```
===============================================================================
            Everything You Always Wanted To Know About GAME BOY *
===============================================================================

                        * but were afraid to ask


                      Written by Dr. -Pan-/Anthrox

 Forward: The following was typed up for informational purposes regarding
          the inner workings on the hand-held game machine known as
          Game Boy, manufactured and designed by Nintendo Co., LTD.
          All files found in this package are presented to inform
          a user on how their Game Boy works and what makes it "tick".
          Game Boy is copyrighted by Nintendo Co., LTD.
          Any reference to copyrighted material is not presented
          for monetary gain, but for educational purposes and higher
          learning.


          If you do not ask, you will not know - -Pan-/ATX
          No askie, no learnie - someone must've said this...

1. What are the Game Boy's true specs?
   -----------------------------------

 The Game Boy really is a powerful machine! FAQ files on Game Boy can be
 found on many Internet sites. Most FAQs contains wrong information.
 Such as claiming it only has 8 sprites.  Here are the correct facts:

 The Game Boy uses a custom/updated/or modified Z80 processor. Comparing
 the Game Boy's Z80 instruction set with a book on the Z80 (circa 1982)
 shows that the GB Z80 has a few different instructions. Such as
 LD (HLI),#$xx
 LD (HLD),#$xx
 SWAP A through L
 LD A,($xx)

 Screen Size: Physical screen: 160*144  VRAM screen image: 256*256
 Screen scrolling is wrap around type; when a part of the image is off the
 screen it will be shown on the opposite side of the screen.

 Although the screen can contain 1024 tiles, only 256 of them may be UNIQUE.
 Each tile may have up to 4 colors. You may change the color of the pixel
 value.  There are 4 shades of gray. You can select which shade you want
 for that pixel value. However, when you change the color for that pixel value
```

```
EVERY tile that has a pixel with the same value will also be affected.
This is good for a routine which fades out the screen or performs a GLOWING
effect of some kind.
The tile graphics are 8*8 pixels, each pixel contains 2 bits of data to
create 4 numbers. Each number is the color value for that pixel.
The graphics are stored as as bitmapped tiles much like the SNES.
This is the confusing part for me. It is either like the
4 color SNES tile or it is INTERLEAVED.

SNES 4 Color Tile: A            INTERLEAVED 4 Color Tile: A

.11111..                         .11111..   <- first plane
11...11.                         ........   <- second plane
11...11.                         11...11.
1111111.  <- first plane         ........
11...11.                         11...11.
11...11.                         ........
11...11.                         1111111.   <- first plane
........                         ........   <- second plane

........                         11...11.
........                         ........
........                         11...11.
........                         ........
........  <- second plane        11...11.
........                         ........
........                         ........
........                         ........

You'll have to figure it out by yourself since I haven't gotten into it :)

Graphics vram location for OBJ and BG tiles start at $8000 and end at $97FF



Sprites: 40 Sprites! They may be 8*8 or 8*16.

Each sprite has up to 4 colors. There are 2 palettes to chose from

The sprites can be flipped on the X and/or Y axis

Sprite OAM ram:  Location: $FE00->$FE9F

Each sprite data contains 4 bytes of info. They are:

Byte 1: Y screen position; 8 bits
Byte 2: X screen position; 8 bits
Byte 3: Character code; tile number $00-$FF
Byte 4: Palette, X, Y, Priority; Most Significant 4 bits.
```

```
          First 4 bits are NOT USED!

          Bit 7 - Priority
          Bit 6 - Y flip
          Bit 5 - X flip
          Bit 4 - Palette number; 0,1
          Bit 3-0 - NOT USED!

  Display Ram size: 64k bit
  Work Ram size: 64k bit

  There are 2 Memory Bank Controllers (MBC) that can be used. MBC1 is the
  standard that is used on most cartridges.
  MBC2 is used with cartridges which need Save-Ram.
  It controls extended Save-RAM banks.

  Extended Ram may go up to 256k bit.

  MBC1 - When controlling ROM only you may read up to 16 megabits! (2 MBYTES)
         When controlling RAM only you may read up to 4 megabits (512 kbytes)
                                       and read up to 256kbit RAM

  MBC2 - Controls Back-Up Ram (Save-RAM) (512 * 4 bit) which can be extended
         to 2 megabits (16 kbyte * 16)  256k byte

  Sound: There are only 2 channels; left and right.
