What This Video Assumes

- You know how to put CFW on your Nintendo Switch
- You have Atmosphere installed on your Nintendo Switch
- You know how to put Homebrew on your Nintendo Switch. We'll be using the homebrew app Edizon for most of these tutorials. This is because SX OS's current code finder is glitched and doesn't display values properly.
- You know how to use Edizon and cheat files to cheat in-game
- Cheats made in Atmosphere, for the most part, should work in SX OS, so if you prefer SX OS, make your cheats with Atmosphere and switch back to SX OS after you're finished.

Video Contents

- Decimal vs Hexadecimal
- Unsigned vs Signed
- Data-Types
- Address Types
- How to Find Known-Value Addresses
- Code Types (Code Type 0)
- And examples and explanations of each

Decimal vs Hexadecimal

- Decimal:

  - The system most people are used to using. It expresses numbers through the digits 0-9, with zero being the smallest digit and 9 being the largest.

  - The prefix dec is 10, so there are ten characters.

- Hexadecimal:

  - The system we will be using. It expresses numbers through the digits 0-9, and then the letters A-F. Zero is the smallest digit and F is the largest digit.

  - The prefixes hex is 6, and dec is 10, so there are 16 characters

Decimal vs Hexadecimal (EX)

- Count from 0 to 10 in both decimal and hexadecimal

  Decimal: 0...1...2...3...4...5...6...7...8...9...10

  Hexadecimal:
  0...1...2...3...4...5...6...7...8...9...A...B...C...D...E...F...10
- 0-9 in hexadecimal is equal to 0-9 in decimal
- A-F in hexadecimal is equal to 10-15 in decimal
- 10 in hexadecimal is equal to 16 in decimal

Decimal vs Hexadecimal (Cont)

- From here on out, hexadecimal values will be represented by an 0x in front of them (EX: 10 in hexadecimal is now 0x10 and is 16 in decimal)
- We will use a calculator to go from decimal to hexadecimal
- http://calc.penjee.com/
- Set calculator to “dec” to convert decimal to hexadecimal
- Set calculator to “hex” to convert hexadecimal to decimal
- You can also set Data-Type and if a value is signed or not. More on that next slide.

Unsigned vs Signed

- We're going to go over what are called data-types. Data-types can be unsigned or signed.
- An unsigned data-type only uses positive numbers while signed data-types can use positive and negative numbers.
- I've never seen a signed Data-Type on the Nintendo Switch before, so you'll likely be dealing with an unsigned value.

Data-Types

- An address is where your value is located in memory, much like a house address.
- Data-Types represent how big numbers can be in a particular address. Your house can only hold so much stuff.
- These tutorials will center around where to find where the value we want to modify, like ammo or money, is located.
- We will work with 4 different Data-Types. To find out which one we're dealing with, we'll need to go through the process of elimination. Here are the different Data-Types:

  8-Bit, 16-Bit, 32-Bit, and 64-Bit

Data-Type: 8-Bit

- If it is unsigned, it holds decimal numbers from 0 to 255 (0x00-0xFF).
- If it is signed, it holds integer numbers from 0 to 127 (0x00-0x7F) and then integers from -128 to -1 (0x80-0xFF).
- A byte is two hexadecimal characters. An 8-bit address has one byte.

Data-Type: 16-Bit

- If it is unsigned, it holds decimal numbers from 0 to 65535 (0x0000-0xFFFF).
- If it is signed, it holds integer numbers from 0 to 32767 (0x0000-0x7FFF) and then integers from -32768 to -1 (0x8000-0xFFFF).
- Has two bytes

Data-Type: 32-Bit

- If it is unsigned, it holds decimal numbers from 0 to 4294967295 (0x00000000-0xFFFFFFFF).
- If it is signed, it holds integer numbers from 0 to 2147483647 (0x00000000-0x7FFFFFFF) and then integers from -2147483648 to -1 (0x80000000-0xFFFFFFFF).
- Has four bytes
- The most common data-type, at least for the Nintendo Switch.

