Учебник по ромхакингу Sega Mega Drive/Genesis

Дополнительная информация: при написании книги использовалось freeware программное обеспечение и open-source игра. Автор не несет ответственности за действия, которые может совершить читатель, используя полученные знания. Знания - сила, используй её с умом! Книга защищена законом об авторском праве. Изменение книги запрещено. Копирование отдельных частей книги запрещено.

Содержание

Часть 1. Введение

Приветствую тебя, дорогой друг! Если ты любишь it-технологии, приставку Sega Mega Drive/Genesis, достиг совершеннолетия, хочешь научиться чему-то новому, и стать компьютерным волшебником, то...читай дальше.))

Книга рассчитана на новичков в ромхакинге и в ней некоторые понятия, в целях более лучшего усвоения материала, будут выглядеть не так, как, в официальной терминологии.

Часть 2. Знакомство с основными понятиями

RAM - энергозависимая(оперативная) память. Да, у сежки есть оперативная память, как и у обычного компьютера.
ROM – энергонезависимая память. Это скопированный(сдампленный) картридж, в виде компьютерного файла, с расширениями .bin/.smd/.gen. Можно сравнить с жестким диском компьютера.
Эмулятор — это программа, которая имитирует “железо” игровой приставки сега (ну и не только её, может и нескольких других), с помощью неё ты можешь поиграть в игры сеги.
Дебаггер (отладчик) — это инструмент, который помогает разработчику игры отыскивать баги (ошибки, недоработки) игры и модифицировать её.
Hex-редактор – это программа для изменения ROM.
Байт- это единица информации, записываемое в специальное, необычное с виду, шестнадцатеричное двузначное число (для перевода обычного числа в «байтовый» вид в книге будет использоваться калькулятор программиста).
Ромхакер - это человек, настолько любящий игру, что хочет стать её разработчиком))
Опкод, он же operation code - это операция процессора, состоящая из одного или нескольких байтов.

Часть 3. Инструментарий

Нам понадобится эмулятор со встроенным дебаггером, например, Gens r57shell mod. Хотя можно использовать BizHawk, ведь он посовременнее будет. Также понадобится hex-редактор, например, HxD. Ну, и, конечно же, калькулятор. Ссылка на инструменты будет в последней части книги.

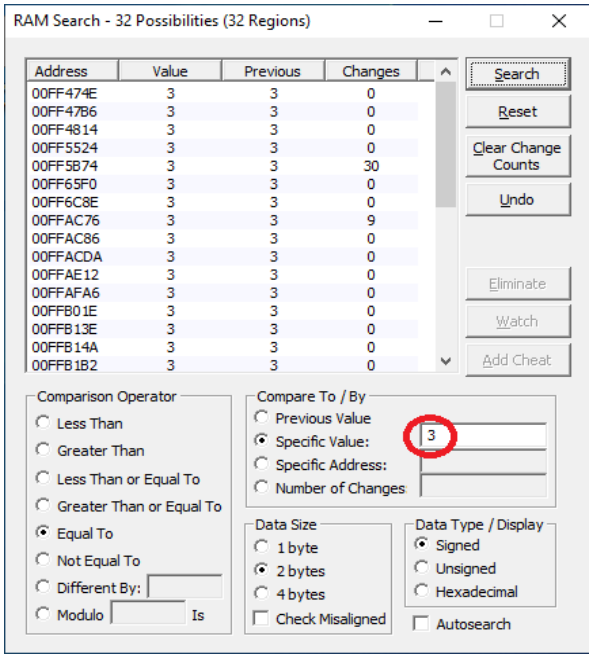
Часть 4. Твоя первая модификация

Процессор сеги, во время игры, выполняет различные операции. Например, он выполняет операцию записи числового значения очков здоровья персонажа в игре Cave Story из одного из своих регистров в определённую ячейку оперативной(RAM) памяти, имеющую определённый адрес. Ссылка на эту игру будет в конце книги. Мы будем делать модификацию игры с увеличенным запасом очков здоровья игрока.

Давайте найдем адрес ячейки в RAM-памяти, в которой хранится здоровье персонажа.

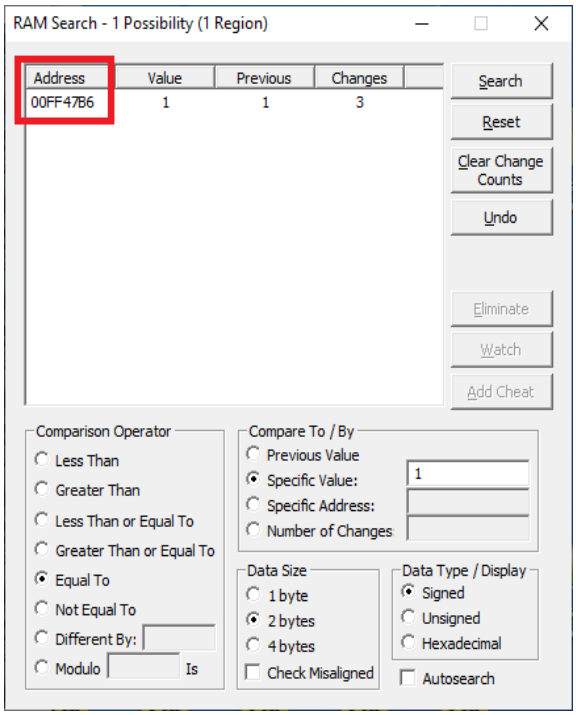
Запустим игру в эмуляторе Gens r57shell mod, пройдём до первых противников, это летучие мыши. Далее, в эмуляторе, нажимаем Tools, RAM search. Это означает, что мы запускаем функцию поиска очков здоровья в оперативке.

Пишем цифру 3(ведь у персонажа 3 очка, это видно в левой верхней части экрана), проставляем галочки как на картинке, и нажимаем Search:



Кстати, «Equal To» означает, что мы ищем значение, равное 3, «Specific Value» – то, что ищем конкретное известно нам значение, «2 bytes» -то, что ищем значение, выраженное в двух байтах.