         But there are 4 different ways to produce sound:
         Sound 1: produces quadrangular wave patterns with sweep and envelope
                  functions
         Sound 2: produces quadrangular wave patterns with envelope functions
         Sound 3: produces a voluntary wave pattern (samples can be possible
                                                   if done right)
         Sound 4: produces white noise

         You tell the channel which sound number you want to use and it
         will produce the sound when you've set the according data.

         Since I'm not a sound programmer I really can't get into the details
         of how to create waves patterns and such. It would confuse you more
         than me!
```

```
 2. The Game Boy seems like a nice little system! But what are the registers?
    And what about the memory mapping?
    -------------------------------------------------------------------------

    You can check out the memory mapping by viewing the 2 graphic diagrams
    included with this informational tutorial. They are called:
    Address1.IFF, Address1.PCX, Address2.IFF, Address2.PCX

    .IFF is for Amiga users.   .PCX is for PC users.

    They will display how the memory is mapped out and how banking works.


    The registers:

 ------------------------------------------------------------------------------
    Address  - $FF00
    Name     - P1
    Contents - Register for reading joy pad info.    (R/W)

            Bit 7 - Not used
            Bit 6 - Not used
            Bit 5 - P15 out port
            Bit 4 - P14 out port
            Bit 3 - P13 in port
            Bit 2 - P12 in port
            Bit 1 - P11 in port
            Bit 0 - P10 in port

          This is a very strange way of reading joypad info.
          There are only 8 possible button/switches on the Game Boy.
          A, B, Select, Start, Up, Down, Left, Right.
          Why they made their joypad registers in this way I'll never know.
          They could have used all 8 bits and you just read which one is on.

          This is the matrix layout for register $FF00:


            P14                P15
             |                  |
 --P10-------O-Right------------O-A---------
             |                  |
 --P11-------O-Left-------------O-B---------
             |                  |
 --P12-------O-Up---------------O-Select----
             |                  |
 --P13-------O-Down-------------O-Start-----
```

```
  |                  |


This is the logic in reading joy pad data:

Turn on P15 (bit 5) in $ff00
Wait a few clock cycles
read $ff00 into A
invert A   - same as EOR #$FF - just reverse all bits
             apparently the joy pad info returned is like the C64
             info. 0 means on, 1 means off. But logic tells us
             that it should be the other way around. So to make it
             less confusing we just flip the bits!

AND A with #$0F - get only the first four bits
                  By turning on P15 we are trying to read column
                  P15 in the matrix layout. It contains A,B,SEL,STRT

SWAP A - #$3f becomes #$f3, it swaps hi<->lo nibbles

store A in B for backup


Turn on P14 (bit 4) in $ff00
Wait a few more clock cycles
read $ff00 into A
invert A - just as above
AND A with #$0F - get first 4 bits
                - By turning on P14 we get the data for column P14
                  in the matrix layout. It contains U,D,L,R

OR A with B - put the two values together.

turn on P14 and P15 in $ff00 to reset.

The button values using the above method are such:
$80 - Start            $8 - Down
$40 - Select           $4 - Up
$20 - B                $2 - Left
$10 - A                $1 - Right

Let's say we held down A, Start, and Up.
The value returned in accumulator A would be $94


Let's see this method in action!
