

SCORE 100
TIME 3:45
0

Reversing Engineering SEGA Megadrive Games

Edd Barrett

April 29, 2012

Twitter: @vext01



SCORE 100
TIME 3:45
0

Reversing Engineering SEGA Megadrive Games

Introduction



Why?

- ▶ When I was a kid I had a SEGA Megadrive (didn't we all).
- ▶ Since then I have learned a lot about reverse engineering.
- ▶ Curiosity led me to look at how these systems work.



Starting Out

TIME 3:45
0

I find the best way to learn about something is to have a clear goal.

Goal 1

Sonic 3 – Reverse the save game mechanism.

And if I succeed:

Goal 2

Make some tooling to help reverse other games too.



SCORE 100
TIME 3:45
0

Reversing Engineering SEGA Megadrive Games

SEGA Megadrive – Overview



Basic Architecture

TIME 3:45

A quick overview of the SEGA megadrive:

CPU cores

Basically a glorified m68k:

- ▶ Motorola m68000 – 64K RAM, 7.61/7.67 MHz
- ▶ Zilog Z80 – 3.58 MHz

The rest

- ▶ Yamaha YM2612 FM (Main sound chip)
- ▶ Texas Instruments SN76489 PSG (Sq. Wave / White noise)
- ▶ Custom graphics chip (VDP)



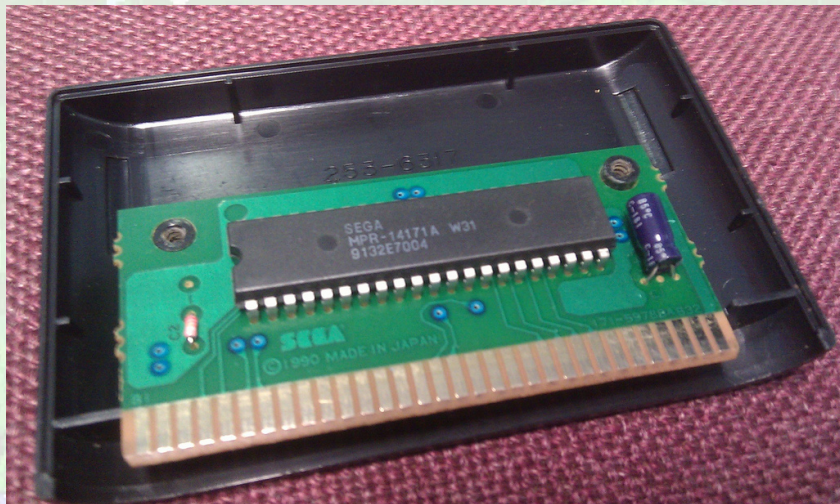
SONIC
3

Game Cartridges



Game Cartridges

TIME 3:45
0



3

Game Cartridges

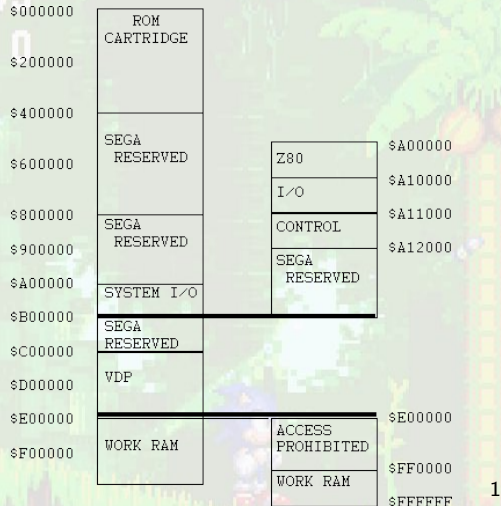
Inside a Typical Cart

- ▶ ROM
 - ▶ Game instructions
 - ▶ Sprites
 - ▶ Music
- ▶ Save RAM (Optional)
 - ▶ Stores persistent state. High scores, saves etc.
 - ▶ Usually a lithium cell retain memory
- ▶ Additional graphics hardware (Optional)
 - ▶ For any “special” graphics capabilities
 - ▶ Eg. Sega Virtua Processor



