Contents

No$nes Controls

General Joypad Controls
Keyboard controls for up to four players can be defined in setup. Buttons of USB Gamepads & Joysticks & the like can be also changed per player.

Special Controls

```
F1..F5  Enable/Disable Sound Channel 1-5
F9      Enable/Disable Showing Screen Splits as dotted line
NUM *   Hard Reset
NUM /   Soft Reset
NUM -   Switch to Debug Mode
NUM +   Disable Realtime Delays and use 10% frameskip (while held down)
BS      Disable Realtime Delays (same as above, for notebook keyboards)
```

Famicom Disk System (FDS)

```
INS     Flip Disk (some games ignore fast flips; hold down for some seconds)
```

VS Unisystem (Arcade Cabinet)

```
INS       Credit Left Coin Slot
HOME      Credit Right Coin Slot
PGUP      Credit Service Button
1..4      Button 1..4 (aliases for NES Start/Select buttons)
0         DIP-Switch Window
```

Playchoice 10 (Arcade Cabinet)

```
INS       Credit Coin Slot 1
HOME      Credit Coin Slot 2 (works only if enabled via DIP switches)
PGUP      Service Button (add credit) (during reset: bookkeeping)
DEL       Reset Button (if any)
END       Enter Button (also works via Num-Enter)
PGDN      Channel Select Button
0         DIP-Switch Window
Mouse     Zapper (Lightgun Move/Trigger)
```

FamicomBox

```
INS         XXX Insert Coin
HOME      RESET Button
PGUP        XXX GAME/TV Button
123456    1st..6th Keyswitch Position (from left)     [=INTENDED, see Note]
0         DIP-Switch Window
Mouse     Zapper (Lightgun Move/Trigger)
```

Note: The ordering of the keyswitch positions from left-to-right is still unknown; currently keys 1..6 are simply mapped to keyswitch bit0..5.
Multi-cart loading isn't emulated yet, so you'll only see the menu & self-test feature, without any games.

Famicom Keyboard

```
BS/DEL  DEL
HOME    CLR
TAB     ESC
```

```
END     STOP
L-ALT   GRPH
R-ALT   KANA
```

Party Tap Push Buttons

```
123456  Buttons for Player 1-6
```

Konami Hyper Shot Push Buttons

```
Joypad Button A --> "RUN"-Button (or, used as "Shoot-Right" in Hyper Sports)
Joypad Button B --> "JUMP"-Button (or, used as "Shoot-Left" in Hyper Sports)
```

The games do replace joypad data by button data; no$nes somewhat undoes that replacement.

Lightguns (NES Zapper, Famicom Beam Gun, VS Zapper, Bandai Hyper Shot Gun)

```
Mouse                Move
Left Mouse Button    Trigger
```

Paddle (NES and both Famicom versions)

```
Mouse (left/right)   Move
Left Mouse Button    Button
```

Oeka Kids Tablet

```
Mouse                Move & Touch
Left Mouse Button    Click
```

Mouse / Trackball

```
Mouse (move)         Move
Left/Right Button    Left/Right Button (A/B Button on Trackball)
Middle Button or ESC Return mouse to operating system
```

Power Glove

```
Gamepad Analog X/Y/Z/R   Position X/Y/Z and Wrist Rotation
Gamepad Button 1..4      Thumb/Index/Middle/Ring Finger Flex
Gamepad Digital POV      Keypad DPAD
Keyboard 0..9            Keypad 0..9
Keyboard Buttons/DPAD    Keypad A,B,Start,Select,DPAD
XXX                      Keypad Prog,Center,Enter
```

UForce

```
Page Down         Top-most sensor                       ;\for push-button
NUM-/ NUM-*       Upper Left/Right sensors in upper field ; style usage (with
NUM-8 NUM-9       Lower Left/Right sensors in upper field ; sensor=min/max)
NUM-5             Upper Left       sensor  in lower field ; (eg. Nuclear Rat,
NUM-2 NUM-3       Lower Left/Right sensors in lower field ; and Rock on Air)
Page Up           Bottom-most sensor                    ;/
Gamepad Analog X  Top-most sensor                       ;\for analog usage
Gamepad Analog Y  Bottom-most sensor, or,               ; (eg. Hose'em Down
Gamepad Analog Y  Lower-Left sensor in lower field      ;/and Rock on Air)
```

Plus Start/Select as configured in joypad setup. Analog mode is activate when moving the analog gamepad (and will then override the corresponding keyboard keys). Analog input emulation is matched to the existing "UForce Power Games" cartridge (and may mismatch with other/homebrew UForce games).

Top-Rider Bike

```
Gamepad Analog X         Steering Handles Left/Right
Gamepad Analog Y         Forward=GearLo, Back=GearHi
Gamepad Analog Z         Forward=Accelerate Slow/Fast, Back=Accelerate Turbo
Gamepad Button 1,2,3,4   Brake,Wheelie,Start,Select
```

Pachinko

```
Digital Joypad/Keyboard  Pachinko Joypad Buttons
Analog Joypad Forward    Pachinko Analog Dial
```

Barcode Reader (Barcode Batter and Datach)

```
INS-Key       Paste numeric ASCII string from clipboard and Send barcode
DEL-Key       Manually Clear input buffer (without sending)
Keypad 0..9   Manually Key-in Barcode digits
Keypad Dot    Manually Confirm input and Send (or re-send) barcode
```

Note: For the Barcode Battler, only EAN-13 is implemented (since it's unknown how/if it supports shorter UPC-A, UPC-E, or EAN-8 barcodes). For Datach, EAN-13, UPC-A, EAN-8 are supported (however, the Datach software seems to support a further non-standard unknown barcode format, which isn't emulated).

Capcom Mahjong Controller

```
A..N          Drop Card A..N (or N=Draw new card)
Space         Same as N
F1..F7        Function Keys (Select, Start, and five "japanese" keys)
Tab/LeftCtrl  Same as F1 (Select)
Enter         Same as F2 (Enter)
```

Piano Keyboard

```
12345...      1st Octave (12 keys) (only last 7 keys used on Doremikko)
QWERT...      2nd Octave (12 keys) (all keys used)
ASDFG...      3rd Octave (12 keys) (all keys used)
ZXCVB...      4th Octave (10 keys) (only first 5 keys used on Doremikko)
Rshift/Ctrl   4th Octave (last 2)  (Miracle only)
NUM-Dot       5th Octave (1 key)   (Miracle only)
NUM-0..7      Control Buttons 0..7 (Miracle only)
Left Shift    Sustain Foot Pedal   (Miracle only)
```

Note: The miracle sound isn't emulated (as far as known there aren't any dumps of it's BIOS and SOUND-ROM existing yet). The FDS sound for Doremikko isn't emulated yet (when I last checked, there seems to have been little known about FDS sound hardware).

Mats (Power Pad/Family Trainer/Tap-tap Mat)

```
QWER --> Upper row (Side B)     RTYU --> Upper Row (Side A, or Tap-tap Mat)
ASDF --> Middle row (Side B)    FGHJ --> Middle Row (Side A, or Tap-tap Mat)
ZXCV --> Lower row (Side B)     VBNM --> Lower Row (Side A, or Tap-tap Mat)
```

Safety Note: Put the keyboard on the floor if you are using feet or hammer.

Exciting Boxing Bag

```
Num-7,8,9   Left Hook,   N/A    ,Right Hook
Num-4,5,6   Left Jabb, Straight ,Right Jabb
Num-1,2,3   Left Push,  Body    ,Right Push
```

RacerMate Bicycle Trainer

```
Gamepad Analog X   Pulse
Gamepad Analog Y   Speed
Gamepad Analog Z   Watts
F1 or Enter        Button F1 (START)
F2                 Button F2 (DISPLAY)
F3                 Button F3 (SET)
F4 or End          Button RESET
PGUP or Up         Button Plus (UP)
PGDN or Down       Button Minus (DOWN)
```

XED Editor

XED About

About XED

XED is a text editor, the executable is fast and small, and it includes comfortable editing functions. It is both intended for standard .TXT files (or any other ASCII files, such like .ASM for assembler source code). Also, the line-wrapping support (.XED files) can be used for authoring stories, specifications, etc. Most of the features are much the same as for other text editors, some special functions are pointed out below:

Block Selection

XED supports good old quality block mechanisms, allowing to copy/move the selection directly to cursor position by Ctrl+K,C/V hotkeys (without needing to use paste). For data exchange with other programs or text files, the block can be directly written to (or loaded from) file by Ctrl+K,R/W. And, mainstream copy/cut/paste functions are supported as well, by Ctrl+Ins, Shift+Del, Shift+Ins.
Note: The block remains selected even when typing text, and it won't get deleted when touching Del-key.

Condensed Display Mode

Condensed mode is activated by "F6,C" key combination. In this mode, only lines beginning with colon ":", or (for assembler source code files) with semicolon-colon ";:", for example:

```
:Chapter IV
;:---Sound Engine---
```

Normal block functions can be used in this mode to Move, Copy, or Delete whole 'chapter(s)'. Cursor keys can be used to move the cursor to a specific chapter. Pushing Enter or Escape terminates condensed mode.

Column Block Mode

Column mode is activated by "Ctrl+K,N" key combination. In this mode, the block selection appears as a rectangular area, allowing to deal with tables & columns in text files by using copy/delete, indent/unindent block functions.
Typing "Ctrl+K,N" again will return to normal block mode (in which any lines between begin/end of the block will be selected at full length).

Blank Space

Unlike most other editors, XED allows to move the cursor to any screen location, including at horizontal positions after the end of the current line. Entering

space characters at such locations advances the cursor position, but does not actually store space characters in the file.
When typing text, spaces are automatically inserted between line-end and cursor position. Respectively, ending spaces are automatically deleted (eg. assume that the line contains "Hello !", deleting "!" will also remove the space character, internally).
That is of course all happening behind your back, you won't have to care about it - but you can be sure that there'll be no hidden spaces filling up disk space.

Tabulation Marks / TAB-Key
The TAB Key advances the cursor to the next higher tabulation location (usually in steps of eight columns, counted from leftmost screen border), and the appropriate number of spaces is inserted into the file if necessary.
In overwrite mode (de-/activated by INS Key), the TAB Key simply advances the cursor without actually inserting spaces (and without overwriting existing text by spaces).

Tabulation Marks / CHR(9)
When enabled in setup (default), TAB marks are automatically expanded into appropriate number of spaces (ie. towards next "8-column" position) when loading a file.
The file is normally saved by using raw SPC characters, without any TABs. Optionally, it can be saved by using "best-fill" SPCs and TABs (disabled by default), that feature may conflict with third party tools (assemblers, compilers, etc). In order to reduce the risk of such problems, best-fill is suppressed in quoted lines (by using ' or " or <> quotation marks, eg. db 'Hello !').

Line Wrapping
Line wrapping is enabled/disabled by "F5+W" key combination. Wrapping is automatically enabled when loading a file with extension ".XED".
In the file, wrapped lines are using CR characters as soft linebreaks, paragraphs are terminated by normal CR,LF characters.
Note: It'd be recommended to convert .XED files into 'standard' formats such like .TXT or .HTM before releasing them, but preferably NOT into disliked bloated file formats such like .PDF or .DOC.

Word Processing
Aside from the above line-wrapping support, no other 'word processing' features are included, the program provides normal 'type writer' functions, not more, not less. In particular, any overload such like bold or colored text, big and small fonts, bitmaps and different fonts are disliked.

XED Hotkeys

XED recognizes both CP/M Wordstar hotkeys (also used by Borland PC compilers), and Norton editor hotkeys (NU.EXE). The "Ctrl+X,Y" style hotkeys are wordstar based, particulary including good block functions. The F4,X and Alt/Ctrl+X type hotkeys are norton based, particulary very useful for forwards/backwards searching.

Standard Cursor Keys

```
Up          Move line up
Down        Move line down
Left        Move character left
Right       Move character right
Pgup        Scroll page up / to top of screen
Pgdn        Scroll page down / to bottom of screen
Ctrl+Pgup   Go to start of file (or Ctrl+Home)
Ctrl+Pgdn   Go to end of file (or Ctrl+End)
Home        Go to start of line
End         Go to end of line
Ctrl+Left   Move word left
Ctrl+Right  Move word right
Ins         Toggle Insert/Overwrite mode
Del         Delete char below cursor
Backspace   Delete char left of cursor
Tab         Move to next tabulation mark
Enter       New line/paragraph end
Esc         Quit (or Alt+X, F3+Q, Ctrl+K+D, Ctrl+K+Q, Ctrl+K+X)
```

Note: Pgup/Pgdn are moving the cursor to top/bottom of screen, page scrolling takes place only if the cursor was already at that location.

Editor Keys

```
Ctrl+Y      Delete line (or Alt+K)
Alt+L       Delete to line end (or Ctrl+Q,Y)
Alt+V       Caseflip to line end
Ctrl+V      Caseflip from line beginning
```

Norton Search/Replace Functions

```
Alt+F       Norton - search/replace, forwards
Ctrl+F      Norton - search/replace, backwards
Alt+C       Norton - continue search/replace, forwards
Ctrl+C      Norton - continue search/replace, backwards
```

Search: Type "Alt/Ctrl+F, String, Enter".
Search+replace: "Type Alt/Ctrl+F, String1, Alt+F, String2, Enter".
Non-case sensitive: Terminate by Escape instead of Enter.

Wordstar Search/Replace Functions

```
  Ctrl+Q,F   Wordstar - search
  Ctrl+Q,A   Wordstar - replace
  Ctrl+L     Wordstar - continue search/replace
```
Search options: B=Backwards, G=Global, N=No query,
U=non-casesensitive, W=whole words only, n=n times.

Disk Commands
```
  F3,E       Save+exit
  F3,S       Save (or Ctrl+K,S)
  F3,N       Edit new file
  F3,A       Append a file
```
See also: Block commands (read/write block).

Block Selection
```
  Shift+Cursor  Select block begin..end
  Ctrl+K,B   Set block begin (or F4,S)
  Ctrl+K,K   Set block end   (or F4,S)
  Ctrl+K,H   Remove/hide block markers (or F4,R)
  F4,L       Mark line including ending CRLF (or Ctrl+K,L)
  F4,E       Mark line excluding ending CRLF
  Ctrl+K,T   Mark word
  Ctrl+K,N   Toggle normal/column blocktype
```

Clipboard Commands
```
  Shift+Ins  Paste from Clipboard
  Shift+Del  Cut to Clipboard
  Ctrl+Ins   Copy to Clipboard
  Ctrl+Del   Delete Block
```

Block Commands
```
  Ctrl+K,C   Copy block (or F4,C)
  Ctrl+K,V   Move block (or F4,M)
  Ctrl+K,Y   Delete block (or F4,D)
  Ctrl+K,P   Print block  (or F7,B)
  Ctrl+Q,B   Find block begin (or F4,F)
  Ctrl+Q,K   Find block end   (or F4,F)
  Ctrl+K,R   Read block from disk towards cursor location
  Ctrl+K,W   Write block to disk
  Ctrl+K,U   Unindent block (delete one space at begin of each line)
  Ctrl+K,I   Indent block (insert one space at begin of each line)
  F5,F       Format block (with actual x-wrap size) (or ;Ctrl+B)
  F8,A       Add values within column-block
```

Setup Commands
```
  F11        Setup menu (or F8,S)
  F5,S       Save editor configuration
  F5,L       Set line len for word wrap (or Ctrl+O,R)
  F5,W       Wordwrap on/off (or Ctrl+O,W) (*)
  F5,I       Auto indent on/off (or Ctrl+O,I)
  F5,T       Set tab display spacing
```
(*) Wrapped lines will be terminated by CR, paragraphs by CRLF.

Other
```
  F1         Help
  F2         Status (displays info about file & currently selected block)
  F8,M       Make best fill tabs
  F8,T       Translate all tabs to spaces
  SrcLock    Freeze cursor when typing text ("useful" for backwards writing)
  Ctrl+O,C   Center current line
  Ctrl+K,#   Set marker (#=0..9)
  Ctrl+Q,#   Move to marker (#=0..9)
  Ctrl+Q,P   Move to previous pos
  F6,C       Condensed display mode on/off (*)
  Ctrl+G     Go to line nnnn (or F6,G) (or commandline switch /l:nnnn)
```
(*) only lines starting with ':' or ';:' will be displayed. cursor and block commands can be used (e.g. to copy a text-sequence by just marking it's headline)

Hex-Edit Keys (Binary Files)
This mode is activated by /b commandline switch, allowing to view and modify binary files. Aside from normal cursor keys, the following hotkeys are used:
```
  Tab        Toggle between HEX and ASC mode (or Shift+Left/Right)
  Ctrl+Arrow Step left/right one full byte (instead one single HEX digit)
  Ctrl+G     Goto hex-address
  Ctrl+K,S   Save file (as usually)
```

Printer Commands
```
  F7,P       Print file
  F7,B       Print block (or Ctrl+K,P)
  F7,E       Eject page
  F7,S       Set page size
```

More printer options can be found in setup. Printing was working well (at least with my own printer) in older XED versions, but it is probably badly bugged (at least untested) for years now.

XED Assembler/Debugger Interface

Nocash Debuggers
The XED editor provides simple but very useful interaction with the various nocash debuggers/emulators (nogba, nogmb, nocpc, nomsx, no$c64, no$2k6, nozx, nones, nosns, nox51).
The editor can be launched from inside of the debugger (by Alt+E hotkey, by retaining the recently edited line number when re-launching the editor).
And, when editing assembler source code files, F9-key can used to launch the assembler from inside of XED. That is, the file is saved to disk, the A22i assembler is started (built-in in all debuggers), and, in case of successful assembly, the program is loaded & started in the emulator. Otherwise, the assembler displays a list of errors, and the editor is moved directly to the source code line number in which the first error has occured.

16bit DOS debuggers
The XED editor is included built-in in all nocash windows debuggers, and in the no$gba 32bit DOS version only.
For use with other nocash 16bit DOS debuggers the XED editor must be downloaded separately at http://problemkaputt.de/xed.htm, and must be installed in a directory which is included in your PATH statement.

XED Commandline based standalone version

Standalone 16bit DOS version
This version is written in turbo pascal, nevertheless fast enough to work on computer with less than 10MHz. It uses 16bit 8086 code, and works with all 80x86 compatible CPUs, including very old XTs.
The downside is that it is restricted to Conventional DOS Memory, so that the maximum filesize is 640K (actually less, because the program and operating system need to use some of that memory).

Using the 32bit debugger-built-in version as 32bit standalone editor
I haven't yet compiled a 32bit standalone version, however, any of the no$xxx 32bit debuggers can be used for that purpose. By commandline input:

```
no$xxx /x <filename>   Edit text file in standalone mode
no$xxx /b <filename>   Edit binary file in standalone hexedit mode
```

Standalone Commandline Syntax
Syntax: XED <filename> [/l:<line number>] | /?

```
<name>    Filename, optionally d:\path\name.ext
/?        Displays commandline help
/l:<nnn>  Moves to line number nnn after loading
```

The filename does not require to include an extension, the program automatically loads the first existing file with any of following extensions appended: XED, ASM, ASC, INC, BAT, TXT, HTM, DOC, A22, PAS.

Standalone DOS Return Value
XED returns a three-byte return value after closing the program. This data is used when calling XED as external editor from inside of nocash DOS debuggers, but it might be also useful for other purposes.
Because normal DOS return values are limited to 8bits only, the three bytes are written into video RAM at rightmost three character locations in first line:

```
VSEG:77*2  Exit code   (00h=Exit normal, F9h=Exit by F9-key)
VSEG:78*2  Line number (Lower 8bits, 1..65536 in total)
VSEG:79*2  Line number (Upper 8bits)
```

The color attribute for these characters is set to zero (invisible, black on black). Use INT 10h/AH=0Fh to determine the current video mode (AL AND 7Fh), if it is monochrome (07h) then use VSEG=B000h, otherwise VSEG=B800h.

No$nes Emulation Files

SLOT Folder
Default location for Game ROM-Images is "SLOT" folder (in same directory as "no$nes.exe").
ZIPped ROM-Images can be loaded if PKUNZIP.EXE is installed (ie. it must be somewhere in your "PATH") (if you don't know what that means, put it into a folder like C:\WINDOWS\COMMAND or so).

BIOS Folder
The "BIOS" folder (in same directory as "no$nes.exe") should contain following files:

```
DISKSYS.ROM   8Kbytes  Famicom Disk System (FDS) BIOS
PC10BIOS.ROM  16Kbytes Playchoice 10 (PC10) BIOS    (IC "8T")
PC10CHAR.ROM  24Kbytes Playchoice 10 (PC10) Charset (ICs "8P+8M+8K")
```

Playchoice 10 Games
PC10 games require the BIOS files (see above), and completely dumped ROMs. As of 2012, most or all existing PC10 dumps are incomplete: They

contain only the NES PRG/CHR-ROMs, and the Z80 INST-ROM, but do miss the Z80 PROM (which is absolutely required to use/decrypt the INST-ROMs). A tool for adding the missing PROM data to existing ROM-images can be found here: http://problemkaputt.de/pc10make.zip

Tech Data

Overall Specs
CPU 2A03 - customized 6502 CPU - audio - does not contain support for decimal
The NTSC NES runs at 1.7897725MHz, and 1.662607MHz for PAL.

NMIs may be generated by PPU each VBlank.
IRQs may be generated by APU and by external hardware.
Internal Memory: 2K WRAM, 2K VRAM, 256 Bytes SPR-RAM, and Palette/Registers

The cartridge connector also passes audio in and out of the cartridge, to allow for external sound chips to be mixed in with the Famicom audio.

Original Famicom (Family Computer) (1983) (Japan)
60-pin cartridge slot, with external sound-input, without lockout chip.
Two joypads directly attached to console, Joypad 1 with Start/Select buttons, Joypad 2 with microphone, but without Start/Select. 15pin Expansion port for further controllers. Video RF-Output only.
"During its first year, people found the Famicom to be unreliable, with programming errors and freezing rampant. Yamauchi recalled all sold Famicom systems, and put the Famicom out of production until the errors were fixed. The Famicom was re-released with a new motherboard."

Original NES (Nintendo Entertainment System) (1985) (US, Europe, Australia)
Same as Famicom, but with slightly different pin-outs on cartridge slot, and controllers/expansion ports: Front-loading 72-pin cartridge slot, without external sound-input on cartridge slot, without microphone on joypads, with lockout chip.

Newer Famicom, AV Famicom (1993-1995)
60-pin cartridge slot, with external sound-input, without lockout chip.
Includes NES-style joystick connectors, plus the original 15pin Famicom Expansion port. Doesn't have microphone. Video AV-Output only.