Game: Ms. Pacman
Address: $3b1
```

```
 0003B1: 0003B1: 3E 20          LD A,#$20      <- bit 5 = $20
 0003B3: 0003B3: EA 00 FF       LD ($FF00),A   <- turn on P15
 0003B6: 0003B6: FA 00 FF       LD A,($FF00)
 0003B9: 0003B9: FA 00 FF       LD A,($FF00)   <- wait a few cycles
 0003BC: 0003BC: 2F             CPL            <- complement (invert) EOR #$ff
 0003BD: 0003BD: E6 0F          AND #$0F       <- get only first 4 bits
 0003BF: 0003BF: CB 37          SWAP A         <- swap it
 0003C1: 0003C1: 47             LD B,A         <- store A in B
 0003C2: 0003C2: 3E 10          LD A,#$10      <- bit 4 = $10
 0003C4: 0003C4: EA 00 FF       LD ($FF00),A   <- turn on P14
 0003C7: 0003C7: FA 00 FF       LD A,($FF00)
 0003CA: 0003CA: FA 00 FF       LD A,($FF00)
 0003CD: 0003CD: FA 00 FF       LD A,($FF00)
 0003D0: 0003D0: FA 00 FF       LD A,($FF00)
 0003D3: 0003D3: FA 00 FF       LD A,($FF00)
 0003D6: 0003D6: FA 00 FF       LD A,($FF00)   <- Wait a few MORE cycles
 0003D9: 0003D9: 2F             CPL            <- complement (invert)
 0003DA: 0003DA: E6 0F          AND #$0F       <- get first 4 bits
 0003DC: 0003DC: B0             OR B           <- put A and B together
```

The following routine is common on SNES as well. It clarifies that you've only pressed the specified button(s) once every other frame. That way the Joypad is less sensitive to wrong/bad/false movements.

```
 0003DD: 0003DD: 57             LD D,A         <- store A in D
 0003DE: 0003DE: FA 8B FF       LD A,($FF8B)   <- read old joy data from ram
 0003E1: 0003E1: AA             XOR D          <- toggle w/current button bit
 0003E2: 0003E2: A2             AND D          <- get current button bit back
 0003E3: 0003E3: EA 8C FF       LD ($FF8C),A   <- save in new Joydata storage
 0003E6: 0003E6: 7A             LD A,D         <- put original value in A
 0003E7: 0003E7: EA 8B FF       LD ($FF8B),A   <- store it as old joy data



 0003EA: 0003EA: 3E 30          LD A,#$30      <- turn on P14 and P15
 0003EC: 0003EC: EA 00 FF       LD ($FF00),A   <- RESET Joypad?!
 0003EF: 0003EF: C9             RET            <- Return from Subroutine
```

```
   Address  - $FF01
   Name     - SB
   Contents - Serial transfer data (R/W)

              8 Bits of data to be read/written

   Address  - $FF02
   Name     - SC
```

```
    Contents - SIO control  (R/W)

               Bit 7 - Transfer start flag
                       0: Non transfer
                       1: Start transfer

               Bit 0 - Shift Clock
                       0: External Clock
                       1: Internal Clock

 ------------------------------------------------------------------------------

    Address  - $FF04
    Name     - DIV
    Contents - Divider Register (R/W)

 ------------------------------------------------------------------------------

    Address  - $FF05
    Name     - TIMA
    Contents - Timer counter (R/W)

               The timer generates an interrupt when it overflows.

    Address  - $FF06
    Name     - TMA
    Contents - Timer Modulo (R/W)

               When the TIMA overflows, this data will be loaded.

    Address  - $FF07
    Name     - TAC
    Contents - Timer Control

               Bit 2 - Timer Stop
                       0: Stop Timer
                       1: Start Timer

               Bits 1+0 - Input Clock Select
                          00: 4.096 khz
                          01: 262.144 khz
                          10: 65.536 khz
                          11: 16.384 khz

 ------------------------------------------------------------------------------

    Address  - $FF0F
    Name     - IF
    Contents - Interrupt Flag (R/W)
```

```
               Bit 4: Transition from High to Low of Pin number P10-P13
               Bit 3: Serial I/O transfer end
               Bit 2: Timer Overflow
               Bit 1: LCDC (see STAT)
               Bit 0: V-Blank

    Address  - $FFFF
    Name     - IE
    Contents - Interrupt Enable (R/W)

               Bit 4: Transition from High to Low of Pin number P10-P13
               Bit 3: Serial I/O transfer end
               Bit 2: Timer Overflow
               Bit 1: LCDC (see STAT)
               Bit 0: V-Blank

               0: disable
               1: enable

    Address  - XXXX (CPU instruction command)
    Name     - IME
    Content  - Interrupt Master Enable

               To prohibit ALL interrupts use CPU instruction DI
               To acknowledge interrupt settings use CPU instruction EI
               DI - Disable Interrupts
               EI - Enable Interrupts

    The priority and jump address for the above 5 interrupts are:

     Interrupt        Priority        Start Address

     V-Blank             1              $0040
     LCDC Status         2              $0048 - Modes 0, 01, 10
                                                LYC=LY coincide (selectable)

     Timer Overflow      3              $0050
     Serial Transfer     4              $0058 - when transfer is complete
     Hi-Lo Of Pin        5              $0060

     * When more than 1 interrupts occur at the same time ONLY the interrupt
       with the highest priority can be acknowledged.