SONIC
x 3

Memory Map of the Megadrive



► Thanks to the leaked [sega2.doc](#) we know about the memory layout

¹Image borrowed from Nemesis

Memory Map of a Game Cart

- ▶ First 512 bytes are the “cart header”.
- ▶ The layout of the rest of the cart is specified inside the header.

```
% ./dgm_hdump ~/roms/Sonic\ the\ Hedgehog\ 3.bin
Console Name : [SEGA GENESIS      ]
Copyright    : [(C)SEGA 1993.NOV]
Domestic Name: [SONIC THE          HEDGEHOG 3      ]
Overseas Name: [SONIC THE          HEDGEHOG 3      ]
Game Type    : [GM]
Product Code : [ MK-1079 -00]
Checksum     : a8 f2
IO Support   : [J          ]
ROM Start    : 00 00 00 00
ROM End      : 00 1f ff ff
RAM          : 00 ff 00 00 00 ff ff ff 52 41 f8 20 00 20 00 01 00 20 03 ff
RAM Present? : [RA]
RAM Start    : 0x200001
RAM End      : 0x2003ff
Modem Data   : [          ]
Memo         : [          ?          ]
Release Country: []
```



SONIC
3

SCORE 100
TIME 3:45
0

Reversing Engineering SEGA Megadrive Games

Reversing the Sonic 3 Save RAM



How Do We Start – The Electronic Engineer's Approach

- ▶ Interface with Sonic 3 cart.
- ▶ Dump save ram to disk somehow.
- ▶ Identify field storing the level number in the save RAM.
- ▶ Modify this field.
- ▶ Upload modified RAM to cart.
- ▶ Game on!

Pretty difficult and probably requires extra hardware.



How Do We Start – The Software Engineer's Approach

Instead:

- ▶ Use emulator supporting save RAM emulation (Dgen).
- ▶ Examine on-disk save RAM dump.
- ▶ Identify field storing the level number in the save RAM.
- ▶ Modify save directly on disk.
- ▶ Game on!

Requires no special hardware or electronics knowledge :)



Bindiffing Save RAM

- ▶ First we need to find the “interesting” parts of save RAM.
- ▶ We can use a bindiff tool find these.

Sonic 3 Example

- ▶ In emulator, start game as Sonic – Dump save RAM
- ▶ Now start game as Tails – Dump save RAM
- ▶ Bindiff the two



Bindiffing Save RAM

- ▶ I used radiff2 from Radare2.
 - ▶ <http://radare.org/y/>

Begin game with Sonic vs. Begin game with tails

0x0000016c 01 => 02

0x000001cc 70 => c7

0x000001ce 4f => fd

0x000001f8 01 => 02

0x00000258 70 => c7

0x0000025a 4f => fd



SONIC
x 3

Bindiffing Save RAM

- ▶ I used radiiff2 from Radare2.
 - ▶ <http://radare.org/y/>

Begin game with Sonic vs. Begin game with tails

```
0x0000016c 01 => 02    <--- Looks like character field
0x000001cc 70 => c7     <--- ?
0x000001ce 4f => fd     <--- ?

0x000001f8 01 => 02    <--- Same as above
0x00000258 70 => c7     <--- just at different offset.
0x0000025a 4f => fd     <--- For redundancy?
```



SONIC
x 3

Bindiffing Save RAM

- ▶ Tried changing the character field
- ▶ Either 0x0 or 0x3 is likely to be Sonic+Tails
- ▶ Cart resets save RAM.
- ▶ In all likelihood the unknown bytes are a checksum.

?



Bindiffing Save RAM

- ▶ Tried changing the character field
- ▶ Either 0x0 or 0x3 is likely to be Sonic+Tails
- ▶ Cart resets save RAM.
- ▶ In all likelihood the unknown bytes are a checksum.

?