Newer NES (1993-1995)
Top-loading 72-pin cartridge slot, without external sound-input, without lockout chip. Poorer video signal than old NES. Video WHAT?-output only.

VS Unisystem
Arcade Machine. Coin-detect inputs, eight Dip-switches, different palette.

Play Choice 10
Arcade Machine with 10 cartridge slots. Uses Z80 as second CPU.

Memory Maps

Internal Memory: 2K WRAM, 2K VRAM, 256 Bytes SPR-RAM, and Palette/Registers

CPU Memory Map (16bit buswidth, 0-FFFFh)

```
  0000h-07FFh   Internal 2K Work RAM (mirrored to 800h-1FFFh)
  2000h-2007h   Internal PPU Registers (mirrored to 2008h-3FFFh)
  4000h-4017h   Internal APU Registers
  4018h-5FFFh   Cartridge Expansion Area almost 8K
  6000h-7FFFh   Cartridge SRAM Area 8K
  8000h-FFFFh   Cartridge PRG-ROM Area 32K
```

CPU Reset vector located at [FFFC], even smaller carts must have memory at that location. Larger carts may use whatever external mappers to access more than the usual 32K.

PPU Memory Map (14bit buswidth, 0-3FFFh)

```
  0000h-0FFFh   Pattern Table 0 (4K) (256 Tiles)
  1000h-1FFFh   Pattern Table 1 (4K) (256 Tiles)
  2000h-23FFh   Name Table 0 and Attribute Table 0 (1K) (32x30 BG Map)
  2400h-27FFh   Name Table 1 and Attribute Table 1 (1K) (32x30 BG Map)
  2800h-2BFFh   Name Table 2 and Attribute Table 2 (1K) (32x30 BG Map)
  2C00h-2FFFh   Name Table 3 and Attribute Table 3 (1K) (32x30 BG Map)
  3000h-3EFFh   Mirror of 2000h-2EFFh
  3F00h-3F1Fh   Background and Sprite Palettes (25 entries used)
  3F20h-3FFFh   Mirrors of 3F00h-3F1Fh
```

Note: The NES contains only 2K built-in VRAM, which can be used for whatever purpose (for example, as two Name Tables, or as one Name Table plus 64 Tiles). Palette Memory is built-in as well. Any additional VRAM (or, more regulary, VROM) is located in the cartridge, which may also contain mapping hardware to access more than 12K of video memory.

SPR-RAM Memory Map (8bit buswidth, 0-FFh)

```
  00-FF         Sprite Attributes (256 bytes, for 64 sprites / 4 bytes each)
```

Sprite RAM is directly built-in in the PPU chip. SPR-RAM is not connected to CPU or PPU bus, and can be accessed via I/O Ports only.

I/O Map

I/O Map

```
  2000h - PPU Control Register 1 (W)
  2001h - PPU Control Register 2 (W)
  2002h - PPU Status Register (R)
  2003h - SPR-RAM Address Register (W)
  2004h - SPR-RAM Data Register (RW)
  2005h - PPU Background Scrolling Offset (W2)
  2006h - VRAM Address Register (W2)
  2007h - VRAM Read/Write Data Register (RW)
  4000h - APU Channel 1 (Rectangle) Volume/Decay (W)
  4001h - APU Channel 1 (Rectangle) Sweep (W)
  4002h - APU Channel 1 (Rectangle) Frequency (W)
  4003h - APU Channel 1 (Rectangle) Length (W)
  4004h - APU Channel 2 (Rectangle) Volume/Decay (W)
  4005h - APU Channel 2 (Rectangle) Sweep (W)
  4006h - APU Channel 2 (Rectangle) Frequency (W)
  4007h - APU Channel 2 (Rectangle) Length (W)
  4008h - APU Channel 3 (Triangle) Linear Counter (W)
  4009h - APU Channel 3 (Triangle) N/A (-)
  400Ah - APU Channel 3 (Triangle) Frequency (W)
  400Bh - APU Channel 3 (Triangle) Length (W)
  400Ch - APU Channel 4 (Noise) Volume/Decay (W)
  400Dh - APU Channel 4 (Noise) N/A (-)
  400Eh - APU Channel 4 (Noise) Frequency (W)
  400Fh - APU Channel 4 (Noise) Length (W)
  4010h - APU Channel 5 (DMC) Play mode and DMA frequency (W)
  4011h - APU Channel 5 (DMC) Delta counter load register (W)
  4012h - APU Channel 5 (DMC) Address load register (W)
  4013h - APU Channel 5 (DMC) Length register (W)
  4014h - SPR-RAM DMA Register (W)
  4015h - DMC/IRQ/length counter status/channel enable register (RW)
  4016h - Joypad #1 (RW)
  4017h - Joypad #2/APU SOFTCLK (RW)
```

Additionally, external hardware may contain further ports:

```
  4020h - VS Unisystem Coin Acknowledge
  4020h-40FFh - Famicom Disk System (FDS)
  4100h-FFFFh - Various addresses used by various cartridge mappers
```

Picture Processing Unit (PPU)

Based on "Nintendo Entertainment System Documentation" Version 2.00 by Jeremy Chadwick aka Y0SHi aka JDC. Which was itself based on "Nintendo Entertainment System Architecture" by Marat Fayzullin.

3D Glasses

PPU Reset

Registers

Some registers are reset to zero upon reset; namely NMIs are disabled. Other registers are kept unchanged (or contain semi-stable initial values on Power-Up).

Locked Registers

During first frame after Reset, Ports 2000h, 2001h, 2005h, and 2006h are reportedly write-protected. And, Port 2007h is read-protected (always returns

00h, even if all VRAM at 0000h..3FFFh is FFh-filled, and even if the 2007h prefetch latch was pre-loaded with a nonzero value before reset; the origin of the 00h is unknown, it might be an open-bus value).
The read/write protection is released when:

```
NTSC:  At END of First Vblank (261 scanlines after reset)
PAL:   At END of First Vblank (311 scanlines after reset)
```

PPU Control and Status Registers

2000h - PPU Control Register 1 (W)

```
Bit7  Execute NMI on VBlank             (0=Disabled, 1=Enabled)
Bit6  PPU Master/Slave Selection        (0=Master, 1=Slave) (Not used in NES)
Bit5  Sprite Size                       (0=8x8, 1=8x16)
Bit4  Pattern Table Address Background  (0=VRAM 0000h, 1=VRAM 1000h)
Bit3  Pattern Table Address 8x8 Sprites (0=VRAM 0000h, 1=VRAM 1000h)
Bit2  Port 2007h VRAM Address Increment (0=Increment by 1, 1=Increment by 32)
Bit1-0 Name Table Scroll Address        (0-3=VRAM 2000h,2400h,2800h,2C00h)
(That is, Bit0=Horizontal Scroll by 256, Bit1=Vertical Scroll by 240)
```

2001h - PPU Control Register 2 (W)

```
Bit7-5 Color Emphasis       (0=Normal, 1-7=Emphasis) (see Palettes chapter)
Bit4  Sprite Visibility     (0=Not displayed, 1=Displayed)
Bit3  Background Visibility (0=Not displayed, 1=Displayed)
Bit2  Sprite Clipping       (0=Hide in left 8-pixel column, 1=No clipping)
Bit1  Background Clipping   (0=Hide in left 8-pixel column, 1=No clipping)
Bit0  Monochrome Mode       (0=Color, 1=Monochrome)  (see Palettes chapter)
```

If both sprites and BG are disabled (Bit 3,4=0) then video output is disabled, and VRAM can be accessed at any time (instead of during VBlank only). However, SPR-RAM does no longer receive refresh cycles, and its content will gradually degrade when the display is disabled.

2002h - PPU Status Register (R)

```
Bit7   VBlank Flag    (1=VBlank)
Bit6   Sprite 0 Hit   (1=Background-to-Sprite0 collision)
Bit5   Lost Sprites   (1=More than 8 sprites in 1 scanline)
Bit4-0 Not used       (Undefined garbage)
```

Reading resets the 1st/2nd-write flipflop (used by Port 2005h and 2006h).
Reading resets Bit7, can be used to acknowledge NMIs, Bit7 is also automatically reset at the end of VBlank, so manual acknowledge is normally not required (unless one wants to free the NMI signal for external NMI inputs) (and unless one wants to disable/reenable NMIs during NMI handling, in that case Bit7 MUST be acknowledge before reenabling NMIs, else NMI would be executed another time).

Status Notes

VBlank flag is set in each frame, even if the display is fully disabled, and even if NMIs are disabled. Hit flag may become set only if both BG and OBJ are enabled. Lost Sprites flag may become set only if video is enabled (ie. BG or OBJ must be on). For info about the "Not used" status bits, and some other PPU bits see:
Unpredictable Things

VS System

Some VS System PPUs have Port 2000h/2001h swapped, and do have a Chip ID in LSBs of 2002h. For details, see
VS System

PPU SPR-RAM Access Registers

2003h - SPR-RAM Address Register (W)

```
D7-D0: 8bit address in SPR-RAM  (00h-FFh)
```

Specifies the destination address in Sprite RAM for use with Port 2004h (Single byte write), and Port 4014h (256 bytes DMA transfer).
This register is internally used during rendering (and typically contains 00h at the begin of the VBlank period).

2004h - SPR-RAM Data Register (Read/Write)

```
D7-D0: 8bit data written to SPR-RAM.
```

Read/write data to/from selected address in Sprite RAM.
The Port 2003h address is auto-incremented by 1 after each <write> to 2004h.
The address is NOT auto-incremented after <reading> from 2004h.
Caution: Single byte access via 2004h is somewhat working, but DMA via 4014h is more robust.

4014h - Sprite DMA Register (W)

Transfers 256 bytes from CPU Memory area into SPR-RAM. The transfer takes 512 CPU clock cycles, two cycles per byte, the transfer starts about immediately after writing to 4014h: The CPU either fetches the first byte of the next instruction, and then begins DMA, or fetches and executes the next instruction, and then begins DMA. The CPU is halted during transfer.

```
Bit7-0  Upper 8bit of source address (Source=N*100h) (Lower bits are zero)
```

Data is written to Port 2004h. The destination address in SPR-RAM is thus [2003h], which should be normally initialized to zero - unless one wants to "rotate" the target area, which may be useful when implementing more than eight (flickering) sprites per scanline.

Notes
SPR-RAM should be accessed during VBlank only, preferably by issuing Sprite DMA at begin of Vblank.
SPR-RAM is dynamic memory, refreshed during rendering, it does no longer receive refresh cycles (and will lose its content quickly within about a frame) when the display is disabled (by clearing both Bit3+Bit4 in Port 2001h).
PAL issues another refresh somewhere mid-vblank even when display disabled? NTSC doesn't do so and tends to forget data more easily - using DMA to rewrite sprite memory on vblank is somewhat forcefully fixing that issue.

Port 2003h/2004h are internally used during rendering. The unlicensed game Micro Machines is reading 2004h to determine the currently rendered line, however, that doesn't work with all PPU revisions and PPU clones.

PPU VRAM Access Registers

Registers used to Read and Write VRAM data, and for Background Scrolling.
The CPU can Read/Write VRAM during VBlank only - because the PPU permanently accesses VRAM during rendering (even in HBlank phases), and because the PPU uses the VRAM Address register as scratch pointer. Respectively, the address in Port 2006h is destroyed after rendering, and must be re-initialized before using Port 2007h.

1st/2nd Write
Below Port 2005h and 2006h require two 8bit writes to receive a 16bit parameter, the current state (1st or 2nd write) is memorized in a single flipflop, which is shared for BOTH Port 2005h and 2006h. The flipflop is reset when reading from PPU Status Register Port 2002h (the next write will be then treated as 1st write) (and of course it is also reset after any 2nd write).

2005h - PPU Background Scrolling Offset (W2)
Defines the coordinates of the upper-left background pixel, together with PPU Control Register 1, Port 2000h, Bits 0-1).
```
Port 2005h-1st write: Horizontal Scroll Origin (X*1) (0-255)
Port 2005h-2nd write: Vertical Scroll Origin   (Y*1) (0-239)
Port 2000h-Bit0: Horizontal Name Table Origin  (X*256)
Port 2000h-Bit1: Vertical Name Table Origin    (Y*240)
```
Caution: The above scroll reload settings are overwritten by writes to Port 2006h. See PPU Scrolling chapter for more info.

2006h - VRAM Address Register (W2)
Used to specify the 14bit VRAM Address for use with Port 2007h.
```
Port 2006h-1st write: VRAM Address Pointer MSB (6bit)
Port 2006h-2nd write: VRAM Address Pointer LSB (8bit)
```
Caution: Writes to Port 2006h are overwriting scroll reload bits (in Port 2005h and Bit0-1 of Port 2000h). And, the PPU uses the Port 2006h register internally during rendering, when the display is enabled one should thus reinitialize Port 2006h at begin of VBlank before accessing VRAM via Port 2007h.

2007h - VRAM Read/Write Data Register (RW)
The PPU will auto-increment the VRAM address (selected via Port 2006h) after each read/write from/to Port 2007h by 1 or 32 (depending on Bit2 of $2000).
```
Bit7-0  8bit data read/written from/to VRAM
```
Caution: Reading from VRAM 0000h-3EFFh loads the desired value into a latch, and returns the OLD content of the latch to the CPU. After changing the address one should thus always issue a dummy read to flush the old content. However, reading from Palette memory VRAM 3F00h-3FFFh, or writing to VRAM 0000-3FFFh does directly access the desired address.
Note: Some (maybe all) RGB PPUs (as used in Famicom Titler) do not allow to read palette memory (instead, they appear to mirror 3Fxxh to 2Fxxh).

Caution
The APU (if DMC sound is used) can conflict with joypad reads! For details, see:
APU DMC-DMA Glitch

PPU Scrolling

The PPU allows to scroll the background pixelwise horizontally and vertically. The total scroll-able area is 512x480 pixels (though the full size can be usedwith external memory only, see Name Tables chapter), of which circa 256x240 pixels are displayed (see visible screen resolution).

Vertical offsets 240-255 (aka Tile Rows 30-31) will cause garbage Tile numbers to be fetched from the Attribute Table (instead of from Name Table), after line 255 it will wrap to line 0, but without producing a carry-out to the Name Table Address.

Scroll Pointer and Reload Registers
Scrolling relies on a Pointer register (Port 2006h), and on a Reload register (Port 2005h, and Bit0-1 of Port 2000h). The Pointer is automatically incremented by the hardware during rendering, and points to the currently drawn tile row, the same pointer register is also used by software to access VRAM during VBlank or when the display is disabled. The Reload value defines the horizontal and vertical origin of upper-left pixel, the reload value is automatically loaded into the Pointer at the end of the vblank period (vertical reload bits), and at the begin of each scanline (horizontal reload bits). The relation between Pointer and Reload bits is:
```
VRAM-Pointer            Scroll-Reload
A8  2006h/1st-Bit0 <--> Y*64  2005h/2nd-Bit6
```

```
A9  2006h/1st-Bit1 <--> Y*128 2005h/2nd-Bit7
A10 2006h/1st-Bit2 <--> X*256 2000h-Bit0
A11 2006h/1st-Bit3 <--> Y*240 2000h-Bit1
A12 2006h/1st-Bit4 <--> Y*1   2005h/2nd-Bit0
A13 2006h/1st-Bit5 <--> Y*2   2005h/2nd-Bit1
-   2006h/1st-Bit6 <--> Y*4   2005h/2nd-Bit2
-   2006h/1st-Bit7 <--> -     -
A0  2006h/2nd-Bit0 <--> X*8   2005h/1st-Bit3
A1  2006h/2nd-Bit1 <--> X*16  2005h/1st-Bit4
A2  2006h/2nd-Bit2 <--> X*32  2005h/1st-Bit5
A3  2006h/2nd-Bit3 <--> X*64  2005h/1st-Bit6
A4  2006h/2nd-Bit4 <--> X*128 2005h/1st-Bit7
A5  2006h/2nd-Bit5 <--> Y*8   2005h/2nd-Bit3
A6  2006h/2nd-Bit6 <--> Y*16  2005h/2nd-Bit4
A7  2006h/2nd-Bit7 <--> Y*32  2005h/2nd-Bit5
-   -              <--> X*1   2005h/1st-Bit0
-   -              <--> X*2   2005h/1st-Bit1
-   -              <--> X*4   2005h/1st-Bit2
```

Port 2006h-1st Write (VRAM Pointer MSB)
As one might (not) have expected, this does NOT change the VRAM Pointer, instead, the written value is stored in the corresponding Reload bits (Port 2005h/2000h settings), the VRAM pointer is left unchanged for now.

Port 2006h-2nd Write (VRAM Pointer LSB)
The written value is stored in the VRAM Pointer LSB Bits (and maybe also in the corresponding Reload bits ?). And, the VRAM Pointer MSB is now loaded from the corresponding Reload bits (ie. the value from the previous Port 2006h-1st Write is applied now).

Port 2005h-1st Write (Horizontal Scroll Origin, X*1, 0-255)
Port 2005h-2nd Write (Vertical Scroll Origin, Y*1, 0-239)
Port 2000h-Bit0 (Horizontal Name Table Origin, X*256)
Port 2000h-Bit1 (Vertical Name Table Origin, Y*240)
Writing to these registers changes the Reload value bits only, the VRAM Pointer is left unchanged (except for indirect changes at times when the Reload value is loaded into the Pointer during rendering).

Full-screen and Mid-frame Scrolling
Simple full-screen scrolling can be implemented by initializing the Reload value via Ports 2005h and 2000h. Many games change the scroll settings mid-frame to split the screen into a scrolled and non-scrolled area: The Horizontal bits can be changed by re-writing the Reload value via Ports 2005h and 2000h, the vertical bits by re-writing the Pointer value via Port 2006h. Changing both horizontal and vertical bits is possible by mixed writes to Port 2005h and 2006h, for example:

```
[2006h.1st]=(X/256)*4 + (Y/240)*8
[2005h.2nd]=((Y MOD 240) AND C7h)
[2005h.1st]=(X AND 07h)
[2006h.2nd]=(X AND F8h)/8 + ((Y MOD 240) AND 38h)*4
```

Notes: In that example, most bits are updated twice, once via 2006h and once via 2005h, above shows only the relevant bits, the other bits would be don't care (eg. writing unmasked values to 2005h would be faster, and wouldn't change the functionality). The 1st/2nd-write-flipflop is toggled on each of the four writes, so that above does <first> change 2005h-2nd-write, and <then> 2005h-1st-write.

Pointer Increment/Reload during Rendering
During rendering, A4-A0 is incremented per tile, with carry-out to A10, at end of HBlank A4-A0 and A10 are reset to the Reload value. "A14-A12" are used as LSBs of Tile Data address, these bits are incremented per scanline, with carry-out to tile row A9-A5, the tile row wraps from 29 to 0 with carry-out to A11.
Note: Initializing the tile row to 30 or 31 will display garbage tiles (fetched from Attribute table area), in that case the row wraps from 31 to 0, but without carry-out to A11.

PPU Tile Memory

PPU 0000h-0FFFh - Pattern Table 0 (4K) (256 Tiles)
PPU 1000h-1FFFh - Pattern Table 1 (4K) (256 Tiles)

Pattern Table Format
Each pattern table contains 256 tiles. When using both pattern table 0 and 1, up to 512 tiles can be used for Background and Sprites.
Each tile consists of a 8x8 pixel bitmap with 2bit depth (4 colors). Each tile occopies 16 bytes, the first 8 bytes contain color bit 0 for each pixel, the next 8 bytes color bit 1. Each byte defines a row of 8 pixels (MSB left).

Pattern Table Memory
The console does NOT include built-in Pattern Table Memory. Instead, Pattern tables are located in the cartridge, usually in a separate ROM chip, or (less often) in a SRAM chip. Cartridges with more than 8K Pattern memory may contain whatever mapping mechanisms to map the memory into the PPU 8K Pattern Memory area.
There are some special mappers that do automatically change the CHR banks during rendering (allowing to access more than 8K in one frame):
Mapper 5: MMC5 - BANKING, IRQ, SOUND, VIDEO, MULTIPLY, etc.

Mapper 9: MMC2 - PRG/24K/8K, VROM/4K, NT, LATCH
Mapper 10: MMC4 - PRG/16K, VROM/4K, NT, LATCH
Mapper 96: 74161/32 - PRG/32K, CHR/16K/4K, LATCH
Cartridges with CHR-RAM (instead CHR-ROM) usually have 8K RAM, but there also a few with 16K, 32K or 64K RAM:
Mapper 13: CPROM - 16K VRAM
Mapper 96: 74161/32 - PRG/32K, CHR/16K/4K, LATCH
Mapper 168: RacerMate PRG/16K, VRAM/4K, IRQ

PPU Background

PPU 2000h-23FFh - Name Table 0 and Attribute Table 0 (1K)
PPU 2400h-27FFh - Name Table 1 and Attribute Table 1 (1K)
PPU 2800h-2BFFh - Name Table 2 and Attribute Table 2 (1K)
PPU 2C00h-2FFFh - Name Table 3 and Attribute Table 3 (1K)
PPU 3000h-3EFFh - Mirror of 2000h-2EFFh

Name Table Format
Each Name Table occupies 3C0h bytes, containing 8bit tile numbers for 32x30 tiles (256x240 pixels). The tiles are fetched from Pattern Table 0 or 1 (depending on Bit 4 in PPU Control Register 1). Note that NTSC displays may be unable to display the whole 256x240 pixels, basically the relevant portion of screen output should be in the <middle> 32x28 tiles (256x224 pixels) see PPU Dimensions and Timings chapter for more info.

Attribute Table Format
Each Name Table is directly followed by an Attribute Table of 40h bytes, containing 2bit background palette numbers for each 16x16 pixel field. Each byte in the Attribute table defines palette numbers for a 32x32 pixel area:

```
Bit0-1  Palette Number for upperleft 16x16 pixels of the 32x32 area
Bit2-3  Palette Number for upperright 16x16 pixels of the 32x32 area
Bit4-5  Palette Number for lowerleft 16x16 pixels of the 32x32 area
Bit6-7  Palette Number for lowerright 16x16 pixels of the 32x32 area
```

Note: Attributes for each 8x1 pixel row are fetched from cartridge bus. The MMC5 Mapper with EXRAM allows to use different palettes for each 8x8 pixel tile, instead of sharing one palette for above 16x16 areas.

Background Scrolling
Scrolling origin is defined by the Name Table selection in Bit0-1 of $2000, and by offsets in $2005, of which Horizontal offsets may range in 0-255, vertical offsets in 0-239; values above 239 are considered negative (eg. 248 is -8). The picture wraps to the next Name Table when drawing exceeds the boundaries of the current Name Table...

