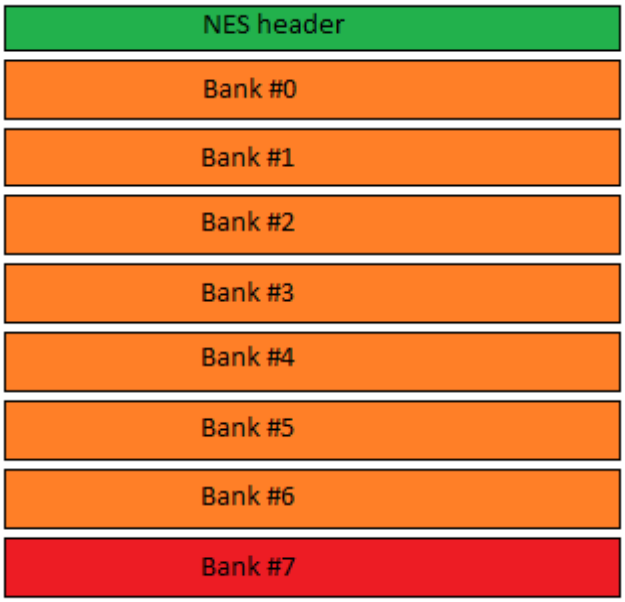
Prince of Persia [NES] Technical documentation

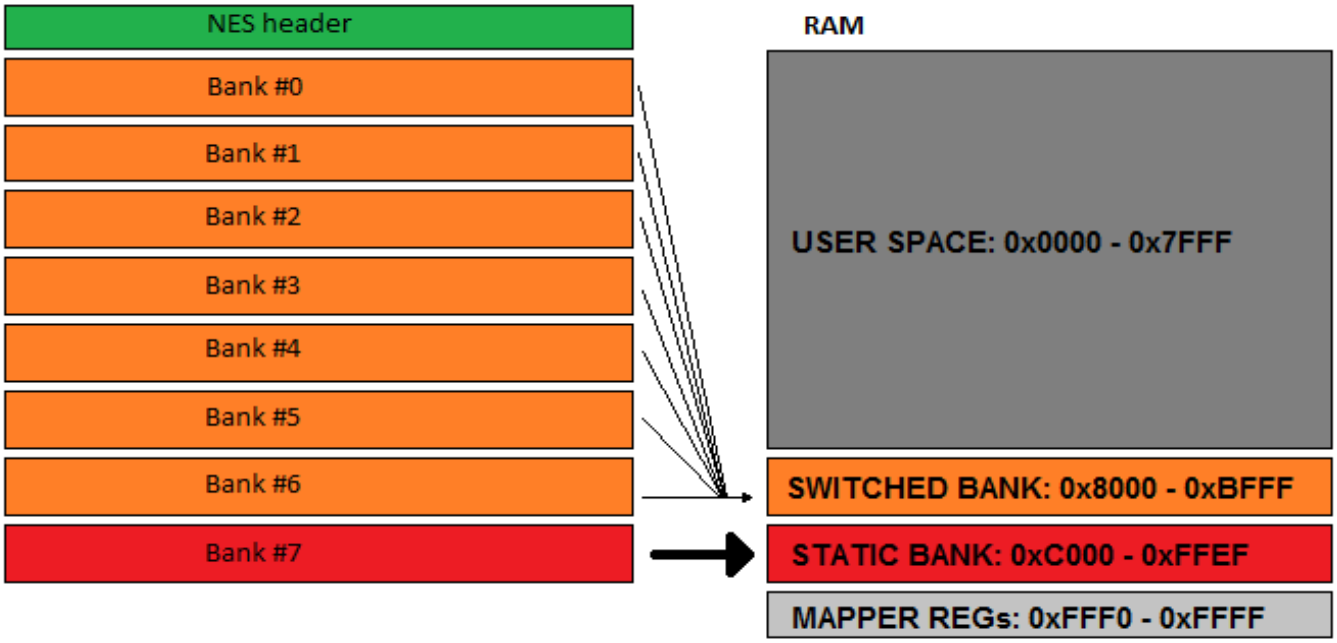
1. Формат и содержимое ROM-файла.



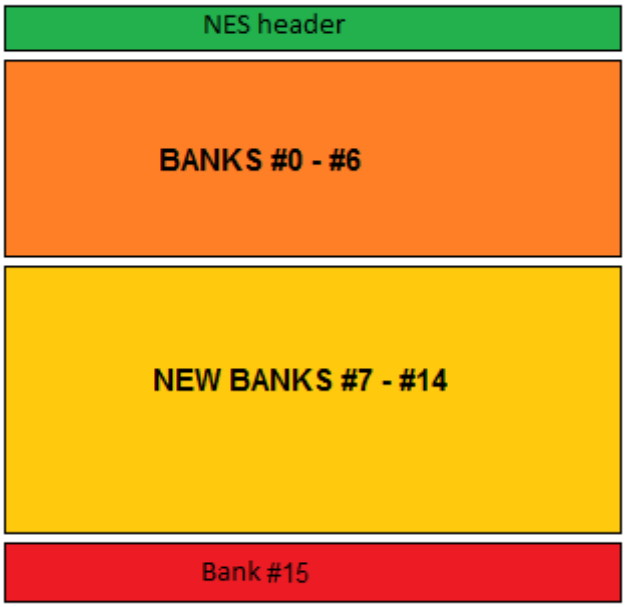
Сам файл NES содержит заголовок и данные ROM сразу за ним. Данные ROM – это непосредственно те данные, которые сохранены в ПЗУ картриджа. Заголовок имеет лишь NES файл и предназначен он для правильной интерпретации данных самого ROM.

Prince of Persia имеет Mapper #2 (UNROM) *Mapper – алгоритм переключения кусков (bank – банков) памяти в игре.* Содержит 8 банков памяти, последний из которых не переключаемый и хранится всегда в конце области памяти, адресуемой процессором.

В процессе игры банки памяти динамически отображаются в предпоследние 16 кБ памяти, адресуемой процессором. Таким образом, в оперативной памяти в последних 32 кБ находится один из первых 7 банков и 1 последний — 8, который всегда в конце области памяти.



Последний банк грузится в конец и его начало всегда является процедурой запуска (main()) при запуске игры. Исходя из всего этого, ROM-файл можно расширить. Таким образом без каких-либо изменений кода можно получить еще 8x16кБ ROM-памяти для игры, где можно хранить и данные, и код.



