed in the Event Bits Document, yet they are often set or cleared in the Event Script Dump. These event bits are used to determine whether NPCs are immediately shown when you load a map. For example, bit $31C controls whether Vargas appears on Mt. Kolts. This bit is cleared after you fight him so that he is not shown if you return to the map. You can view and modify which bit corresponds to a particular NPC in the LE. Just select any NPC and look at the “SHOW IF MEM SET” field.



You can cross reference this with the event dump by looking at the following part of the line:

`Set event bit $1E80($31C) [`**`$1EE3, bit 4`**`]`

In this way, you can tell that “$1EE3, bit 4” corresponds to event bit “$31C”.