interested in set on '0'. In this case, mask would be 1111111111111101 in binary, which we convert to 0xFFFD in Hex.

If you use the condition 9XXXXXXX FFFD0002, it will always activate when the value of the second bit is '1' at address XXXXXXXX.

With this mask in place, the following values would have the following effects:

CODE ACTIVE		CODE IN-ACTIVE	
Hex	Binary	Hex	Binary
0x0002	0000000000000010	0xFFF0	1111111111110000
0x0003	0000000000000011	0x0001	0000000000000001
0xFFFF	1111111111111111		
0XFFF2	1111111111110010		

The values in the left column activate the code because they all have the second bit from the right set to '1', regardless of what other bits are set to. The values in the right column will not activate the code because they all have the second bit to the right set to '0', again, regardless of what other bits are set to.

7.b. Button Press Locations

Sophisticated Action Replay codes often use button presses to activate or deactivate them or to control them in some way (like cycling through items using 'LEFT' or 'RIGHT'). In order to use button presses in our codes, we need a way to know their state.

In the previous section, we looked at using a mask on a 16-bit conditional code type and used the example of the location that stores the current pressed state of the DS's buttons. If you have not done so, read Section 6.a. for an understanding of how masks can give you better precision when working with button-press states.

To set up a condition on a button press in your own codes, you can find a generic memory location in the hardware registers at address 0x4000130.

Use the Hex viewer to jump to that address and enable auto-refresh. Try pressing each button on your DS console and watch the value at that address change.

Button(s)	Value	Button(s)	Value	Button(s)	Value
A	FE	A + SELECT	FA	START + UP	B7
B	FD	A + UP	BE	START + DOWN	77
X	n/a	A + DOWN	7E	START + LEFT	D7
Y	n/a	A + LEFT	DE	START + RIGHT	E7
START	F7	A + RIGHT	EE	SELECT + UP	BB
SELECT	FB	B + START	F5	SELECT + DOWN	7B
UP	BF	B + SELECT	F9	SELECT + LEFT	DB
DOWN	7F	B + UP	BD	SELECT + RIGHT	EB
LEFT	DF	B + DOWN	7D	UP + LEFT	9F
RIGHT	EF	B + LEFT	DD	UP + RIGHT	AF
A + B	FC	B + RIGHT	ED	DOWN + LEFT	5F
A + START	F6	START + SELECT	F3	DOWN + RIGHT	6F

8. Useful Information

8.a. Decimal to Hexadecimal Conversion Table

Here are the hexadecimal equivalents of some decimal values you might use frequently when developing new codes.

Decimal	Hex	Decimal	Hex	Decimal	Hex
0	0	15	F	200	C8
1	1	16	10	300	12C
2	2	17	11	400	190
3	3	18	12	500	1F4
4	4	19	13	600	258
5	5	20	14	700	2BC
6	6	30	1E	800	320
7	7	40	28	900	384
8	8	50	32	999	3E7
9	9	60	3C	1,000	3E8
10	A	70	46	9,999	270F
11	B	80	50	10,000	2710
12	C	90	5A	99,999	1869F
13	D	99	63	100,000	186A0
14	E	100	64	1,000,000	F4240

Tip: Use Windows' built-in calculator (in Scientific Mode) to quickly convert any decimal value you may need into hexadecimal if the number you need isn't in the table above.

8.b. Complete List of Action Replay Engine Code-types

For explanation of symbols used in the code-type table, see key below table.

Code	Function
0XXXXXXX YYYYYYYY	32bit write of YYYYYYYY to location: (xxxxxxx + 'offset')
1XXXXXXX ????YYYY	16bit write of YYYY to location: (XXXXXX + 'offset')
2XXXXXXX ??????YY	8bit write of YY to location: (XXXXXXX + 'offset')
3XXXXXXX YYYYYYYY	**32bit 'If less-than' instruction.** If the value at (XXXXXXX or 'offset' when address is 0) < YYYYYYYY then execute the following block of instructions. Conditional instructions can be nested.
4XXXXXXX YYYYYYYY	**32bit 'If greater-than' instruction.**