В Prince of Persia первые два банка содержат в себе тайлы (изображения объектов). Тайлы не несут в себе никакой цветовой информации. Каждый пиксель тайла состоит из 2 бит. Эти биты кодируют номер цвета в палитре, которая динамически подгружается в PPU (видеопамять) вместе с тайлами. Оставшиеся 6 банков содержат в себе данные и код (причем нет разделения между данными и кодом — они вполне могут идти рядом).

Адресное пространство, доступное процессору выглядит следующим образом:

Адрес	Размер	Назначение
$0000-$07FF	2k	RAM
$0800-$1FFF	6k	RAM Mirror (x3)
$2000-$2007	$8 = 8 байт	Registers Video
$2008-$3FFF	$1FF8 = 8184 байта	Registers Video Mirror (x1023)
$4000-$4017	$20 = 32 байта	Registers Audio & DMA & I/O
$4018-$4FFF	$0FE8 = 4072 байта	Not used
$5000-$5FFF	4k	Not used
$6000-$7FFF	8k	Not used
$8000-$BFFF	16k	PRG-ROM (dyn)
$C000-$FFFF	16k	PRG-ROM (0)

Непосредственно пользовательская область — адреса $0000 - $07FF, из которых адреса $0100-$1FF не используются.

Переключение банков памяти в Mapper #2 осуществляется следующим образом: по любому адресу в диапазоне $8000 – $FFFF записывается число (от 0 до N-1, где N – число банков), после чего, в диапазон $8000 – $BFFF отображается содержимое соответствующего банка (поэтому, банки лучше всего переключать в коде последнего банка, т. к. при переключении банка, по адресу, следующему за текущей исполняемой инструкцией окажутся произвольные данные).

Пример кода переключения банков:

```
$F2D3:84 41     STY $0041 = #$00
$F2D5:A8        TAY
$F2D6:8D D1 06  STA $06D1 = #$02
$F2D9:99 F0 FF  STA $FFF0,Y @ $FFF0 = #$00
$F2DC:A4 41     LDY $0041 = #$00
$F2DE:60        RTS
```

В регистре A передается номер включаемого банка.

2. Регистры и инструкции процессора.

Регистры и адресация памяти:

Процессор имеет 3 регистра общего назначения (A – аккумулятор, X и Y).

- Регистр A: Используется для общих целей, в частности для переноса данных между ячейками памяти.
- Регистры X, Y: Также используются для общих целей и, кроме того, могут являться индексами при адресации массивов.

Адресация массивов бывает двух типов: абсолютная и относительная.

- Абсолютная адресуется от указанной ячейки;
- Относительная адресуется от указателя, лежащего в указанной и соседней ячейках (адрес состоит из двух байт).

Например:

```
$0000     15 00 00 01 02 03 04 05  0A 0B 0C 0D 15 18 19 20
$0010     20 30 40 51 62 73 84 95  AA BB CC DD 15 18 19 20


          LDX 5                ; X = 5
          LDA $0000, X         ; загружаем в A число по смещению
                               ; $0000 + 5 = $0005: #03

          LDA ($00), X         ; загружаем в A число по смещению
                               ; ($0000: 15 00 → указатель $0015)
                               ; $0015 + 5 = $001A: #CC
```

Процедуры:

Вызов процедуры выполняется инструкцией JSR, возврат — RTS. Соглашения по передачи параметров (типа stdcall или cdecl) тут нет. Передавать можно как угодно. Обычно через переменные или регистры.

Инструкции условного перехода:
Наиболее часто используемые: BEQ, BNE, BCC, BCS.
Работают следующим образом:

- BEQ – переход, если FlagZ = 1;
- BNE – переход, если FlagZ = 0;
- BCS – переход, если FlagC = 1;
- BCC – переход, если FlagC = 0.

Флаги выставляются так:

- LDA, LDX, LDY: FlagZ = 1, если операнд равен 0, иначе FlagZ = 0;
- ADC, SBC: Здесь FlagC считается 9 битом при выполнении сложения или вычитания. Т.е. при сложении туда помещается 1 при переполнении 8 бита, при вычитании он наоборот оттуда заимствуется.
- CMP, CPX, CPY: FlagZ = 1, если операнды равны, иначе FlagZ = 0, FlagC = 1, если операнд больше, либо равен, иначе FlagC = 0.

Например:

```
          ; if ( $0001 >= $0002 ) goto exit;

          LDA $0001
          CMP $0002
          BCS exit
          ; something code

     exit:
          RTS
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
          ; if ( $0001 < $0002 ) goto exit;
          LDA $0001
          CMP $0002
          BCS do_something
          JMP exit

     do_something:
          ; something code

     exit:
          RTS
```

Полный набор инструкций 6502 opcodes (на русском).

3. Псевдокод.

В псевдокоде используется упрощенный синтаксис C:

```
void sub_main()
{
     if ( !$0001 ) goto label_EXIT;

     $0400 = #15;
     X = $0020;
     Y = $0030;
     #0015[X] = $0072[Y];

label_EXIT:
     return;
}
```

В данном случае:

```
     #0015[X]       // абсолютная адресация (см. выше)
     $0072[Y]       // относительная. В ячейках $0072:$0073
                    // лежит указатель.
                    // $0073 – старший байт адреса
                    // $0072 – младший байт адреса
```

Пример перевода ASM в псевдокод:

```
     $F2D3:84 41     STY $0041 = #$00
     $F2D5:A8        TAY
     $F2D6:8D D1 06  STA $06D1 = #$06
     $F2D9:99 F0 FF  STA $FFF0,Y @ $FFF0 = #$00
     $F2DC:A4 41     LDY $0041 = #$00
     $F2DE:60        RTS
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

void sub_SwitchBank()
{
     // bank counter in A
     $0041 = Y;              // save Y register to var0041
     Y = A;                  // TAY: transfer A to Y
     $06D1 = A;              // save current bank counter
                             // to var0641
     #FFF0[Y] = A;           // write bank counter
                             // to mapper register
     Y = $0041;              // restore Y from var0041
     return;
}
```