Multiple Name Tables
The NES has the capability of addressing up to four Name Tables (NT0-3), allowing to define backgrounds of up to 512x480 pixels, arranged as such:

```
Square    Horizontal Scroll    Vertical Scroll
NT0 NT1   NT0 left/right NT1   NT0 above/below NT2
NT2 NT3   NT2 left/right NT3   NT1 above/below NT3
```

However, the NES includes only 2K VRAM, so that not more than two Name Tables can be used (unless the cartridge includes external Name Table memory).

Name Table Mapping/Mirroring
The NES outputs the desired Name Table number (NT0-3) to the cartridge, which may then respond by selecting one of the two internal 1K RAM blocks (BLK0-1), or by presenting an external RAM/ROM block (eg. BLK2-3). Examples:

```
_Name Table_____________NT0___NT1___NT2___NT3___Purpose_____________
 Horizontal Mirroring  BLK0  BLK0  BLK1  BLK1  Vertical Scrolling
 Vertical Mirroring    BLK0  BLK1  BLK0  BLK1  Horizontal Scrolling
 Four-screen           BLK0  BLK1  BLK2  BLK3  Four-Way Scrolling
```

When using only the internal blocks, the cartridge may use a simple hardwired connection between two pins to select horizontal or vertical mirroring. Also, the cartridge may contain whatever circuits to map Single-Screen or CHR-ROM to whatever addresses dependently or independently of the selected NT number.

Background Clipping
The PPU allows to mask the left 8 pixels of BG, allowing to use horizontal scrolling with only one Name Table, the 16pix-width palette attribute isn't fully clipped though. Also, BG could be vertically clipped by software, which would require accurate timing though. Aside from that, it'd be no problem to implement four-way scrolling by using only one name table.

PPU Sprites

SPR-RAM 00-FF - Sprite Attributes (256 bytes, for 64 sprites / 4 bytes each)

The PPU supports 64 sprites, which can be either 8x8 or 8x16 pixels in size, only 8 sprites can be displayed per scanline. The sprite Tile bitmaps are kept within the Pattern Table region of VRAM (which is also used for BG Tiles).

Sprite-RAM is built-in in the PPU-chip, and can be accessed via I/O ports only (it is not part of the PPU or CPU memory area). Each four bytes in SPR-RAM define attributes for one sprite, bytes 0-3 for sprite 0, up to bytes FCh-FFh for sprite 63.

SPR-RAM Byte 0 - Y Coordinate Minus 1

```
Vertical Position-1 (FFh,00h..EEh=Scanline 0..239, EFh..FEh=Not displayed)
```

The sprites can be moved bottom-offscreen, but cannot be moved top-offscreen.

SPR-RAM Byte 1 - Tile Number

In 8x8 pixel mode (PPU Control Register 1, Bit5=0):

```
Bit7-0  Specifies 8bit tile number
And,    Pattern Table selected by Bit 3 in PPU Control Register 1
```

In 8x16 pixel mode (PPU Control Register 1, Bit5=1):

```
Bit7-1  Upper 7bit of tile number (N=0-127 uses Tiles N*2 and N*2+1)
Bit0    Pattern Table Address     (0=VRAM 0000h, 1=VRAM 1000h)
```

SPR-RAM Byte 2 - Attributes

```
7    Vertical Flip        (0=Normal, 1=Mirror)
6    Horizontal Flip      (0=Normal, 1=Mirror)
5    Background Priority  (0=Sprite in front of BG, 1=Sprite Behind BG)
4-2  Not used             (Always zero when reading from SPR-RAM)
1-0  Sprite Palette       (0-3=Sprite Palette 0-3)
```

SPR-RAM Byte 3 - X Coordinate

```
Horizontal Position (00h..FFh)
```

Sprites can be moved right-offscreen, and clipping via Port 2001h also allows to move sprites somewhat left-offscreen.

Sprite Priorites

If two or more non-transparent sprite-pixels overlap, then only the sprite with highest priority is processed. If the sprites background priority bit is set to "Behind BG", then it will be hidden behind any non-transparent background pixels.

```
Sprite 0 = highest priority
Sprite 63 = lowest priority
```

Mind that the PPU processes ONLY the sprite with highest priority, eg. if a non-transparent pixel of sprite 5 hides "Behind BG", then sprite 6-63 won't be displayed (even if they are "In Front of BG").

Sprite 0 Hit Flag (Collision Check between BG and Sprite 0)

The Hit Flag, Bit 6 of register 2002h, gets set when the cathode ray beam passes a non-transparent Sprite 0 pixel which is overlapping a non-transparent BG pixel (regardless the sprites BG priority).

The Hit Flag is automatically reset at the end of the VBlank period, it cannot be set or reset by software, that means one can detect only one Hit per frame.

Aside from a normal collision detection, the Hit Flag is also useful to detect when the cathode ray beam has reached a specific screen location, eg. to split the picture into a scrolled and non-scrolled section.

Color 1 or color 2 are (NOT) non-nontransparent (?)

PPU Palettes

PPU 3F00h-3F1Fh - Background and Sprite Palettes

Palette Memory (25 entries used)

```
3F00h        Background Color (Color 0)
3F01h-3F03h  Background Palette 0 (Color 1-3)
3F05h-3F07h  Background Palette 1 (Color 1-3)
3F09h-3F0Bh  Background Palette 2 (Color 1-3)
3F0Dh-3F0Fh  Background Palette 3 (Color 1-3)
3F11h-3F13h  Sprite Palette 0 (Color 1-3)
3F15h-3F17h  Sprite Palette 1 (Color 1-3)
3F19h-3F1Bh  Sprite Palette 2 (Color 1-3)
3F1Dh-3F1Fh  Sprite Palette 3 (Color 1-3)
```

Palette Gaps, Mirrors, and Unused Entries

```
3F04h,3F08h,3F0Ch - Three general purpose 6bit data registers.
3F10h,3F14h,3F18h,3F1Ch - Mirrors of 3F00h,3F04h,3F08h,3F0Ch.
3F20h-3FFFh - Mirrors of 3F00h-3F1Fh.
```

Palette Entries

```
Bit7-6  Not used    (contains garbage when reading palette memory)
Bit5-4  Luminance   (Grayscale) (0-3)
Bit3-0  Chrominance (Color)     (0-F)
```

The Color values are based on the NTSC color-wheel (even on PAL consoles):

```
|__0__|__1___2___3___4___5___6___7___8___9___A___B___C_|_D__|__E___F__|
|White|..Blue..Magenta..Red......Yellow...Green....Blue|Gray|  Black  |
```

Color and Grayscale (0-3) can be combined as such:

```
Luminance__0Xh_______1Xh__________2Xh_________3Xh____________________
Color 0:   Med Gray, Light Gray,  White,      White
Color 1-C: (Dark),   (Normal),    (Brighter), (Brightest/Pastelized)
Color D:   Reserved, Black,       Dark Gray,  Lighter Gray
Color E-F: Black,    Black,       Black,      Black
```

Some black/white colors are duplictated, one should normally use 0Eh or 0Fh as black. Of the two Light Grays, 3Dh is slightly brighter than 10h. The Reserved color is Blacker-than-black, producing a very low voltage, which some monitors may or may not treat to be a sync signal rather than a color.

Monochrome Television Set (nine different grayshades)

On mono TV sets, Colors 0 and D-F can be used for Black/Gray/White colors as usually, and Colors 1-C are all displayed as grayshades: 01h-0Ch=Black, 11h-1Ch=Med/Dark Gray, 21h-2Ch=Med/Light Gray, 31h-3Ch=Bright Gray. Intensity ramp example: 0Eh, 2Dh, 11h, 00h, 21h, 01h, 3Dh, 31h, 30h. Also, the Color Emphasis bits are somewhat affecting the luminance output, even on mono television sets.

Monochrome Bit (three different grayshades)

When Port 2001h/Bit0 is set: The lower 4bits of all palette entries are treated to be zero; this affects both memory reads and video output.
For memory reads, the 6bit values are ANDed with 30h when trying to read palette data via Port 2007h.
For video output, only 3 colors can be displayed: Gray, Light Gray, and White. All other colors cannot be used (even Black and Dark Gray are disabled).
The Color Emphasis bits can be still used (eg. to change above 3 gray-shades into 3 pastelized green-shades).

Color Emphasis Bits

Port 2001h/Bit7-5 allow to adjust the palette, eg. with setting 001b the whole picture becomes more green.

```
000b  Normal
001b  Green
010b  Brown
100b  Blue
```

To play by the rules, one should reportedly not set more than one of the emphasis bits at once, setting two or more bits may shortcut something inside the PPU, though it doesn't seem to damage the chip.

Border Color and Clipping Colors

The screen border is black, regardless of any palette settings.
The Color 0 setting is displayed when the BG is disabled, when the whole display is disabled, and when the left BG row is clipped.

RGB-Palettes (VS System, Play Choice 10, Famicom Titler)

These systems are using PPUs with RGB output. The PPUs are having a MUCH more colorful palette than the pastelized NES palette; this "feature" may be seen as a hardware glitch, or as an improvement.
Some of the VS System's PPUs are additionally having "scrambled" colors (rearranged color numbers, so games will have wrong colors when not buying the correct PPU; intended to prevent piracy and unlicensed games). For details, see
VS System

PPU Cartridge Bus Note

When software accesses palette RAM via Port 2006h/2007h, the palette address (PPU 3Fxxh) is output to the PPU address bus (usually a mirror of PPU 2Fxxh), but the /WR signal isn't activated on palette writes (so VRAM remains unchanged), however, the /RD signal is activated on palette reads, and the addressed VRAM data is latched (ie. if the [2006h] gets changed to a non-palette address, then the latched value will be returned on the next read from 2007h).

PPU Dimensions & Timings

Below are timings for Nintendo's NTSC and PAL consoles, and for the Dendy (a russian Famicom clone with PAL output and Famicom-like NTSC-style timings).

Clock Speeds

```
Type               NTSC               PAL                  Dendy
Master Clock (X1)  21.47727MHz        26.6017125MHz        Like PAL
Color Clock        3.579545MHz=X1/6   4.43361875MHz=X1/6   Like PAL
Dot Clock          5.3693175MHz=X1/4  5.3203425MHz=X1/5    Like PAL
CPU Clock          1.7897725MHz=X1/12 1.66260703MHz=X1/16  1.773448MHz=X1/15
Frame Rate         60.09914261Hz      50.00697891Hz        Like PAL
```

Note: Above NTSC values are rounded. The exact NTSC Color Clock should be 315/88MHz.

Vertical Timings (in scanlines)

```
NTSC PAL  Dendy
?    1    ?          Pre-Render Time
240? 239  240?       Rendering Time
1    1    51?        Post-Render Time
20   70   20?        Vblank
1    1    1?         Post-Blank Time
---- ---- ----       -------
262  312  312?       Total number of Scanlines
```

Horizontal Timings (in dots)

```
NTSC PAL  Dendy
```

```
25x  252  25x         Rendering Time
xx   89   xx          Hblank
---- ---- ----        -------
341  341  341?        Total number of dots per scanline
```
In each second frame, NTSC does output a short scanline with only 340 dots, this happens only if BG and/or OBJ are enabled.

CPU vs PPU Timings

Type	NTSC	PAL	Dendy
Dots per CPU Clk	3.0 (12/4)	3.2 (16/5)	3.0 (15/5)
CPU Clks per Scanline	113.6666	106.5625	
CPU Clks per One Frame	29780.66	33247.5	
CPU Clks per Other Frame	29780.33	33247.5	
CPU Clks per Two Frames	59561.0	66495.0	

Modulator vs PPU Timings

Type	NTSC	PAL	Dendy
Dots per Color Clk	1.5 (6/4)	1.2 (6/5)	1.2 (6/5)
Color Clks per Scanline	227.3333	284.1666	
Color Clks per One Frame	59561.33	88660.0	
Color Clks per Other Frame	59560.66	88660.0	
Color Clks per Two Frames	119122.0	177320.0	

Visible Screen Resolution

The logical screen resolution processed by the PPU is 256x240 pixels. However the visible screen resolution is somewhat smaller, due to improper blanking periods, and eventually due to exceeding the physical dimensions of (NTSC) displays.
On PAL hardware, the upper 1 scanline, the left 2 pixels, and the right 2 pixels are invisible (displayed as black border). On NTSC hardware, the upper 8 scanlines, and the lower 8 scanlines are reportedly often invisible (224 lines visible), eventually some NTSC screens are hiding only the upper 3 scanlines (237 lines visible).
To be compatible with all types of displays, output valid 256x240 pixels, but have the relevant information in the <middle> 240x224 pixels (30x28 tiles) only.

Synchronizing Software with the Cathode Ray Beam

The PPU sets the VBlank flag in PPU Status Register (and optionally produces an NMI) once per frame. It doesn't support scanline interrupts, or a current scanline number register, which is making it a bit difficult to access PPU registers at specific Cathode Ray Beam locations (for example, to split the the screen into two sections with different colors or scroll offsets). However, a couple of methods could be used for that purpose:

```
1) Delay Loops synchronized with NMI (badly wasting CPU time) or using
   meaningful code with fixed non-conditional execution time instead delays.
2) Producing a "Sprite 0 Hit", or a "More Than 8 Sprites Per Scanline"
   situation at specific screen location (which sets corresponding flag in
   PPU Status Register, one cannot reset the flag manually, so either works
   only once per frame)
3) Using PCM Sound IRQs as Timer (synchronized with NMI)
4) Using external Timers (contained in some Cartridge Mappers)
```

The APU also contains a so-called "Frame" counter - that counter is NOT physically synchronized with the PPU, and it doesn't even match the exact number of clock cycles per frame. There's a limited chance that one could program it to produce an IRQ at a specific screen location (and to resynchronize it for the next frame).

More detailed Timing Info

PPU 2C02 Timings

PAL Vblank Start Timing

```
Line   <--Line238--><--Line239--><--Line240--><--Line241--><--Line242-->
/NMI   ---------------------------------------_____________________________
Video: PPPPPPPPP-_C-PPPPPPPPP-_C-----------_C-----------_C-----------_C-
```

PAL Vblank Middle Timing

```
Line   <--Line268--><--Line269--><--Line270--><--Line271--><--Line272-->
/NMI   ____________________________________________________________________
Video: ----------_C-----------____________-____________-____________-_C-
```

PAL Vblank End Timing

```
Line   <--Line310--><--Line311--><--Line000--><--Line001--><--Line002-->
/NMI   _____________-----------------------------------------------------
Video: ----------_C-----------_C-----------_C-PPPPPPPPP-_C-PPPPPPPPP-_C-
```

Whereas

For Video: P=Picture, -=Blank, _=Sync, C=ColorBurst
/NMI is toggled on & off at the begin of the line (after the ending "-_C-" blanking part of the previous line).
VSYNC is toggled at HSYNC, while active, it changes the "-_C-" part to "-___", and also changes the next lines "picture" part from "---" to "___".

PPU 2C02 Timings

PPU base timing - NTSC

the 21.48 MHz signal is divided by 4 to get 5.37 MHz, and is used as the smallest unit of timing in the PPU. All following references to PPU clock cycle (abbr. "cc") timing in this document will be in respect to this timing base, unless otherwise indicated.

```
  - Pixels are rendered at the same rate as the base PPU clock.
    In other words, 1 clock cycle= 1 pixel.
  - One frame consists of 262 scanlines.
    This equals 341*262 PPU cc's per frame (divide by 3 for # of CPU cc's).
  - 341 PPU cc's make up the time of a typical scanline (or 341/3 CPU cc's).
```

All PPU memory access cycles are 2 clocks long, and can be made back-to-back (typically done during rendering). Here's how the access breaks down:

At the beginning of the access cycle, PPU address lines 8..13 are updated with the target address. This data remains here until the next time an access cycle occurs.

Miscellanious PPU info

```
  - Reading from $2002 clears the vblank flag (bit 7), and resets the
    internal $2005/6 flip-flop. Writes here have no effect.
  - $2002.5 and $2002.6 after being set, stay that way for the first 20
    scanlines of the new frame, relative to the VINT.
  - Pin /VBL on the 2C02 is the logical NAND between 2002.7 and 2000.7.
```

Frame rendering details

The following describes the PPU's status during all 262 scanlines of a frame. Any scanlines where work is done (like image rendering), consists of the steps which will be described in the next section.

0..19: Starting at the instant the VINT flag is pulled down (when a NMI is generated), 20 scanlines make up the period of time on the PPU which I like to call the VINT period. During this time, the PPU makes no access to it's external memory (i.e. name / pattern tables, etc.).

20: After 20 scanlines worth of time go by (since the VINT flag was set), the PPU starts to render scanlines. This first scanline is a dummy one; although it will access it's external memory in the same sequence it would for drawing a valid scanline, no on-screen pixels are rendered during this time, making the fetched background data immaterial. Both horizontal *and* vertical scroll counters are updated (presumably) at cc offset 256 in this scanline. Other than that, the operation of this scanline is identical to any other. The primary reason this scanline exists is to start the object render pipeline, since it takes 256 cc's worth of time to determine which objects are in range or not for any particular scanline.

21..260: after rendering 1 dummy scanline, the PPU starts to render the actual data to be displayed on the screen. This is done for 240 scanlines, of course.

261: after the very last rendered scanline finishes, the PPU does nothing for 1 scanline (i.e. the programmer gets screwed out of perfectly good VINT time). When this scanline finishes, the VINT flag is set, and the process of drawing lines starts all over again.

Scanline rendering details

As explained before, external PPU memory can be accessed every 2 cc's. With 341 cc's per scanline, this gives the PPU enough time to make 170 memory accesses per scanline (and it uses all of them!). After the 170th fetch, the PPU does nothing for 1 clock cycle. Remember that a single pixel is rendered every clock cycle.

Note that the PPU fetches an attribute table byte for every 8 sequential horizontal pixels it draws. This essentially limits the PPU's color area (the area of pixels which are forced to use the same 3-color palette) to only 8 horizontally sequential pixels.

Memory fetch phase 1 thru 128 - BG Fetch

Fetches 4x32 bytes; one Name Table entry, one Attribute Table entry, and two Pattern Table bytes; for 3rd..34th tile in scanline (33th tile may be parts visible if BG scrolled, 34th is never visible). From the time of the Name Table fetch, it takes (16-n) clock cycles until the first pixel of the fetched tile is processed (drawn on screen, unless being covered by a Sprite-pixel, and eventually setting the Hit Flag) (n=0..7, horizontal scroll offset).
Simultaneously with above BG fetch, the PPU pre-processes SPR-RAM for the NEXT scanline by searching for Sprite Y-cooridnates that are visible in that scanline, only the first eight matches will be recursed, if the search finds additional matching entries then bit5 of $2002 will get set to indicate that one or more entries have been ignored.

Memory fetch phase 129 thru 160 - Sprite Fetch

Fetches 4x8 bytes; two dummy Name Table entris, and two Pattern Table bytes; for 1st..8th sprite in NEXT scanline (fetches dummy patterns if the scanline contains less than 8 sprites).

Memory fetch phase 161 thru 168 - BG Fetch

Fetches 4x2 bytes; one Name Table entry, one Attribute Table entry, and two Pattern Table bytes; for 1st..2nd tile in NEXT scanline.

Memory fetch phase 169 thru 170 (and a half) - Padding

Fetches 2 bytes; two dummy reads from the Name Table address of the 3rd tile in next scanline. After that fetches, the PPU rests for one "dead" cycle here (or the equivelant of 1/2 memory access cycle) before repeating the whole pixel/scanline rendering process. Scanline 20 is the only scanline that has variable length, on every odd frame, this scanline is only 340 cycles (the dead cycle at the end is removed). This is done to cause a shift in the NTSC colorburst phase.

3D Glasses

Famicom 3D System (Japan) (LCD shutter glasses)
Japanese Famicom games are using fairly expensive LCD shutter glasses, controlled via the 15pin Controller Expansion Port. Left & Right images are transferred in separate frames (and the left/right shutters are opened/closed accordingly).

```
Write to [4016h].Bit1   (0=Shut Which? Eye, 1=Shut WhichOther? Eye)
```

The advantage is that the 3D picture is fully colored. The downside is that the hardware is expensive, and must be purchased separately. Another restriction is that it requires the old 15pin Famicom connector (or an adapter for newer Famicom and NES 7pin controller ports).

Simple 3D Glasses (US and Europe) (red/cyan glasses)
US and European NES games are using much cheaper "passive" 3D glasses with colored "lens" instead of LCD shutters. Theoretically, this would allow to transfer the left & right picture in one frame, but that'd require more CPU load and color depth, so, in practice the games are drawing the left/right frames separately (similar as in the LCD shutter method).

```
Right Eye -- Red
Left Eye  -- Blue/cyan or so
```

The advantage is that the hardware is so cheap that it can have been included with the game for free. Cheapest variant would be "cardboard" glasses, though there seems to have been also a yellow stylish plastic variant.
The downside of mis-using colors for 3D effects is that the picture will appear more or less monochrome.

Pulfrich Effect 3D Glasses (US) (dark/clear glasses)
Another approach is using the Pulfrich effect with dark/clear glasses, this is some sort psychophysical stuff, related to different signal timings for bright/dark colors.

```
Right Eye -- Dark        ;\as so for NES Orb-3D
Left Eye  -- Clear       ;/(that is, opposite as for SNES Jim Power)
```

The advantage is that it's cheap, and that it can be even used for colored images (unlike the red/cyan glasses method). The disadvantage is that it does work only with permanently moving objects.

Games with support for 3D Glasses

```
Attack Animal Gakuen (J) (Pony Canyon)
Cosmic Epsilon (J) (Asmik)
Falsion (Konami)
Famicom Grand Prix: 3D Hot Rally (Nintendo)
Highway Star (J) aka Rad Racer (U) (E) (Square)
Tobidase Daisakusen (J) aka 3-D World Runner (U) (Square)
JJ Tobidase Daisakusen Part II (J) (Square)
Orb-3D (U) (Pulfrich Effect)
```

In most of that games, 3D mode is activated by pressing SELECT button. Game versions that were released outside of Japan are customized to work with the red/cyan glasses (or dark/clear glasses for Orb-3D).

Audio Processing Unit (APU)

Based on "2A03 technical reference by Brad Taylor" (1st release, April 2004).

