

# Code marker

## Mark at start

1. If you need to trace all the functions involved in the game.
  2. If you want to trace the code from the current moment, or loaded the savestate.
- \* If end point of marking is known, you can set a stop point.

## Mark from

1. If you want to trace the code from the specified moment until completion at the specified point.

## Patch

1. If you want to manually change the code execution order.

## The marked data:

1\_call\_marked.bin – marked functions calls.  
1\_code\_marked.bin – marked code and used variables.

Marked data is stored in a folder “dump”.

dump\1\_call\_marked.bin  
dump\1\_code\_marked.bin

## Marks codes:

Call:

"2" - function.

Code:

"1" - the read byte.  
"2" - the written byte.  
"3" - the read and written byte.  
"7" - the executed instruction.

## Console messages of the marker:

Mark at start:

*Code Marker: marking start.*  
*Code Marker: marking stoped.*  
*Code Marker: marking restarted* – the markup continues, after any manipulation.

Mark from:

*Code Marker: marking enabled* – the markup will start from the specified point.(PC = start)  
*Code Marker: marking disabled .*  
*Code Marker: marking reset* – the markup start point is reset.

\* Enable the debug console:

Configuration->CPU->Enable Console Output

## Analysis:

By comparing the dumps, looking for used functions\variables.  
Then, using the debugger and IDA (The Interactive Disassembler), we look for a concrete function\variable.

**Example:**

**Game:** Tomb Rider III Adventures of Lara Croft.

**Game ID:** SLUS-00691

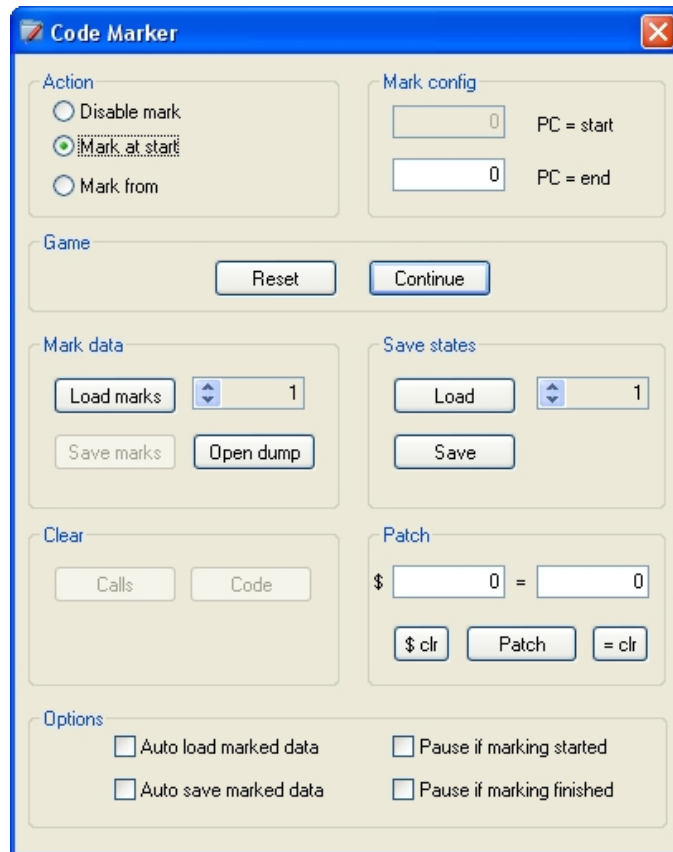
**Target:** to find the door opening code with the lever in the first level.