Data-Type: 64-Bit

- If it is unsigned, it holds decimal numbers from 0 to 18446744073709551615 (0x0000000000000000-0xFFFFFFFFFFFFFFFF).
- If it is signed, it holds integer numbers from 0 to 9223372036854775807 (0x0000000000000000-0x7FFFFFFFFFFFFFFF) and then integers from -9223372036854775808 to -1 (0x8000000000000000-0xFFFFFFFFFFFFFFFF).
- Has eight bytes
- The least common data-type, at least for the Nintendo Switch.

How Do I Know Which Data-Type I'm Dealing With?

- Have you ever seen a negative number in the value you're looking for? If not, it's unsigned. If you have, it's signed.
- How high have you seen the value go? Here, we'll use the process of elimination.

How Do I Know Which Data-Type I'm Dealing With (Cont)?

- If you've never seen the value above 255 or 127 and/or never seen the lowest negative value be below -128, then it's probably 8-bit. It could potentially be any of the other Data-Types, but likely not.
- If you've never seen the value above 65535 or 32767 and/or never seen the lowest negative value below -32768, but have seen it above 255 and/or have seen it below the negative value -128 (i.e. -129), then it's probably 16-bit. It could be 32 or 64-bit, but not 8-bit.
- If you've never seen the value above 4294967295 or 2147483647 and/or never seen the lowest negative value below -2147483648, but have seen it above 65535 and/or have seen it below the negative value -32768, then it's probably 32-bit. It could also be 64-bit. It cannot be 8 or 16-bit.
- If you've never seen the value above 18446744073709551615 or 9223372036854775807 and/or have never seen the lowest negative value below -9223372036854775808, but have seen the value above 4294967295 and/or have seen it below the negative value -2147483648, then it's 64-bit. It cannot be 8, 16, or 32-bit.
- When in doubt, assume 32-Bit as it’s the most common data-type.

Address Types

- Just like you can move from your house to get a new address, so too can values in a game's memory.
- Addresses are classified by how much they move in the memory.
- MAIN Address: Addresses of this type never moves. Typically associated with game code itself. Can also be associated with in-game values like ammo and money.
- HEAP Address: Addresses of this type typically moves whenever you switch to another level or quit the game and relaunch it. However, in some games, there are HEAP Addresses that never move. Typically associated with in-game values like ammo and money.
- BASE Address: Addresses of this type always move whenever you switch to another level or quit the game and relaunch it. These values change so much it might change location before the level even ends or every few seconds. Typically associated with in-game values like ammo and money. BASE is all other addresses that aren’t in MAIN or HEAP.

Address Types (Cont)

- Addresses will be represented as [Address Type+0x??????????] for the Nintendo Switch. Our cheating software will always tell what address type you're dealing with.
- The value on the right side of the plus sign will always be 10 digits long.
- If an address in our software (Edizon, SX OS) returns as [MAIN+0x0000F00000], then that means the location of the address is 0xF00000 away from the start of MAIN. Look at it like an addition problem. You don’t have to do any math like this until we start working with Noexs. So, when making the types of codes shown in this video, you don’t have to do this math, as Edizon/SX OS will do it for you.
- If MAIN is equal to 0x800, the actual address is 0x0000F00800 in the example above.
- For the first part of this video, we will deal with different Data-Types, but all addresses will be MAIN or Static HEAP. We will eventually learn how to deal with Dynamic HEAP and BASE.

Address Types (Cont)

- If a MAIN Address is, say, [MAIN+0x0000F00000], it will always be [MAIN+0x0000F00000].
- If a HEAP Address is, say, [HEAP+0x004F000000], it may be [HEAP+0x004F000000] on the next restart of the game/level change. It could also be [HEAP+0x0050000000] or any number of things.
- If a BASE Address is, say, [BASE+0x8F00000000], it will definitely be something else on the next restart of the game/level change. It could be [BASE+0x8EF0000000] or any number of things.
- There are exceptions to MAIN Addresses and Static HEAP Addresses, though. If the game updates, even static addresses will change, rendering the last code you made unfunctional. However, after this change, you likely would only have the find the address once more and it’d be static again. No guarantees on this, though.