       When an interrupt is used a '0' should be stored in the IF register
       before the IE register is set.


 ----------------------------------------------------------------------------
```

```
     Address  - $FF40
     Name     - LCDC
     Contents - LCD Control (R/W)

                Bit 7 - LCD Control Operation
                        0: Stop completely (no picture on screen)
                        1: operation

                Bit 6 - Window Screen Display Data Select
                        0: $9800-$9BFF
                        1: $9C00-$9FFF

                Bit 5 - Window Display
                        0: off
                        1: on

                Bit 4 - BG Character Data Select
                        0: $8800-$97FF
                        1: $8000-$8FFF <- Same area as OBJ

                Bit 3 - BG Screen Display Data Select
                        0: $9800-$9BFF
                        1: $9C00-$9FFF

                Bit 2 - OBJ Construction
                        0: 8*8
                        1: 8*16

                Bit 1 - OBJ Display
                        0: off
                        1: on

                Bit 0 - BG Display
                        0: off
                        1: on


     Address  - $FF41
     Name     - STAT
     Contents - LCDC Status   (R/W)

                Bits 6-3 - Interrupt Selection By LCDC Status

                Bit 6 - LYC=LY Coincidence (Selectable)
                Bit 5 - Mode 10
                Bit 4 - Mode 01
                Bit 3 - Mode 00
                        0: Non Selection
                        1: Selection
```

```
               Bit 2 - Coincidence Flag
                       0: LYC not equal to LCDC LY
                       1: LYC = LCDC LY

               Bit 1-0 - Mode Flag
                         00: Entire Display Ram can be accessed
                         01: During V-Blank
                         10: During Searching OAM-RAM
                         11: During Transfering Data to LCD Driver


      STAT shows the current status of the LCD controller.
      Mode 00: When the flag is 00 it is the H-Blank period and the CPU can
               access the display RAM ($8000-$9FFF)
               When it is not equal the display ram is being used by the
               LCD controller

      Mode 01: When the flag is 01 it is the V-Blank period and the CPU can
               access the display RAM ($800-$9FFF)

      Mode 10: When the flag is 10 then the OAM is being used ($FE00-$FE90)
               The CPU cannot access the OAM during this period

      Mode 11: When the flag is 11 both the OAM and CPU are being used.
               The CPU cannot access either during this period

 -----------------------------------------------------------------------------

    Address  - $FF42
    Name     - SCY
    Contents - Scroll Y   (R/W)

               8 Bit value $00-$FF to scroll BG Y screen position

    Address  - $FF43
    Name     - SCX
    Contents - Scroll X   (R/W)

               8 Bit value $00-$FF to scroll BG X screen position

    Address  - $FF44
    Name     - LY
    Contents - LCDC Y-Coordinate (R)

             The LY indicates the vertical line to which the present data
             is transferred to the LCD Driver
             The LY can take on any value between 0 through 153. The values
             between 144 and 153 indicate the V-Blank period. Writing will
```

```
                  reset the counter.

                  This is just a RASTER register. The current line is thrown
                  into here. But since there are no RASTERS on an LCD display.....
                  it's called the LCDC Y-Coordinate.

         Address  - $FF45
         Name     - LYC
         Contents - LY Compare  (R/W)

                  The LYC compares itself with the LY. If the values are the same
                  it causes the STAT to set the coincident flag.

         Address  - $FF47
         Name     - BGP
         Contents - BG Palette Data  (W)

                    Bit 7-6 - Data for Dot Data 11
                    Bit 5-4 - Data for Dot Data 10
                    Bit 3-2 - Data for Dot Data 01
                    Bit 1-0 - Data for Dot Data 00

                    This selects the shade of gray you what for your BG pixel.