Еще пример:

```
          $F302:20 D3 F2  JSR $F2D3
          $F305:8E 06 20  STX $2006 = #$41
          $F308:A9 00     LDA #$00
          $F30A:8D 06 20  STA $2006 = #$41
          $F30D:A2 10     LDX #$10
          $F30F:A0 00     LDY #$00
          $F311:B1 17     LDA ($17),Y @ $020E = #$03
          $F313:8D 07 20  STA $2007 = #$00
          $F316:C8        INY
          $F317:D0 F8     BNE $F311
          $F319:E6 18     INC $0018 = #$02
          $F31B:CA        DEX
          $F31C:D0 F1     BNE $F30F
          $F31E:4C 10 D0  JMP $D010
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
          $D010:A9 05     LDA #$05
          $D012:4C D3 F2  JMP $F2D3
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

Перевод (следующая страница):

```
void sub_F302()
{
     sub_F2D3();    // switch bank. Bank counter in A register

                    // $2006 – PPU Address register
                    // $2007 – PPU data write
                    // В $2006 записываем адрес в видеопамяти
                    // (старший байт, затем младший),
                    // после чего в регистр 2007 записываем данные
                    // в PPU. После каждой записи, адрес PPU
                    // автоматически увеличивается на 1.
     $2006 = X;     // старший байт адреса
     $2006 = #00    // младший байт адреса
     X = #10;

label_F30F:
     Y = #00;

label_F311:
                    // в ячейках $0017:$0018 лежит указатель
                    // на данные, которые будем записывать в PPU
     $2007 = $0017[Y];
     Y++;
     if ( Y != #00 ) goto label_F311;
     $0018++;
     X--;
     if ( X != #00 ) goto label_F30F;
     return sub_D010();
}

void sub_D010()
{
     A = #05;            // bank counter = #05
     return sub_F2D3()   // switch bank
}
```

Можно и далее облагородить код:

```
void WriteDataIntoPPU(char Bank, char PPULine)
{
     switch_bank(Bank);       // switch bank.
     PPUADDRESS = PPULine;    // старший байт адреса
     PPUADDRESS = #00;        // младший байт адреса

     for(X = #10; X > 0; X--)
     {
          for(Y = #00; Y <= #FF; Y++)
          {
               PPUDATA = $Tiles[Y];
          }
          $Tiles += #100;     // переходим на следующую строку
     }
     return switch_bank5();
}

void switch_bank5()
{
     return switch_bank(#05);
}
```

Таким образом видно, что это процедура вывода тайлов на экран в количестве 10 строк по адресу PPULine, причем данные (расположенные в массиве $Tiles) расположены в банке Bank (переданному в качестве параметра).

Регистры управления PPU
Регистры управления звуковым устройством
Регистры джойстика

4. Конвертирование адресов между RAM и смещениями в ROM-файле.

Банки располагаются последовательно в файле (сразу после заголовка, который имеет размер 16 байт). Т.е. Bank #00: 0x00010 – 0x0400F, Bank #01: 0x04010 – 0x0800F и т. д. Таким образом, если в коде встречается конструкция:

```
LDY       $06D1          ; например #02
LDX       #00            ; X = #00
LDA       $EBA7, X       ; A = #EBA7[X]
```

То это однозначно обращение к ROM-памяти. В данном случае, алгоритм конвертирования действителен только для Prince of Persia, т. к. код сохраняет номер текущего банка в переменную $06D1.

```
ROMFileOffset = $06D1 * 0x4000 + 0x10 + (RAMAddress - 0x8000);
                                  // (0x10 – размер заголовка)

ROMFileOffset_EBA7 = #02 * 0x4000 + 0x10 + (0xEBA7 - 0x8000) = 0xEBB7;
```

Т.е. данные, к которым обращается код из фрагмента, лежат в файле по смещению 0xEBB7. Конвертирование адреса памяти в адрес в ROM-файле неоднозначно (т. к. нужно знать номер текущего банка памяти, что без анализа кода неизвестно). В то же время, конвертирование смещения данных в ROM-файле в адрес RAM однозначно:

```
RAMAddress = (ROMFileOffset-0x10) - 0x4000*(ROMFileOffset/0x4000-1) + 0x8000;

RAMAddress_EBB7 = (0xEBB7 - 0x10) - 0x4000*(0xEBB7/0x4000 - 1) + 0x8000 =
      = FileOffset_EBB7 = #02 * 0x4000 + 0x10 + (0xEBB7 - 0x8000) = 0xEBA7;
```

Однако, как правило, код, который загружает данные из банка памяти располагается в том же банке, что и данные. Поэтому, если в файле указатель лежит по адресу (например) 0xE5A1, а указатель, который он использует — E6A1, то в файле он (чаще всего) будет в том же банке, т. е. 0xE6A1 + 0x10 = 0xE6B1.

5. Данные и их формат для Prince of Persia в ROM.

Здесь и далее приводятся адреса и смещения для версии Prince of Persia (U) [!]. Для остальных версий имеют место различия только для смещений в ROM младше 0xBFFF.

```
NES_DEMO_PLAY_PTR = 0x0A71F
; Этот указатель в редакторе извлекается из кода по смещеням:
;    0x0A51B(LO):0xA51C(HI)
; Формат: XX:YY XX:YY … #FF, где XX – время действия, YY – маска
; бит кнопок джойстика (см. Регистры джойстика).

NES_LEVEL_HEALTH_DATA = 0x0B4DA
; Формат: XX, где XX – максимальное число единиц для уровня
; по одному байту на уровень. Т.е. HEALTH[LEVEL] = #0B527[LEVEL]
; Итого (14x1 = 14 байт)

NES_LEVEL_PALETTE_PTR = 0x0B6B4
; Формат: массив указателей по одному указателю (2 байта) на
уровень (итого: 14x2 = 28 байт)
; Указатель, в свою очередь, указывает еще на один указатель,
который указывает на непосредственно данные палитры (каждая по 16
байт)

NES_GUARDS_TYPES = 0x13BEB
; Тип врагов в уровне (стражник или скелет)
; Формат: #00 или #24. #00 – стражник, #24 – скелет. Иные значения
приводят к глюкам в игре. (итого: 14x1 = 14 байт)

NES_ACTIONS_PTRS_LIST = 0x15602
; Формат: указатели на данные, которые определяют текущее
поведение спрайта. Всего указателей (а значит возможных действий
персонажа) – 95. Первый указатель = #0000

NES_LEVEL_LEFT_ROOM = 0x1EB12
; Формат: номера комнат, правые тайлы которых будут рисоваться с
левой границы комнат, которые расположены с левого края уровня
(т. е. не имеют комнат слева). (итого: 14x1 = 14 байт)