Далее, нам нужно потратить здоровье в игре, потом снова перейти в окошко Ram Search, ввести цифру 2, и снова нажать Search. Далее, проделать эту процедуру ещё раз, пока не увидим это:



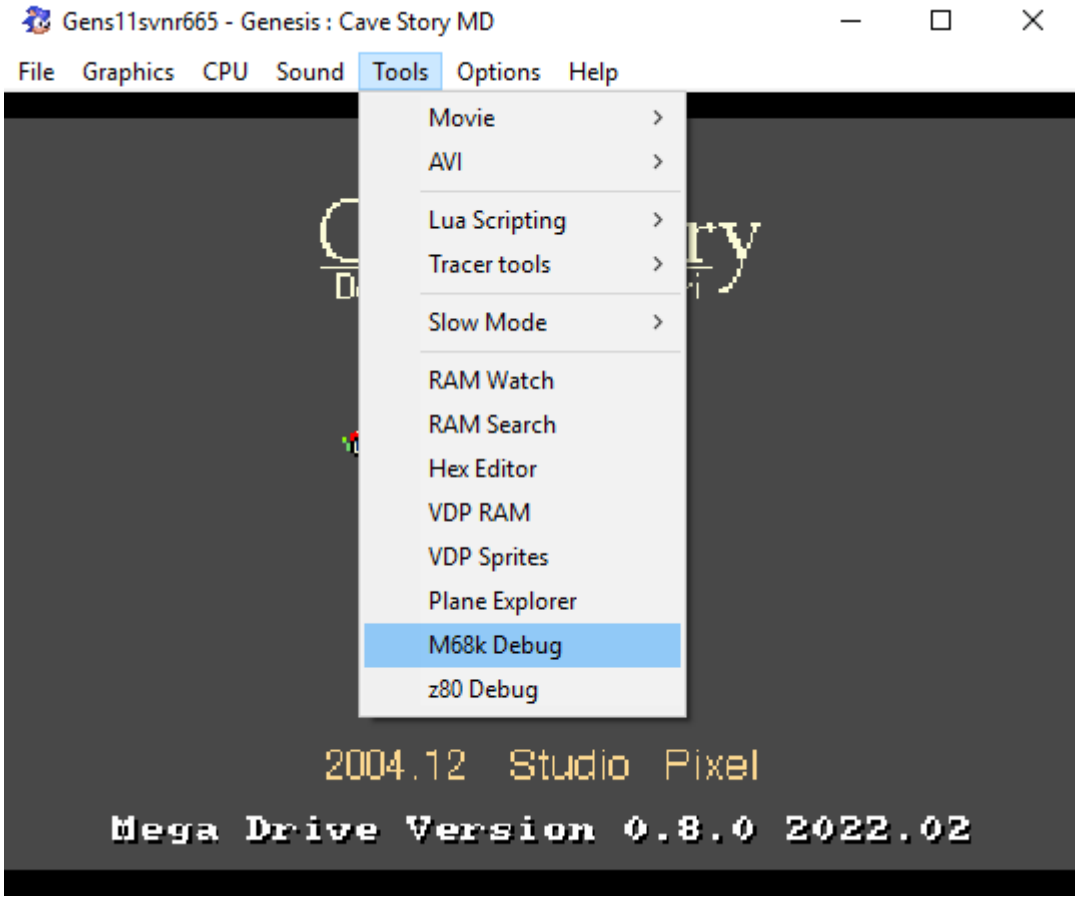
Вот мы и нашли адрес ячейки памяти FF47B6, в котором хранится значение здоровья!

Теперь, чтобы изменить числовое значение очков здоровья, которое устанавливается при старте игры, мы должны найти эту самую процессорную инструкцию записи числового значения количества очков здоровья по адресу FF47B6 данной ячейки RAM памяти.

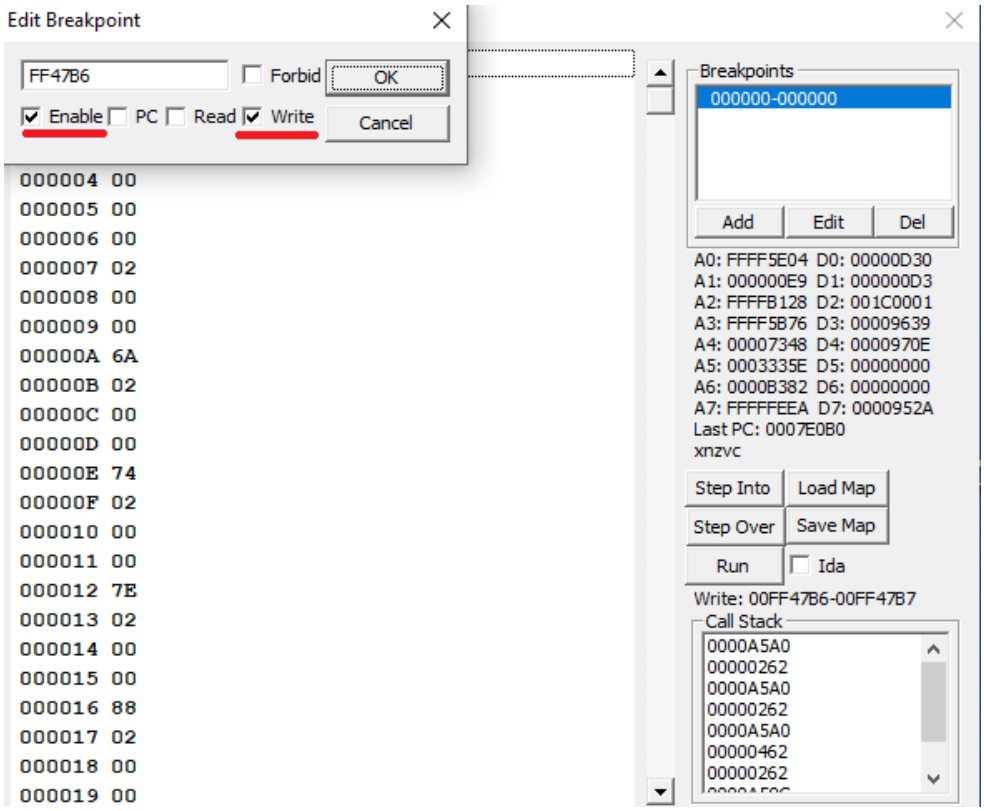
Так вот, для нахождения этой операции, нам нужно дать команду компьютеру. Команда по смыслу будет такая: «компьютер, останови игру в то время, когда следующей для выполнения будет операция записи стартовых очков здоровья по адресу FF47B6, покажи мне её, чтобы я её изменил». Для этого мы должны выйти на титульный экран игры, и установить так называемую **точку остановки**. Почему на титульный

экран? Потому что, находясь на титульном экране игры, процессор ещё не установил стартовые очки здоровья персонажа.