1. Through the window of the marker, load the savestate near the lever. (Load button)



2. Turn on the markup. (Mark at start)



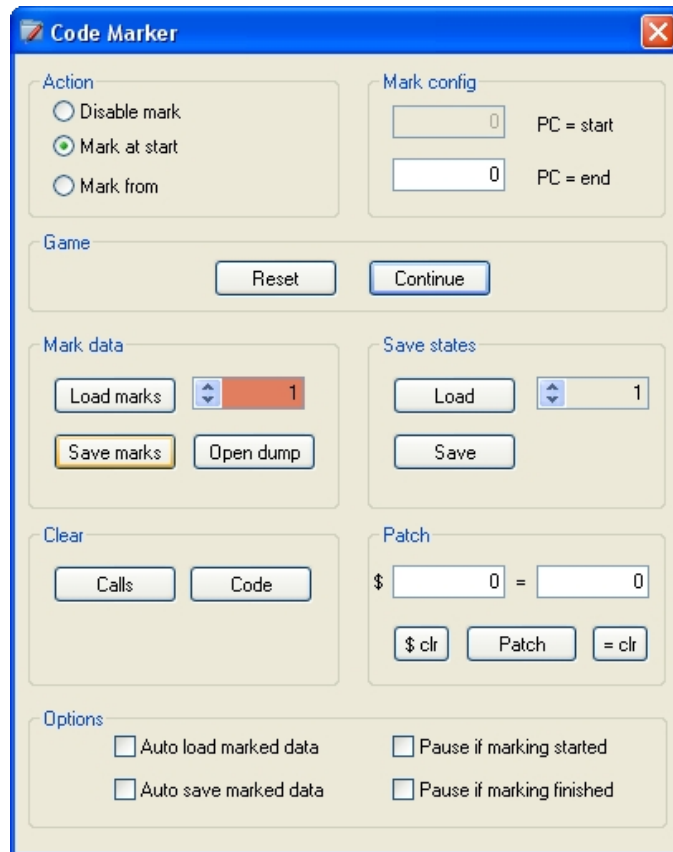
3. Click the Continue button.

4. Press X button.

5. Approach the lever closely.(R1 + UP)



6. Save the marked data at **number 1**.(Save marks)



The screenshot shows the 'Code Marker' application window. It has a blue title bar with a close button. The interface is divided into several sections: 'Action' with radio buttons for 'Disable mark', 'Mark at start' (selected), and 'Mark from'; 'Mark config' with two input fields both set to '0', labeled 'PC = start' and 'PC = end'; 'Game' with 'Reset' and 'Continue' buttons; 'Mark data' with 'Load marks' (disabled), 'Save marks' (highlighted), and 'Open dump' buttons, along with a red bar showing '1'; 'Save states' with 'Load' (disabled) and 'Save' buttons, and a blue bar showing '1'; 'Clear' with 'Calls' and 'Code' buttons; 'Patch' with a '\$' input field set to '0', an '=' input field set to '0', and '\$ clr', 'Patch', and '= clr' buttons; and 'Options' with four checkboxes: 'Auto load marked data', 'Auto save marked data', 'Pause if marking started', and 'Pause if marking finished'.

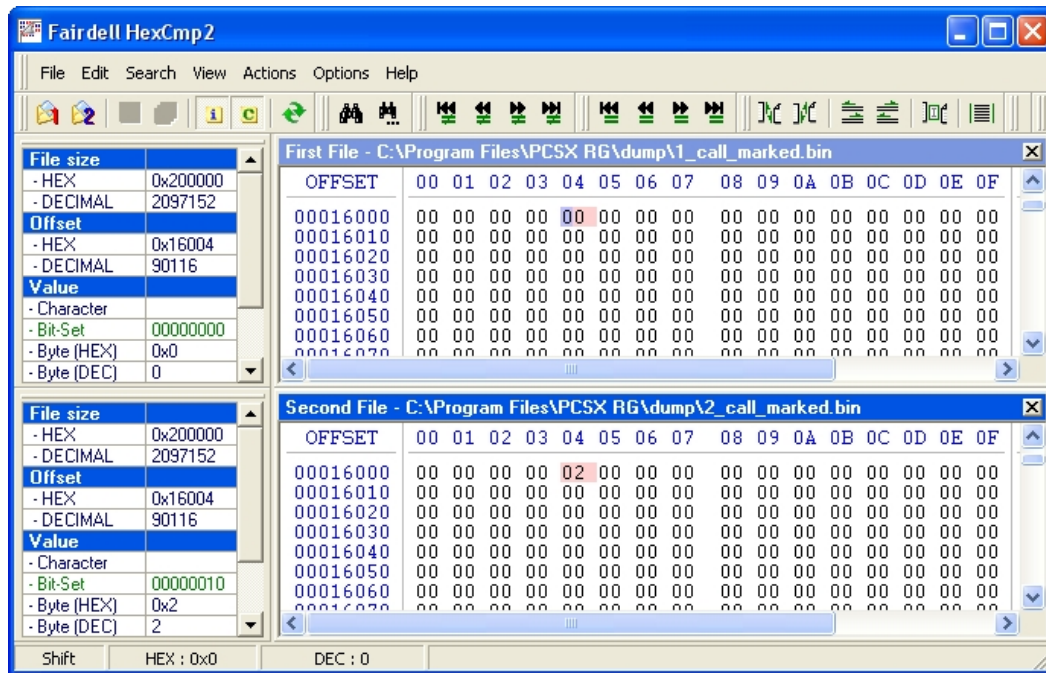
7. Pull the lever, wait until the door opens.