NES_LEVEL_TOP_ROOM = 0x1EB20
; Формат: номера комнат, нижние тайлы которых будут рисоваться с
правой границы комнат, которые расположены с верхнего края уровня
(т. е. не имеют комнат сверху). (итого: 14x1 = 14 байт)

NES_LEVEL_PTR_ARR = 0x1EB4A
; Формат: указатели на данные тайлов комнат уровня (итого 14x2 =
28 байт)
; Формат данных комнат: всего комнат в уровне – 24. Если первый
байт в массиве = #FF, то комната пуста и следующий байт – 1-ый
байт последующей комнаты, иначе – следующие 30 байт содержат тайлы
комнаты: 10 столбцов*3 строки.
```

```
NES_LEVEL_HEADERS_PTR_ARR = 0x1EB66
; Формат: указатели на заголовки уровней (итого 14x2 = 28 байт)
; Формат данных заголовков:
;    3 байта – комната (1-24), позиция (0-29),
      направление (0x00 – смотрит влево, 0xFF - смотри вправо)
;    24 байта – расположение стражников по комнатам. По одному
байту на комнату. Байт: [0x00-0x1D]+0x20*R (R - случайное число),
если Байт & 0x1E == 0x1E - стражник в комнате отсутствует, иначе:
X = (Байт & 0x1E) Mod 10, Y = (Байт & 0x1E) / 10
;    Массив не фиксированного значения:
           B1 B2 B3 B4 B5 C1 C2 C3 C4 C5 … 0xFF
;    Линки между решетками и кнопками.
;    Каждый линк состоит из 5 байт:
;         B1 B2 - комната и позиция кнопки, которая связана с
дверью
;         B3 - действие кнопки (0 – открывает дверь, 1 –
закрывает)
;         B4 B5 - комната и позиция двери, с которой связана
кнопка
;         0xFF - означает конец данных

NES_LEVEL_DIMENSIONS_PTR_ARR = 0x1EB82
; Формат: указатели на данные геометрического расположения комнат
в уровне (итого 14x2 = 28 байт)
; Формат данных: массив имеет фиксированный размер 4x24 = 96 байт
; По 4 байта на комнату. Где:
;         B1 - номер комнаты, находящейся слева (0 – комнаты нет)
;         B2 - номер комнаты, находящейся справа
;         B3 - номер комнаты, находящейся сверху
;         B4 - номер комнаты, находящейся снизу от указанной