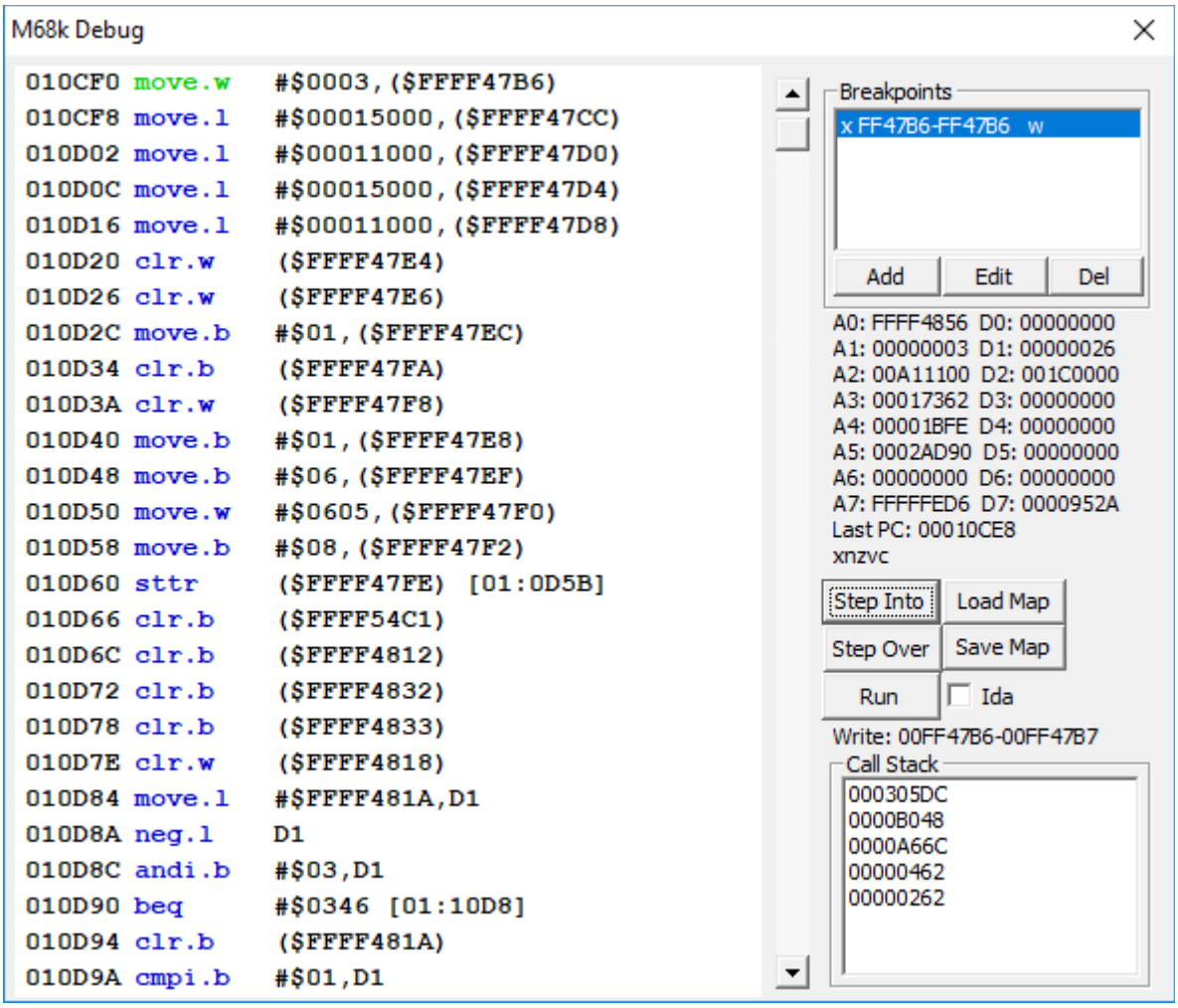
Итак, выйдем на титульный экран игры, и откроем **дебаггер**, встроенный в эмулятор. Нажимаем Tools, M68k Debug:



Далее нужно нажать на кнопку Add, чтобы добавить точку остановки на запись в ячейку памяти по её адресу. Это означает то, что мы сейчас скажем компьютеру адрес здоровья, чтобы он, при установке очков здоровья, остановился и показал нам операцию процессора. Нам нужно вписать адрес ячейки памяти где очки здоровья, который мы нашли ранее, поставить галочку Enable, что означает «включить», поставить галочку «Write», что означает, что мы просим компьютер остановить игру, когда в этот адрес запишется стартовое значение здоровья, и нажать ОК:



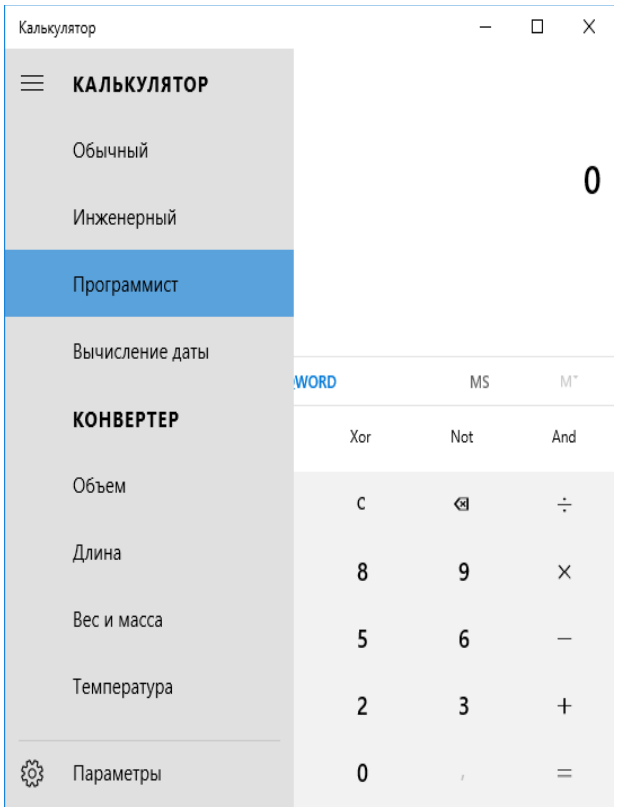
Нажимаем старт. Игра остановится, и в окошке M68k Debug мы увидим операцию процессора, которая отвечает за перемещение данных в ячейку памяти здоровья:

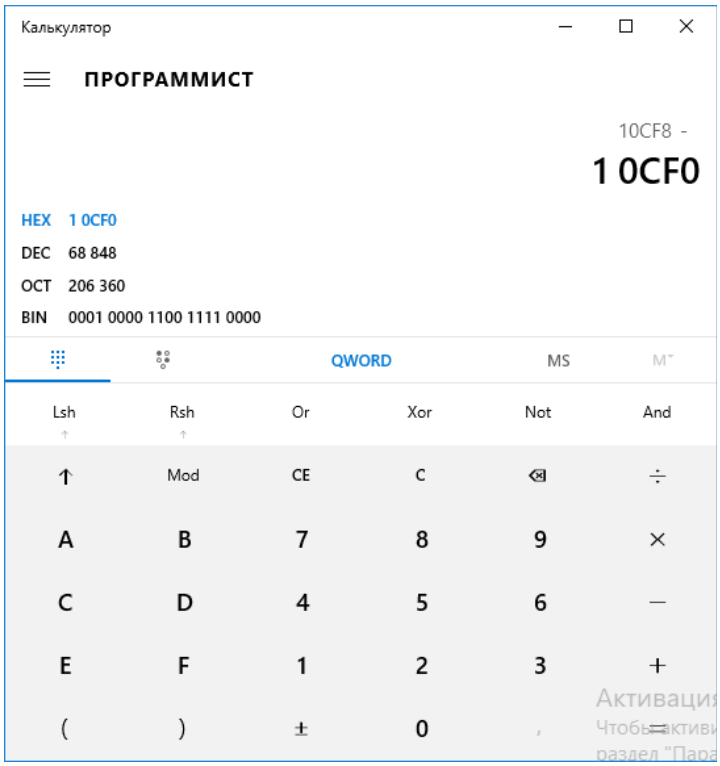


Разберем эту инструкцию, которая выделена зеленым цветом на скриншоте. Для этого запомните, что вы находитесь на 6-ой странице, и перейдите на 10-ую страницу. В 7-ой части изучите 5-ый пример, (а точнее 5.1) и снова вернитесь сюда.