APU Channel 1-4 Register 0 (Volume/Decay)

4000h - APU Volume/Decay Channel 1 (Rectangle)
4004h - APU Volume/Decay Channel 2 (Rectangle)
400Ch - APU Volume/Decay Channel 4 (Noise)

```
0-3    Volume / Envelope decay rate
        When Bit4=1: Volume (0=Silent/None..F=Loud/Max)
        When Bit4=0: Envelope decay rate, NTSC=240Hz/(N+1), PAL=192Hz/(N+1)
4      Envelope decay disable (0=Envelope/Decay, 1=Fixed Volume)
5      Length counter clock disable / Envelope decay looping enable
        When Bit4=1: length counter clock disable
        When Bit4=0: envelope decay looping enable
         0: Disable Looping, stay at 0 on end of decay [ \_____ ]
         1: Enable Looping, restart decay at F         [ \\\\\\ ]
         (Does this still affect Length counter clock disable ?)
6-7    Duty cycle type (unused on noise channel)
```

```
0  [--______________] 12.5%   Whereas,
1  [----____________] 25.0%   [_] = LOW  (zero) (0)
2  [--------________] 50.0%   [-] = HIGH (volume/decay) (0..F)
3  [------------____] 75.0%   Noise randomly outputs LOW or HIGH
```

The Duty Cycle counter is reset when the length counter of the same channel is written to (via 4003h/4007h).

Initial Decay Volume:
Only a write out to 4003h/4007h/400Fh will reset the current envelope decay counter to a known state (to 0Fh=max) for the appropriate channel's envelope decay hardware. Otherwise, the envelope decay counter is always counting down (by 1) at the frequency currently contained in the volume / envelope decay rate bits (even when envelope decays are disabled by setting bit 4), except when the envelope decay counter contains a value of 0, and envelope decay looping (bit 5) is disabled (0).

4008h - APU Linear Counter Channel 3 (Triangle)

```
0-6     linear counter load register
7       length counter clock disable / linear counter start
```

Linear counter incremented-or-decremented? at NTSC=240Hz, PAL=192Hz, same purpose, but higher resolution than length counter, so Triangle channel is ALWAYS using either linear-counter or length-counter, and cannot be used with both counters disabled?

The Triangle channel does not have a variable volume, nor variable duty cycle. Instead, it produces the following fixed 32-step output level stream:

```
0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F,F,E,D,C,B,A,9,8,7,6,5,4,3,2,1,0
```

On 2A03 reset, the stream starts at 0. The stream will be halted (at the current position) whenever the Triangle channel's length or linear counter contains a count of 0, and will continue at whatever old position when it is restarted.

APU Channel 1-4 Register 1 (Sweep)

4001h - APU Sweep Channel 1 (Rectangle)
4005h - APU Sweep Channel 2 (Rectangle)

```
0-2     Sweep right shift amount (S=0..7)
3       Sweep Direction          (0=[+]Increase, 1=[-]Decrease)
4-6     Sweep update rate        (N=0..7), NTSC=120Hz/(N+1), PAL=96Hz/(N+1)
7       Sweep enable             (0=Disable, 1=Enable)
```

At specified Update Rate, the 11bit Wavelength will be modified as such:

```
Wavelength = Wavelength +/- (Wavelength SHR S)
(For Channel 1 Decrease only: minus an additional 1)
(Ie. in Decrease mode: Channel 1 uses NOT, Channel 2 uses NEG)
```

Wavelength register will be updated only if all 3 of these conditions are met:

```
Bit 7 is set (sweeping enabled)
The shift value (which is S in the formula) does not equal to 0
The channel's length counter contains a non-zero value
```

Sweep end: The channel gets silenced, and sweep clock is halted, when:

```
1) current 11bit wavelength value is less than 008h
2) new 11bit wavelength would become greater than 7FFh
```

Note that these conditions pertain regardless of any sweep refresh rate values, or if sweeping is enabled/disabled (via Bit7).

4009h - APU N/A Channel 3 (Triangle)
400Dh - APU N/A Channel 4 (Noise)

```
0-7     Unused (No Sweep support for these channels)
```

APU Channel 1-4 Register 2 (Frequency)

4002h - APU Frequency Channel 1 (Rectangle)
4006h - APU Frequency Channel 2 (Rectangle)
400Ah - APU Frequency Channel 3 (Triangle)

```
0-7     Lower 8 bits of wavelength (upper 3 bits in Register 3)
```

F = CPUCLK/(N+1)/16 for Rectangle channels
F = CPUCLK/(N+1)/32 for Triangle channel
CPUCLK is 1.789773MHz (NTSC), or 1.662607MHz (PAL).

400Eh - APU Frequency Channel 4 (Noise)

```
0-3     Noise frequency, F=1.79MHz/2/(N+1)
         Value 0..F corresponds to following 11bit clock cycle value:
         N=002,004,008,010,020,030,040,050,065,07F,0BE,0FE,17D,1FC,3F9,7F2
4-6     Unused
7       Random number type generation (0=32767 bits, 1=93 bits)
```

XXX conflicting info suggests these frequency values:

```
NTSC: 004,008,010,020,040,060,080,0A0,0CA,0FE,17C,1FC,2FA,3F8,7F2,FE4
PAL:  004,008,00E,01E,03C,058,076,094,0BC,0EC,162,1D8,2C4,3B0,762,EC2
(XXX but, this won't match the "/2/(N+1)" part of the above formula)
```

The random number generator consists of a 15bit shift register. The MSB (Bit14) is output/inverted (1=Low/Zero, 0=High/Decay/Volume). At the specified frequency, Bit14 is XORed with Bit13 (32767-bit mode) or with Bit8 (93-bit mode), the register is then shifted to the left, with the result of the XOR

operation shifted-in to Bit0.

On 2A03 reset, this shift register is loaded with a value of 1.
Not sure if it is reset when switching from 32767-bit mode to 93-bit mode? If it isn't reset then 93-bit mode will act unstable: produce different 93-bit patterns, or even a 31-bit pattern, depending on old shift register content.

APU Channel 1-4 Register 3 (Length)

4003h - APU Length Channel 1 (Rectangle)
4007h - APU Length Channel 2 (Rectangle)
400Bh - APU Length Channel 3 (Triangle)
400Fh - APU Length Channel 4 (Noise)
Writing to the length registers restarts the length (obviously), and also restarts the duty cycle (channel 1,2 only), and restarts the decay volume (channel 1,2,4 only).

```
0-2   Upper 3 bits of wavelength (unused on noise channel)
3-7   Length counter load register (5bit value, see below)
```

The above 5bit value is translated to the actual 7bit counter value as such:

```
Bit3=0 and Bit7=0 (Dividers matched for use with PAL/50Hz)
  Bit6-4  (0..7 = 05h,0Ah,14h,28h,50h,1Eh,07h,0Dh)
Bit3=0 and Bit7=1 (Dividers matched for use with NTSC/60Hz)
  Bit6-4  (0..7 = 06h,0Ch,18h,30h,60h,24h,08h,10h)
Bit3=1 (General Fixed Dividers)
  Bit7-4  (0..F = 7Fh,01h..0Fh)
```

The 7bit counter value is decremented once per frame (PAL=48Hz, or NTSC=60Hz) the counter and sound output are stopped when reaching a value of zero. The counter can be paused (and restarted at current location) by Length Counter Clock Disabled bit in Register 0.

APU Channel 5 - DMC Sound

4010h - DMC Play mode and DMA frequency

```
7    IRQ Enable, when Length=0 AND Loop=Disabled  (0=Disable, 1=Enable)
      DMC IRQs can be acknowledged by writing 0 to Bit7 of 4010h, or
      by writing any value to 4015h
6    Loop when reaching Length=0                  (0=Stop, 1=Loop)
      In looped mode, the sample block is restarted by reloading the
      DMA Start Address and Length values, IRQs are not generated.
5-4  Appear to be unused
3-0  DMC frequency (00h..0Fh, see below)
```

For frequency 00h..0Fh, the number of cycles/samplebyte are:

```
NTSC: D60,BE0,AA0,A00,8F0,7F0,710,6B0,5F0,500,470,400,350,2A8,240,1B0
PAL:  XXX
```

For frequency 00h..0Fh, the number of cycles/samplebit are:

```
NTSC: 1AC,17C,154,140,11E,0FE,0E2,0D6,0BE,0A0,08E,080,06A,054,048,036
PAL:  18E,162,13C,12A,114,0EC,0D2,0C6,0B0,094,084,076,062,04E,042,032
```

4011h - DMC Delta counter load register

```
7       Appears to be unused
6-1     MSBs of 7bit DAC   (6bit "Delta Counter")
0       LSB of 7bit DAC
```

Used to initialize the Delta Counter (for DMC usage), or to output 7bit data directly (for PCM usage). Another use of this register has been to somewhat control the volume of the Triangle & Noise sound channel outputs. Please see NESSOUND.TXT for more information.

4012h - DMC address load register
Specifies the DMA Start Address. The Start Address is loaded to the actual DMA pointer, when the DMC is activated from an inactive state, or when restarting looped playback.

```
7-0  DMA Start Address for DMC   (Address = C000h+N*40h)
```

The DMA pointer is 15 bits in size, and wraps from FFFFh to 8000h (not C000h).

4013h - DMC length register

```
7-0  DMA Length DMC   (Length = N*10h+1 Bytes = N*80h+8 Bits)
```

When it arrives at 0, the DMC will take action(s) based on the 2 MSB of $4010. This counter will be loaded with the current calculated address value of $4013 when the DMC is activated from an inactive state.

Usage as Delta Modulation Channel (DMC) with Direct Memory Access (DMA)
This method uses 1-bit samples, processed at the specified sample-bit-rate,

```
1 = Increment Delta counter by 1  (unless result would be greater than 3Fh)
0 = Decrement Delta counter by 1  (unless result would be less than 0)
```

Every eight sample-bits, the DMC will halt the CPU for 2 clock cycles to retrieve the next sample-byte per DMA, each sample byte is processed bit-by-bit (LSB first).

Usage as Pulse Code Modulation (PCM) Channel

Alternately, 7bit samples can be written directly to the DAC.
Advantages are that all 7bit can be used (instead only the upper six Delta bits), and that the DAC can be directly changed from one value to any other value (which would take up to 64 increment/decrement steps in DMC mode).
Disadvantages are that it requires exact software timings and more CPU load than the DMA method.
No idea if it is required to "enable" the DMC channel (eg. by outputting a looped dummy 55h sample byte) in order to use PCM ?

On 2A03 reset, the DMC's IRQ flag is cleared (disabled), and the [DMC] channel is disabled. On 2A03 reset, all 7 used bits of $4011 are reset to 0.

Caution
Using DMC-DMA can conflict with other I/O ports (such like VRAM or joypad reads)! For details, see:
APU DMC-DMA Glitch

APU Control and Status Registers

4015h - DMC/IRQ/length counter status/channel enable register

```
0       Status/Enable rectangle wave channel 1
1       Status/Enable rectangle wave channel 2
2       Status/Enable triangle wave channel 3
3       Status/Enable noise channel 4
4       Status/Enable DMC channel 5
5       Not used (returns garbage on reading)
6       Frame IRQ status (active when set)
7       DMC's IRQ status (active when set)
```

Reading Bit4-0 returns 0 for a zero count status in the length counter (channel's sound is disabled), and 1 for a non-zero status. Reading Bit7-6 returns IRQ status flags.
Writing to Bit4-0: Writing 0 forces to disable the channel, it will get stopped and become silent (as if its length counter has reached 0), writing 1 de-activates the forced-stop (without changing the length counter, playback continues at whatever value have been in the length counter).
Writing to Bit7-5: Unknown.
DMC IRQs are acknowledged by WRITING any value to 4015h.
Frame IRQs are acknowledged by READING from 4015h.

Note that all 5 writable bits in 4015h will be set to 0 upon 2A03 reset.

4017h - APU Low frequency timer control (W)
Any write to 4017h resets both the frame counter, and the clock divider.
Sometimes, games will write to this register in order to synchronize the sound hardware's internal timing, to the sound routine's timing (usually tied into the NMI code). The frame IRQ frequency is slightly smaller than the PPU's vertical retrace frequency, so you can see why games would desire this synchronization.

```
bit6:  Frame IRQ Disable  (0=Enable Frame IRQ, 1=Disable Frame IRQ)
bit7:  Frame Rate Select  (0=NTSC=60Hz=240Hz/4, 1=PAL=48Hz=240Hz/5)
  ...XXX... Frame IRQ works ONLY if bit6 and bit7 are BOTH zero!
```

On 2A03 reset, Bit6-7 will be cleared. That means that Frame IRQs are enabled by default (though usually not executed since the CPUs IRQ-Disable-Flag is set on reset).
Frame IRQs are acknowledged by reading from 4015h.

Frame Counter
Several audio timings are referred to as "Frames" and "PAL" and "NTSC",

```
! These timings are NOT physically related to actual PPU VBlank/NMI timings !
```

The Audio "Frame" counter can be switched into "PAL" or "NTSC" mode by software - regardless of whether the game does run on a PAL/NTSC console. Audio-frames may be (more or less) synchronized with video-frames by chosing PAL/NTSC audio-mode matching to PAL/NTSC console type respectively.
The frame counter is based on a 240Hz signal which is gained from dividing the 1.78Mhz PHI2 clock edges (2*1.78M edges/second) by 14915. In PAL Mode (4017h Bit7=1), the 240Hz signal is divided by 1.25 (by simply leaving out each fifth clock pulse), resulting in a somewhat dirty 192Hz signal. This PAL/NTSC adjustment mechanism counts through 4 or 5 steps, producing output as such:

```
0/NTSC: 4,0,1,2,3,0,1,2,3,0,1,2,3  |  1/PAL: 0,1,2,3,4,0,1,2,3,4,0,1,2,3,4
240Hz:  __-_-_-_-_-_-_-_-_-_-_-_-  |  192Hz: -_-_-_-___-_-_-_-___-_-_-_-__
120Hz:  ____-___-___-___-___-___-  |  96Hz:  __-___-_____-___-_____-___-__
60Hz:   (above somehow div by 2)   |  48Hz:  (above somehow divided by 2)
```

Frame Counter is reset on writing to 4017h, and does then restart sequences as shown above (in NTSC mode starting with a skipped step, whilst directly starting with a non-skipped step in PAL mode).
Linear counter (triangle) and envelope decay counters (rectangle/noise) are clocked by 240Hz/192Hz. Frequency sweep (rectangle) clocked by 120Hz/96Hz. Length counters (all channels) and Frame IRQ clocked by 60Hz/48Hz.

4014h - SPR-RAM DMA
Sprite RAM DMA Function contained in 2A03 chip. See PPU description.

4016h - Write
Three bit general purpose output latch contained in 2A03 chip.
Used to strobe joysticks. See Controllers chapter for more info.

4016h/4017h - Read
The 2A03 chip does not actually contain read-able registers at these addresses, however, it does output read-request signals for these addresses, which are used to activate on-board joystick inputs.
See Controllers chapter for more info.

Note: 4015h is the only R/W register in the 4000h-4017h area, all other registers in this area are write-only, and do not respond to read cycles (except for the external read-able 4016h/4017h registers).

APU 4-bit DAC

Channel 1-4 are (each) using a standard 4-bit DAC with 16 steps of output voltage resolution. On the 2A03, rectangle wave 1 & 2 are mixed together, and are available via pin 1. Triangle, noise, and DMC are available on pin 2.
Signals are then merged via 20KOhm (pin 1) and 12KOhm (pin2), respectively, rectangle channels have different output levels than equivalent volume settings on triangle/noise channels?

The output waveforms have some linear asymmetry (the desired output voltage would increase on a linear scale, the actual outputted voltage increases less and less each step).

The side effect of this is that the DMC's 7-bit DAC port ($4011) is able to indirectly control the volume (somewhat) of both triangle & noise channels. When $4011=0, triangle & noise volume outputs are at maximum. When $4011=7F, triangle & noise channel outputs operate at only 57% total volume. A few games actually take advantage of this "volume" feature, and write values to $4011 in order to regulate the amplitude of the triangle wave channel's output.

Forced Zero Volume:
When hardware in the channel wants to disable it's sound output (like the length counter, or sweep unit).

APU Various

After 2A03 reset, the sound channels are unavailable for playback during the first 2048 CPU clocks.

The rectangle channel(s) frequency in the range of 54.6 Hz to 12.4 KHz.
The triangle wave channel range of 27.3 Hz to 55.9 KHz.
The random wavelength channel range anywhere from 29.3 Hz to 447 KHz.

RP2A03E quirk
I have been informed that revisions of the 2A03 before "F" actually lacked support for the 93-bit looped noise playback mode. While the Famicom's 2A03 went through 4 revisions (E..H), I think that only one was ever used for the front loading NES: "G". Other differences between 2A03 revisions are unknown. Is that Quirk True and Confirmed ?

APU DMC-DMA Glitch

DMC-DMA can conflict with other I/O ports. If the DMA read occurs at the same time as a CPU read, then the CPU read is executed TWICE. In case of CPU reads from read-sensitve I/O ports this can result in "lost" bits and bytes.

```
2002h  ppu status     ;-vblank flag acknowledged (lost flag)
2007h  vram data read ;-vram address incremented twice (one byte skipped)
4016h  joypad1 read   ;\two CLKs sent to joypad, causing one shift-register
4017h  joypad2 read   ;/step to be skipped
xxxxh  other external read-sensitive mapper registers
```

The bug occurs only on CPU reads (not on CPU writes). And, it occurs only on NTSC consoles (with RP2A03xx APU/CPU), not on PAL consoles (with RP2A07xx APU/CPU).

Workarounds
In case of joypad & vram reads, the workaround would be to read the data repeatedly, until receiving twice the same values. This should also work for most other controllers - unless the controller transfers the data only ONCE, or unless it does internally reset the data after reading.
For ppu status read, it is generally better to set a nmi_flag in the NMI handler, and to poll that flag in the main program (rather than polling 2002h directly, which is unstable, even without the DMC glitch).
And, of course, one could disable DMC. Or, sense the collision time (eg. by examining VRAM reads), and then place critical reads into gaps where DMC isn't occuring).

APU External Sound Channels

The Famicom 60-pin cartridge slot includes a SND_IN pin, allowing external sound controllers (as included in some Cartridge Mappers, and in Famicom Disk System) to produce additional sound channels which are merged with the normal APU channels:

Mapper 5: MMC5 - BANKING, IRQ, SOUND, VIDEO, MULTIPLY, etc.
Mapper 19: Namcot 106 - PRG/8K, VROM/1K/VRAM, IRQ, SOUND
Mapper 20: Disk System - PRG RAM, BIOS, DISK, IRQ, SOUND
Mapper 24: Konami VRC6A - PRG/16K/8K, VROM/1K, NT, IRQ, SOUND
Mapper 26: Konami VRC6B - PRG/16K/8K, VROM/1K, NT, IRQ, SOUND
Mapper 85: Konami VRC7A/B - PRG/16K/8K, VROM/1K, NT, IRQ, SOUND
Mapper 188: UNROM-reversed

However, the NES 72-pin cartridge slot DOES NOT include a SND_IN pin, even though it does have more (more or less unused) pins than Famicom.

And, there are a few devices with external speakers (not routed through NES SND_IN) - a "beep" function Power Glove, and fully featured synthesizer in the Miracle:
Controllers - Piano Keyboards
Controllers - Power Glove

And, the FamicomBox contains a beep function (when money is inserted); the beep sound is merged with the APU sound, and then passed to TV-Set.
FamicomBox

Controllers

Controller Interface
Controllers - I/O Ports
Controllers - Pin-Outs
Controllers - Summary of Controller Types
Controllers - Summary of Controller Signals
Controllers - Detection

Controllers
Controllers - Joypads
Controllers - Four-Player Adaptors
Controllers - Lightguns (Zapper)
Controllers - Paddles
Controllers - Push Buttons
Controllers - Typewriter Keyboards
Controllers - Piano Keyboards
Controllers - Keypads
Controllers - Mats
Controllers - Inflatable Controllers
Controllers - RacerMate Bicycle Training System
Controllers - Tablets
Controllers - Trackball and Mouse
Controllers - Power Glove
Controllers - UForce
Controllers - Barcode Readers
Controllers - Pachinko
Controllers - Microphones
Controllers - Reset Button
Controllers - Arcade Machines

Other Devices that connect to Controller I/O Ports
3D Glasses
Storage Data Recorder
Storage Turbo File
Storage Battle Box
Hori Game Repeater
Headphones

Another special device (not directly connected to Controller Ports)
R.O.B. (Robotic Operating Buddy)

Controllers - I/O Ports

Port 4016h-4017h control 3 general purpose outputs (OUT0/1/2), plus 2 clock outputs (PORT0/1-CLK), and several inputs (number of inputs varies depending on the console type), all inputs are inverted inside of the console, ie. LOW arrives as "1" at the CPU.

4016h - Joypad Output Register (W)

2-0	OUT2-0	Expansion Port Outputs
0	OUT0	NES/Famicom: Joypad 1+2 Strobe (for BOTH joypads)

4016h - Joypad Input Register 0 (R)

Bit	Name	NES	Famicom	Purpose
7-5	N/A	Not used (undefined)	Not used (undefined)	-
4	PORT0-4	Expansion/Gameport	Not used (undefined)	Zapper 1 Button
3	PORT0-3	Expansion/Gameport	Not used (undefined)	Zapper 1 Light
2	PORT0-2	Expansion	Microphone Input	Microphone
1	PORT0-1	Expansion	Expansion	Exp.
0	PORT0-0	Expansion/Gameport	Joypad	Joypad 1

Reading from 4016h generates a PORT0-CLK signal (used to clock joypads).

4017h - Joypad Input Register 1 (R)

Bit	Name	NES	Famicom	Purpose
7-5	N/A	Not used (undefined)	Not used (undefined)	-
4	PORT1-4	Expansion/Gameport	Expansion	Zapper 2 Button
3	PORT1-3	Expansion/Gameport	Expansion	Zapper 2 Light
2	PORT1-2	Expansion	Expansion	Exp.
1	PORT1-1	Expansion	Expansion	Exp.
0	PORT1-0	Expansion/Gameport	Expansion/Joypad	Joypad 2

Reading from 4017h generates a PORT1-CLK signal (used to clock joypads).

4017h - APU Low frequency timer control (W)
Not joypad/expansion related. See APU chapter for more info.