NES_LEVEL_TYPE_DATA = 0x1EB05
; Формат: тип уровня (0 – подземный, 1 – дворец) (итого: 14x1 = 14
байт)
```

Прочие данные (массивы для сборки тайлов в объекты, либо звуковые данные) редактором не используются, а потому здесь не рассматриваются.

6. Переменные в RAM

Из тех, что мне известны :)

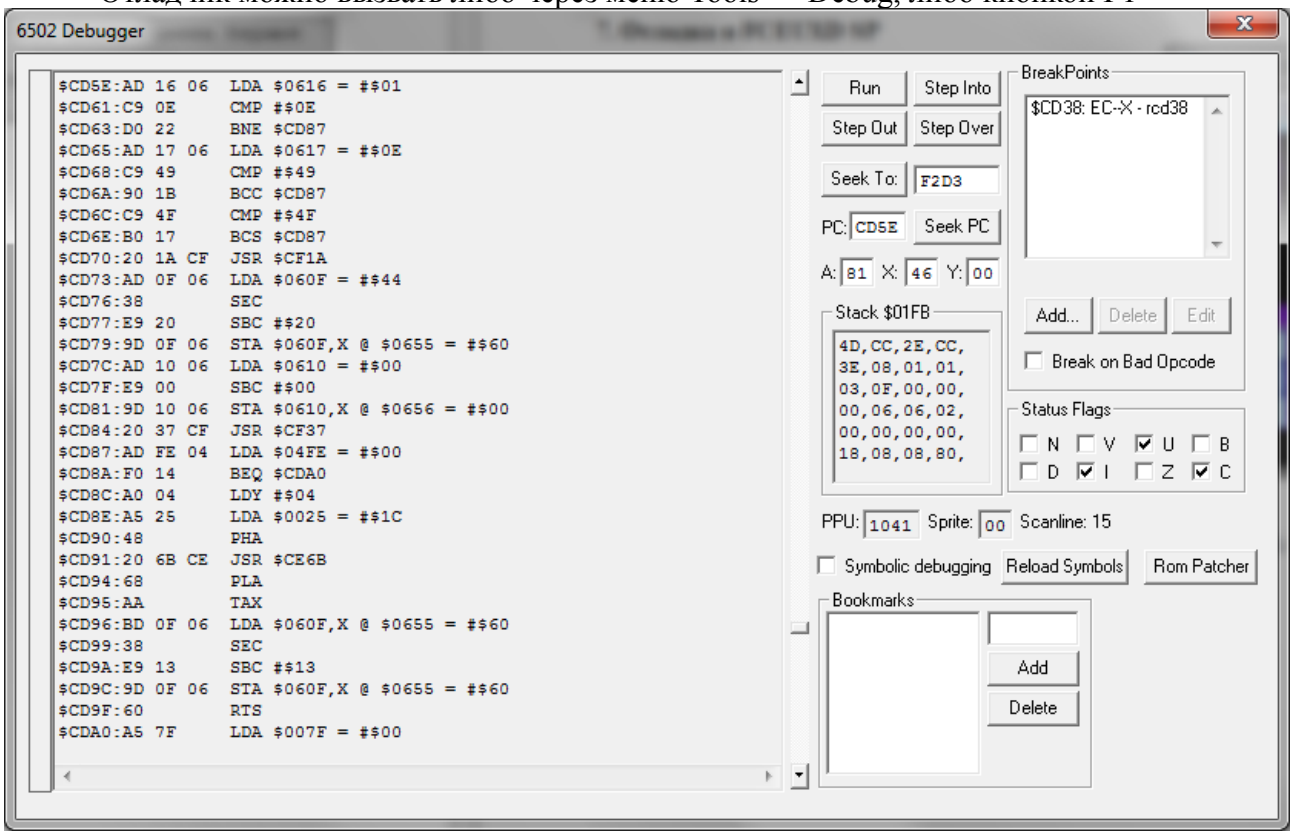
```
$0000-$0001 - различные указатели (используются локально в функциях)
$000C-$000D - указатели на данные комнаты
$000E-$000F - указатели на данные комнаты
$0017, $0018 — различные указатели (также как и $0000:$0001)
$002c - номер текущего обработчика событий (используется где-то внутри main)
$0041 - сюда сохраняются X или Y
$0070 — номер текущего уровня (начиная с 0)
$0051 — номер текущей комнаты (начиная с 0)
$0401 — флаг присутствия в комнате двойника принца
$0402 — флаг, устанавливается в 1, когда открывается выход (используется в 5
уровне, как флаг появления двойника)
$0407-$0408 — указатель на какие-то данные (как указатель нигде не
используется). Используется как флаг того, что принц находится в комнате, в
которой расположен выход
$040a — копия байта с джойстика
$040c — копия номера банка(?)
$04b1 — текущая позиция принца в комнате (0-29). Некоторыми функциями
выставляется в иное значение для обозначения позиции какого-либо объекта в
комнате
$04d6 — сюда записывается текущее значение скролла PPU (но затем обнуляется,
поэтому зачастую равно 0)
$04c5 — таймер (десятки минут в ascii. Т.е. десятки + 0x30)
$04c6 — таймер (единицы минут в ascii. Т.е. Единицы + 0x30)
$04e4 — флаг demo-play. Если установлен, то проигрывается demo-play.
$04e7 - button pushed(?)
$04db-$04dc — текущая позиция (видимо, в пикселях)
$04fc — установлен, если в 4 уровне есть зеркало (в 4 комнате после нажатия на
кнопку открывания выхода)
$04fd — флаг, обозначающий, что двойник принца вышел из зеркала
$04fe — флаг, обозначающий, что через зеркало нельзя пройти
$05e7-$05ff — массив таймеров закрытия решетки (после нажатия на кнопку
открытия решетки). Формат по 3 байта: комната:позиция:таймер. После того, как
таймер обнуляется, в байт с номером комнаты записывается 0xFF, после чего сюда
могут быть записаны новые данные.
0x60E-0x617 - структура текущего состояния принца
0x60E - если бит 7 выставлен, то принц отображается, иначе - нет
$060F — младший байт текущей позиции по X (в пикселях?)
$0610 — старший байт текущей позиции по X
$0611 — младший байт текущей позиции по Y
$0612 — старший байт текущей позиции по Y
$0613-$0614 — указатель на текущее действие (см. NES_ACTIONS_PTRS_LIST)
$0615 — направление (0x00 — смотрит влево, 0xFF – смотрит вправо)
$0616 — номер текущего действия (по сути индекс в массиве указателей
NES_ACTIONS_PTRS_LIST)
$0617 — текущая поза
… далее следуют еще несколько структур того же формата, описывающие состояние
(активных?) объектов в комнате. Сюда же (начиная с адреса $0617) записывается