- ▶ Asked for help on ASSEMBler Games forum.
 - ▶ <http://www.assemblergames.com/forums/>



SONIC
x 3

A Response

Someone called "Jorge Nuno" replied to my post:

For the "checksum" this was from an old conversation between me and him [Tmee]:

---8<---

OK Save Ram it is copied into 0xFFFFE600. And I think this is the code that verifies the magic checksum:

```
sub_C362:
    moveq #0,d7
loc_C364:
    move.w (a6)+,d5
    eor.w d5,d7
    lsr.w #1,d7
    bcc.s loc_C370
    eori.w #$8810,d7
loc_C370:
    dbf d6,loc_C364
    rts
```

Need to checkout a6...

Probably d7 contains the result.

---8<---

A Response



A Response

TIME 3:45

Oh, and...

Sonic the Hedgehog 3 SRAM research by TmEE co.™ (2006)

	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16
01	0	0	0	0	0	0	0	0	---	Time 1	---	---	Time 2	---	---	---
02	---	Time 3	---	---	Chars	---	---	Time 1	---	---	Time 2	---	---	---	---	---
03	---	Time 3	---	---	Chars	---	---	Time 1	---	---	Time 2	---	---	---	---	---
04	---	Time 3	---	---	Chars	---	---	Time 1	---	---	Time 2	---	---	---	---	---
05	---	Time 3	---	---	Chars	---	---	Time 1	---	---	Time 2	---	---	---	---	---
06	---	Time 3	---	---	Chars	---	---	76	68	ChkSum	0	0	---	---	---	---
07	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---
08	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---
09	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---
10	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---
11	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---
12	ChkSum	0	0	---	---	---	---	Save slot 1	---	---	---	---	Save slot 2	---	---	---
13	---	---	---	---	---	---	---	Save slot 3	---	---	---	---	Save slot 4	---	---	---
14	---	---	---	---	---	---	---	Save slot 5	---	---	---	---	Save slot 6	---	---	---
15	---	---	---	---	66	68	ChkSum	0	0	0	0	0	0	0	0	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
17	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---
18	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---
19	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---
20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
21	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---
22	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---
23	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---
24	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---
25	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---
26	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---
27	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---
28	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---
29	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---
30	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---
31	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---
32	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

NOTE: ALL NUMBERS ARE DECIMAL !!!

Azure Lake times

Balloon Park times

Desert Palace times

Chrome Gadget times

Endless nine times

Notice the stage sequence. 2 copies of same stuff are used to prevent possible data corruption (?)

Time Attacks stuff

Time Format:

Byte Description

- 0 - Completed Flag (0/128)
- 1 - Minutes (0...9)
- 2 - Seconds (0...59)
- 3 - Hundreths (0...99)

Char format

- 0 - Place 1
- 1 - Place 2
- 2 - Place 3
- 3 - Unused (0)

Characters

- 0 - Sonic
- 1 - Tails
- 2 - Knuckles

Save slot format

Save slots stuff

Byte Description

- 0 - Use flag (128 unused / 0 used, same as ↑)
- 1 - Unknown (always(?) 0)
- 2 - Character (0-S&T, 1-Sonic, 2-Tails)
- 3 - Current zone (0...5)
- 4 - Special stage count (0...don't remember)
- 5 - Emerald count (0...6)
- 6 - Collected emeralds (1 BIT = 1 emerald)
- 7 - Special stages played (each BIT means 1 specific special stage ring. IF BIT is set, it means that ring has been entered)

I have no idea how the checksums are calculated. It is not like ones on ROMs as no WORD/LWORD access is possible. It is not read byte, add byte type checksum either.

& S3&K SRAM will be added here soon (if people want of course).