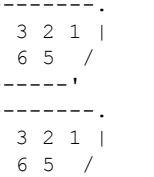
For info about the "Not used (undefined)" bits, see:
Unpredictable Things

Caution
The APU (if DMC sound is used) can conflict with joypad reads! For details, see:
APU DMC-DMA Glitch

Controllers - Pin-Outs

Controller ports - NES (and newer Famicom models) - male, front side

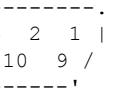
Pin	Dir	Player 1	Player 2	Expl./Usage
1	Out	GND	GND	Ground
2	Out	PORT0-CLK	PORT1-CLK	Joystick Clock (CPU Port Read)
3	Out	OUT0	OUT0	Joystick Serial-Start (Strobe)
4	In	PORT0-0	PORT1-0	Joystick Serial-Data
5	Out	+5VDC	+5VDC	Supply
6	In	PORT0-3	PORT1-3	Zapper Light, Paddle Button
7	In	PORT0-4	PORT1-4	Zapper Button, Paddle Position



All controller inputs are inverted inside of the console, LOW arrives as "1".
Note: Older Famicom consoles do not include controller ports, instead the joypad cables are directly attached to the console (without plugs/sockets).

Famicom Expansion Port (standard db15, female, front side)
Included in both older and newer Famicom consoles, not in NES consoles.

1	Out	GND	
2	Out	SOUND OUT	(headphone adaptors)
3	I/O	/IRQ	
4	In	port1-D4	(zapper button)
5	In	port1-D3	(zapper light)
6	In	port1-D2	(barcode battler) (turbo file data.in)
7	In	port1-D1	(joystick 4 serial input) (paddle ADC serial input)
8	In	port1-D0	(joystick 2 serial input)
9	Out	port1-CLK	(joystick 2+4 clock read)
10	Out	OUT2	(turbo file data.clock) (tape output)
11	Out	OUT1	(turbo file reset address)
12	Out	OUT0	(joystick 1+2+3+4 start) (turbo file data.out)
13	In	port0-D1	(joystick 3 serial input) (paddle button) (tape input)
14	Out	port0-CLK	(joystick 1+3 clock read)
15	Out	+5V	



Used to connect a 3rd and 4th joystick, and various other expansion hardware.
Keep in mind that older Famicom Joypads cannot be disconnected, so the input at Pin 8 may be disturbed by joypad 2 signals.

Note: Joypads/PowerPads/etc are normally using standard 4021 parallel-in serial-out shift registers.

Controllers - Summary of Controller Types

NES Controllers

```
NES Joypad 1/2
NES Joypad 3/4
NES Joypad 4-player adaptor Satellite (for joypad 3-4) (wireless)
NES Joypad 4-player adaptor Four-Score (for joypad 3-4) (wired)
NES Lightgun Zapper
NES Arkanoid I Paddle
NES Miracle Piano Keyboard (49 keys)
NES Mat - Power Pad (dance mat)
NES RacerMate Bicycle Training System
NES Power Glove
NES UForce
NES Add-on 3D-Glasses (colored lens, without any electronics)
NES Add-on Robot
```

Note: Most of the NES controllers do also exist for the Famicom. However, observe that they are connected differently, and thus need to be accessed differently at software side.

Famicom Controllers

```
Famicom Joypad 1/2 (hardwired to console, pad2 with mic instead start/select)
Famicom Joypad 3/4 (player 3/4, or alternate joypads for player 1/2)
Famicom Joypad 4-player adaptor (for joypad 4, not needed for joypad 3)
Famicom Joypad Add-ons (gimmicks mounted on top of standard joypads)
Famicom Lightgun Beam-Gun
Famicom Lightgun Hyper Shot (extra buttons, not fully compatible trigger)
Famicom Arkanoid I Paddle (with 9bit overflow)
Famicom Arkanoid II Paddle (raw 8bit without overflow)
Famicom Arkanoid II Secondary Paddle (add-on for second player)
Famicom Push Buttons Party Tap (6 players, 1 button/player)
Famicom Push Buttons Hyper Shot (2 players, 2 buttons/player)
Famicom Keyboard (72 keys type-writer keyboard)
Famicom Doremikko Piano Keyboard (36 keys)
Famicom Keypad TV-Net ("remote control" style) (different versions exist)
Famicom Keypad Famicom Network (joypad with numeric keypad)
Famicom Keypad Mahjong Controller (keys "A..M", plus 7 functions keys)
Famicom Mat - Family Trainer (dance mat)
Famicom Mat - Tap-tap Mat (mat with hammer)
Famicom Inflatable Exciting Boxing Bop Bag
Famicom Inflatable Top-Rider Bike
Famicom Tablet Oeka Kids (touchpad)
Famicom Trackball Hori Track (joypad with trackball)
Famicom Power Glove
Famicom Barcode Reader - Barcode Battler (controller port)
Famicom Barcode Reader - Datach (cartridge slot)
Famicom Pachinko - Joypad with pachinko dial
Famicom Microphone - Standard Microphone in japanese Joypad 2
Famicom Microphone - Bandai Cartridge with "Stage" Microphone & buttons
Famicom Disk System (with Eject Button; aka no-disk sensor)
Famicom Add-on 3D-Glasses (with LCD shutters, connected to controller port)
Famicom Add-on Storage Turbo File
Famicom Add-on Storage Battle Box
Famicom Add-on Storage Data Recorder
Famicom Add-on Robot
```

Console Variants with special Controllers

```
Subor Keyboard (NES-clones with built-in keyboard)
Subor Mouse (NES-clone bundled with mouse)
Sharp Famicom Titler (console with built-in tablet and keypad)
VS System Joysticks (swapped signals; other than normal joypads)
VS System Dip-Switches / Coin-Inputs / Service Button
VS System Lightgun (with serially injected data)
Play Choice 10
FamicomBox: Dip Switches, Keyswitch, Coin Input, Joypad/Zapper Control
Nintendo M82 - Shop Demo Unit for 12 games (similar to FamicomBox)
```

Controllers - Summary of Controller Signals

========== Port 4016h.Write ==========

OUT-0 (4016h.W.Bit0)

```
NES/Famicom Strobe Joypads, Paddle, Keyboard, etc. (load Shift-Registers)
NES Miracle Piano Data Out (and Long/Short Strobe)
NES RacerMate Bicycle Trainer - Forwarded to TX1/TX2 on 4016h/4017h.reads
```

```
  Famicom Battle Box: CLK to EEPROM, and, when set, enable chipselect toggle
  Famicom Doremikko Piano: Clock for next row
  Famicom Turbo File Data Out
  Famicom Oeka Kids Tablet Strobe (inverse of normal joypad strobe)
```

OUT-1 (4016h.W.Bit1)

```
  Famicom 3D System (select left/right shutter for 3D Glasses)
  Famicom Bandai Hyper Shot Gun: Vibration Feature Enable
  Famicom Doremikko Piano: Start transfer (select 1st row)
  Famicom Exciting Boxing (row select)
  Famicom Keyboard Clock for Nibbles
  Famicom Konami Hyper Shot Must be LOW for Player 2 "JUMP"/"RUN" Buttons
  Famicom Newer-Paddle-Versions: Manually start next A/D-Conversion?
  Famicom Oeka Kids Tablet Clock (manually clocked, unlike joypads)
  Famicom Top-Rider Bike - Start some conversion or so?
  Famicom Turbo File Reset Address to 0000h
  NES Unused (not connected to Controller Port)
  VS Dualsystem: Send IRQ to other CPU (0=No, 1=IRQ) (and map Shared-RAM)
```

OUT-2 (4016h.W.Bit2)

```
  Famicom Bandai Hyper Shot Gun: Sound Feature Enable
  Famicom Keyboard Tape Data Out
  Famicom Konami Hyper Shot Must be LOW for Player 1 "JUMP"/"RUN" Buttons
  Famicom Turbo File Data Clock
  NES Unused (not connected to Controller Port)
  VS Unisystem Mapper 99, Select 8K VROM Bank
```

========== Port 4016h.Read ==========

PORT0-0 (4016h.R.Bit0)

```
  Famicom Joypad 1 (always connected)
  NES/Famicom Power Glove (per-byte strobe, and data-out after LONG strobe)
  NES Hori Track (8bit joypad, 2x4bit trackball, 4bit switches/ID) (unreleased)
  NES Joypad 1 (if connected)
  NES Miracle Piano Data In
  NES Power Glove (data-in) (Mattel)
  NES RacerMate Bicycle Trainer - Get RX1 (and forward OUT0 to TX1) (Player 1)
  NES Satellite & Four-Score (Joypad 1, Joypad 3, ID_A)
  VS Unisystem Joypad 2 (not Joypad 1)
  VS Unisystem Lightgun (5th Bit=ID=1, 7th Bit=Light, 8th Bit=Trigger)
```

PORT0-1 (4016h.R.Bit1)

```
  Famicom Hori Track (8bit joypad, 2x4bit trackball, 4bit switches/ID)
  Famicom Joypad 3 (when 4-player adaptor used)
  Famicom Pachinko Controller (8bit Joypad 3 data, followed by 8bit ADC data)
  Famicom Paddle 1 Button (1bit)
  Famicom Keyboard Tape Data In
  NES Unused (not connected to Controller Port)
```

And, probably: Famicom Power Glove (data-in) (PAX)

PORT0-2 (4016h.R.Bit2)

```
  Famicom Microphone (built-in in Joypad 2)
  NES Unused (not connected to Controller Port)
  VS Unisystem Credit Service Button
```

PORT0-3 (4016h.R.Bit3)

```
  Famicom Unused (not connected to Controller nor Expansion Port)
  NES Second-Zapper Light Sensor
  NES Power Pad Bits 2,1,5,9,6,10,11,7
  VS Unisystem Dip Switch 1
```

PORT0-4 (4016h.R.Bit4)

```
  Famicom Unused (not connected to Controller nor Expansion Port)
  NES Second-Zapper Trigger Button
  NES Power Pad Bits 4,3,12,8, and four "1"-bits
  VS Unisystem Dip Switch 2
```

PORT0-5/6 (4016h.R.Bit5/6)

```
  NES/Famicom Unused (not connected to Controller nor Expansion Port)
  VS Unisystem Credit Left/Right Coin Slot
```

PORT0-7 (4016h.R.Bit7)

```
  NES/Famicom Unused
  VS Dualsystem: Master/Slave ID (0=Slave CPU, 1=Master CPU)
```

========== Port 4017h.Read ==========

PORT1-0 (4017h.R.Bit0)

```
Famicom Joypad 2 (always connected) (without Start/Select)
NES Joypad 2 (if connected)
NES Satellite & Four-Score (Joypad 2, Joypad 4, ID_B)
NES RacerMate Bicycle Trainer - Get RX2 (and forward OUT0 to TX2) (Player 2)
Subor Clone: Subor Mouse Data (in conjunction with OUT-0,OUT-1,OUT-2 ?)
VS Unisystem Joypad 1 (not Joypad 2)
```

PORT1-1 (4017h.R.Bit1)

```
Famicom Doremikko Piano: Data fragment for current half-row
Famicom Joypad 4 (when 4-player adaptor used)
Famicom Exciting Boxing (column 1)
Famicom Keyboard Bit0 of 4-bit Nibble
Famicom Konami Hyper Shot Player 1 "RUN" Button
Famicom Mahjong Controller: Data (in conjunction with OUT-0,OUT-1,OUT-2 ?)
Famicom Paddle 1 Position (8bits)
NES Unused (not connected to Controller Port)
```

PORT1-2 (4017h.R.Bit2)

```
Famicom Barcode Battler (20-byte ASCII String at 1200 Baud, 8N1)
Famicom Doremikko Piano: Data fragment for current half-row
Famicom Exciting Boxing (column 2)
Famicom Keyboard Bit1 of 4-bit Nibble
Famicom Konami Hyper Shot Player 1 "JUMP" Button
Famicom Oeka Kids Tablet Ack (confirm Strobe/Clock signals)
Famicom Party Tap (Button 1, Button 4, ID "1"-Bit)
Famicom Turbo File Data In
NES Unused (not connected to Controller Port)
VS Unisystem Dip Switch 3
```

PORT1-3 (4017h.R.Bit3)

```
Famicom Battle Box: Dta.In (data from EEPROM)
Famicom Doremikko Piano: Data fragment for current half-row
Famicom Exciting Boxing (column 3)
Famicom Keyboard Bit2 of 4-bit Nibble
Famicom Konami Hyper Shot Player 2 "RUN" Button
Famicom Oeka Kids Tablet Data (18bits)
Famicom Paddle 2 Button (1bit)
Famicom Party Tap (Button 2, Button 5, ID "0"-Bit)
Famicom Top-Rider Bike - First 8bit shift-register
NES/Famicom Zapper Light Sensor
NES/Famicom Power Pad Bits 2,1,5,9,6,10,11,7
NES Paddle Button (1bit)
NES Power Pad Bits 2,1,5,9,6,10,11,7
VS Unisystem Dip Switch 4
```

PORT1-4 (4017h.R.Bit4)

```
Famicom Battle Box: Status of Dta.out (can be toggled by [4016h].reads)
Famicom Doremikko Piano: Data fragment for current half-row
Famicom Exciting Boxing (column 4)
Famicom Keyboard Bit3 of 4-bit Nibble
Famicom Konami Hyper Shot Player 2 "JUMP" Button
Famicom Paddle 2 Position (8bits)
Famicom Party Tap (Button 3, Button 6, ID "1"-Bit)
Famicom Top-Rider Bike - Second 8bit shift-register
NES/Famicom Zapper Trigger Button
NES/Famicom Power Pad Bits 4,3,12,8, and four "1"-bits
NES Power Pad Bits 4,3,12,8, and four "1"-bits
NES Paddle Position (8bits)
VS Unisystem Dip Switch 5
```

PORT1-5/6/7 (4017h.R.Bit5/6/7)

```
NES/Famicom Unused (not connected to Controller nor Expansion Port)
VS Unisystem Dip Switch 6/7/8
```

========== Other Controller Port Signals ==========

PORT0-CLK (4016h.R) and PORT1-CLK (4017h.R)

Automatically clocked when reading 4016h and 4017h accordingly, usually clocking serial shift registers (joypad, paddle). Other uses are:

```
Famicom Battle Box: Toggle Dta.out (always), Toggle /CS (when [4016h].W.0=1)
Famicom Doremikko Piano: Clock for 1st/2nd half of current row
FamicomBox: Watchdog Reload
NES RacerMate Bicycle Trainer - Forward OUT0 to TX0/TX1 (by 4016h/4017h.Read)
```

/IRQ

```
Famicom Unused (expansion port /IRQ-pin not used by any known controllers)
NES Unused (not connected to Controller Port)
```

SOUND OUT

```
Famicom Headphones (used by some headphone adaptors)
```

```
  NES Unused (not connected to Controller Port)
```

========== Controller Signals on Cartridge Slot ==========

Controller I/O Ports

```
  4032h FDS Disk Status Register 1 (Disk Insert/Eject Sensor)
  6000h Bandai Microphone and Buttons ?
  6000h.Bit3 - Bandai Datach - Joint ROM System (Barcode Sensor)
  500xh FamicomBox: Dip Switches, Keyswitch, Coin Input, Joypad/Zapper Control
```

And, video out: Used by Lightguns and R.O.B. Robot.

Controllers - Detection

Opcodes

```
  88 8C 16 40 A9 08 2D 17 40 C9 08 C8 8C 16 40  ;Battle Box
  AD 17 40 4A 4A 29 01 09 06 8D 16 40           ;Turbo File (older games)
  AD 17 40 29 04 4A 4A 09 06 8D 16 40           ;Turbo File (newer games)
```

Controllers - Joypads

Joypads (or Joysticks)

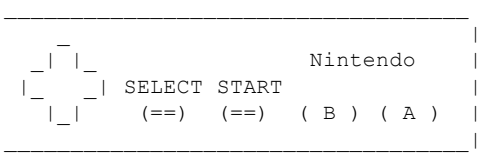
Each joypad includes an 8bit shift register, set Port 4016h/Bit0=1 to reload the button states into the shift registers of both joypads, then reset Port 4016h/Bit0=0 to disable the shift reload (otherwise all further reads would be stuck to the 1st bit, ie. Button A). Joypad data can be then read from bit 0 of 4016h (joypad 1) and/or bit 0 of 4017h (joypad 2) as serial bitstream of 8bit length, ordered as follows:

```
  1st         Button A         (0=High=Released, 1=Low=Pressed)
  2nd         Button B         (0=High=Released, 1=Low=Pressed)
  3rd         Select           (0=High=Released, 1=Low=Pressed)
  4th         Start            (0=High=Released, 1=Low=Pressed)
  5th         Direction Up     (0=High=Released, 1=Low=Moved)
  6th         Direction Down   (0=High=Released, 1=Low=Moved)
  7th         Direction Left   (0=High=Released, 1=Low=Moved)
  8th         Direction Right  (0=High=Released, 1=Low=Moved)
  9th and up  Unused (all 1=Low)  ;(or all 0=High when no joypad connected)
```

The console automatically sends a clock pulse to the Joypad 1 shift register after each read from 4016h (and to joypad 2 after read from 4017h). There are no timing restrictions, joypads can be handled as fast, or as slow, as desired.

Note that older Famicom controllers include Select & Start buttons only on joypad 1, whilst joypad 2 probably returns unused dummy bits instead (the values of that dummy bits are unknown, probably 0=High=Released?).

Joypad Layout



Jump and Run Games conventionally use A=Jump, B=Fire.

Third-Party Joypads/Joysticks (NES)

Third-Party NES Joypads/Joysticks can be simply plugged into normal controller ports.

Third-Party Joypads/Joysticks (Famicom)

The Famicom comes with two hardwired joypads, making it difficult to use third-party joypads or joysticks. One simple solution is to mount add-on hardware (mechanically) on the standard joypads:

```
  Climber Stick NBF-CY (Nichibutsu) (mini-dildo/nipple to be mounted on DPAD)
  FamiCoin (colored "coins" to be mounted on DPAD; for better grip or so)
  Super Controller (Bandai?)(joystick to be mounted on DPAD of standard joypad)
  Ultech 3 Meijin-kun (joystick to be mounted on DPAD+A+B of standard joypad)
```

Another solution is to connect the devices to the 15pin expansion port (the 15pin connector lacks the joypad 1 signal, so at best, they could be shortcut with the joypad 2 signal, or wired as joypad 3/4; aside from 3/4-player games, many 1/2-player games also support that kind of input). Known devices are:

```
  ASCII Stick L5 - one-handed joypad (not joystick) (ASCII)
  Joypad with numeric keypad for Famicom Modem (connects to the 15pin port)
```

Reportedly, following further devices exist (unknown if they are mechanically attached, or electronically connected to 15pin port, or to 7pin ports of late Famicoms):

```
  Joystick-7 and Joystick-7 Mk II
  Joycard Sanusui SSS (Hudson) (with adapter for headphones)
  Reggies's Joystick (with turbofire)
  Super Controller II (Bandai) (with LCD screen)
  Toyo Stick (Toyo)
```

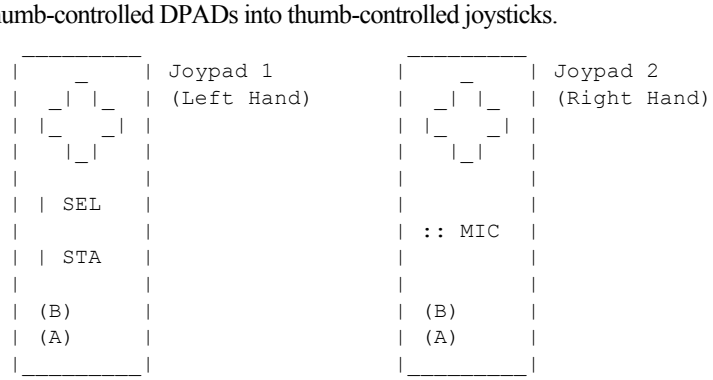
Attention:

To support external joypads, single-player games should read joypad 3, and treat it is alternate input for player 1. Two-player games should also support

joypad 4 (via 4-player adaptor) as alternate input for player 2.

Crazy Climber Stick

Crazy Climber is played with two regular joypads, but both turned by 90 degrees. Optionally, "Climber Sticks" can be mounted on the joypads, turning the thumb-controlled DPADs into thumb-controlled joysticks.



Controllers - Four-Player Adaptors

NES Four-player devices (Satellite and Four Score)

```
NES Satellite (wireless) (Nintendo) (1989)
Four-Score (wired) (Nintendo) (1990)
```

The device is connected to both of the consoles two controller ports, and up to four controllers can be connected to the device.

The device is accessed much like normal joypads, except that the returned bitstream consist of 24 bits instead of normal 8 bits:

```
write "1-then-0" to (4016h) (that only once, for all 24 bits)
read 1st 8 bits: controller 1    (4016h) / controller 2   (4017h) (as normal)
read 2nd 8 bits: controller 3    (4016h) / controller 4   (4017h) (new ports)
read 3rd 8 bits: 0,0,0,1,0,0,0,0 (4016h) / 0,0,1,0,0,0,0,0 (4017h) (ID codes)
further bits: unknown (probably all 0, or all 1)
```

The ID codes can be used to detect if the 4-player adapter is connected (used by RC Pro Am 2, not used by Gauntlet II). Otherwise the ID field typically contains all ones (normal/single controller), or all zeros (no controller connected at all).

Satellite & Four Score Games

```
Bomberman II
Danny Sullivan's Indy Heat
Gauntlet II
Greg Norman's Golf Power
Harlem Globetrotters
Kings of the Beach
Magic Johnson's Fast Break
Monster Truck Rally
M.U.L.E.
NES Play Action Football
Nightmare on Elm Street
Nintendo World Cup
Rackets & Rivals
R.C. Pro-Am II
Rock 'n' Ball (?)
Roundball: 2 on 2 Challenge
Smash TV
Spot
Super Jeopardy!
Super Off Road
Super Spike V'Ball
Swords and Serpents
Top Players' Tennis
```

Famicom Four-player device (Two extra joypads at Expansion Port)

Older Famicom consoles have 2 "built-in" joypads, additional 2 joypads can be connected to the expansion port. The procedure for reading Famicom 4-player data is similar as for 2-player data: As normal, write "1-then-0" to 4016h, then read 8 times from 4016h, that simultaneously receives data for two pads, Bit 0 for joypad 1, and additionally Bit 1 for joypad 3. Respectively, Bit 0 and 1 of 4017h are for pad 2 and 4.

Japanese 3-4 player games:

```
Downtown Nekketsu Koshinkyoku: Soreyuke Daiundokai
Ike Ike! Nekketsu Hockey Bu: Subette Koronde Dai Ranto
Nekketsu Kakutou Densetsu
Nekketsu Koukou Dodge Ball Bu (in Bean Ball mode of japanese version only)
Nekketsu Street Basket: Ganbare Dunk Heroes
Kunio-kun no Nekketsu Soccer League
U.S. Championship V'Ball
Wit's
```

Note: For 3-players, joypad 3 can be plugged directly to the Expansion Port (without needing a 4-player adaptor).