структура состояния двойника принца того же формата …
$06cf — количество оставшихся единиц здоровья
$06d0 — количество оставшихся единиц здоровья стражника
$06d1 — текущий банк памяти
$06dd — флаг «смерти» принца. Если 1, то начинает мигать Push start
$06e0 — в комнате присутствует стражник (если $0401 == 0) или двойник (если
$0401 == 1)
$06ee — наличие меча у принца. 0 — нет, 1 — есть, 2 — используется в данный
момент (для драки со стражником, например)
$06f2 — общее количество единиц здоровья принца
$06f3 — общее количество здоровья стражника
$06f7 — еще один флаг, обозначающий, что выход открыт
```

$06ff — текущая позиция принца (0-29). В отличие от $04B1 всегда имеет верные данные
$0735 — некая дельта, обозначающая, насколько далеко принц убежал за пределы текущей комнаты (измеряется в количестве комнат)

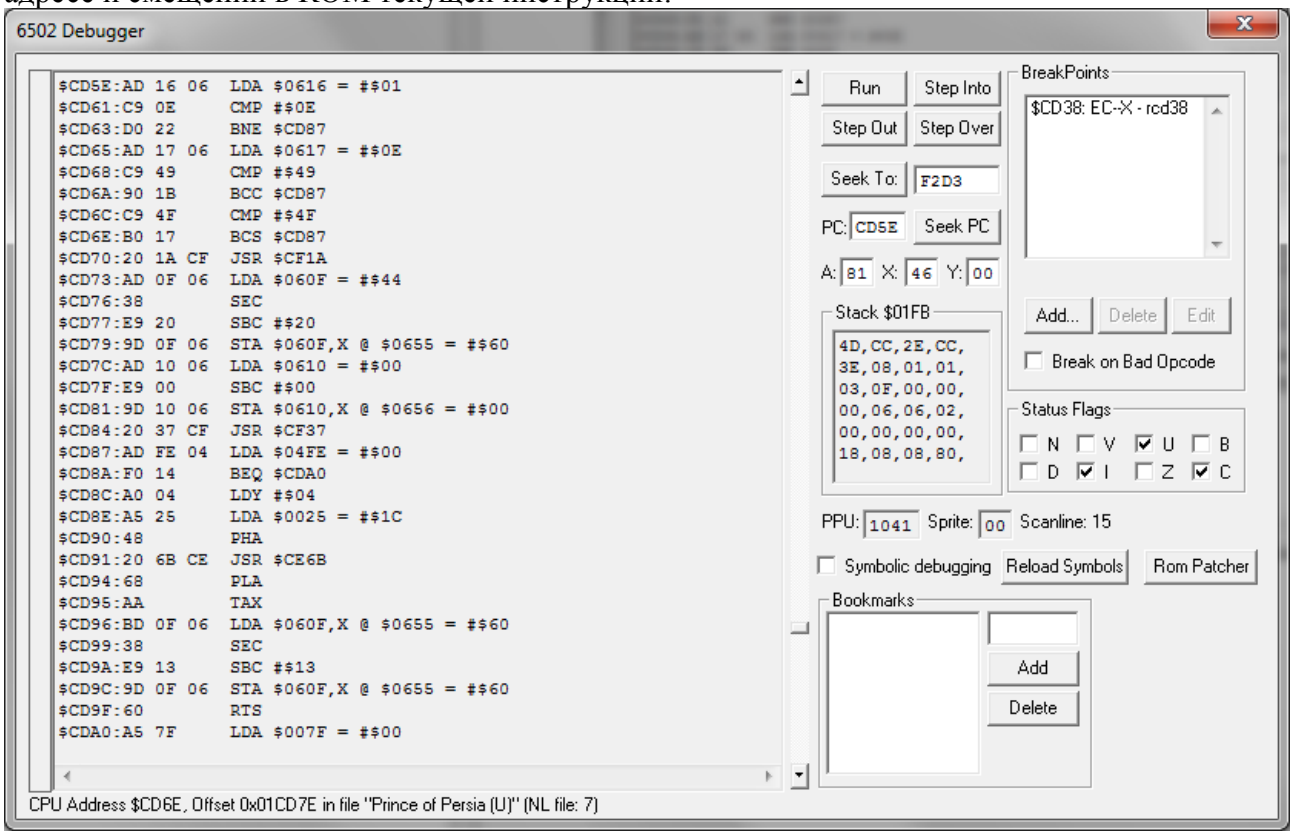
7. Отладка в FCEUXD SP

Само окно эмулятора выглядит так.

Отладчик можно вызвать либо через меню Tools → Debug, либо кнопкой F1



Наведя мышку на столбец слева от кода (выделен серой рамкой) можно получить данные об адресе и смещении в ROM текущей инструкции:



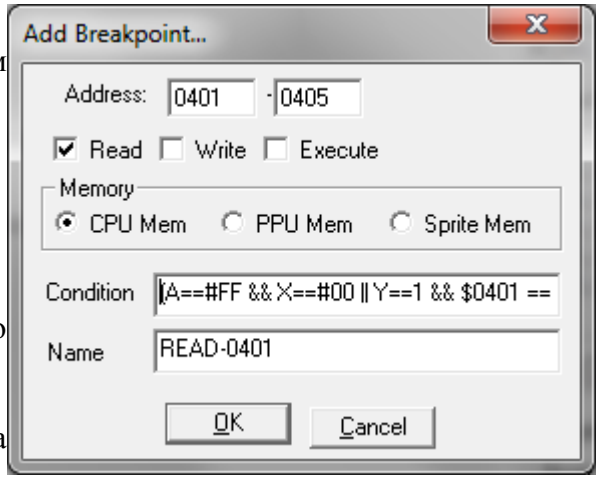
Кнопки в отладчике:

- Run – запустить код на выполнение;
- Step Out – выйти из текущей выполняемой функции и остановится на следующей инструкции;
- Step Into – пошагово выполнять все инструкции со входом во все функции;
- Step Over – пошагово выполнять все инструкции, не входя внутрь функций;
- Seek To – отобразит в отладчике слева код, начинающийся с указанного адреса;
- Seek PC – начать исполнять код с инструкции по указанному адресу;
- Add/Delete/Edit Breakpoints – установить бряк по определенному условию.

Определить адрес инструкции можно по окну Stack. Поскольку при выполнении инструкции JSR в стек помещается адрес возврата (2 байта), то в данном случае эту функцию вызвала инструкция по адресу, предшествующему $CC4D, а вышестоящую, соответственно, $CC2E (первые 4 байта в окне Stack).

Также, тут можно менять значения флагов и регистров.

Установка breakpoints:



Позволяет установить бряки по определенным условиям.
Можно поставить безусловный:

- Address – определенный адрес;
- Ставим Execute;
- Указываем Name.

Выполнение остановится на инструкции по указанному адресу.

Прервать выполнение, если память была считана или записана по указанному адресу:

- Address – указываем адрес или интервал адресов (например с 0401 по 0405);
- Ставим Read или Write (или все вместе)
- Указываем Name.

Выполнение остановится, если было считано или записано значение по указанным адресам.

Прервать выполнение, если выполнено определенное условие:

- Указываем Address, ставим галочки R/W/E, указываем Name
- Пишем условие, по которому необходимо прервать выполнение.

Условие должно быть в скобках и может содержать в себе имена регистров, ячейки памяти и т. д. Синтаксис подобен синтаксису Си.
Например:

```
(A==#FF && X==#00 || Y==1 && $0401 == #01 && P != #FF1A)
```

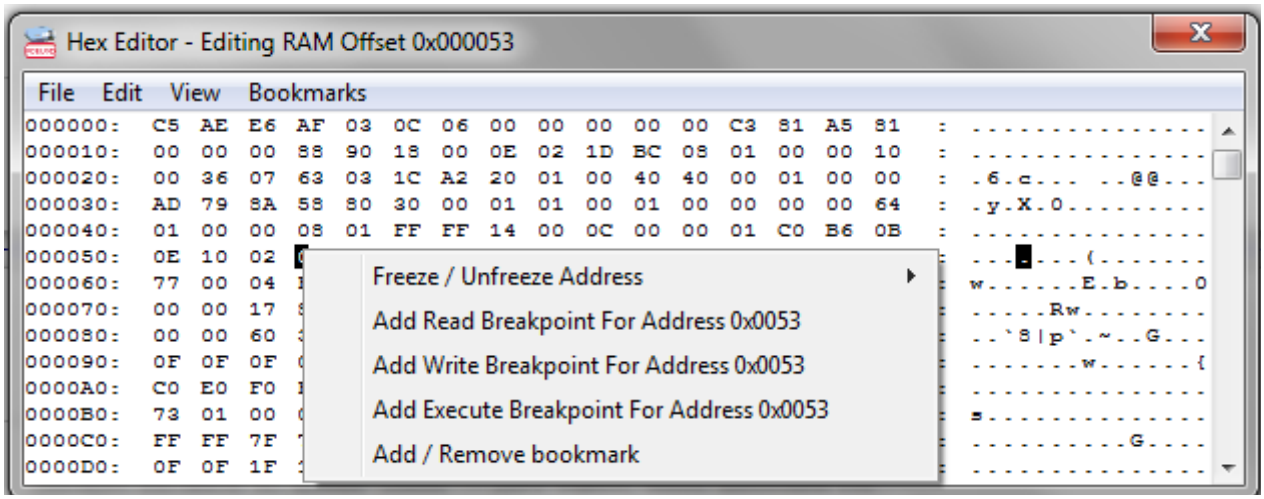
A, X, Y – регистры;
$0401 – ячейка памяти;
P – адрес текущей выполняемой инструкции.

8. Hex editor, он же редактор памяти RAM.

Можно вызвать из меню Tools → Hex editor, либо кнопкой F6.

Отображает текущее состояние памяти в реальном времени. При этом можно записывать любые значения в любые ячейки памяти. Например, для переключение банка памяти достаточно записать по любому адресу с $8000 по $FFFF число от 0 до 7 и в память с $8000 по $BFFF отобразится соответствующий банк.

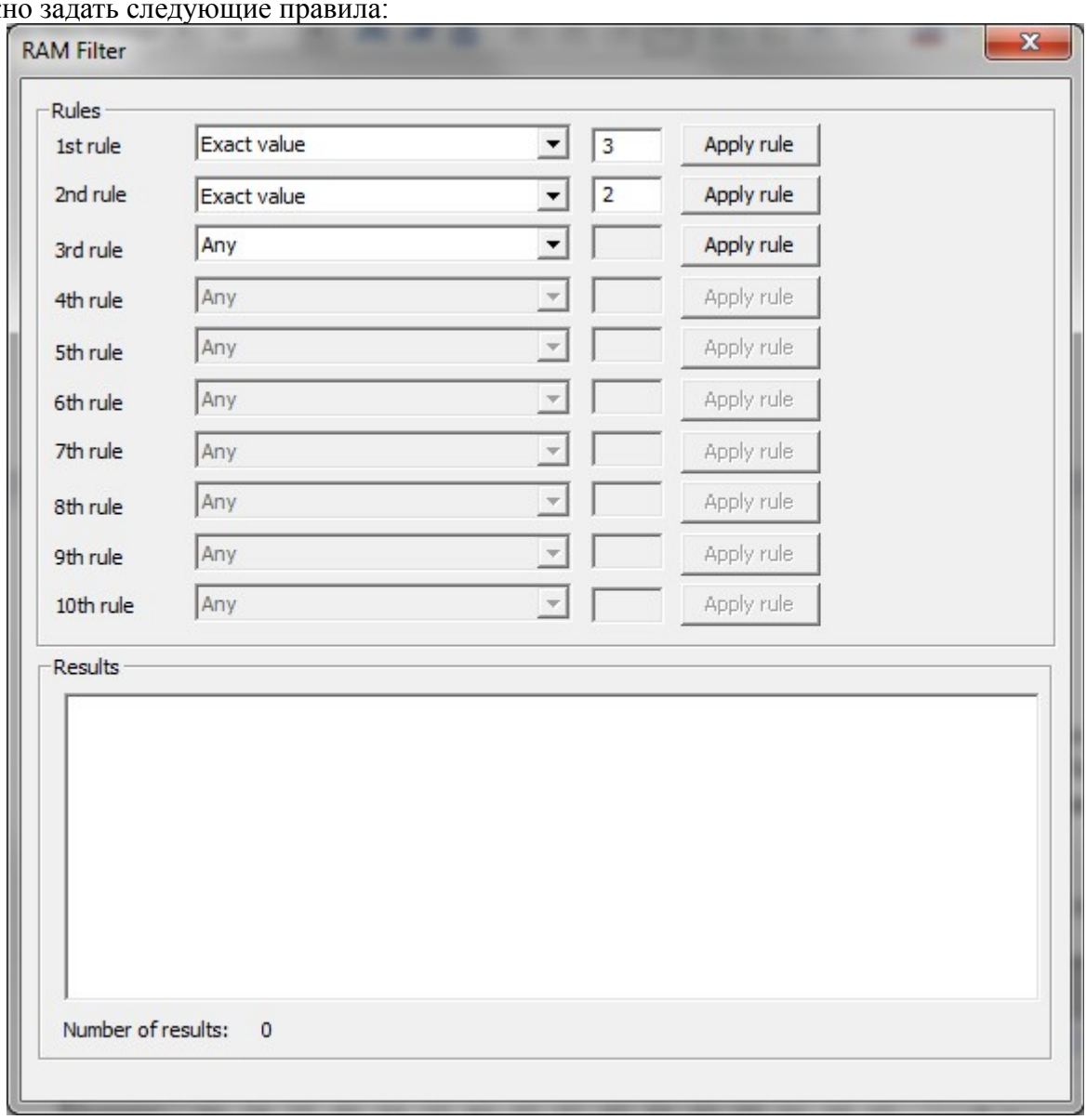