Итак, если вы поняли 5-ый пример седьмой главы, то теперь можно догадаться, что нам нужно изменить тройку на желаемое нами число.

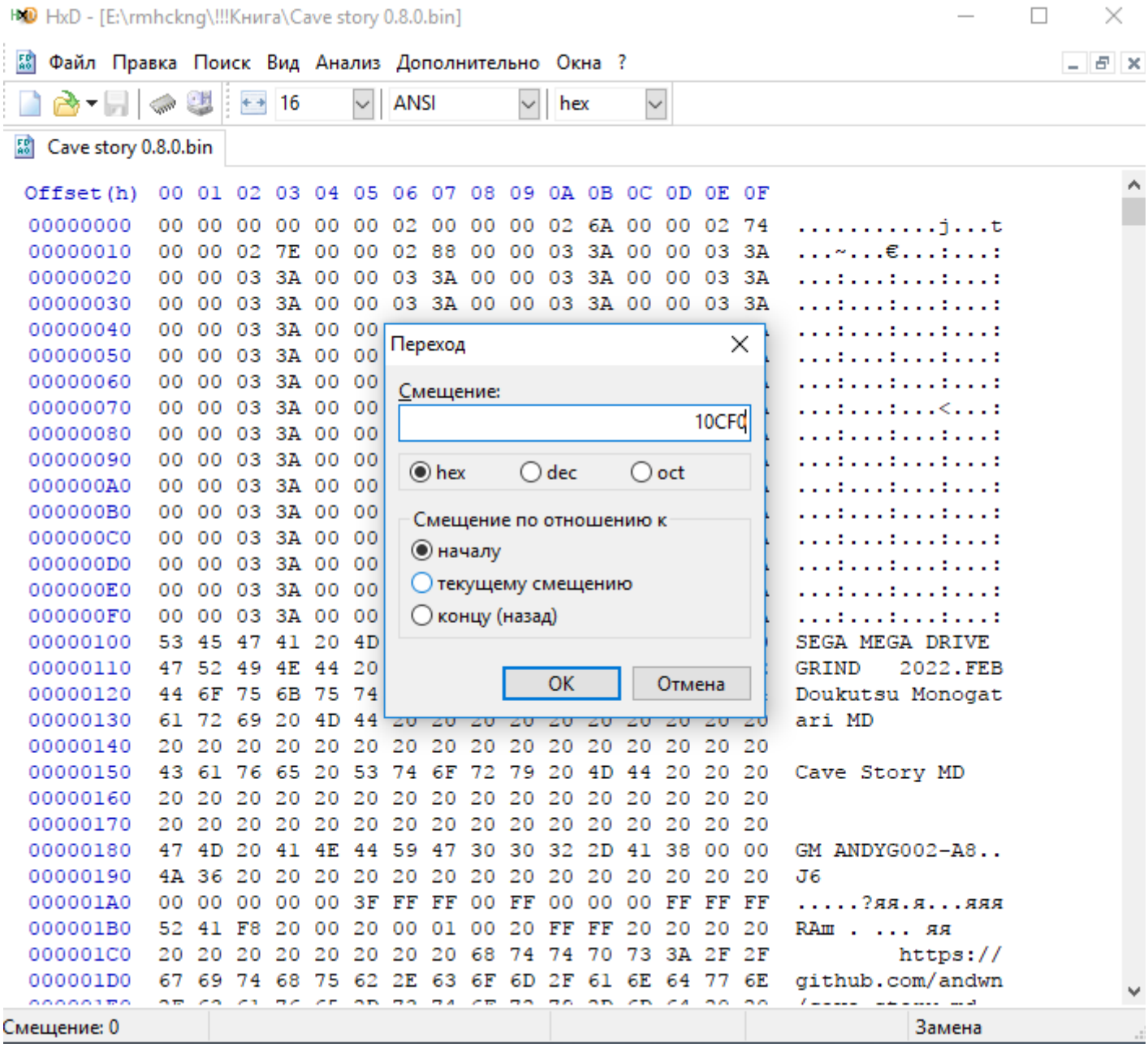
Теперь, чтобы изменить инструкцию процессора в дампе игры, нам нужно для начала узнать её длину. Она может состоять из нескольких байтов. Если не знать её длину, то будет непонятно, какой опкод у инструкции. Для того, чтобы вычислить длину инструкции, мы, из картинки выше, возьмем адрес инструкции, что идет ниже нашей инструкции записи здоровья, а именно “10CF8”, и вычтем из неё адрес инструкции, подсвеченной зеленым цветом, а именно 10CF0. Всё это проделаем в калькуляторе, в котором заранее выбран режим программиста, и нажата кнопка HEX:



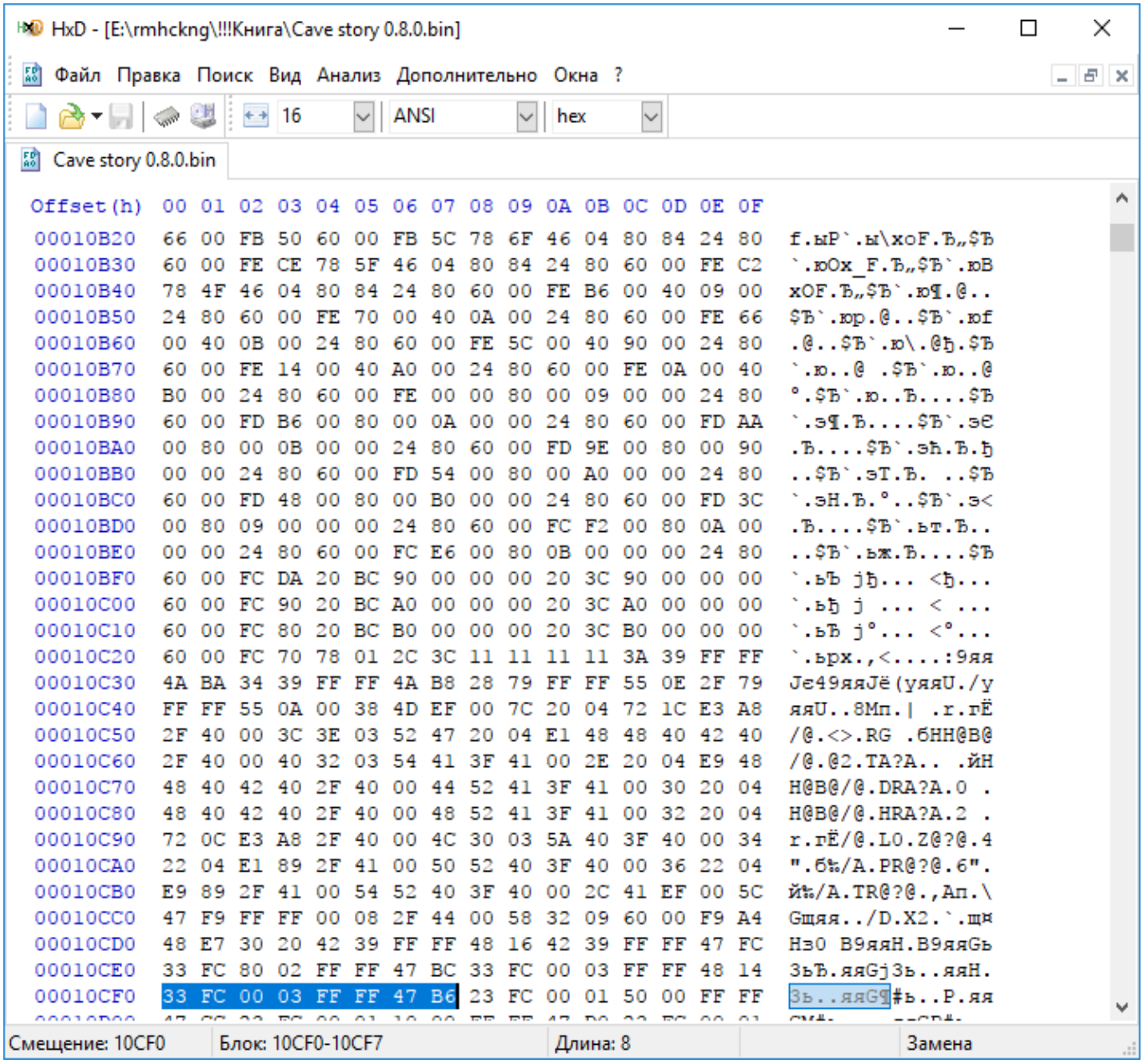


Результат оказался равен восьми. Итак, мы вычислили длину инструкции, она оказалась равна восьми байтам.

Теперь пойдем её изменять. Для этого нужно запустить hex-редактор HxD, открыть файл игры, нажать «Поиск», «Перейти», вписать адрес 10CF0, который мы нашли ранее, и нажать ОК


HxD - [E:\rmhckng\!!!Книга\Cave story 0.8.0.bin]
Файл Правка Поиск Вид Анализ Дополнительно Окна ?
16
ANSI
hex
Cave story 0.8.0.bin
Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
00000000 00 00 00 00 00 00 02 00 00 00 02 6A 00 00 02 74 ....j...t
00000010 00 00 02 7E 00 00 02 88 00 00 03 3A 00 00 03 3A ...~...€...:...:
00000020 00 00 03 3A 00 00 03 3A 00 00 03 3A 00 00 03 3A ...:...:...:...:
00000030 00 00 03 3A 00 00 03 3A 00 00 03 3A 00 00 03 3A ...:...:...:...:
00000040 00 00 03 3A 00 00
00000050 00 00 03 3A 00 00
00000060 00 00 03 3A 00 00
00000070 00 00 03 3A 00 00 ...:...:...<...:
00000080 00 00 03 3A 00 00
00000090 00 00 03 3A 00 00
000000A0 00 00 03 3A 00 00
000000B0 00 00 03 3A 00 00
000000C0 00 00 03 3A 00 00
000000D0 00 00 03 3A 00 00
000000E0 00 00 03 3A 00 00
000000F0 00 00 03 3A 00 00
00000100 53 45 47 41 20 4D SEGA MEGA DRIVE
00000110 47 52 49 4E 44 20 GRIND 2022.FEB
00000120 44 6F 75 6B 75 74 Doukutsu Monogat
00000130 61 72 69 20 4D 44 ari MD
00000140 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
00000150 43 61 76 65 20 53 74 6F 72 79 20 4D 44 20 20 20 Cave Story MD
00000160 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
00000170 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
00000180 47 4D 20 41 4E 44 59 47 30 30 32 2D 41 38 00 00 GM ANDYG002-A8..
00000190 4A 36 20 20 20 20 20 20 20 20 20 20 20 20 20 20 J6
000001A0 00 00 00 00 00 3F FF FF 00 FF 00 00 00 FF FF FF .....?яя.я...яяя
000001B0 52 41 F8 20 00 20 00 01 00 20 FF FF 20 20 20 20 RAш . ... яя
000001C0 20 20 20 20 20 20 20 20 68 74 74 70 73 3A 2F 2F https://
000001D0 67 69 74 68 75 62 2E 63 6F 6D 2F 61 6E 64 77 6E github.com/andwn
Переход
Смещение:
10CF0
hex
dec
oct
Смещение по отношению к
началу
текущему смещению
концу (назад)
OK
Отмена
Смещение: 0
Замена

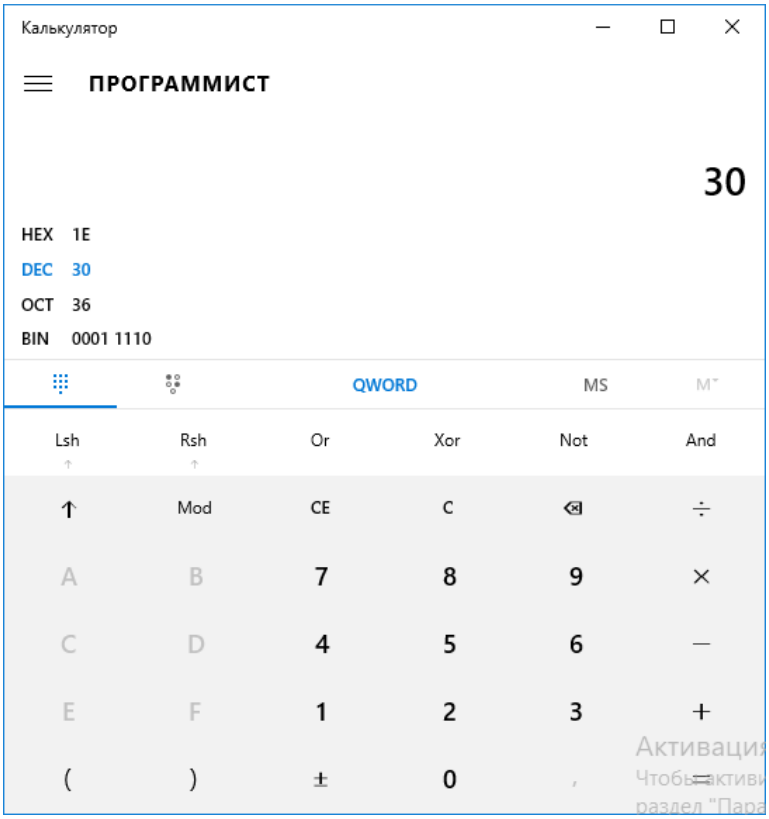
Курсор мыши окажется на адресе 10CF0, и мы увидим те самые *8* байт, которые будем изменять:

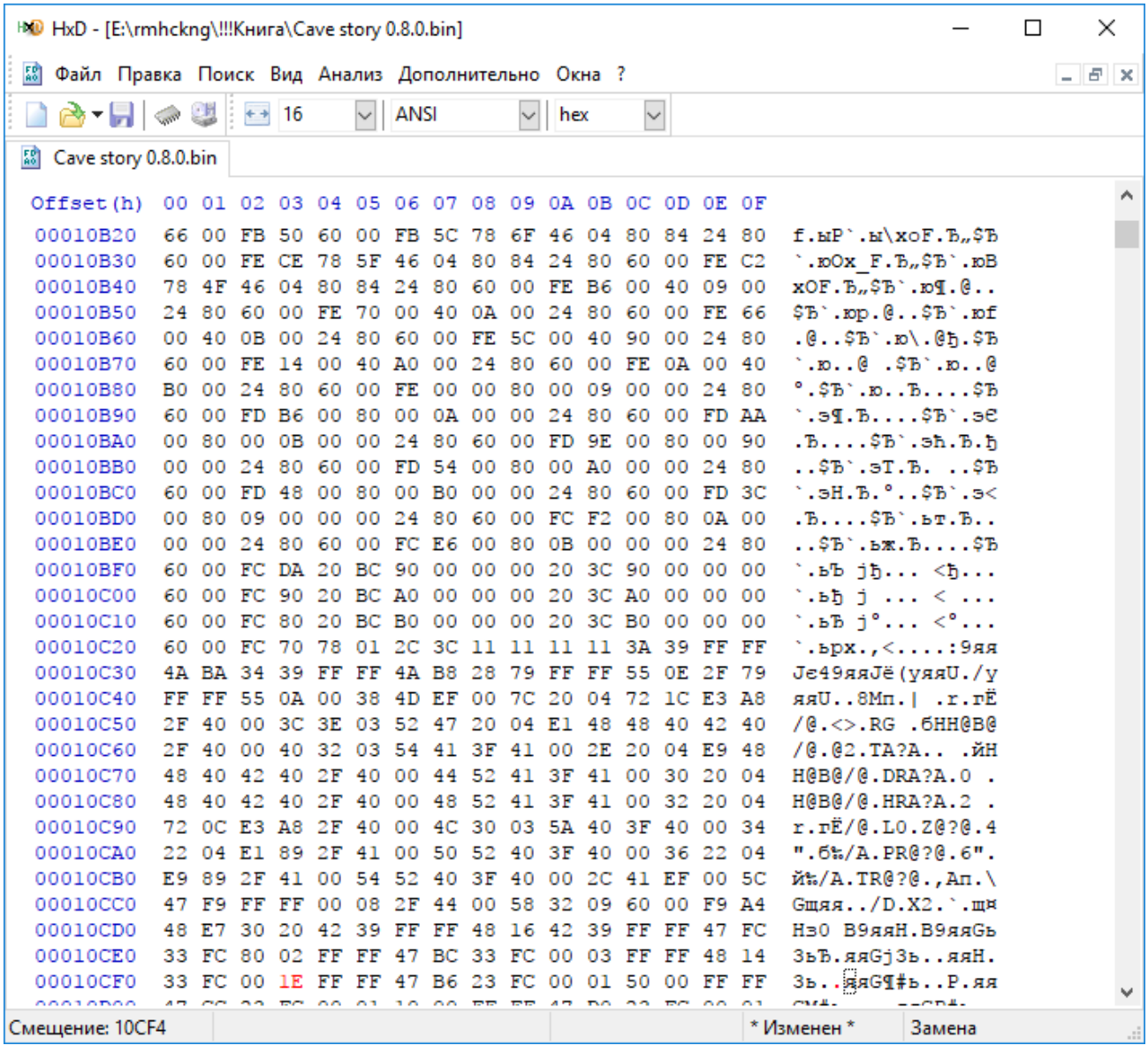


Итак, мы видим 33 FC 00 03 FF FF 47 B6. Разберем эту команду процессора.

«33 FC» означает запись числа word, «00 03» означает три очка здоровья, «FF FF 47 B6» – это адрес очков здоровья в RAM памяти.

Давайте изменим эту тройку на тридцать. Так будет намноооого легче играть)). Но не забывайте, что здесь другая система исчисления, так что 30 это будет не 30, а «1E», мы же не забыли про специальный калькулятор?





Далее нужно сохранить этот файл с помощью «файл»-«сохранить как», дать название, например «cave story мод здоровья.gen».

Иногда, получается так, что мод(модификация) не запускается сразу, т.к. вместо игры появляется красный экран. Чтобы вылечить это, нам нужна программа Fixheadr. В таком случае, тянем на значок этой программы наш файл мода игры, и получаем рабочий мод. Ссылка на программу будет в конце книги.

Также хочется отметить, что если увеличить здоровье персонажа в игре Cave Story, то при восстановлении здоровья на «контрольном пункте» оно снова станет неизменным. Это связано с тем, что игра, дополнительно использует ещё одно значение – значение **максимального** здоровья персонажа, которого он достиг в игре. На контрольном пункте, при восстановлении здоровья, игра считывает текущее максимальное значение очков здоровья, и копирует это значение в ячейку здоровья. Так что, придётся ещё и вычислить, тем же самым путем, которым мы уже вычисляли, адрес максимального стартового значения здоровья, и сделать его также равным 30, при старте игры. Подсказка: ram-адрес максимального стартового значения здоровья FF4814.

Не всегда всё так легко получается при модификации игр, т.к. одна и та же инструкция может использоваться игрой для разных целей. Например, бывает так: функция уменьшения здоровья персонажа и здоровья босса использует одну и ту же инструкцию. И, если ты, желая сделать бессмертие в игре, сотрешь инструкцию, то персонаж станет бессмертным, но и босс тоже, и, при этом, игру будет невозможно пройти. В таком случае, нужно уметь пользоваться трейсером, понимать команды процессора более глубоко, а возможно, придется даже писать свой код вместо кода игры(вероятно, в следующих версиях книги, будет присутствовать информация о том, как это делать).

Часть 5. Сборник трюков

Трюки не универсальны, в одних играх они срабатывают, а в других нет. Если тебе будет этого мало, придумывай свои трюки.

1)Как найти стартовое количество жизней?

Найди в памяти кол-во жизней персонажа. Перезагрузи игру. На титульном экране останови эмулятор кнопкой pause на клавиатуре. Установи точку остановки на жизни персонажа. Снова нажми pause, начни игру. Если повезёт (ну, вообще, это дело не везения, а знаний и опыта.)))), ты сразу увидишь инструкцию, в которой будет присутствовать цифра, отвечающая за стартовое количество жизней.

2)Как уменьшить здоровье босса, чтобы он был слабее?