                    Since each pixel uses 2 bits, the corresponding shade will
                    be selected from here. The Background Color (00) lies at
                    Bits 1-0, just put a value from 0-$3 to change the color.

         Address  - $FF48
         Name     - OBP0
         Contents - Object Palette 0 Data (W)

                    This selects the colors for sprite palette 0.
                    It works exactly as BGP ($FF47).
                    See BGP for details.

         Address  - $FF49
         Name     - OBP1
         Contents - Object Palette 1 Data (W)

                    This Selects the colors for sprite palette 1.
                    It works exactly as BGP ($FF47).
                    See BGP for details.


         Address  - $FF4A
         Name     - WY
         Contents - Window Y Position  (R/W)

                    0 <= WY <= 143
```

```
               WY must be greater than or equal to 0 and must be less than
               or equal to 143.

    Address  - $FF4B
    Name     - WX
    Contents - Window X Position  (R/W)

               7 <= WX <= 166

               WX must be greater than or equal to 7 and must be less than
               or equal to 166.


               Lets say WY = 80 and WX = 80.
               The window would be positioned as so:

                0                  80                          159
                _________________________________________________
             0 |                   |                             |
               |                   |                             |
               |                   |                             |
               |                   |                             |
               |                   |                             |
               |                   |                             |
               |                   |                             |
               |                   |                             |
               |                   |                             |
               |                   |80                           |
            80 |-------------------+-----------------------------|
               |                80 |                             |
               |                   |                             |
               |                   |      Window Display         |
               |                   |                             |
               |                   |                             |
               |                   |           Here              |
               |                   |                             |
               |                   |                             |
               |                   |                             |
           143 |___________________|_____________________________|

           OBJ Characters (Sprites) can still enter the window
           So can BG characters

 -----------------------------------------------------------------------------

    Address  - $FF46
    Name     - DMA
    Contents - DMA Transfer and Start Address (W)

    The DMA Transfer (40*28 bit) from internal ROM or RAM ($0000-$F19F)
    to the OAM (address $FE00-$FE9F) can be performed. It takes 160 nano-seconds
    for the transfer.

    40*28 bit = #140  or #$8C.  As you can see, it only transfers $8C bytes
    of data. OAM data is $A0 bytes long, from $0-$9F.

    But if you examine the OAM data you see that 4 bits are not in use.

    40*32 bit = #$A0, but since 4 bits for each OAM is not used it's
    40*28 bit.

    It transfers all the OAM data to OAM RAM.

    The DMA transfer start address can be designated every $100 from address
    $0000-$F100.   That means $0000, $0100, $0200, $0300....

    Example program:
       DI             <- Disable Interrupt
       LD A,#$04      <- transfer data from $0400
       LD ($FF46),A   <- put A into DMA registers
       LD A,#40       <- #40 is the value to wait for. we need to wait 160
 Wait:                <- nano seconds
       DEC A          <- decrease A by 1
       JR NZ,Wait     <- branch if Not Zero to Wait
       EI             <- Enable Interrupt
       RET            <- RETurn from sub-routine

 ------------------------------------------------------------------------------

    Address  - $FF10
    Name     - NR 10
    Contents - Sound Mode 1 register, Sweep register (R/W)

               Bit 6-4 - Sweep Time
               Bit 3 - Sweep Increase/Decrease
                       0: Addition    (frequency increases)
                       1: Subtraction (frequency increases)
               Bit 2-0 - Number of sweep shift (# 0-7)

               Sweep Time:

               000: sweep off
               001: 7.8 ms
               010: 15.6 ms
               011: 23.4 ms
               100: 31.3 ms
```

```
                101: 39.1 ms
                110: 46.9 ms
                111: 54.7 ms


     Address  - $FF11
     Name     - NR 11
     Contents - Sound Mode 1 register, Sound length/Wave pattern duty (R/W)

                Only Bits 7-6 can be read.

                Bit 7-6 - Wave Pattern Duty
                Bit 5-0 - Sound length data (# 0-63)

                Wave Duty:

                00: 12.5%
                01: 25%
                10: 50%
                11: 75%

     Address  - $FF12
     Name     - NR 12
     Contents - Sound Mode 1 register, Envelope (R/W)

                Bit 7-4 - Initial value of envelope
                Bit 3 - Envelope UP/DOWN
                        0: Decrease
                        1: Range of increase
                Bit 2-0 - Number of envelope sweep (# 0-7)

                Initial value of envelope is from %0000 to %1111

     Address  - $FF13
     Name     - NR 13
     Contents - Sound Mode 1 register, Frequency lo (W)

                lower 8 bits of 11 bit frequency.