How to Find Known-Value Addresses

- Set your Data-Type and whether or not the value is signed.
- Search for the number that your value is currently at.
- Change the value in some way by making it go up or down. Search for the new value.
- This method uses the process of elimination to find the address we're looking for. The software checks all addresses for the value you input and then eliminates the ones that don't match.
- Eventually, you'll get down to only a few addresses. If you cannot get it down to one address, you'll need to test every address that's left. Some addresses might only be associated with the visual value, not the true value, so you need to test each until you get the outcome you want in game. Something might show as having 5 bullets in-game with an address written to 5, but have to reload on the 6th bullet. That means you only found the value for the visual ammo counter, not the actual ammo counter.
- Once you find the address, create a code to change it to whatever you want. In some cases, you might need to set multiple addresses to the value you want to change your value to. This is uncommon, and usually, your value is handled by only one address, but things like character IDs might need multiple addresses to be set to the same thing (EX: Super Smash Bros. Ultimate Character Modifier Code).

How to Find Known-Value Addresses (EX)

- Coin Value While Racing in Mario Kart

  (Unsigned HEAP 8-Bit Address)
- Pokemon Shield Watts (Signed HEAP 32-Bit Address)
- A Fabricated MAIN 16-Bit Address (As they’re so uncommon in my experience that the only one’s I’ve found are too complex for this video.)

Code Types

- Code types are what are in cheat codes to tell the cheat software what to do.
- In this video, we'll be dealing with Code Type 0: Write to Memory. We'll use the addresses we found in the last part and hack them using Code Type 0.

Code Type 0: Write to Memory

0TMR00AA AAAAAAAA YYYYYYYY

- T = Data-Type (1 for 8-bit, 2 for 16-bit, 4 for 32-bit, 8 for 64-bit. T represents the amount of bytes a data-type has.)
- M = Address Type (0 = MAIN, 1 = HEAP)
- R = Register (Hexadecimal value from 0x0-0xF. Registers, in the context of our cheating software, are special addresses we can store things in to make things easier on ourselves. For Code Type 0, using 0 will suffice.)
- A = Address Relative to MAIN/HEAP (Whatever is to the right of the plus sign in the address. This is without the 0x part as the program will assume you're working in hexadecimal already. Doesn’t work for BASE.)
- Y = What to Write to the Address

Explaining Code Type 0

- Let's say you used the code
  04100000 0000F000 7FFFFFFF
- T = 4, so the address is 32-Bit and we're writing 4 Bytes.
- M = 1, so the address is a HEAP Address.
- A = 0x000000F000, so the address we're writing to is [HEAP+0x000000F000]
- R = 0, so the address found from the calculation above becomes the address stored in Register 0.
- Let's say HEAP = 0x0010000000 (what HEAP equals will change, but the distance from HEAP shouldn't if the code is static. Therefore, the code continues to work).
- 0x0010000000+0x000000F000 = 0x001000F000 (the real address). 0x001000F000 becomes Register 0.
- The value 0x7FFFFFFF is then moved into Register 0 (which means 0x7FFFFFFF is written or moved into 0x001000F000). For those of you who've used x86, the code can be represented by the function:

  MOV r0, 7FFFFFFF

Explaining Code Type 0 (Cont)

- Code Type 0 changes slightly based on the data-type.
- If the Data-Type is 8-Bit, the code type is as follows:

  01MR00AA AAAAAAAA 000000YY (only the last byte is written)
- If the Data-Type is 16-Bit, the code type is as follows:

  02MR00AA AAAAAAAA 0000YYYY (the last two bytes are written)
- If the Data-Type is 32-Bit, the code type is as follows:

  04MR00AA AAAAAAAA YYYYYYYY
- If the Data-Type is 64-Bit, the code type is as follows:

  08MR00AA AAAAAAAA YYYYYYYY YYYYYYYY

Code Type 0:
Write to Memory (EX)

- We’ll now take the three examples that we made from the How to Find The Address of Your Value (EX) slide and use them in Code Type 0 to write them to whatever we want.