	If the value at (XXXXXXX or 'offset' when address is 0) > YYYYYYYY then execute the following block of instructions. Conditional instructions can be nested.
5XXXXXXX YYYYYYYY	**32bit 'If equal' instruction.** If the value at (XXXXXXX or 'offset' when address is 0) == YYYYYYYY then execute the following block of instructions. Conditional instructions can be nested.
6XXXXXXX YYYYYYYY	**32bit 'If not equal' instruction.** If the value at (XXXXXXX or 'offset' when address is 0) != YYYYYYYY then execute the following block of instructions. Conditional instructions can be nested.
7XXXXXXX ZZZZYYYY	**16bit 'If less-than' instruction.** If the value at (XXXXXXX or 'offset' when address is 0) masked by ZZZZ < YYYY then execute the following block of instructions. Conditional instructions can be nested.
8XXXXXXX ZZZZYYYY	**16bit 'If greater-than' instruction.** If the value at (XXXXXXX or 'offset' when address is 0) masked by ZZZZ > YYYY then execute the following block of instructions. Conditional instructions can be nested.
9XXXXXXX ZZZZYYYY	**16bit 'If equal' instruction.** If the value at (XXXXXXX or 'offset' when address is 0) masked by ZZZZ == YYYY then execute the following block of instructions. Conditional instructions can be nested.
AXXXXXXX ZZZZYYYY	**16bit 'If not equal' instruction.** If the value at (XXXXXXX or 'offset' when address is 0) masked by ZZZZ != YYYY then execute the following block of instructions. Conditional instructions can be nested.
BXXXXXXX ????????	**Load offset register.** Loads the offset register with the data at address (XXXXXXX + 'offset') Used to perform pointer relative operations.
C??????? NNNNNNNN	**Repeat operation.** Repeats a block of codes for NNNNNNNN times. The block can include conditional instructions. Repeats blocks cannot contain further repeats.
D0?????? ????????	**End-if instruction.** Ends the most recent conditional block.
D1?????? ????????	**End-repeat instruction.** Ends the current repeat block. Also implicitly ends any conditional instructions inside the repeat block.
D2?????? ????????	**End-code instruction.** Ends the current repeat block (if any), then End-if's any further outstanding

	conditional statements. Also sets ‘offset’ and ‘stored’ to zero.
D3?????? YYYYYYYY	**Set offset register.** Loads the offset register with the value YYYYYYYY.
D4?????? YYYYYYYY	**Add to ‘stored’.** Adds YYYYYYYY to the ‘stored’ register.
d5?????? YYYYYYYY	**Set ‘stored’.** Loads the value YYYYYYYY into the ‘stored’ register.
D6?????? XXXXXXXX	**32bit store and increment.** Saves all 32 bits of ‘stored’ register to address (XXXXXXXX + ‘offset’). Post-increments ‘offset’ by 4.
D7?????? XXXXXXXX	**16bit store and increment.** Saves bottom 16 bits of ‘stored’ register to address (XXXXXXXX + ‘offset’). Post-increments ‘offset’ by 2.
D8?????? XXXXXXXX	**32bit store and increment.** Saves bottom 8 bits of ‘stored’ register to address (XXXXXXXX + ‘offset’). Post-increments ‘offset’ by 1.
D9?????? XXXXXXXX	**32bit load "stored" from address.** Loads ‘stored’ with the 32bit value at address (XXXXXXXX + ‘offset’)
DA?????? XXXXXXXX	**16bit load ‘stored’ from address.** Loads ‘stored’ with the 16bit value at address (XXXXXXXX + ‘offset’)
DB?????? XXXXXXXX	**8bit load "stored" from address.** Loads ‘stored’ with the 8bit value at address (XXXXXXXX + ‘offset’)
EXXXXXXX NNNNNNNN VVVVVVVV VVVVVVVV * ((NNNNNNNN+7)/8)	**Direct memory write.** Writes NNNNNNNN bytes from the list of values VVVVVVVV to the addresses starting at (XXXXXXX + ‘offset’)
FXXXXXXX NNNNNNNN	**Memory copy.** Copies NNNNNNNN bytes from addresses starting at the ‘offset’ register to addresses starting at XXXXXXXX.

Key to symbols used in the code-type table

Symbol	Meaning
????	Values here don’t matter
xxxx	Address
yyyy	Data
zzzz	Mask
nnnn	Count
vvvv	Direct values
“Offset”	A code-engine register used to hold an address offset