                Next 3 bit or in NR 14 ($FF14)

     Address  - $FF14
     Name     - NR 14
     Contents - Sound Mode 1 register, Frequency hi (R/W)

                Only Bit 6 can be read.

                Bit 7 - Initial (when set, sound restarts)
                Bit 6 - Counter/consecutive selection
                Bit 2-0 - Frequency's higher 3 bits
     Address  - $FF16
     Name     - NR 21
     Contents - Sound Mode 2 register, Sound Length; Wave Pattern Duty (R/W)

                Only bits 7-6 can be read.

                Bit 7-6 - Wave pattern duty
                Bit 5-0 - Sound length (# 0-63)

     Address  - $FF17
     Name     - NR 22
     Contents - Sound Mode 2 register, envelope (R/W)

                Bit 7-4 - Initial envelope value
                Bit 3 - Envelope UP/DOWN
                        0: decrease
                        1: range of increase
                Bit 2-0 - Number of envelope step (# 0-7)


     Address  - $FF18
     Name     - NR 23
     Contents - Sound Mode 2 register, frequency lo data (W)

                Frequency's lower 8 bits of 11 bit data
                Next 3 bits are in NR 14 ($FF19)

     Address  - $FF19
     Name     - NR 24
     Contents - Sound Mode 2 register, frequency hi data (R/W)

                Only bit 6 can be read.

                Bit 7 - Initial
                Bit 6 - Counter/consecutive selection
                Bit 2-0 - Frequency's higher 3 bits

     Address  - $FF1A
     Name     - NR 30
     Contents - Sound Mode 3 register, Sound on/off (R/W)

                Only bit 7 can be read

                Bit 7 - Sound OFF
                        0: Sound 3 output stop
                        1: Sound 3 output OK

     Address  - $FF1B
     Name     - NR 31
     Contents - Sound Mode 3 register, sound length (R/W)

                Bit 7-0 - Sound length

     Address  - $FF1C
     Name     - NR 32
     Contents - Sound Mode 3 register, Select output level

                Only bits 6-5 can be read

                Bit 6-5 - Select output level
                          00: Mute
                          01: Produce Wave Pattern RAM Data as it is
                              (4 bit length)
                          10: Produce Wave Pattern RAM data shifted once to the
                              RIGHT (1/2)  (4 bit length)
                          11: Produce Wave Pattern RAM data shifted twice to the
                              RIGHt (1/4)  (4 bit length)

         * - Wave Pattern RAM is located from $FF30-$FF3f

     Address  - $FF1D
     Name     - NR 33
     Contents - Sound Mode 3 register, frequency's lower data (W)

                Lower 8 bits of an 11 bit frequency

     Address  - $FF1E
     Name     - NR 34
     Contents - Sound Mode 3 register, frequency's higher data (R/W)

                Only bit 6 can be read.