Сделай сохранение игры кнопкой F5 до встречи с боссом. Запусти RAM Search. Дойди до босса, ударь его, далее снова зайди в RAM Search. Проставь галочки таким образом: Less Than, Previous Value, 2 bytes (а, может, и не обязательно на это, но лучше начинать пробовать с этого), и нажми Search. Бей босса ещё раз, и снова жми Search. Таким образом отсеивай значения, пока не увидишь адрес его здоровья. Загрузи сохранение, поставь на паузу, установи точку остановки на адрес ячейки памяти с числовым значением здоровья босса, отожми паузу, дойди до босса и дебаггер покажет тебе инструкцию, в которой ты увидишь стартовые очки здоровья босса.

3)Здоровье босса и здоровье главного персонажа используют одну инструкцию, а мне хочется сделать бессмертие. Что делать?

а) Можно попробовать найти в игре какую-нибудь функцию, которая не очень нужна, например, подсчитывание очков. Сделай точку остановки на подсчитывание очков. Сотри несколько инструкций для своего кода, начиная с места остановки, и впиши свой код, в котором ты будешь записывать полное здоровье в ячейку памяти главного персонажа.

б) Более тяжелый вариант, при котором ты сохранишь функцию подсчитывания очков. Сделай точку остановки на подсчитывание очков. Вместо этой инструкции напиши код прыжка JMP (если ты уже научился это делать) на свой код, в котором пополняй здоровье.

4)Хочу сделать высокий прыжок. Как?

Этот вариант не всегда срабатывает. Однако можешь попробовать его. Уменьши скорость игры с помощью кнопки минус на клавиатуре до 10%. Прыгни персонажем, нажми pause на клавиатуре, и пока он взлетает вверх, ищи несколько раз значения в памяти, которые увеличиваются, т.е. нужно поставить галочки Greater Than, Previous Value при поиске. Отсеивай значения, пока не найдешь адрес высоты прыжка персонажа. Открой Tools, Hex Editor эмулятора, перейди по найденному адресу, и попробуй изменять значение высоты прыжка во время игры в большую сторону. Тут, может случится так, что персонаж улетит в небо)). Ускорь эмулятор кнопкой плюс на клавиатуре и верни персонажа на землю, далее поставь точку остановки на адрес высоты прыжка. Прыгни персонажем, и, попробуй изучить инструкции, которые находятся выше точки остановки, где, вероятно, ты увидишь сравнивание текущей высоты с максимальной высотой прыжка (смотри операцию сравнивания в соответствующей части книги). Если тебе повезет (хотя, не в везении дело)))), то ты найдёшь адрес максимальной высоты прыжка и, теперь, осталось только выйти на титульный экран, поставить точку остановки на адрес максимальной высоты прыжка, и начать игру.

5)Как мне найти код игры, отвечающий за паузу, чтобы я смог поставить вместо паузы что-нибудь своё?
а) поищи значение 0 в памяти игры, до нажатия кнопки старт. Далее нужно искать значение 1, после нажатия кнопки старт. Работает не всегда, в зависимости от того, как написана игра.
б) найди адрес нажатия клавиш с помощью следующей полезной информации. В сеге, обычно так: клавиша A — hex-значение 40, B — 10, C — 20, Start - 80. Сделай точку остановки на чтение этого адреса памяти, т. е. поставь галочку read вместо галочки write. Изучай близлежащие к точке остановки инструкции, в которых нужно найти место, в котором идёт сравнивание со значением 80.
6)Хочу сделать прыжок JMP на свой код, но где найти место под свой код?
В сеге всё очень просто на этот счет. Добавляй свой код в hex-editor-е в конец ROM-а(или в "пустом" месте ROM-а, обычно оно забито значениями FF). А прыжок осуществляется так: 4E B9 xx xx xx, где xx xx xx — адрес(смещение) твоего кода в ROM-е.

Часть 6. Интересные примеры модификаций игр

~~Здесь будет реклама.~~

Есть такой сайт, называется romhacking.net, на котором можно найти огромное количество интересных модификаций, сделанных людьми из разных стран. Вот ссылка на него: https://www.romhacking.net/?page=hacks&genre=&platform=11&game=&category=1&perpage=20&order=&dir=&title=&author=&hacksearch=Go
Модификации на этом сайте выложены в виде патчей, для которых, соответственно, нужен патчер (обычно это программа lunar ips) и ром игры.

Часть 7. Примеры работы процессора. Опкоды

Для начала нужно понимать, что в памяти лежат данные, выраженные в:
а) byte -число от 0 до 255. Например, 10 патронов в игре будут выглядеть как «0A». Почему не «10»? Потому, что здесь другая система исчисления- шестнадцатеричная. О ней ты можешь почитать здесь: https://ru.wikipedia.org/wiki/Шестнадцатеричная_система_счисления
б) word -это два байта, т. е. Число от 0 до 65536
в) longword -это 4 байта. Число от 0 до...(посчитай сам сколько это будет, в калькуляторе, настроенным на режим программиста, подсказка - FF FF FF FF)

Сейчас будут рассмотрены примеры работы процессора, который использует byte, word:
1)вычитание единицы (byte) по адресу FFE10A.
Смысловая, ассемблерная инструкция, которая показывается в **дебаггере**, при срабатывании точки остановки будет такая:
SUBQ.B #1,($E10A), где **SUB** означает вычитание, **B** означает byte, **#1** означает единицу, **$E10A**-адрес памяти, опкод всей инструкции вместе: 53 38 E1 0A.
Напомню, что опкод — это шестнадцатеричный вид инструкции процессора, который виден в hex-редакторе.
2)прибавление единицы (byte)по адресу FFE10A:
ассемблерная инструкция **ADDQ.B #1,($E10A)**, где **ADD**-это прибавление,
опкод: 52 38 E1 0A
После прибавления или уменьшения, значение адреса FFE10A изменится так: если было 05 яблок в памяти, то стало 06
3) прибавление единицы (word)по адресу FFF024
ADD.W #1,$00FFF024, где W означает word
опкод: 52 79 00 FF F0 24
После прибавления, значения адресов FFF024-FFF025 изменятся так: если было 00 05 патронов, то стало 00 06
Это важно понимать, т. к. 06 будет находится в адресе FFF025, а не по адресу FFF024, ведь используется word, а не byte.