Moreover, many normal 1-2 player games do support joypad 3-4 as alternate player 1-2 inputs (allowing to use external joysticks/joypads, including such with autofire functions, instead of the standard famicom joypads).

Controllers - Lightguns (Zapper)

Zapper (Light Gun) Ports / Connection

Zapper state can be obtained by reading Bit3-4 of Port 4017h and/or 4016h.
Famicom Zapper connected to Famicom Expansion Port (Inputs at 4017h).
NES Zappers connected to 1st and/or 2nd Joypad Port (Inputs at 4016h, 4017h).
Most or all NES games are using 2nd Joypad Port because Famicom uses 4017h.

```
Bit3  State of the gun sight (0=High=Light detected, 1=Low=None)
Bit4  Trigger (0=High=Released, 1=Low=Pulse) (shoot on 1-to-0 transition)
```

The trigger mechanics are working as so:

```
pulled part way: soft "click" sound: 0-to-1 transition (ignored by games)
pulled full way: loud "CLANG" sound: 1-to-0 transition (games do fire)
```

One exception are Bandai games, which do accidently shoot at 0-to-1 (either because of misunderstanding the zapper hardware, and/or for compatibility with Bandai's own Hyper Shot gun).
The light detection flag gets set when sensing light emission from the display, ie. when the cathode ray beam outputs a bright color (preferably white) at the location where the gun is pointed to.
Handling the light sensor is usually done as so:

```
Output a black picture with a white-rectangle at target location
If the gun senses light, then it's a "hit".
(to reduce unnecessary flickering, do that only when trigger is pulled)
```

To avoid "hits" on other light sources, verify that there is NO light at times when the screen should be dark (eg. shortly before end of vblank).
When sensing light, the sensor bit does reportedly stay set for 10-25 scanlines (during hit-checks, one should check the bit around every 5 scanlines).
Note: The NES PPU doesn't have any H/V-latches for lightpen/lightgun coordinates, so there's little chance to measure the H-position by software.
However, one may measure the vertical position by counting the time between light and vblank (this would allow handling multiple targets at different vertical positions within a single frame).
Note: Shooting at offscreen locations is advancing menu cursors in some games (the games are typically also allowing to do this via joypad-select, so it isn't stritcly necessary to support offscreen positions in emulators).

Games compatible with the NES Zapper:

```
Adventures of Bayou Billy (gun optional) (U) 1989 (E) 1990 Konami
Baby Boomer (unlicensed) (gun optional) (U) 1989 Jim Meuer/Color Dreams
Barker Bill's Trick Shooting (U) 1989 Nintendo
Chiller (unlicensed) (gun optional) (U) (HES) 1986 Exidy
Day Dreamin' Davey (gun optional) (U) 1990 HAL
Duck Hunt (JUE) (PC10) (VS) 1984 Nintendo
Freedom Force (U) (VS) 1988 Sunsoft
Gotcha! The Sport! (U) 1987 LJN
Gumshoe (UE) (VS) 1986 Nintendo
Gun-Nac (U) (J) 1990 Tonkin House/Tokyo Shoseki/Nexoft
Hogan's Alley (JU) (PC10) (VS) 1984 Nintendo
Laser Invasion (U) aka Gun Sight (for LaserScope or Zapper) (J) 1991 Konami
The Lone Ranger (gun optional) 1991 Konami
Mechanized Attack (gun optional) (U) 1991 SNK
Operation Wolf (gun optional) (J) (U) 1989 Taito
Shooting Range (for Hyper Shot or Zapper+Joypad) (U) 1989 Bandai
Space Shadow (for Hyper Shot or Zapper+Joypad) (J) 1989 Bandai
To The Earth (U) 1989 Nintendo
Track & Field II (in one section of the game) (U)(E) 1988/89 Konami
Wild Gunman (U) (PC10) Wairudo Ganman (J) 1984 Nintendo
```

NES/Famicom Lightguns

```
Nintendo Zapper (US) (old gray/dark gray version) (1985)
Nintendo Zapper (US) (new gray/orange version)
Nintendo Beam Gun (Japan) (black/brown revolver) (1985)
Nintendo VS Unisystem Lightgun (orange revolver) (2bit serial transmit)
Bandai Hyper Shot (black machine gun: zapper + joypad-like buttons) (1989)
Hyperkin FC Super Loader Gun (made by Hyperkin, NES/Zapper compatible?)
Konami LaserScope/Gun Sight (headset: headphones + voice-activated zapper)
Nexoft The Dominator: Pro Beam Light Gun (wireless lightgun)
```

As additional add-on gimmicks, Quickshot has made Sighting Scopes and Deluxe Sighting Scopes that can be mounted on the Nintendo Zapper.

Bandai Hyper Shot Gun (for Space Shadow) (1989)

A black machine pistol, working (more or less) like a normal zapper combined with an additional joypad.

```
4016h.R.Bit1 Serial 8bit joypad3-style button data
4017h.R.Bit3 Light   (0=High=Yes, 1=Low=No)
4017h.R.Bit4 Trigger (0=High=Released, 1=Low=Pressed/Held) (shoot while 1)
4016h.W.Bit1 Gun Move aka Body Vibration System (0=Off, 1=On)
4016h.W.Bit2 Sound (0=Off, 1=On)
```

The serial "joypad3" data is used as so (by Space Shadow):

```
joypad3 button B --> throw grenade
joypad3 up       --> move forward (after defeating enemy)
joypad3 select   --> toggle sound/gun-move (in title screen)
joypad3 start    --> start/pause
```

The trigger is a simple push-button without the normal zapper-mechanics, this allows Space Shadow to support continous fire when the button is held down, but isn't fully compatible with normal zapper games (which will fire on 1-to-0 transitions, ie. when <releasing> the Hyper Shot trigger, rather than pressing it).
The light sensor may have different sensitivity as normal zappers (Space Shadow does use white-rectangles, but doesn't output black-background; the

tunnel-backgrounds are fairly dark, but the backgrounds at tunnel-end are very bright, even including some white pixels; so apparently, the hypershot gun senses light only when aimed at all-white-pixels areas).
There seems to be some sort of rumble/vibration feature (called "Body Vibration System" on the gun, and "Gun Move" in Space Shadow).
And, some "Sound" feature (can be enabled/disabled in Space Shadow title screen), details are unknown; maybe causing the gun to produce sounds when pulling the trigger, or maybe simply outputting the APU sound/music to a speaker in the gun?
Note: Bandai's "Hyper Shot" Lightgun is not to be confused with Konami's "Hyper Shot" Push Buttons.

VS System Lightguns (for VS Unisystem/Dualsystem arcade machines)
These are resembling the japanese revolver-shaped Beam Guns. The VS Unisystem can't use the "normal" Zapper I/O lines (because it has DIP-Switches connected to that ports), and, instead, it's injecting Lightgun data to the 8bit "Joypad" shift-register:

```
Port.Bit      (VS-Joypad)  VS-Zapper
4016h.5th.Bit (Joy2 Up)    Connect OK   (0=Alert sound, 1=Connected)
4016h.7th.Bit (Joy2 Left)  Light Sensor (0=No,1=Light ;INVERSE of NES Zapper)
4016h.8th.Bit (Joy2 Right) Trigger      (0-to-1=Fire ;INVERSE of NES Zapper)
```

The Connect bit is used to generate an Alert sound if people are trying to steal the gun by cutting the cable.
The gun lacks a transistor that is found in normal Zappers (so the Light sensor will output opposite voltage).
Also mind that the VS Unisystem uses different palette(s) than NES consoles, so the PPU color number for White (20h or 30h on NES) may differ from game to game.

FamicomBox Zapper
The FamicomBox is using a standard NES-Zapper, but with some extras on the mainboard: A somewhat bidirectional Supply GND pin on the Zapper connector (allows to do a zapper connect check), an Disconnect-Alert signal (on a special "CATV" connector), and a DIP-switch for disabling the Zapper.
For details, see:
FamicomBox

Zapper Schematic / Handheld Pistol NES004

```
.----------.
|         1|--------------GND                          VCC -----NES.Pin5
| Sharp   2|---|<|--------GND (light sensor "43 Pi")    GND -----NES.Pin1
| IR3T07A 3|---[22]--||---GND (cap: 1/50)                    ___
|         4|---------||---GND (cap: 3.3/50)  VCC        GND -o o-NES.Pin7
|         5|--------------GND                 |    2SC459  Trigger
|         6|--[390K]------VCC               [10K] .------.
|         7|---------||---GND (cap: .001)     |   |     C|------ NES.Pin6
|         8|-----------------------------------+---|B     |
|         9|--------------VCC                      |     E|---GND
'----------'    VCC--||---GND (cap: 10/16)         '------'
```

Arcade Gun "062G2 GUN" (for VS and/or PC10 or so)

```
.----------.
|         1|--------------GND            GND ----------- GND, RedPlug.pin1
|         2|---|<|--------GND (light sensor "43PI")  .-- GND, RedPlug.pin2
| IR3T07  3|--[6R8]--||---GND (C1: 105)              '-- ALM, RedPlug.pin3
|         4|---------||---GND (C2: 335)
|         5|--------------GND
|         6|--[394]-------VCC                    ___
|         7|---------||---GND (C3: 102)  GND ---o o--- TRG, WhitePlug.pin1
|         8|------------------------------------------ HIT, WhitePlug.pin2
|         9|--------------VCC            VCC --[101]-- +5V, WhitePlug.pin3
'----------'   GND--||--VCC (C4: 476)      GND--||--+5V (C5: 476)
```

Note: Aside from the electronics, there's also a lens the gun barrel (not shown in the schematics).

Controllers - Paddles

The Paddle (aka Arkanoid Controller or Vaus Controller) consists of a Push Button, and a Dial (a potentiometer, which can be turned left/right by about 270 degrees, connected to an A/D-converter and 8bit shift-register), some (or maybe all) paddles also have a second potentiometer (for calibrating the dial-range).

Paddle Games

```
Arkanoid (J) 1986 (Taito) (bundled with OLD paddle for 15pin Famicom)
Arkanoid (U) 1987 (Taito) (bundled with OLD paddle for 7pin NES)
Arkanoid 2 (J) 1988 (Taito) (with NEW paddle; with connector for 2nd paddle)
Chase H.Q. (J) 1989 (Taito) (compatible with NEW paddle)
```

Old Paddles (automatic ADC-start, max 1 paddle)
For Famicom-version (Japan):

```
4016h.W.Bit0 Load shift-register     (strobe 1-then-0)
4016h.R.Bit1 Paddle Button   (1bit)  (0=High=Released, 1=Low=Pressed)
4017h.R.Bit1 Paddle Position (8bits) (0=High=One, 1=Low=Zero) (MSB first)
```

For NES-version (US/Europe):

```
4016h.W.Bit0 Load shift-register     (strobe 1-then-0)
4017h.R.Bit3 Paddle Button   (1bit)  (0=High=Released, 1=Low=Pressed)
```

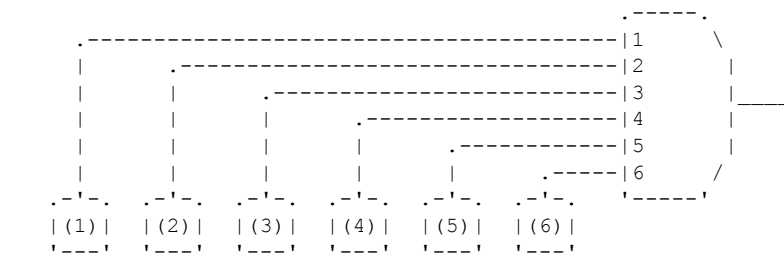
```
  4017h.R.Bit4 Paddle Position (8bits) (0=High=One, 1=Low=Zero) (MSB first)
```
For the Position Values, first of, undo inversion of the ADC value (XOR by FFh), then handle the result as so:
```
  treat 00h..1Fh as position 100h..11Fh  ;\repair 8bit/9bit "overflows"
  treat 20h..FEh as position 020h..0FEh  ;/
  treat FFh as NOT CONNECTED (although FFh may ALSO occur at position 0FFh)
  clip position 020h..11Fh to min/max range 062h..102h  ;<-- Arkanoid
```
Values are small=left, large=right. With old paddles, A/D-conversion seems to start automatically (unlike as new paddles); unknown how that works (if it's poorly implemented, read-errors might occur if the ADC value is updated during shift-register loading).

New Paddles (different ADC-range, manual ADC-start, max 2 paddles)
For Famicom-version (new paddles were released ONLY in Japan):
```
  4016h.W.Bit0 Load shift-register       (strobe 1-then-0)
  4016h.R.Bit1 Paddle 1 Button   (1bit)  (0=High=Released, 1=Low=Pressed)
  4017h.R.Bit1 Paddle 1 Position (8bits) (0=High=One, 1=Low=Zero) (MSB first)
  4017h.R.Bit3 Paddle 2 Button   (1bit)  (0=High=Released, 1=Low=Pressed)
  4017h.R.Bit4 Paddle 2 Position (8bits) (0=High=One, 1=Low=Zero) (MSB first)
  4016h.W.Bit1 Start next A/D-conversion (strobe 1-then-0)
```
For the Position Values, first of, undo inversion of the ADC value (XOR by FFh), then handle the result as so:
```
  treat 00h..FFh as position 00h..FFh (no 9bit "overflows" in new paddles)
  optionally treat as FFh as NOT CONNECTED (Arkanoid 2/Chase H.Q. don't so)
  clip position 00h..FFh to min/max range 4Eh..BAh ;<--Arkanoid 2, TINY-screen
  clip position 00h..FFh to min/max range 4Eh..F2h ;<--Arkanoid 2, WIDE-screen
  clip position 00h..FFh to min/max range 54h..DBh ;<--Arkanoid 2, PONG-view
  clip position 00h..FFh to min/max range 38h..E7h ;<--Chase H.Q.
```
Note: For reaching the Arkanoid 2 doors at left/right screen sides, the values received from paddle must EXCEED the 4Eh..F2h min/max range.
Values are small=left, large=right (or, in Arkanoid 2's PONG-view, small=up for player 1, and small=down for player 2).

Note
Emulators (or hardware) can auto-detect the Paddle version by sensing OUT1=1.

Component List (Old Paddle)
```
  IC1 14pin NE556 (dual 555 universal timers)
  IC2 16pin NEC D4040BC (12-bit asynchronous binary counter with reset)
  IC3 16pin NEC D74HC00C (quad NAND gates)
  IC4 14pin NEC D74HC165C (8-bit parallel-in serial-out shift register)
  CN  6pin connector (VCC,GND,STB,DTA,CLK,BUTTON)
  VR1 paddle-dial potentiometer
  VR2 calibration potentiometer
  SW1 push-button
  plus, 11 capacitors, and 6 resistors
```

Controllers - Push Buttons

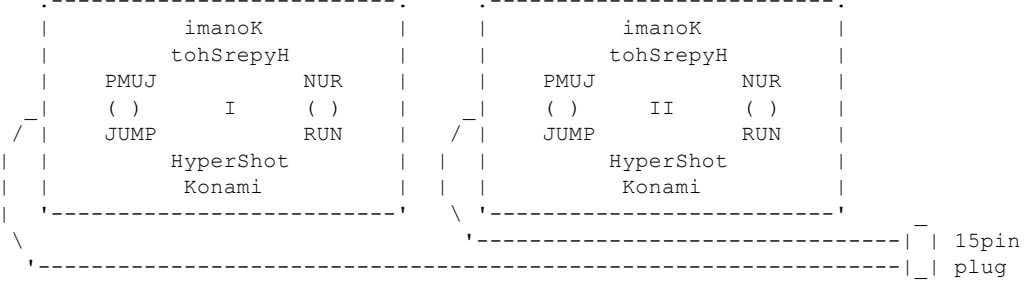
Party Tap (Six Players, one Button per player) (Yonezawa/Partyroom 21)
Set of six controllers, each with one push button. Used by following games:
```
  Casino Derby (J) (19xx)
  Gimmi a Break - Shijou Saikyou no Quiz OuKetteisen (J) (TBS/S'PAL) (1991)
  Gimmi a Break - Shijou Saikyou no Quiz OuKetteisen 2 (J) (TBS/S'PAL) (1992)
  Project Q (J) (Hect/Hector) (1992)
```


The thing has somehow fragile timings; the different games all have different delays before/after strobing and between reads. With max delays, it's accessed somewhat like so:
```
  wait 500 clks                                ;-lead delay (with strobe=0)
  [4016h]=01h, [4016h]=00h, wait 160 clks      ;-strobe 1-then-0 with delay
  a=[4017h], wait 80 clks, b=[4017h]           ;-read 2x3bit with delay
  data = (a AND 1Ch)/4 + (b AND 1Ch)*2         ;-merge (Bit0-5 = Button 1-6)
```
Moreover, at whatever time (apparently somewhere after above 1st/2nd reads), a detection value can be read: "([4017h.R] AND 1Ch)=14h"

Konami Hyper Shot (Two Players, two Buttons per player) (Konami)
Set of two controllers, each with two push buttons. Used by following games:
```
  Hyper Olympic (J) Konami (1985)
  Hyper Sports (J) Konami (1985)
```
Accessed as shown below.
```
  4016h.W.Bit1  Select Player 2 "JUMP"/"RUN" Buttons (0=Low=Yes) ;\usually, set
  4016h.W.Bit2  Select Player 1 "JUMP"/"RUN" Buttons (0=Low=Yes) ;/both to 0
  4017h.R.Bit1  Player 1 "RUN" Button  (0=High=No, 1=Low=Pressed and OUT-2=LOW)
  4017h.R.Bit2  Player 1 "JUMP" Button (0=High=No, 1=Low=Pressed and OUT-2=LOW)
```

```
4017h.R.Bit3  Player 2 "RUN" Button  (0=High=No, 1=Low=Pressed and OUT-1=LOW)
4017h.R.Bit4  Player 2 "JUMP" Button (0=High=No, 1=Low=Pressed and OUT-1=LOW)
```

Note: For whatever reason, the buttons are wired to OUT-2/OUT-1 (rather than being wired to GND), this would allow to select/deselect the buttons; in the existing software both OUT-2 and OUT-1 are always set to LOW.



The pads don't explicitly have "front/back" sides (and can be turned either way around); however, for shooting left/right in Hyper Sports they should be turned this way: JUMP=Left, RUN=Right.

Theoretically, one could as well use normal joypad buttons, however, Konami has intentionally crippled them to be working ONLY with their Hyper Shot: The games are actually reading normal joypad data, but are then erasing all joypad bits (except start/select), and are then replacing them by the Hyper Shot Button bits.

Note: Konami's "Hyper Shot" Buttons are not to be confused with Bandai's "Hyper Shot" Lightgun.

Controllers - Typewriter Keyboards

Keyboard with 72 Keys, and tape read/write port, connected to 15-pin Famicom Expansion port. Used by Famicom BASIC and Playbox BASIC. Also, the Study and Game 32-in-1 and Education 18-in-1 cartridges use an identical protocol, but with different keyboard matrix.

Keyboard Access Pseudo Code

```
[4016h]=05h:WAIT(16clks)                  ;reset (force row 0)
FOR i=0 TO 8                              ;loop 9 rows
  [4016h]=04h:WAIT(56clks)                ;request LSB of NEXT row
  Row[i]=(([4017h] SHR 1) AND 0Fh)        ;read LSB
  [4016h]=06h:WAIT(56clks)                ;request MSB of SAME row
  Row[i]=(([4017h] SHL 3) AND F0h)+Row[i] ;read MSB
NEXT                                      ;loop next
```

Column 0-7 are then in Bit0-7 of each row. Bits are 0=Pressed, 1=Released (unlike for most other NES/Famicom controllers, which are 1=Pressed). When reading more than 9 rows, the 10th read (row 9) returns garbage data, and then starts over at row 0.

Famicom Keyboard Matrix

Row	Bit0	Bit1	Bit2	Bit3	Bit4	Bit5	Bit6	Bit7
0	F8	RETURN	[	]	KANA	R-SHFT	\(Yen)	STOP
1	F7	@	:	;	_	/	-	^
2	F6	O	L	K	.	,	P	0
3	F5	I	U	J	M	N	9	8
4	F4	Y	G	H	B	V	7	6
5	F3	T	R	D	F	C	5	4
6	F2	W	S	A	X	Z	E	3
7	F1	ESC	Q	CTRL	L-SHFT	GRPH	1	2
8	CLR	UP	RIGHT	LEFT	DOWN	SPACE	DEL	INS

Famicom Keyboard Layout (HVC-007)

```
.-----------------------------------------------.
|    F1  F2  F3  F4  F5  F6  F7  F8             |
|    1 2 3 4 5 6 7 8 9 0 - ^ \ STOP             |
| ESC Q W E R T Y U I O P @ [ ENTER  CLR INS DEL |
| CTRL A S D F G H J K L ; : ] KANA      UP     |
| SHIFT Z X C V B N M , . / _ SHIFT  LEFT  RIGHT |
|      GRPH    SPACE                  DOWN      |
'-----------------------------------------------'
```

32-in-1 Study and Game / Education Keyboard Matrix

Row	Bit0	Bit1	Bit2	Bit3	Bit4	Bit5	Bit6	Bit7
0	4	G	F	C	F2	E	5	V
1	2	D	S	END	F1	W	3	X
2	INS	BS	PGDN	RIGHT	F8	PGUP	ESC	HOME
3	9	I	L	,	F5	O	0	.
4	]	ENTER	UP	LEFT	F7	[	\	DOWN
5	Q	CAPS	Z	Pa	ESC	A	1	CTRL
6	7	Y	K	M	F4	U	8	J
7	-	;	'	/	F6	P	=	SHIFT
8	T	H	N	SPACE	F3	R	6	B

The 32-in-1 menu also checks Bit4 in Row 9, if that bit is zero then it does additionally read row 0Ah..0Ch. Aside from the menu, most or all games in the 32-in-1 cartridge don't seem to use that extra rows though.

32-in-1 Study and Game Keyboard Layout (as shown in Typing School I)

```
.-------------------------------------------------------.
```

```
| ESC F1 F2 F3 F4 F5 F6 F7 F8 F9 F10 F11 F12 Pa. Br  Nu  Re. |
| ~  1   2   3   4   5   6   7   8   9   0   -   +  BS  HOME |
| TAB Q   W   E   R   T   Y   U   I   O   P   [   ]   \  END |
| CAPS A   S   D   F   G   H   J   K   L   ;   '  ENTER PGUP |
| SHIFT Z   X   C   V   B   N   M   ,   .   /  SHIFT UP PGDN |
| ### CTRL ALT ## [      SPACE      ] ALT INS DEL LT DN RIGH |
'-----------------------------------------------------------'
```

Above does only show how software in "Typing School I" depicts the keyboard, unknown how the real hardware looks like... there are some NES-console clones named GA-M16 and GLK-2016 with built-in keyboard... maybe these clone(s) are compatible with the 32-in-1 cartridge?