                Bit 7 - Initial flag
                Bit 6 - Counter/consecutive flag
                Bit 2-0 - Frequency's higher 3 bits



     Address  - $FF20
     Name     - NR 41
     Contents - Sound Mode 4 register, sound length (R/W)

                Bit 5-0 - Sound length data (# 0-63)


     Address  - $FF21
     Name     - NR 42
     Contents - Sound Mode 4 register, envelope (R/W)

                Bit 7-4 - Initial value of envelope
                Bit 3 - Envelope UP/DOWN
                        0: decrease
                        1: range of increase
                Bit 2-0 - number of envelope step (# 0-7)

     Address  - $FF22
     Name     - NR 43
     Contents - Sound Mode 4 register, polynomial counter (R/W)

                Bit 7-4 - Selection of the shift clock frequency of the
                          polynomial counter
                Bit 3 - Selection of the polynomial counter's step
                Bit 2-0 - Selection of the dividing ratio of frequencies

                Selection of the dividing ratio of frequencies:
                000: f * 1/2^3 * 2
                001: f * 1/2^3 * 1
                010: f * 1/2^3 * 1/2
                011: f * 1/2^3 * 1/3
                100: f * 1/2^3 * 1/4
                101: f * 1/2^3 * 1/5
                110: f * 1/2^3 * 1/6
                111: f * 1/2^3 * 1/7           f = 4.194304 Mhz

                Selection of the polynomial counter step:
                0: 15 steps
                1: 7 steps

                Selection of the shift clock frequency of the polynomial
                counter:

                0000: dividing ratio of frequencies * 1/2
                0001: dividing ratio of frequencies * 1/2^2
                0010: dividing ratio of frequencies * 1/2^3
                0011: dividing ratio of frequencies * 1/2^4
                      :                           :
                      :                           :
                      :                           :
                0101: dividing ratio of frequencies * 1/2^14
                1110: prohibited code
                1111: prohibited code

     Address  - $FF30
     Name     - NR 30
     Contents - Sound Mode 4 register, counter/consecutive; inital (R/W)
```

```
               Only bit 6 can be read.

               Bit 7 - Inital
               Bit 6 - Counter/consecutive selection


    Address  - $FF24
    Name     - NR 50
    Contents - Channel control / ON-OFF / Volume (R/W)

               Bit 7 - Vin->SO2 ON/OFF
               Bit 6-4 - SO2 output level (volume) (# 0-7)
               Bit 3 - Vin->SO1 ON/OFF
               Bit 2-0 - SO1 output level (volume) (# 0-7)

               Vin->SO1 (Vin->SO2)

               By synthesizing the sound from sound 1 through 4, the voice
               input from Vin terminal is put out.
               0: no output
               1: output OK

     Address  - $FF25
     Name     - NR 51
     Contents - Selection of Sound output terminal (R/W)

                Bit 7 - Output sound 4 to SO2 terminal
                Bit 6 - Output sound 3 to SO2 terminal
                Bit 5 - Output sound 2 to SO2 terminal
                Bit 4 - Output sound 1 to SO2 terminal
                Bit 3 - Output sound 4 to SO1 terminal
                Bit 2 - Output sound 3 to SO1 terminal
                Bit 1 - Output sound 2 to SO1 terminal
                Bit 0 - Output sound 0 to SO1 terminal

     Address  - $FF26
     Name     - NR 52
     Contents - Sound on/off (R/W)

                Only Bit 7, 3-0 can be read.

                Bit 7 - All sound on/off
                        0: stop all sound circuits
                        1: operate all sound circuits
                Bit 3 - Sound 4 ON flag
                Bit 2 - Sound 3 ON flag
                Bit 1 - Sound 2 ON flag
                Bit 0 - Sound 1 ON flag
```

```
 3. Ok, so the Game Boy runs on a Z80, and that's an 8 bit processor...
    So how is it possible for an 8 bit processor to access 4 Mbits?
    ------------------------------------------------------------------

    Simple! Bank switching! If you examine the graphic diagram Address1.xxx
    you will see some very strange stuff!

    The Z80 can only work with 16 bit addresses $0-$FFFF

    So to access the other data you must trick the machine into pointing
    to another piece of memory.

    ROM is located from $0000 - $7FFF, RAM is from $8000-$FFFF

    All game programs are ROM so we know it is from $0000-$7FFF

    But the Game Boy has a fixed memory area from $0000-$3FFF; when you
    access it, it will always be BANK 0. It is called the FIXED HOME ADDRESS.

    That means the only other ROM addresses available are $4000-$7FFF.

    Bank 0 is read by the CPU as being at $0000-$3FFF
    Bank 1 is read by the CPU as being at $4000-$7FFF
    Bank 2 is read by the CPU as being at $4000-$7FFF
    Bank 3 is read by the CPU as being at $4000-$7FFF

    See the pattern? Only the FIXED HOME ADDRESS has it's own special
    location.

    Banks and addresses starting at $4000 is called the CPU address.

    CPU Address $014000 is actually Bank #$01 address $4000

    CPU Address $014000 is equal to ROM address (offset) $004000

    CPU Address $024000 is equal to ROM address (offset) $008000

    CPU Address $044000 is equal to ROM address (offset) $010000
    The CPU uses the CPU ADDRESS.