Это самые простые примеры, в которых процессор увеличивает или уменьшает значения в памяти сразу, напрямую, без использования регистров. А что такое регистр, спросите вы. Отвечу. Процессор использует свою встроенную, сверхбыструю память, и регистры - это её ячейки. Поэтому сейчас рассмотрим более сложный, четвертый пример, когда процессор вычитает или прибавляет (патроны/яблоки/очки здоровья и т.д.) с использованием регистров. Итак, пример, состоящий из трёх последовательных инструкций (кстати, этот пример относится к уменьшению здоровья в игре Cave Story):

```
move.w ($FF47B6),d0 (опкод 30 39 FF FF 47 B6)
здесь процессор сначала переместил значение из адреса FF47B6 в свой регистр d0
sub.w    d2,d0 (опкод 90 42)
далее процессор провел вычитание значения регистра d2 из регистра d0. Кстати, в это время, в
```

регистре d2 хранилось кол-во урона, т. е. получилось вычитание урона из здоровья. Если стереть эту инструкцию в игре, с помощью опкода «4E 71», то получим бессмертие в игре.
move.w d0,$FF47B6 (опкод 33 C0 FF FF 47 B6)
здесь процессор переместил значение регистра d0 в адрес FF47B6 обратно
Думаю, уже понятно, что ADD -это прибавление, SUB-вычитание, а MOVE-перемещение
Список инструкций можно почитать в предпоследней части книги.
5.1) Перемещение значения 3(word) по адресу FF47B6:
move.w #$0003, ($FFFF47B6)
опкод 33 FC 00 03 FF FF 47 B6
5.2) Перемещение значения 999(word) по адресу FFEE30:
move.w #$03E7, $FFEE30
опкод 33 FC 03 E7 00 FF EE 30
6) Перемещение значения содержимого адреса памяти FFEE32 в ячейку памяти с адресом FFEE30:
move.w $FFEE32, $FFEE30
опкод 33 fc 00 FF EE 32 00 FF EE 30
7) Пропуск определенного количества инструкций в байтах. Ещё он называется безусловным переходом
bra X, где X-количество байтов для пропуска
опкод 60 XX
Т.е., если, например, поставить опкод 6004, то процессор, выполняя эту инструкцию, пропустит 4 байта инструкций, следующих за этой инструкцией 6004
8) Прыжок (безусловный прыжок в подпрограмму). Это означает, что процессор «перепрыгнет» на другой адрес в ROM-е, и будет выполнять инструкции, уже оттуда
JMP($000F9530)
опкод 4E F9 00 0F 95 30
Чтобы вернуться из подпрограммы обратно, в то место, откуда прыгали, существует следующая инструкция:
RTS
опкод 4E 75
9)Сравнивание CMP (вероятно образовано от английского слова compare)
cmp.b #15,d5
опкод BA 3C 00 0F
Здесь сравнивается число 15 с содержимым регистра d5. Обычно после сравнивания стоит инструкция BEQ «переход если равно», или BNE «переход, если не равно». Ведь мы же не просто так сравниваем. В обычном, высокоуровневом программировании: «если a=b, то сделать то-то, а если a не равно b, то сделать то-то. Полные их названия «Branch If Equivalent» и «Branch If not Equivalent», а их опкоды 67 и 66 соответственно. Например:
cmp.b #15,d5
beq #04
Т.е., если содержимое регистра d5 равно 15, то процессор делает переход на 4 байта.
10)Также часто встречающаяся инструкция это TST. Это проверка на ноль. Например, игра проверяет, умер ли персонаж, а он считается умершим, если его здоровье равно нулю))
Tst.w $FFEA30
Здесь процессор считывает значение адреса FFEA30, и проверяет, равен он нулю, или нет.
После этого, также, обычно стоит инструкция «перехода если равно», или «перехода, если не равно».
11)Отсутствие операции “No operation”. С помощью этой инструкции можно стирать другие инструкции.
Nop
Опкод 4E 71
Например, ты хочешь стереть какую-то инструкцию, состоящую из 6 байт. В таком случае тебе нужно вписать три раза 4E 71 вместо неё. (так, порою и делается бессмертие в играх)

Часть 8. Список инструкций

ADD	Сложение
SUB	Вычитание
NEG	Инверсия
ADDX	Сложение с расширением
SUBX	Вычитание с расширением
NEGX	Инверсия с расширением
ABCD	Десятичное сложение с расширением
SBCD	Десятичное вычитание с расширением
NBCD	Десятичная инверсия с расширением

MULS Умножение со знаком
MULU Умножение без знака
DIVS Деление со знаком
DIVU Деление без знака
EXT Расширение знака
CLR Очистка операнда
CMP Сравнение
TST Проверка
TAS Проверка и установка
MOVE Пересылка операнда
MOVEP Пересылка данных с периферийного устройства
MOVEM Пересылка группы регистров
MOVEQ Быстрая пересылка
SWAP Обмен половин регистра
EXG Обмен регистров
LEA Загрузка действительного адреса
PEA Сохранение в стеке действительного адреса
LINK Связь стека
UNLK Отцепка стека
AND Логическое И
OR Логическое ИЛИ
EOR Логическое Исключающее ИЛИ
NOT Логическое НЕ (инверсия)
ASL Арифметический сдвиг влево
ASR Арифметический сдвиг вправо
LSL Логический сдвиг влево
LSR Логический сдвиг вправо
ROL Циклический сдвиг влево без переноса
ROR Циклический сдвиг вправо без переноса
ROXL Циклический сдвиг влево с переносом
ROXR Циклический сдвиг вправо с переносом
BCLR Проверка и очистка бита
BSET Проверка и установка бита
BCHG Проверка и изменение бита
BTST Проверка бита
JMP Прыжок
BRA *Безусловный переход*
JSR Переход к подпрограмме
BSR Переход к подпрограмме
RTS Возврат из подпрограммы
RTR Возврат из подпрограммы
BCC Переход, если нет переноса
BCS Переход, если есть перенос
BEQ Переход, если равно
BF Переход, если не истинно

BGE Переход, если больше или равно
BGT Переход, если больше
BHI Переход, если выше
BLE Переход, если меньше или равно
BLS Переход, если не выше
BLT Переход, если меньше
BMI Переход, если минус
BNE Переход, если не равно
BPL Переход, если плюс
BT Переход, если истинно
BVC Переход, если нет переполнения
BVS Переход, если есть переполнение
DB усл. Вычитание единицы из регистра данных и переход, если выполняется условие (всего 16 условий, см. *BCC*:BVC)
S усл. Установить при выполнении условия (всего 16 условий, см. *BCC*:BVC)
TRAP Прерывание
TRAPV Прерывание при переполнении
CHK Проверить регистр на границах диапазона
ANDI Логическое И с *регистром состояния*
ORI Логическое ИЛИ с *регистром состояния*
EORI Исключающее ИЛИ с *регистром состояния*
RESET Сброс внешних устройств
STOP Остановка работы
RTE Возврат из исключительной ситуации

Часть 9. Скачать инструменты, используемые в книге

Перейдите по этой ссылке: **https://disk.yandex.ru/d/H8Fo5dVFqbUcqw** и вы увидите архив, где среди программ, дополнительно, лежит pdf-документация на процессор Sega Megadrive/Genesis Motorola 68000.
И, напоследок, ещё полезные ссылки:
https://mrjester.hapisan.com/04_MC68/Index.html
https://gamehacking.org/

На этом, дорогие друзья, эта версия книги окончена. Будет ли другая версия, с более сложными примерами работы процессора, разбором модификации, работой с трейсером, увеличенным количеством трюков, на данный момент неизвестно.

Vers. Моя 1.0