Keyboard I/O Signals

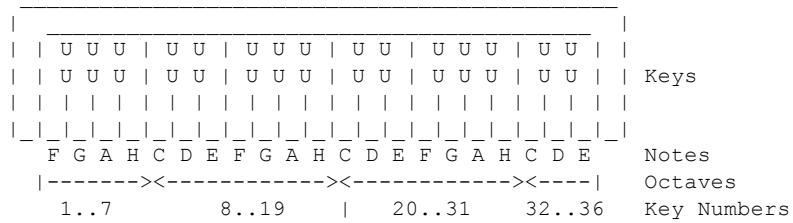
```
OUT.0      Keyboard Strobe/Reset  (0=Normal, 1=Initialize)
OUT.1      Keyboard Clock         (0=LSB, 1=MSB) (1-to-0=Next Row)
OUT.2      Tape Output            (Should be 1 when accessing Keyboard)
PORT0-1    Tape Input
PORT1-4..1 Keyboard Input Bit3..0 (either MSB or LSB of current row)
```

Tape In/Out allow to connect an external tape recorder.
Storage Data Recorder

Subor Keyboard
Details are unknown, maybe same as the Study and Game thing.

Controllers - Piano Keyboards

Doremikko (Three octaves; aka two & two-half octaves) (Konami)
Piano keyboard for the "Doremikko" FDS (Famicom Disk System) game.




```
36 Piano Keys (21 White Keys, 15 Black Keys) (two & two-half octaves)
```

The 36 keys are read like so:

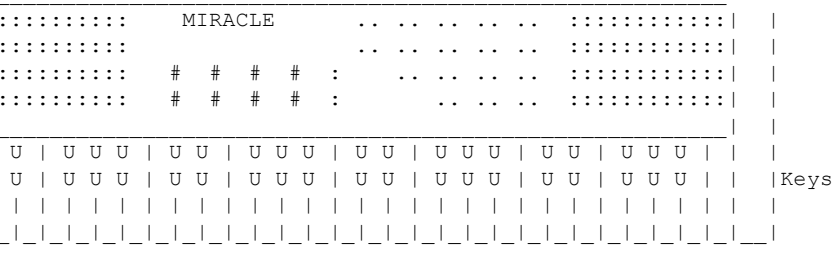
```
[4016h]=02h, [4016h]=01h                                  ;-write 2-then-1
dummy=[4017h],                    Key1,2=[4017h].Bit1,2   ;-read 0+2 bits
[4016h]=00h, [4016h]=01h                                  ;-write 0-then-1
Key3,4,5,6=[4017h].Bit1,2,3,4,    Key7,8=[4017h].Bit1,2   ;-read 4+2 bits
[4016h]=00h, [4016h]=01h                                  ;-write 0-then-1
Key9,10,11,12=[4017h].Bit1,2,3,4, Key13,14=[4017h].Bit1,2 ;-read 4+2 bits
[4016h]=00h, [4016h]=01h                                  ;-write 0-then-1
Key15,16,17,18=[4017h].Bit1,2,3,4, Key19,20=[4017h].Bit1,2 ;-read 4+2 bits
[4016h]=00h, [4016h]=01h                                  ;-write 0-then-1
Key21,22,23,24=[4017h].Bit1,2,3,4, Key25,26=[4017h].Bit1,2 ;-read 4+2 bits
[4016h]=00h, [4016h]=01h                                  ;-write 0-then-1
Key27,28,29,30=[4017h].Bit1,2,3,4, Key31,32=[4017h].Bit1,2 ;-read 4+2 bits
[4016h]=00h, [4016h]=01h                                  ;-write 0-then-1
Key33,34,35,36=[4017h].Bit1,2,3,4                         ;-read 4+0 bits
```

The used controller signals are working somewhat like so:

```
OUT-1      start transfer (select 1st row)
OUT-0      clock for next row
CLK        clock for 1st/2nd half of current row
PORT1.1-4  data for current row (4bit in 1st half, 2bit in 2nd half)
```

Unlike the Miracle, the Doremikko Keyboard doesn't contain its own sound generator. However, since the software was shipped on disk, it may use the FDS sound hardware:
Famicom Disk System (FDS)
The Piano mode (right-most option in main menu) uses normal APU sound as second voice (when pressing two keys).

Miracle (Four Octaves, plus next higher C) (The Software Toolworks)
MIDI Synthesizer connected to NES controller Port. Used by only one NES cartridge: The Miracle Piano Teaching System (1990). Aside from the keyboard, the Miracle contains its own sound generators and speakers.



```
   C D E F G A H C D E F G A H C D E F G A H C D E F G A H C    Notes
  <------------><------------><------------><------------>X    Octaves
      36..47         48..59    |   60..71        72..83    84   Key Numbers
                               |
                           Middle C
49 Piano Keys (29 White Keys, 20 Black Keys) (4 octaves, plus next higher C)
1  Foot Pedal (Sustain)
8  Push Buttons (Mode/Volume Selection)
```

The thing can be connected to NES, SNES, and other consoles/computers.

[Controllers - Piano - Miracle Piano Controller Port]
[Controllers - Piano - Miracle Piano MIDI Commands]
[Controllers - Piano - Miracle Piano Instruments]
[Controllers - Piano - Miracle Pinouts and Component List]

Controllers - Piano - Miracle Piano Controller Port

Miracle Controller Port Transfer

Read Direction (invoked by SHORT Strobe signal)

```
1st         Data Present Flag (0=High=None, 1=Low=Yes)
2nd..9th    Data Bit7..0      (MSB First, inverted 1=LOW=Zero)
10th..12th  Unknown
13th..16th  Unknown (would be ID Bit3..0 on other SNES controllers)
17th and up Unknown
```

Write Direction (invoked by LONG Strobe signal, data output on STROBE line)

```
1st..8th    Data Bit7..0      (MSB First, 0=LOW=Zero)
```

Observe that read/write direction depends on length of initial Strobe signal (so games that are reading joypad with other strobe-lengths might mess up things).

10th bit and up (including the 4bit Controller ID) might be garbage (depending on how the 8051 CPU in the keyboard handles the data transfer). However, with appropriate timings, detecting a Miracle could be done via the "Firmware version request" MIDI command.

Note: The NES and SNES Miracle software expects the piano keyboard connected to Port 1, and a normal joypad connected to Port 2.

miracle_recv_byte

```
[4016h]=01h                           ;strobe on
delay (need strobe on for 12 NES clks) ;short delay = READ mode
[4016h]=00h                           ;strobe off
data_present_flag = [4016h].bit0      ;data present flag (1=LOW=Yes)
for i=7 to 0
  data.bit(i)=NOT [4016h].bit0        ;data bits (MSB first, 1=LOW=Zero)
next i
```

miracle_send_byte

```
[4016h]=01h                           ;strobe on (start bit)
delay (need strobe on for 66 NES clks) ;long delay = WRITE mode
for i=7 to 0
  [4016h].bit0=data.bit(i)            ;data bits (MSB first, 1=HIGH=One)
  dummy=[4016h]                       ;issue short CLK pulse
next i
[4016h]=00h                           ;strobe off (stop/idle)
delay (21 NES clks... plus overload)  ;wait before next byte transfer
```

Controllers - Piano - Miracle Piano MIDI Commands

The Miracle is always using MIDI messages (no matter if the messages are transferred through MIDI or RS232 or NES/SNES/Genesis controller cables). Below lists the supported MIDI messages (including "Undocumented" messages, which are used by the Miracle's SNES software, although they aren't mentioned in the Miracle's Owner's Manual).

MIDI Information Sent FROM/TO The Miracle keyboard

```
Expl.                     Dir  Hex
Note off (Undocumented)     W  8#h,<key>,00h   ;same as Note ON with velo=00h
Note on/off command       R/W  9#h,<key>,<velo>
Main volume level           W  B0h,07h,<vol>
Sustain on/off command    R/W  B#h,40h,<flag>
Local control on/off        W  B0h,7Ah,<flag>
All notes off               W  B#h,7Bh,00h
Patch change command (*)  R ?? C#h,<instr>     ;TO keyboard = Undocumented
Miracle button action     R    F0h,00h,00h,42h,01h,01h,<bb>,F7h
Unknown (Undocumented)      W  F0h,00h,00h,42h,01h,02h,<??>,F7h   ;???
Keyboard buffer overflow  R    F0h,00h,00h,42h,01h,03h,01h,F7h
Midi buffer overflow      R    F0h,00h,00h,42h,01h,03h,02h,F7h
Firmware version request    W  F0h,00h,00h,42h,01h,04h,F7h
Miracle firmware version  R    F0h,00h,00h,42h,01h,05h,<maj>,<min>,F7h
Patch split command         W  F0h,00h,00h,42h,01h,06h,0#h,<lp>,<up>,F7h
Unknown (Undocumented)      W  F0h,00h,00h,42h,01h,07h,F7h        ;???
All LEDs on command         W  F0h,00h,00h,42h,01h,08h,F7h
```

```
  LEDs to normal command      W  F0h,00h,00h,42h,01h,09h,F7h
  Reset (Undocumented)        W  FFh
```

Direction: R=From keyboard, W=To keyboard

Notes: (*) Patch change FROM Keyboard is sent only in Library mode.

```
  N#h          Hex-code with #=channel (#=0 from keyb, #=0..7 to keyb)
  <key>        Key (FROM Miracle: 24h..54h) (TO Miracle: 18h..54h/55h?)
  <velo>       Velocity (01h..7Fh, or 00h=Off)
  <vol>        Volume (00h=Lowest, 7Fh=Full)
  <flag>       Flag (00h=Off, 7Fh=On)
  <instr>      Instrument (00h..7Fh) for all notes
  <lp>         Instrument (00h..7Fh) for notes 24?/36-59, lower patch number
  <up>         Instrument (00h..7Fh) for notes 60-83/84?, upper patch number
  <maj>.<min> Version (from version 1.0 to 99.99)
  <bb>         button on/off (bit0-2:button number, bit3:1=on, bit4-7:zero)
```

Data from piano is always sent on first channel (#=0). Sending data to piano can be done on first 8 channels (#=0..7), different instruments can be assigned to each channel. Although undocumented, the SNES software does initialize 16 channels (#=0..0Fh), unknown if the hardware does support/ignore those extra channels (from the instrument table: it sounds as if one could use 16 single-voice channels or 8 dual-voice channels).

Controllers - Piano - Miracle Piano Instruments

Available Patches (aka Instruments)

The following patches are available through both Library Select Mode and MIDI control:

```
  000 Grand Piano     032 Marimba         064 Synth Bells    096 Tube Bells'
  001 Detuned Piano   033 Glockenspiel'   065 Vox 1          097 Frogs/Ducks
  002 FM Piano        034 Kalimba'        066 Vox 2          098 Banjo'
  003 Dyno            035 Tube Bells      067 Vox 3          099 Shakuhachi'
  004 Harpsichord     036 Steel Drums     068 Mod Synth      100 Piano'
  005 Clavinet        037 Log Drums'      069 Pluck Synth    101 Vibraphone'
  006 Organ           038 Strings 1       070 Hard Synth     102 FM Piano'
  007 Pipe Organ      039 Pizzicato       071 Syntar         103 Clock Belis'
  008 Steel Guitar    040 Strings 2       072 Effects 1 *    104 Harpsichord'
  009 12-StringGuitar 041 Violin 1'       073 Effects 2 *    105 Clavinet'
  010 Guitar          042 Trumpet'        074 Percussion 1 * 106 Organ'
  011 Banjo           043 Trumpets        075 Percussion 2 * 107 Pipe Organ'
  012 Mandolin        044 Horn'           076 Percussion 3 * 108 Metal Guitar'
  013 Koto'           045 Horns           077 Sine Organ'    109 Stick'
  014 Jazz Guitar'    046 Trombone'       078 Organ #        110 Guitar'
  015 Clean Guitar'   047 Trombones       079 Pipe Organ #   111 Xylophone'
  016 Chorus Guitar   048 CupMuteTrumpet' 080 Harpsichord #  112 Marimba'
  017 Fuzz Guitar     049 Sfz Brass 1     081 Synth Pad 1    113 Syn Trombone'
  018 Stop Guitar     050 Sfz Brass 2     082 Synth Pad 2    114 Syn Trumpet'
  019 Harp'           051 Saw Synth       083 Synth Pad 3    115 Sfz Brass 1'
  020 Detuned Harp    052 Tuba'           084 Synth Pad 4    116 Sfz Brass 2'
  021 Upright Bass'   053 Harmonica       085 Synth Pad 5    117 Saw Synth'
  022 Slap Bass'      054 Flute'          086 Synth Pad 6    118 Church Bells'
  023 Electric Bass'  055 Pan Flute'      087 Synth Pad 7    119 Marcato'
  024 Moog            056 Calliope        088 Synth Pad 8    120 Marcato
  025 Techno Bass     057 Shakuhachi      089 Synth Pad 9    121 Violin 2'
  026 Digital Waves   058 Clarinet'       090 Synth Pad 10   122 Strings 3
  027 Fretless Bass'  059 Oboe'           091 Synth Pad 11   123 Synth Bells'
  028 Stick Bass      060 Bassoon'        092 Synth Pad 12   124 Techno Bass'
  029 Vibraphone      061 Sax'            093 Synth Pad 13   125 Mod Synth'
  030 MotorVibraphone 062 Church Bells    094 Synth Pad 14   126 Pluck Synth'
  031 Xylophone       063 Big Bells       095 Synth Pad 15   127 Hard Synth'
```

Notes:

```
 ' These programs are single voice, which lets The Miracle play up to 16
   notes simultaneously. All other programs are dual voice, which lets it
   play up to 8 notes simultaneously.
 * 072..076 See below for a list of Effects/Percussion sounds.
 # 078..080 To be true to the nature of the sampled instrument, these patches
   do not respond to velocity.
```

Effects and Percussion Patches

When selecting instruments 072..076 (Effects 1-2 and Percussion 1-3), a number of different sounds are mapped to each six keyboard keys/notes:

```
  Note      Effects 1   Effects 2    Percussion 1   Percussion 2   Percussion 3
  30-35     Jet         Yes (ding)   -              -              Ratchet
  36-41     Gunshot     No (buzz)    Kick Drum      Rim Shot       Snap 1
  42-47     RoboDeath   Applause     Snare          Exotic         Snap 2
  48-53     Whoosh      Dogbark      Toms           Congas         Dripdrum 1
  54-59     Punch       Door creak   Cymbal         Timbale        Dripdrum 2
  60-65     Slap        Door slam    Closed Hat     Cowbell        Wet clink
  66-71     Duck        Boom         Open Hat       Bongos         Talk Drum
  72-77     Ow! 1       Car skid     Ride           Whistle        Agogo
  78-83     Ow! 2       Goose        Shaker         Clave          Explosion
```

Note: The piano keys are numbered 36..84 (so notes 30..35 can be used only through MIDI messages, not via keyboard).

Controllers - Piano - Miracle Pinouts and Component List

25pin SUBD connector (J6)

```
1  PC/Amiga/Mac RS232 GND (also wired to RTS)
2  PC/Amiga/Mac RS232 RxD
3  PC/Amiga/Mac RS232 TxD
7  NES/SNES/Genesis GND
10 NES/SNES/Genesis Data
13 NES/SNES/Genesis Strobe
14 Sense SENSE0 (0=MIDI Output off, 1=MIDI Output on)
15 Sense SENSE1 (0=9600 Baud; for RS232, 1=31250 Baud; for MIDI)
19 NES/SNES/Genesis Clock
all other pins = not connected
```

For PC/Mac RS232 wire SENSE0=GND, SENSE1=GND

Miracle NES and SNES Cartridges

According to the ROM Headers: The SNES cartridge contains 512Kbyte Slow/LoROM, and no SRAM (nor other storage memory). The NES cartridge contains MMC1 mapper, 256Kbyte PRG-ROM, 64Kbyte CHR-ROM, and no SRAM (nor other storage memory).

Miracle Piano Component List (Main=Mainboard Section, Snd=Sound Engine)

```
U1   Snd  16pin TDA7053 (stereo amplifier for internal speakers)
U2   Snd   8pin NE5532 (dual operational amplifier)
U3   Snd  16pin LM13700 or LM13600 (unclear in schematic) (dual amplifier)
U4   Snd  14pin LM324 (quad audio amplifier)
U5   Main  3pin LM78L05 (converts +10V to VLED, supply for 16 LEDs)
U6   Main 14pin 74LS164 serial-in, parallel-out (to 8 LEDs)
U7   Main 14pin 74LS164 serial-in, parallel-out (to another 8 LEDs)
U8   Main  5pin LM2931CT (converts +12V to +10V, and supply for Power LED)
U9   Main  3pin LM78L05 (converts +10V to +5REF)
U10  Snd  14pin TL084 (JFET quad operational amplifier)
U11  Snd  40pin J004 (sound chip, D/A converter with ROM address generator)
U12  Snd  32pin S631001-200 (128Kx8, Sound ROM for D/A conversion)
U13  Main  3pin LM78L05 (converts +10V to VCC, supply for CPU and logic)
U14  Main 40pin AS0012 (ASIC) Keyboard Interface Chip (with A/D for velocity)
U15  Main 40pin 8032 (8051-compatible CPU) (with Y1=12MHz)
U16  Snd  40pin AS0013 (ASIC)
U17  Main 28pin 27C256 EPROM 32Kx8 (Firmware for CPU)
U18  Main 28pin 6264 SRAM 8Kx8 (Work RAM for CPU)
U19  Main 16pin LT1081 Driver for RS232 voltages
U20  Main  8pin 6N138 opto-coupler for MIDI IN signal
S1-8 Main  2pin Push Buttons
S9   Main  3pin Power Switch (12V/AC)
J1   Main  3pin 12V AC Input (1 Ampere)
J2   Main  2pin Sustain Pedal Connector (polarity is don't care)
J3   Snd   2pin RCA Jack Right
J4   Snd   2pin RCA Jack Left
J5   Snd   5pin Headphone jack with stereo switch (mutes internal speakers)
J6   Main 25pin DB25 connector (RS232 and SNES/NES/Genesis controller port)
J7   Main  5pin MIDI Out (DIN)
J8   Main  5pin MIDI In (DIN)
JP1  Main 16pin Keyboard socket right connector
JP2  Main 16pin Keyboard socket left connector
JP3  Snd   4pin Internal stereo speakers connector
```

Note: The official original schematics are released & can be found in internet.

Controllers - Keypads

TV-NET (21 Buttons)

The TV-NET controllers are roughly resembling TV remote controls (though using a cable connection, no wireless IR-transmission). There are several revisions (with differently labeled buttons):

```
Television Network  Whatever            Daiwa no My Trade
.----------------.  .----------------.  .----------------.
|     TV-NET     |  |      JAP       |  |      JAP       |
| [JP] [F1] [F2] |  | [JP] [JP] [JP] |  | [JP] [JP] [JP] |
| [F3] [F4] [F5] |  | [JP] [JP] [JP] |  | [<|] [|>] [C ] |
| (1)  (2)  (3)  |  | (1)  (2)  (3)  |  | (1)  (2)  (3)  |
| (4)  (5)  (6)  |  | (4)  (5)  (6)  |  | (4)  (5)  (6)  |
| (7)  (8)  (9)  |  | (7)  (8)  (9)  |  | (7)  (8)  (9)  |
| (*)  (0)  (.)  |  | (^)  (0)  (v)  |  | (*)  (0)  (.)  |
| (<-)     (->)  |  | (<|)     (|>)  |  | (# )     (->)  |
|     (JAP)      |  |     (JAP)      |  |     ((o))      |
'----------------'  '----------------'  '----------------'
```

Numeric keypads for use with TV-NET modems:
Modems

Famicom Network Controller (HVC-051) (23 Buttons)

```
.----------------------------------------.
```

```
| (<)   (>)  (1) (2) (3) (*) (C)           |
| SEL _ STA                      (JAP)     |
|   _| |_    (4) (5) (6) (#) (.)           |
|  |_   _|                                 |
|    |_|     (7) (8) (9) (  0  )  (B)  (A) |
'-----------------------------------------'
```

Joypad with numeric keypad for use with Famicom Network modems:
Modems

Power Glove Numeric Keypad
The Power Glove does also have some sort of numeric keypad. In normal mode, the keypad is used to configure the glove (for using it with games without built-in glove-support). When entering a special glove mode, it's also possible to read the keypad buttons by software.
Controllers - Power Glove

Jissen Mahjong Controller (Capcom) (21 Buttons)
The top row has 14 buttons (square "A-M" keys, and a round "N" key), these keys are used to drop one of the 14 cards (or if one has less than 14 cards, then "N" draws a new card).
The bottom row has 7 buttons (SEL, ST, and some japanese symbols), these are function keys (Select and Start, and whatever special "japanese" functions).

```
                      ...--------...
   .------------------''    CAPCOM    ''------------------.
  /                     lt  up  rt                         \
 |  [A] [B] [C] [D] [E] [F] [G] [H] [I] [J] [K] [L] [M] (N)  |
 |                                                          |
 |                      [F1][F2][F3][F4][F5][F6][F7]        |
 |                      lt  dn  rt                          |
  \                                                        /
   '------------------------------------------------------'
```

Reading is done as:

```
[4016h]=5, [4016h]=4, read 8bits from [4017h].Bit1    ;row 0
[4016h]=3, [4016h]=2, read 8bits from [4017h].Bit1    ;row 1
[4016h]=7, [4016h]=6, read 8bits from [4017h].Bit1    ;row 2
```

The bits returned are:

```
Read   Row 0          Row 1          Row 2
1st    H (Right)      None?          None? (Change/Cheat?)   ;\
2nd    G (Up)         None?          Fn Unknown(EndGame?)    ; All buttons:
3rd    F (Left)       N (New/Okay)   Fn Unknown(Tingeling?)  ;  1=Low=Pressed
4th    E              M              Fn Unknown(Jp?)         ;  0=High=Released
5th    D              L              Fn Unknown(Jp?)         ; Unused bits:
6th    C              K              Fn Unknown(Right/Toggle);  ?=What=Unknown
7th    B              J              F2 Start  (Down)        ;
8th    A              I              F1 Select (Left)        ;/
9th..  Unknown(N/A?) Unknown(N/A?) Unknown(N/A?)           ;-Probably padding
```

Translation for the text on the 5 japanese functions keys is unknown (the left-most one apparently toggles setup options); Location of the japanese function keys in the return data is unknown (probably in Row 2, 2nd..6th, probably ordered as 6th read left-most, through 2nd read right-most).
There are only two known supported games:

```
Ide Yousuke Meijin no Jissen Mahjong (J) Capcom (1987)
Ide Yousuke Meijin no Jissen Mahjong 2 (J) Capcom (1991)
```

There are at least three (cosmetic) revisions of the Mahjong controller; mostly different text/logos in lower-left section. The "II" version has a black case, additional arrow symbols (LT-UP-RT above F-G-H keys, and LT-DN-RT below SEL-STA-JP keys) and a blue rectangle around M-key.