Data-Type and Sign Test with Code Type 0

- There’s a possibility the Data-Type of your code is bigger than the one you’ve selected, as mentioned previously.
- To check this, change the Code Type to work with the next highest Data-Type and then write the maximum unsigned value of the new Data-Type to the Address.
- Let’s say you think your value is 8-Bit. Try converting the code to 16-Bit and writing 0x7FFF. If 32767 shows, it’s at least 16-Bit. Progress further from here until you get a Data-Type whose highest value doesn’t show in-game or crashes the game. The Data-Type before this one is your true Data-Type.
- For instance, let’s use an 8-Bit Address as shown above. Let’s convert Code Type 0 to 16-Bit and write 0x7FFF. If 32767 shows up in-game, it’s at least 16-Bit. If the value doesn’t show up in-game or crashes, it’s 8-Bit. Now, Convert Code Type 0 to 32-Bit and write 0x7FFFFFFF if 32767 showed up in-game. If 2147483647 shows up in-game, it’s at least 32-Bit. If the value doesn’t show up in-game or crashes, it’s 16-Bit. Finally, convert Code Type 0 to 64-Bit and write 0x7FFFFFFFFFFFFFFF if 2147483647 showed up in-game. If 9223372036854775807 shows up in-game, it’s 64-Bit. If the value doesn’t show up in-game or crashes, it’s 32-Bit.
- Also, some values might not show up correctly but still indicate a different data-type. For instance, a particular value might only be designed to show three numbers, but be 16-Bit. Writing 32767 in this case could cause 767 to show up, which is still an indication the value is at least 16-Bit.
- There is also a chance a value you think is unsigned is actually signed. To check this, simply write a value that would be negative if the value were signed based on its data-type. For instance, let’s say we have a 8-Bit value that might be signed. We could write 0xFF to that value. If -1 was shown, the value is signed. If 255 was shown, the value is unsigned. You can do the same searches by searching for 0xFFFF in 16-Bit, 0xFFFFFFFF in 32-Bit, and 0xFFFFFFFFFFFFFFFF in 64-Bit, which all yield -1 in those respective data-types. Searching for 0x00FF in a value that is 16-Bit will NOT turn up -1, rather, it’d turn up 255.

Data-Type Hints

- In the format [Address Type+0x??????????Y], where Y is 1, 3, 5, 7, 9, B, D, or F, the data-type of the address is definitely 8-Bit. No need to test for data-type from there.
- In the format [Address Type+0x??????????Y], where Y is 2, 6, A, or E, the data-type of the address is either 8-Bit or 16-Bit.
- In the format [Address Type+0x??????????Y], where Y is 0, 4, 8, or C, the data-type of the address can be anything. Use the processes before this slide to determine what data-type you're working with.

Sections That Carry Over to Other Consoles

- Hexadecimal vs Decimal
- Unsigned vs Signed
- Data-Types (Some consoles don’t use certain data-types. An 8-bit system can only use the 8-Bit Data-Type. A 16-Bit system can use 8-Bit and 16-Bit Data-Types. A 32-Bit system can use from 8-Bit to 32-Bit Data-Types. A 64-Bit system can use all data-types.)
- How to Find Known-Value Addresses
- Also, some other console cheating softwares (EX: Action Replay DS, GameShark, etc.) will use different, but often similar code types. You’d just need to learn their code types. Dealing with code types from one software can make it easier to deal with code types from another software, though.