Можно запретить изменять значение определенной ячейки (или группе ячеек) памяти через контекстное меню → Freeze/Unfreeze.



9. RAM filter

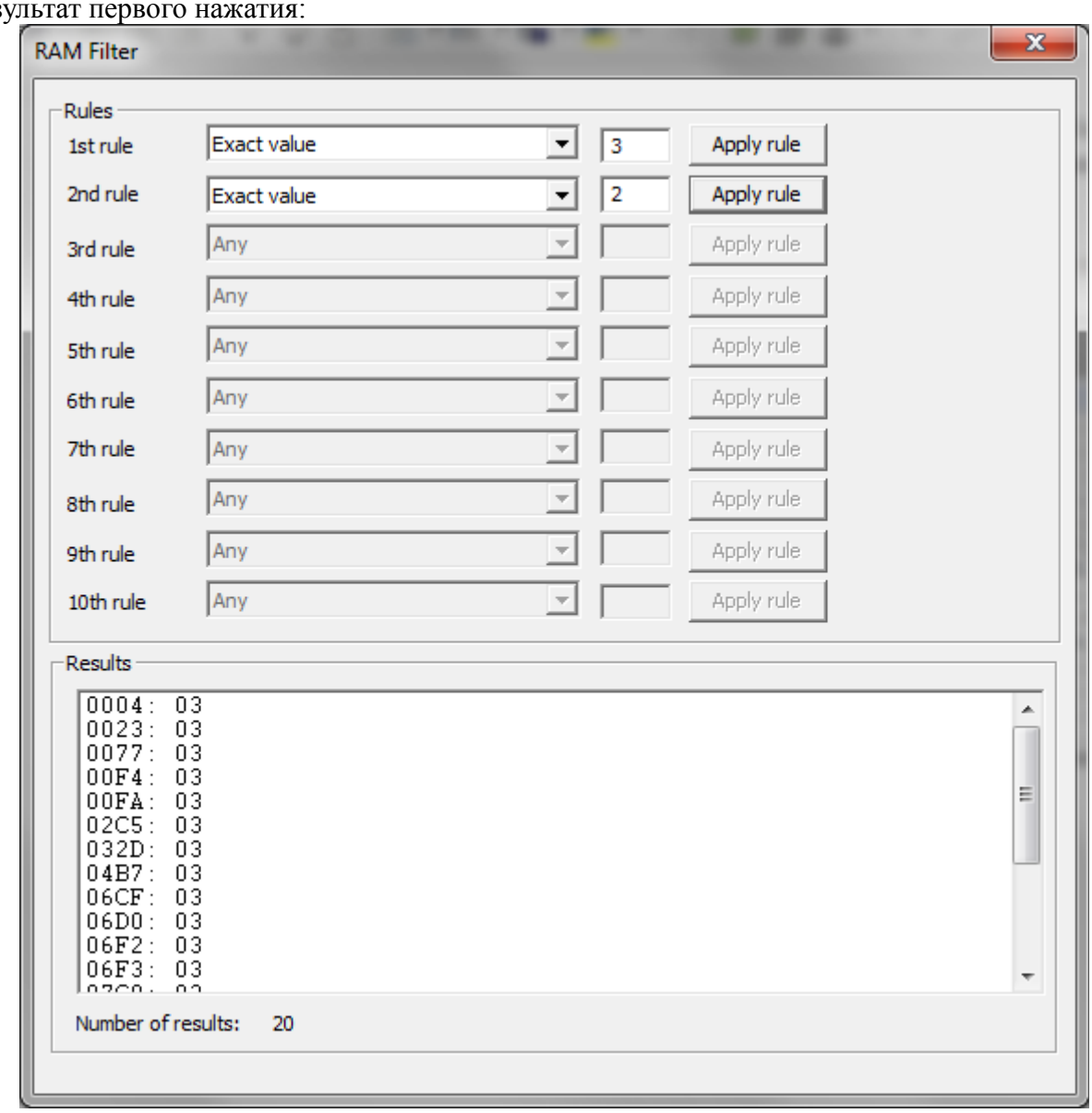
Инструмент, позволяющий фильтровать память по определенным значениям.

Например, чтобы определить, по какому адресу сохраняется текущее значение здоровья, можно задать следующие правила:

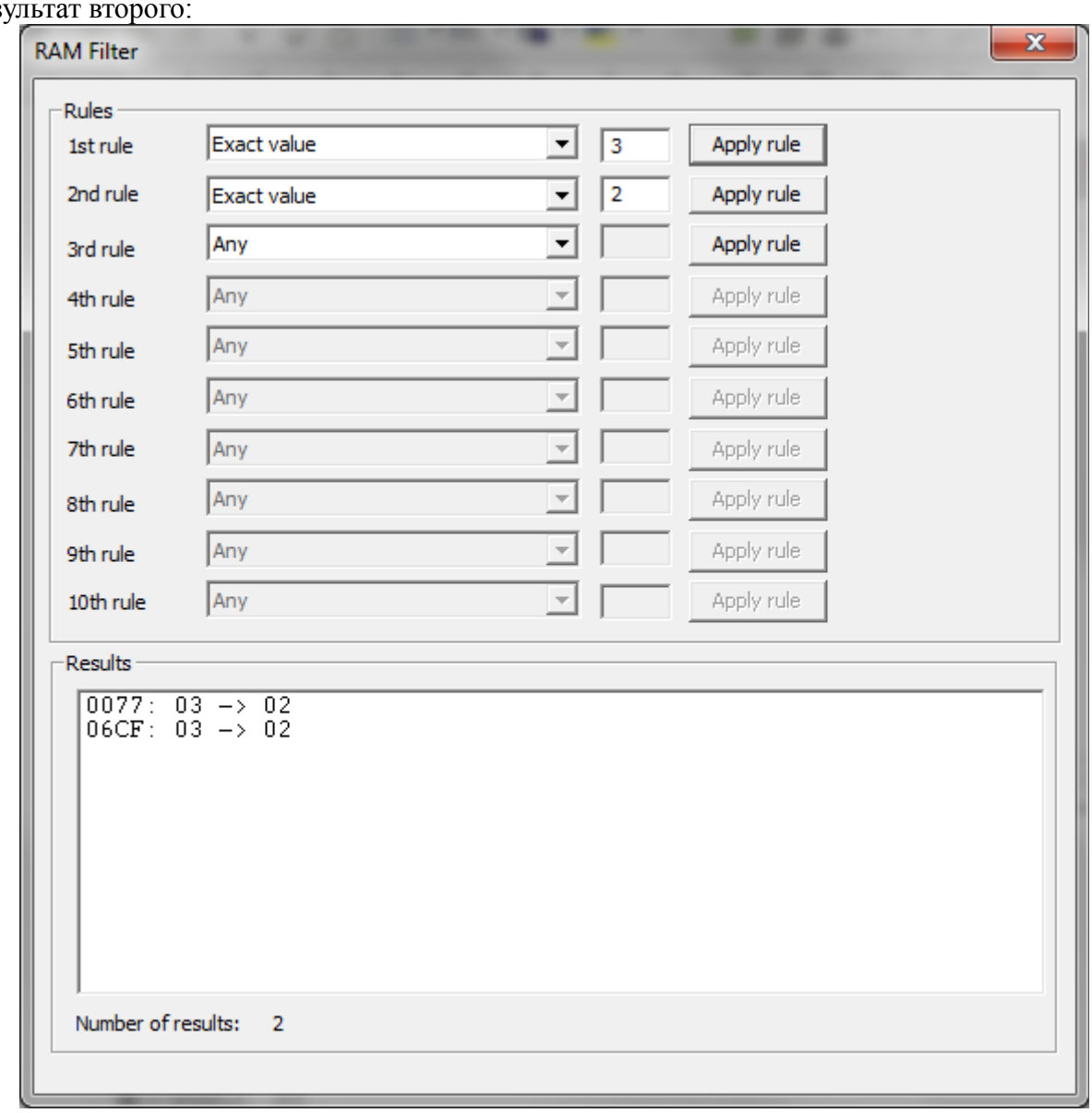


Нажимаем верхнюю кнопку Apply rule (поскольку текущее значение здоровья 3), затем теряем в игре 1 единицу здоровья и нажимаем вторую кнопку Apply rule.

Результат первого нажатия:



Результат второго:



Таким образом видно, что значение сохраняется в двух ячейках — $0077 и $06CF.

(by loginsin aka ALXR. habrahabr.ru/users/loginsin/)