E-mail: tmeeco@gmail.com

Honepage: www.hot.ee/tmeeco

- ▶ Mex and I reimplemented the checksum in C.
- ▶ Wrote a tool to generate custom save RAMs for Sonic 3.

```
Usage:  dgm_s3ramgen [options] outputfile
```

Options:

- | | |
|--------|--------------------------------------|
| -c num | Character select (0=ST, 1=S, 2=T) |
| -e num | Emeralds (8-bitfield) |
| -h | Show help |
| -M | Make a MEGA-RAM (fully complete RAM) |
| -p | Pad (word-align) RAM |
| -s num | Choose slot to change |
| -x num | Debug level (0-3) |
| -z num | Choose zone (0-6) |

Eg. RAM with save slot 1 with S+T on zone 4 with 2 emeralds:

```
dgm_s3ramgen -s1 -c0 -z3 -e3 -p myramfile
```

SCORE 100
TIME 3:45
0

Reversing Engineering SEGA Megadrive Games

Goal 2: Tooling to Help



We Won't Always be So Lucky

- ▶ Now we know roughly what is involved in reversing save RAMs
- ▶ We won't be so lucky for every game.
 - ▶ Ie. If I had not had a response on the forum, what would I do?
- ▶ I would have to have found checksum code myself.
- ▶ A tool which can help can be used to reverse other games.



How to Find the Checksum Code in Sonic 3?

- ▶ We need to know the PC (program counter) when the checksum is written. WATCHPOINTS
- ▶ We need to be able to disassemble code we find here. DISASSEMBLER
- ▶ We need to read registers and memory to understand what code does. INSPECTION
- ▶ Perhaps we need to look at registers and memory just before the checksum is generated. BREAKPOINTS



SONIC
x 3

How to Find the Checksum Code in Sonic 3?

- ▶ We need to know the PC (program counter) when the checksum is written. WATCHPOINTS
- ▶ We need to be able to disassemble code we find here. DISASSEMBLER
- ▶ We need to read registers and memory to understand what code does. INSPECTION
- ▶ Perhaps we need to look at registers and memory just before the checksum is generated. BREAKPOINTS

Looks like I am writing a debugger then...



SONIC
3

Implementing a Debugger

- ▶ Take existing emulator and modify.
- ▶ I chose Dgen/SDL

Dgen/SDL

- ▶ Pretty good (fast) emulator
- ▶ Open-source
- ▶ Mature – 1999-2012 – Dgen originally designed for DOS.
- ▶ Cross platform – well ... UNIX + windows
- ▶ Original developers MIA – Currently maintained by zamaz
- ▶ Written in C++

“Dgen runs well on a P2-233” :P

Implementing a Debugger

- ▶ CPU cores pretty well written and made the task pretty easy.
 - ▶ Musashi for m68k (from Mame project)
 - ▶ CZ80 for z80
- ▶ The only challenging aspect was to make the debugger fast.

Eg. Making Breakpoints Fast

TIME 3:45

```
#define MAX_BREAKPOINTS 64  
struct dgen_bp debug_bp_m68k[MAX_BREAKPOINTS];
```

- ▶ After each instr, we have to check if any of these BPs will fire
- ▶ Checking 64 BPs if only 1 is used is wasteful (and slow)
- ▶ Do as little work as possible by storing BPs cleverly.



Eg. Making Breakpoints Fast Again

Another optimisation:

Too Slow

1. Execute single instruction
2. Check if breakpoint fires
3. Draw screen
4. Goto 1

Faster

1. Register CPU step handler in Musashi core.
2. Execute as many instructions as possible (up to frame limit)
 - ▶ CPU calls a handler after each instruction (last slide)
 - ▶ End current burst of instructions only if we need to break.
3. Draw screen
4. Goto 1

A quick demo:

- ▶ Insert watch point on the checksum bytes
 - ▶ `watch 0x002001cd 4`
- ▶ From here we can find the checksum code



My Code is Free!

TIME 3:45
0

My ROM header dumper and Sonic 3 RAM gen tooling

<https://github.com/vext01/dgmtools>

My Dgen/SDL Debugger

Is included in Dgen/SDL as of version 1.30

<http://dgen.sourceforge.net/>



SCORE 100
TIME 3:45
0

Reversing Engineering SEGA Megadrive Games

What is Next?



Future Distractions

- ▶ Reverse some more games
- ▶ Implement Z80 watch and break points
- ▶ Fix dgen/SDL bugs
- ▶ Make some games for the Megadrive?
 - ▶ In assembler or C

