The game has something called the move name table. And it's a list of every move in the game, separated by terminating characters (0x50). When the game needs to buffer a name of a move, it calls a special subroutine which loads the table, loops through that table until it reaches an indexth move, and writes 20 characters from the found list location to a temporary location $CD6D.
When you open up the fight menu, in order to load the names of moves your Pokemon has, the game uses that subroutine mentioned before. After it completes, the game reads characters from the address $CD6D until a null terminator (0x50) is encountered. Then the name is appended to the move list so it can be displayed on the screen later.
As we can see, there is completely no error checking in any of these parts, the game heavily relies that supplied data will be valid - so what will happen if we try to request a name for an index number greater than 0xA5, like 0xB6? The searching function will start searching beyond the table, as the table has only 0xA5 entries and it has no more data programmed. It will keep searching and searching until it finds a null character (0x50) somewhere. The loop will eventually end up going for so long, that the game will start looking for the names in the RAM. As the RAM’s contents constantly change, it’s impossible to predict the location the function will choose. Let’s say the subroutine luckily and completely by accident finds the character at $CC9F.
This makes the function believe that this move’s name resides at address CC9F. It's obvious that this address does not contain any move names at all – it may contain internal timers, screen data or various buffers, depending on how far did the search go. Then, the subroutine copies these 20 bytes of garbage to the buffer and returns to the main function.
Now, the very important fact: At the time Generation I Pokémon series were developed, every byte and every execution cycle was important. Because every normal, nonglitched move has length of at most 14 bytes (13 characters and a null terminator), and the subroutine always copies 20 bytes, programmers have decided not to make sure the buffer is always terminated, as when dealing with a nonglitched move, the terminating 0x50 character will always get copied along with the name. However, that’s not the case when we are dealing with glitch moves. What happens is the game will try to copy this buffer to the moveset array, and will keep copying until it finds a null character. However, the buffer now contains just garbage and there’s absolutely no guarantee that the game manages to find the terminator.
If the game finds the terminator by accident, nothing will happen, you are safe. But if it won't find a null terminator soon enough, the game will attempt to read and copy characters beyond the buffer. Gameboy has no memory protection, so this operation will be happily accepted and executed, causing bytes after the move array to be overwritten.

So what we have here is a standard buffer overflow.