8. Save the marked data at **number 2**.(Save marks)

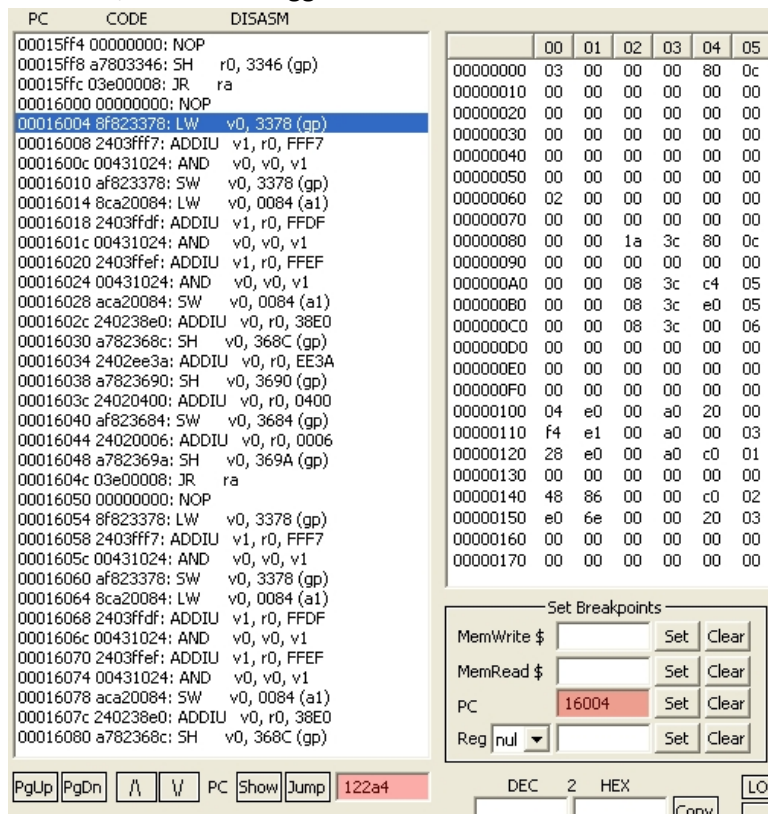


9. By comparing the marks, we get a list of the functions used to open the door.



On the screenshot, we see the mark "2" at \$16004, which means that the function associated with opening the door begins here.

10. To find out who calls this function, use the debugger.



Set the breakpoint PC = 16004, pull the lever, and get the function call code \$122a4.

To find out whether the function f->16004 has a direct relation to opening the door, we need to disable its call:

PC	CODE	DISASM
00012294	3c010009: LUI	at, 0009
00012298	00220821: ADDU	at, at, v0
0001229c	8c22ceac: LW	v0, CEAC (at)
000122a0	00000000: NOP	
000122a4	0040f809: JALR	v0, ra
000122a8	02202821: ADDU	a1, s1, r0
000122ac	96030050: LHU	v1, 0050 (s0)
000122b0	00000000: NOP	
000122b4	246400b6: ADDIU	a0, v1, 00b6
000122b8	3082ffff: ANDI	v0, a0, FFFF
000122bc	2c42016d: SLTIU	v0, v0, 016D
000122c0	10400003: BEQ	v0, r0, 000122D0
000122c4	00031400: SLL	v0, v1, 10
000122c8	080048bb: J	000122EC
000122cc	a6000050: SH	r0, 0050 (s0)
000122d0	00021403: SRA	v0, v0, 10
000122d4	2842ff4a: SLTI	v0, v0, FF4A
000122d8	10400003: BEQ	v0, r0, 000122E8
000122dc	2462ff4a: ADDIU	v0, v1, FF4A
000122e0	080048bb: J	000122EC
000122e4	a6040050: SH	a0, 0050 (s0)
000122e8	a6020050: SH	v0, 0050 (s0)
000122ec	978333d8: LHU	v1, 33D8 (gp)
000122f0	00000000: NOP	
000122f4	2464016c: ADDIU	a0, v1, 016C
000122f8	3082ffff: ANDI	v0, a0, FFFF
000122fc	2c4202d9: SLTIU	v0, v0, 02D9
00012300	10400004: BEQ	v0, r0, 00012314
00012304	00031400: SLL	v0, v1, 10
00012308	a78033d8: SH	r0, 33D8 (gp)
0001230c	080048cd: J	00012334
00012310	00000000: NOP	
00012314	00021403: SRA	v0, v0, 10
00012318	2842fe94: SLTI	v0, v0, FE94
0001231c	10400004: BEQ	v0, r0, 00012330
00012320	2462fe94: ADDIU	v0, v1, FE94

PgUp PgDn PC Show Jump

Start address: PU levels 3, PU control, PU on 0-16, PU off 16-24

Reg Patch: Reg: [dropdown], Reg. Patch

Mem Patch: Address: 122a4, 1 Byte (selected), 2 Bytes, 4 Bytes, Patch

Load the savestate near the lever.

In the Address field, enter: 122a4.

In the Bytes field, enter: 0.

Set the switch to: 4 Bytes.

Click the Patch button.

Again, pulling the lever, the door still opens, but at the same time Lara stopped walking. And this is not what we need, so we move on to the next mark, and repeat the analysis anew.

11. We turn off each function one by one and check if the door opens. If the door did not open, then we found the function we needed. Next, we'll explore it through IDA.

```

TEXT:0001C9DC
TEXT:0001C9DC loc_1C9DC: # DATA XREF: TEXT:000948F8↓o
• TEXT:0001C9DC      sll      $v0, $s2, 4
• TEXT:0001C9E0      addu     $v0, $s2
• TEXT:0001C9E4      lw       $v1, 0x34A4($gp)
• TEXT:0001C9E8      sll      $v0, 3
• TEXT:0001C9EC      li       $a3, 6
• TEXT:0001C9F0      beq      $s4, $a3, loc_1CA04
• TEXT:0001C9F4      addu     $s0, $v1, $v0
• TEXT:0001C9F8      li       $v0, 9
• TEXT:0001C9FC      bne      $s4, $v0, loc_1CA18
• TEXT:0001CA00      nop
TEXT:0001CA04
TEXT:0001CA04 loc_1CA04: # CODE XREF: sub_1C5BC+434↑j
• TEXT:0001CA04      lhu      $v0, 0x28($s0)
• TEXT:0001CA08      nop
• TEXT:0001CA0C      andi     $v0, 0x80
• TEXT:0001CA10      bnez     $v0, loc_1D004
• TEXT:0001CA14      andi     $v0, $s6, 0x8000
TEXT:0001CA18
TEXT:0001CA18 loc_1CA18: # CODE XREF: sub_1C5BC+440↑j
• TEXT:0001CA18      li       $a3, 2
• TEXT:0001CA1C      bne      $s4, $a3, loc_1CA30
• TEXT:0001CA20      nop
• TEXT:0001CA24      lhu      $v0, 0x28($s0)
• TEXT:0001CA28      j        loc_1CA50
• TEXT:0001CA2C      andi     $v0, 0x40
TEXT:0001CA30 # -----
TEXT:0001CA30
TEXT:0001CA30 loc_1CA30: # CODE XREF: sub_1C5BC+460↑j

```

#### Recommendations:

For productive code analysis, the marker needs to be used together with debugger “debugger Ver2”.

Link: <http://www.romhacking.net/utilities/267/>

\* You can copy the executable file "pcsx.exe" to the marker folder.