← lostscrews The STM Release Exploit →

The weird and wonderful CIC

January 17th, 2010 by Segher · 29 Comments

I have been spending some time on reverse engineering the Nintendo CIC ROMs. The CIC is the "lockout" chip in NES/SNES/N64 cartridges, used to ~~get an iron grip on the market~~ prevent people from copying games. It was manufactured by Sharp and is likely one of their old "one-chip microcomputers", used in calculators and TV remotes and the like. I couldn't find a document describing the instruction set it uses (or its architecture!), so I made it all up (combining information from lots of sources: old datasheets, old patents, and the low-res die photographs).

The N64 chips are different, and I haven't seen a ROM dump of those yet, so all of the following is NES/SNES only.

There is one chip inside the console, and one in every cartridge; the code inside the chip decides what to do based on a pin strap (the console one will be the "lock", and the cartridge one will be the "key"). The two chips run off the same clock, and they run the same code, so they run in lockstep (sometimes they execute different codepaths, but the code is careful to take the same number of cycles on both paths in these cases). The chips communicate over two wires, one from key to lock, one from lock to key. Both chips calculate what bits they will send, and what the other guy should send; if what they receive is not the same as what they should have received, they panic, and the lock chip resets the console.

Here is the pinout of the CIC:

```
              +------------------+
 DATA_OUT <-- | 1 P0.0    +5V 16 |
  DATA_IN --> | 2 P0.1        15 |  ?
     SEED --> | 3 P0.2        14 |  ?
LOCK/-KEY --> | 4 P0.3        13 |  ?
              | 5 Xout   P1.3 12 | <-- RESET_SPEED_B
              | 6 Xin    P1.2 11 | <-- RESET_SPEED_A
              | 7 RESET  P1.1 10 | --> SLAVE_CIC_RESET
              | 8 GND    P1.0  9 | --> -HOST_RESET
              +------------------+
```

The LOCK/-KEY pin is the strap pin I talked about above. The SEED pin has a capacitor connected to it; the discharge time of that is supposedly somewhat random, the lock chip times it and uses that as a random generator, to decide which of 16 possible streams to generate. It tells the key chip which one it chose.

The lock chip can reset the key chip (pin 10 on the lock is wired to pin 7 on the key), and it can reset the console. The RESET_SPEED pins are used on the 3195 to decide at what speed to "blink" the reset line (it's connected to a LED as well): about 0.4s, 0.6s, 0.8s, 1.0s each of on/off.

There are dumps of the ROMs here, here, and here. All credits for doing these go to neviksti; thanks!

All the bits in those dumps are inverted (0 vs. 1); if you want to play along with the disassembler I'll give a link to in a second, you'll need to fix that; also, that third ROM is 768 bytes, which I don't handle in my little conversion script, so you'll need to remove the extra columns (they are empty anyway). Or enhance the script if you want to.

Okay then, here is that disassembler. Usage should be self-explanatory.

This ancient CPU looks mighty strange to modern eyes. Let me try to explain the architecture:

First, it is a 4-bit CPU. Yessir. It has an accumulator register, A, and a secondary register, X, both 4 bits. All RAM accesses are done via a single pointer register B, which is 6 bits; the CIC chip only has 32 nybbles of RAM though. There is also a carry flag, C.

Then, there is the process counter, PC. It is 10 bits, but there are only 512 bytes of ROM (except on the 3195, it has 768). The ROM is divided into banks of 128 bytes. When the CPU increments PC, it never touches the bank number.

Well, “increments”. To save chip area, they didn’t use a binary counter, but a polynomial counter; “incrementing” works by shifting the PC by one bit to the right, and setting the the top bit to 1 if and only if the bottom two bits were the same.

There are no conditional branch instructions; instead, various instructions can skip the next instruction if some condition is true (the instruction still takes time, it just doesn’t do anything). Oh, all instructions take one cycle; except for the two byte instructions, which take two cycles.

Finally, there is a four entry stack for the PC; it’s not in RAM, it is separate.

Now the instruction set:

```
"skip" means "do not execute next instruction"
"M" means "the RAM nybble addressed by B"
"BL" means "the low four bits of B"
"BM" means "the high two bits of B"
"PN" means "I/O port number BL"
"x.y" means "bit y of x"

00+N  adi N     "add immediate", A := A + N, skip if overflow   (00 is nop)
10+N  skai N    "skip acc immediate", skip if A = N
20+N  lbli N    "load B low immediate", BL := N
30+N  ldi N     "load immediate", A := N

40    l         "load", A := M
41    x         "exchange", swap A with M
42    xi        "exchange and increment", swap A with M, increment BL, skip if overflow
43    xd        "exchange and decrement", swap A with M, decrement BL, skip if underflow
44    nega      "negate acc", A := -A (two's complement)
46    out       "output", PN := A
47    out0      "output zero", PN := 0
48    sc        "set carry", C := 1
49    rc        "reset carry", C := 0
4a    s         "store", M := A
4c    rit       "return", pop PC from stack
4d    ritsk     "return and skip", pop PC from stack, skip
52    li        "load and increment", A := M, increment BL, skip if overflow
54    coma      "complement acc", A := ~A (ones' complement)
55    in        "input", A := PN
57    xal       "exchange acc and low", swap A with BL
5c    lxa       "load X with acc", X := A
5d    xax       "exchange X and acc", swap X with A
5e     ?        SPECIAL MYSTERY INSTRUCTION

60+N  skm N     "skip memory", skip if M.N = 1
64+N  ska N     "skip acc", skip if A.N = 1
68+N  rm N      "reset memory", M.N := 0
6c+N  sm N      "set memory", M.N := 1

70    ad        "add", A := A + M
72    adc       "add with carry", A := A + M + C
73    adcsk     "add with carry and skip", A := A + M + C, skip if overflow

74+N  lbmi N    "load B high immediate", BM := N
```

```
78+N NN  tl NNN    "transfer long", PC := NNN
7c+N NN  tml NNN   "transfer module long", push PC+2, PC := NNN
80+NN    t NN      "transfer", low bits of PC := NN
```

It would seem that on the 3195, the sc and rc instructions are swapped, as are the coma and nega instructions.

If you look at the code in the ROMs, you'll notice something strange with the ldi instructions: sometimes it runs two in a row. Descriptions for similar CPUs say that if you have two or more ldi instructions in a row, all but the first are skipped. The code still doesn't make sense then; I suspect that this CPU does this skip only if some condition that I do not know yet is true.

This architecture is quite different from what we are used to today, and so it requires quite different programs; I'll leave it to you to discover all the intricacies yourself though, it's more fun that way!

I put a commented disassembly of these ROMs here. Some of that is a work in progress.

I hope you all find this as fascinating as I did!

[edit: fixed op 52 "li" description]

Tags: Other consoles