    How to switch BANKS:

    Using MBC1 (Memory Bank Controller 1)

    Writing to ROM Address (CPU FIXED HOME ADDRESS) $2000-$3FFF
    the ROM bank can be selected. The values are from #$01-#$0F

    LD A,#$01
    LD ($2000),A     <- this selects ROM BANK #$01


    Writing to ROM Address (CPU FIXED HOME ADDRESS) $4000-$5FFF
    the RAM bank can be selected. The values are from #$00-#$03

    LD A,#$03
    LD ($4000),A     <- this select RAM BANK #$03

    Using MBC2 (Memory Bank Controller 2)

    Writing to ROM Address (CPU FIXED HOME ADDRESS) $2100-$21FF
    the ROM bank can be select. The values are from #$01-#$0F






 4. The SNES and Genesis has an some cartridge information in the ROM.
    Does the Game Boy have some info, too?
    ------------------------------------------------------------------

    Sure, why not?!

    The Internal Info block begins at $100 and it's format is as follows:

    $100-$101 - 00 C3  (2 bytes)
    $102-$102 - Lo Hi  (Start Address for Game, usually $150 it would be written
                        as 50 01)
    $100-$133 - Nintendo Character Area, if this does not exist the game
                will not run!

 000100: 00 C3 50 01 CE ED 66 66 CC 0D 00 0B 03 73 00 83
 000110: 00 0C 00 0D 00 08 11 1F 88 89 00 0E DC CC 6E E6
 000120: DD DD D9 99 BB BB 67 63 6E 0E EC CC DD DC 99 9F
 000130: BB B9 33 3E
```

```
     $134-$143 - Title Registration Area (title of the game in ASCII)
     $144-$146 - NOT USED
     $147 - CARTRIDGE TYPE
            0 - ROM ONLY
            1 - ROM+MBC1
            2 - ROM+MBC1+RAM
            3 - ROM+MBC1+RAM+BATTERY
            5 - ROM+MBC2
            6 - ROM+MBC2+BATTERY

    $148 - ROM SIZE
           0 - 256kbit
           1 - 512kbit
           2 - 1M-Bit
           3 - 2M-Bit
           4 - 4M-Bit

    $149 - RAM SIZE
           0 - NONE
           1 - 16kbit
           2 - 64kbit
           3 - 256kbit

    $14A-$14B - Maker Code - 2 bytes
    $14C - Version Number
    $14D - Complement Check
    $14E-$14F - Checksum HI-LO (2 bytes in Big Endian format, high byte first)
```

```
  5. Are there any public development tools available for the Game Boy?
     ------------------------------------------------------------------

     I don't know of many Z80 cross-assemblers available, or if they support
     instructions like SWAP A, or LD (HLI),A.
     It could be possible to use a generic Z80 assembler and to write
     the non-standard instructions by entering the op-code as a Declared Byte.

     The only publicly available disassembler I know of is the one I made
     for the Amiga. It is included in a SNES 65816/SPC700 disassembler
     and supports non-standard Z80 op-codes. The multi-processor disassembler
     is called Super Magic Disassembler, it's current release version
     is 1.4 and can be found on bulletin boards world-wide.
```

```
 6. Are there any Game Boy Emulators available?
    -------------------------------------------

    NO! NO! NONONONO! The only SO-CALLED Emulator was on the AMIGA but was
    a BIG FAKE! If you got it to work, it only played TETRIS.
    It did NOT emulate a Game Boy. It was just a version of Tetris
    played in a graphic rendition of a Game Boy.
    According to some FAQ floating around the Emulator is called Toy-Boy
    and it was made by Argonaut Software.   Come on! Argonaut has 1 Amiga,
    which was used for graphics.. but since they've gotten a Silicon Graphics
    Workstation I highly doubt they use it anymore!




 7. What about Super-Game Boy info? What are it's Specs?
    ----------------------------------------------------

    What am I, Master Brain of the World? I don't have info on everything! :)



 8. Wow! This was some cool info! Now I can sleep at night!
    -------------------------------------------------------

    Yeah! Now you can stop bugging me with all your questions!
```