Famicom Titler
A special NES console from Sharp. The thing has built-in Tablet and Keypad.

Controllers - Mats

Power Pad (Dance Mat) (Bandai/Nintendo)
The Power Pad aka Family Trainer aka Family Fun Fitness is a device made by Bandai/Nintendo (1987/1988) which serves as an "exercising fun center" for the whole family. That is, a large (1x1 meters) vinyl mat with 12 touch-sensitive areas, or actually step-sensitive areas, since it's intended to be put on the floor. The mat has two sides, with different patterns drawn on each side. Supported games are:

```
Athletic World (J) (U) (E) 1986-1988
Class Track Meet/Running Stadium/Stadium Events (J) (U) (E) 1986-1988
Aerobics Studio/Dance Aerobics (J) (U) 1987/1989
Fuuun! Takeshi Shiro 2 (J) 1988
Jogging Race (J) 1987
Meiro Daisakusen (J) 1987
Power Pad Test Program by Tennessee Carmel-Veilleux (PD) (UE)
Rai Rai! Kyonshiizu: Baby Kyonshi no Amida Daiboken (J) 1989
Short Order / Eggsplode! (U) 1989
Street Cop/Manhattan Police (J) (U) 1987/1989
Super Team Games/Daiundoukai (J) (U) 1987/1988
Totsugeki! Fuuun Takeshi Shiro (J) 1987
```

Tap-tap Mat (igs) (Japan only)

Another mat, this one is smaller than the Power Pad mat, and one is supposed to hit the fields with a plastic hammer. Used by only one known game:

```
Super Mogura Tataki!! - Pokkun Moguraa (J) (bundled with mat & hammer)
```

NES Version I/O Access (outside Japan) (Bandai/Nintendo Mats)

Connects to 7pin controller port (typically the mat connects to port 2, and a normal joypad for menu selections to port 1). To read the button states:

```
Output 1-then-0 to Bit0 of Port 4016h
Read eight times from Port 4017h (or 4016h, when plugged into port 1)
```

Each read receives two button states in Bit 3 and 4, in following order:

```
Bit4  4,3,12,8,u,u ,u ,u (0=High=Released, 1=Low=Pressed) (u=Unused always 1)
Bit3  2,1,5 ,9,6,10,11,7 (0=High=Released, 1=Low=Pressed)
```

Whereas, "1-12" are button numbers as drawn on Side B, or equivalent buttons on Side A (horizontally mirrored, of course).

FAMICOM Version I/O Access (Japan) (Bandai/Nintendo Mats and igs Mats)

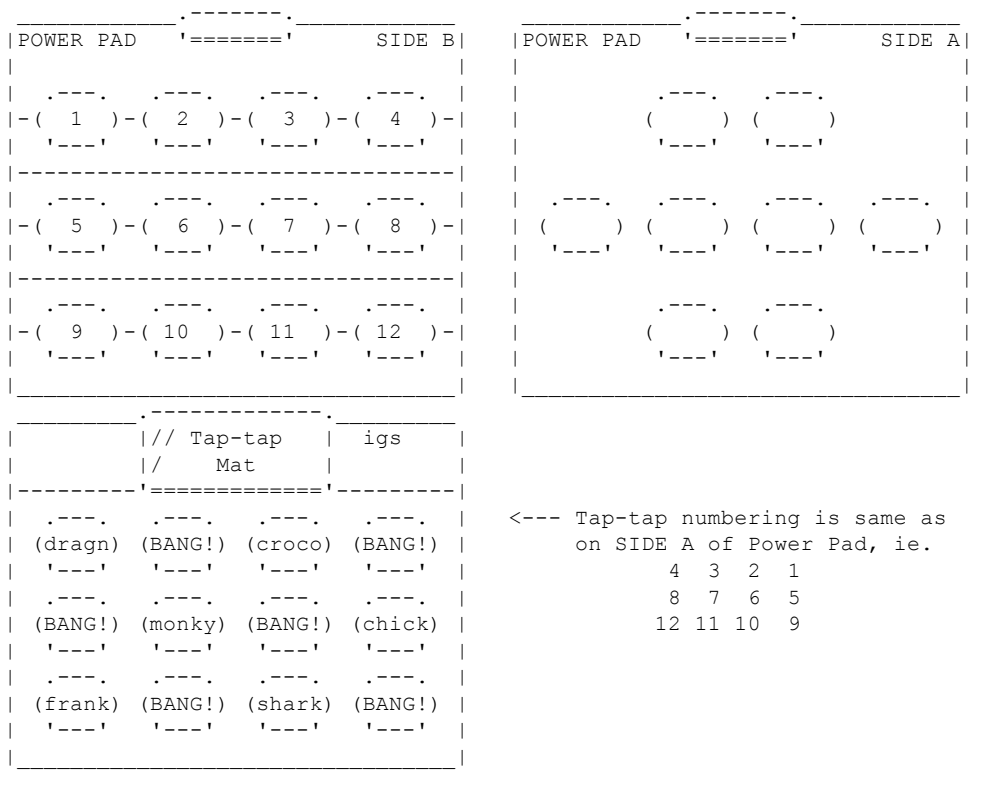
Connects to 15pin expansion port. Unlike in NES version, the buttons are read via a simple row/column matrix (without shift-register). The overall I/O access is same for Nintendo/Bandai-mats and igs-mats:

```
4016h.W.Bit0 Select Lower row  (9..12) (0=Low=Select, 1=High=No)
4016h.W.Bit1 Select Middle row (5..8)  (0=Low=Select, 1=High=No)
4016h.W.Bit2 Select Upper row  (1..4)  (0=Low=Select, 1=High=No)
4017h.R.Bit1 Read Right-most column   (4,8,12) (0=High, 1=Low=Pressed)
4017h.R.Bit2 Read Right-middle column (3,7,11) (0=High, 1=Low=Pressed)
4017h.R.Bit3 Read Left-middle column  (2,6,10) (0=High, 1=Low=Pressed)
4017h.R.Bit4 Read Left-most column    (1,5,9)  (0=High, 1=Low=Pressed)
```

After selecting a Row one should execute some delay before reading Column data: For the Nintendo/Bandai-mats this should be a huge 1800 cycle delay. The igs-mat can be used with shorter 138 cycle delays, but results might be unstable (the tap-tap game reads all data twice, and retries reading until both results are same).

Power Pad Layout (Side A and Side B)

Side A has 2 red fields, 6 blue fields, and 4 hidden fields.
Side B has 12 fields, numbered 1..12, the left fields (1,2,5,6,9,10) are blue, the other are red.



Controllers - Inflatable Controllers

Exciting Boxing Bop Bag (Konami) 1987

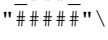
Inflatable boxing bag with 8 sensors, used only by one game (Exciting Boxing).

```
4016h.W.Bit1    Row Select    (0=Hook/Move or 1=Jabb/Straight/Body)
4017h.R.Bit1-4  Column Inputs (0=High=Punch, 1=Low=None)
```

There seems to be a delay required after changing the Row Select bit: The game seems to read one 4bit column per frame, then toggle the row select bit, and then read the other 4bit in next frame.

```
4017h.R  Row0 (OUT1=0)                  Row1 (OUT1=1)
Bit4     Right Hook (left ear)          Straight (breast/chin)
Bit3     Move? (to left screen-edge)    Right Jabb (left shoulder)
Bit2     Move? (to right screen-edge)   Body (belly)
Bit1     Left Hook                      Left Jabb (right shoulder)
```

The location of the 2 move sensors is unknown (not used in Training mode; used only in VS mode). The average location of the 6 punch-sensors can be seen in Training mode, as so:



```
      (ear) Row0.Bit1 --> (| o\ /o |)   <-- Row0.Bit4 (ear)      (hook)
                           \   _   /
                           / \_o_/ \  <---- Row1.Bit4 (breast)   (straight)
  (shoulder) Row1.Bit1 --> |    "    |   <-- Row1.Bit3 (shoulder) (jabb)
                           | ()   () | <---- Row1.Bit4 (belly)    (body)
                           |_/     \_|
                            |=======|
                            |  /-\  |
                            |==| |==|     <-- inflatable boxer
                            |  | |  |
                           _|__|#|__|_    <-- control box
                          (___________)   <-- inflatable ring
                           \           \
                            \___________\   <-- floor mat
```

The type of all 8 sensors is unknown, presumably some kind of pressure or shock sensors. The size of the bag is unknown (looks like around 100..150cm in height on photos). Unknown how the thing is stabilized; maybe one is intended to sit on the floor mat... or the ring or mat are meant to be filled with water... or so?

Top-Rider Bike (Varie) 1988

Inflatable motor-bike with handle-bars mounted on the front end.

```
4016h.W.Bit1    Start some conversion or so?
4016h.W.Bit0    Reload shift-registers
4017h.R.Bit3    8bit shift-register
4017h.R.Bit4    8bit shift-register
```

Accessing the bike seems to require special start-bit and some delays:

```
[4016h]=03h  ;start some conversion or so?
wait 60 clks (game tries so, but bugged code waits only 10 clks)
wait some more (whatever overload, additionally to above delay)
[4016h]=01h  ;strobe on
wait some clks (strobe on should last for 12 clks)
[4016h]=00h  ;strobe off
read 8 times from [4017h].Bit3-4
```

The separate bits are:

```
Read  4017h.R.Bit3                        4017h.R.Bit4
1st   Accel, Bit5 (0=VCC=Zero) Turbo      Unknown/unused?   (?)
2nd   Accel, Bit4 (0=VCC=Zero)            Steer Right, Bit1 (0=VCC=Zero, 1=One)
3rd   Accel, Bit3 (0=VCC=Zero)            Steer Left, Bit1  (0=VCC=Zero, 1=One)
4th   Accel, Bit2 (0=VCC=Zero)            Steer Right, Bit0 (0=VCC=Zero, 1=One)
5th   Accel, Bit1 (0=VCC=Zero) Fast       Steer Left, Bit0  (0=VCC=Zero, 1=One)
6th   Accel, Bit0 (0=VCC=Zero) Slow       Hand Brake  (0=VCC=No, 1=GND=Pulled)
7th   Start Button  (0=VCC=Released)      Gear Switch (0=VCC=Lo, 1=GND=Hi)
8th   Select Button (0=VCC=Released)      Wheelie     (0=VCC=No, 1=GND=What?)
```

Of the 6bit accelerate value, only a few values are actually used: 01h=Slow, 02h=Fast, 20h=Turbo, and 00h/03h..1Fh/21h..3Fh=Stop; details are unknown, it's probably not a 6bit analog values, maybe a 7-position switch with 1bit per position; and no bit for the stop-position)?

The 4bit steering value allows 7 directions (forward, and 3 levels of left/right each) (settings with both left and right bits nonzero are invalid); details are unknown, maybe analog input, or a 7-position switch.

The gear switch is a switch (not a push button), and so, remains in the most recently selected position.

Pulling the Hand Brake obviously pushes a button. Start/Select are on the bases front panel, but unknown is Start is on left or right side.

Unknown how to trigger the Wheelie function; the picture on the box depicts a function near ring finger of left hand, maybe there's a button underneath of the left handle, or maybe it's possible to lift/pull the handles?

The length/height/width of the bike is unknown (judging from the photo on the box, it seems to be made for small children). There's reportedly some risk to pop the bike when used by heavy adults, unknown if that's just panic or an actual problem.

```
       ______ something near ring finger? maybe wheelie-button or so?
      |      ___ gear switch
      |     |          _________ mimmicked km/h and rpm display
      |     |         |      |                ____ handbrake
      |     |         v      v               |
      |     |    .----------------.    __     v          turn handle-bar
     v     v    |  .--.   .--.  |  |  |=====O        to steer left/right?
  ._______ _____| |    | |    | |__|  |/______.
  |________|  |__|  '--' o '--' o|__|__|._______| <--- turn right handle
  '       '|__| :'----------------':             '      to accelerate?
           #   :  (S??)   (S??)  :
              : '._____________.' : <------ base with start/select
              :                   :
              :                   : <------ inflatable pillow to sit on
              :                   :
```

Controllers - RacerMate Bicycle Training System

CompuTrainer by RacerMate. Used only by RacerMate Challenge II.

```
4016h.W.Bit0  OUT0 (to be forwarded to TX1/TX2 on 4016h/4017h.reads)
4016h.R.Bit0  Get RX1 (and forward OUT0 to TX1) (Player 1)
4017h.R.Bit0  Get RX2 (and forward OUT0 to TX2) (Player 2) (if any)
```

The transfer rate is generated by an IRQ timer in the cartridge (the timer's baudrate is unknown; the transfers do require at least 27 IRQs per 60Hz frame, so the IRQ rate must be at least 1620 Hz or higher). In practice, 1024 clks per IRQ seems to be too slow (video glitches), 512 clks too fast (1/10 second

counter increases non-linear), so, the IRQ rate is probably something around 768 clks.

Transfer Packets (per player) (48 TX Bits & 48 RX Bits)

```
 First Frame:
  2 IRQs   Leading Pause (TX levels left unchanged)
  1 IRQ    Output 1st TX Bit (Start Bit "0")   Input nothing
  23 IRQs  Output 2nd..24th TX Bit             Input 1st..23rd RX Bit
  1 IRQ    Output nothing (keep 24th TX bit)   Input 24th RX Bit
  0 IRQs   Ending pause (IRQs should be disabled until next Vblank NMI)
 Second Frame:
  2 IRQs   Leading Pause (TX levels left unchanged)
  1 IRQ    Output 25th TX Bit (Start Bit "1")  Input nothing
  23 IRQs  Output 26nd..48th TX Bit            Input 25th..47rd RX Bit
  1 IRQ    Output nothing (keep 48th TX bit)   Input 48th RX Bit
  0 IRQs   Ending pause (IRQs should be disabled until next Vblank NMI)
```

Both the 48bit-RX and 48bit-TX streams are interleaved:

```
  24 "odd" bits  (1st, 3rd, 5th, 7th, ... 47th bits)
  24 "even" bits (2nd, 4th, 6th, 8th, ... 48th bits)
```

Odd TX Bits (1st, 3rd, 5th, 7th, ... 47th TX bits) (24bit from NES to Bike)

```
  1bit  Must be "0" (start bit of packet-half transferred in First Frame)
  6bit  Unknown (seems to be fixed: 1,1,0,1,0,0)   ;(aka 2Ch/4, LSB first)
  5bit  Data, Bit0..4  (LSB first)
  1bit  Must be "1" (start bit of packet-half transferred in Second Frame)
  7bit  Data, Bit5..11 (LSB first)
  4bit  Index, Bit0..3 (LSB first)
```

The Index/Data Pairs are:

```
  Index=1, Grade (signed)
  Index=2, Wind (signed)
  Index=3, Weight (LBS)
  Index=4, Pulse Target Min (BPM+800h) ;\player 1 only
  Index=5, Pulse Target Max (BPM+800h) ;/
  Index=5, Zero                        ;-player 2 only
  Index=F, Start Race (Data=001h)
```

Unknown if there are further ones... eg. to stop/pause/resume race?

Even RX Bits (2nd, 4th, 6th, 8th, ... 48th RX bits) (24bit from Bike to NES)

```
  1st EvenRxByte --> Button flags (bit0-5) SpecialStart (bit6)
  2nd EvenRxByte --> 8bit Data Fragment
  3rd EvenRxByte --> 4bit Data Fragment (LSBs), 4bit Index (01h..0Eh) (MSBs)
```

The 12bit data fragments are stored in an array (with index 01h..0Eh), there are thus 14x12bit = 168 bits in that array. There are two separate arrays (one per bike).

```
  XXX content of the array is unknown (presumably speed and such things?)
```

The Button bits are:

```
  0  Button RESET                  (0=High=Released, 1=Low=Pressed)
  1  Button F1 (START/PAUSE/LEAVE) (0=High=Released, 1=Low=Pressed)
  2  Button F3 (SET/SELECT)        (0=High=Released, 1=Low=Pressed)
  3  Button +  (PLUS/UP)           (0=High=Released, 1=Low=Pressed)
  4  Button F2 (DISPLAY)           (0=High=Released, 1=Low=Pressed)
  5  Button -  (MINUS/DOWN)        (0=High=Released, 1=Low=Pressed)
  6  Start of Special Odd RX Bits  (?)
  7  Unknown/unused?               (?)
```

The Index/Data Pairs are:

```
  Index=0     Unknown/Ignored
  Index=1     Speed 12bit (0..FFFh = 0..81.9 MPH)  (ie. 1/50 MPH units)
  Index=2     Watts 12bit (0..FFFh = 0..3054 Watts) (roughly 3/4 Watt units)
  Index=3     Pulse 8bit (0..FFh = 0..255 BPM) (and MSBs = 4bit Flags)
  Index=4..E  Unknown/Stored in memory (but seems to have no effect)
  Index=F     Unknown/Ignored
  Unknown if/how/where RPM is transferred.
  Speed (MPH) also affects distance (DST).
```

The 4bit flag-field for the Pulse data is:

```
  Bit8  Blink Heart-Symbol
  Bit9  Show "E"
  Bit10 Show "LO"  ;\when both set: treated as "sensor not connected"
  Bit11 Show "HI"  ;/(ie. showing "--" instead of the pulse value)
```

Even TX Bits (2nd, 4th, 6th, 8th, ... 48th TX bits) (24bit from NES to Bike)

These bits are simply inverse bits that are preceeding the following Odd TX Bits. Ie. a "0" bit will be transferred as "1-then-0" bit-pair, and "1" bit as "0-then-1". The receiver (control panel) may use that transitions for synchronization (eg. in case of baudrate inaccuracies).

In case of the 24th/48th TX bits, this "wraps" to next frame, ie. they are inverse of the of the start-bit of the next frame.

Odd RX Bits (1st, 3rd, 5th, 7th, ... 47th RX bits) (24bit from Bike to NES)

These bits can be inverse data bits that are preceeding the normal Even RX Bits (similar as for TX).

Alternately, the odd RX bits can contain special information (probably an extension of the original transfer protocol). Presence of that special info is determined as so:

```
  (EvenRxBytes <> (OddRxBytes XOR FFFFFFh)) AND Parity(1stOddRxByte)=odd
```

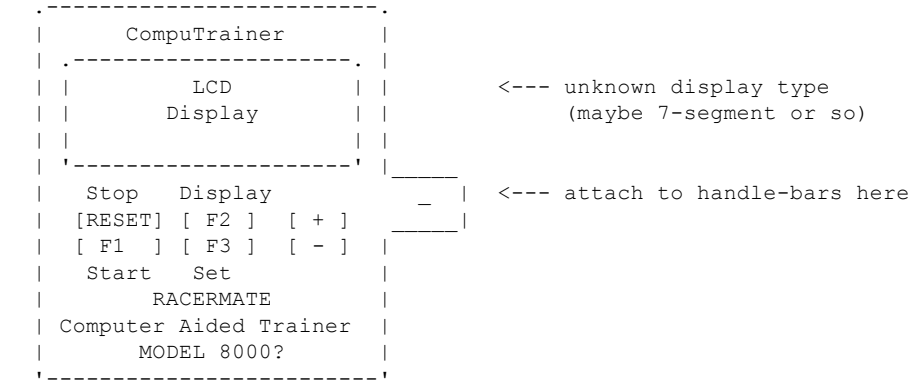
If that condition is true, then the 3 odd bytes are a fragment of a 24-byte special array; it does thus take 8 packets to transfer the whole array; the array-start

is indicated by bit6 of the "button" byte.

```
XXX content of the array is unknown ...
XXX the software seems to manage only ONE array (not 2 arrays for 2 bikes)
```

Control Panel

The control panel connects to both controller ports, which leaves no room for connecting joypads or other controllers. So, menu selections are solely controlled by the 6 buttons on the control panel. The only other inputs are the rear-wheels rotation speed, and the pulse sensor.



Components

Hardware at Bike Side:

```
Control Panel (LCD display, 6 buttons, wired to flywheel, NES, pulse sensor)
Front Stand (small stand, holds front-wheel in straight-forward direction)
Rear Stand with electronic flywheel (resistance, speed-meter, stand upright)
Power Supply (for flywheel)
Pulse sensor, cables, screws
Bicycle (not included) (can be used with regular "ourdoor" bicycles)
```

Hardware at NES Side:

```
NES Connector Box (2x7pin NES connector, 2x3pin 3.5mm sockets; for 2 bikes)
NES Console with CIC disabled (or top-loading NES without CIC)
NES Challenge II cartridge (without CIC, except, old versions had CIC clone)
```

There are several hardware revisions (including some for newer computers/consoles). For the NES, there are at least two front-panel variants (with connectors located in different places), and at least two rear-stand/fly-wheels (marked CompuTrainer, or CompuTrainer Pro), and, there are reportedly several EPROM versions:

```
3.11.088 05-02-1991 20:41:04
5.01.033 02-01-1994 19:28:10
5.01.036 05-15-1996 19:47:57
6.02.002 05-21-1996 12:22:18
9.03.128 (PAL) 03-22-1996 11:57:11
```

The game does also require a special mapper (and a NES with CIC disabled):
Mapper 168: RacerMate PRG/16K, VRAM/4K, IRQ

Controllers - Tablets

Games

```
Oeka Kids - Anpanman no Hiragana Daisuki (J)
Oeka Kids - Anpanman to Oekaki Shiyou!! (J)
```

Controller Access

```
4016h.W.Bit0  Oeka Kids Tablet Strobe (inverse of normal joypad strobe)
4016h.W.Bit1  Oeka Kids Tablet Clock (manually clocked, unlike joypads)
4017h.R.Bit2  Oeka Kids Tablet Ack (confirm Strobe/Clock signals)
4017h.R.Bit3  Oeka Kids Tablet Data (18bits)
```

To start the transmission & read the 18 databits:

```
[4016h]=00h, wait for [4017h].Bit2=0  ;\strobe 0-then-1 (start transfer)
[4016h]=01h, wait for [4017h].Bit2=1  ;/
[4016h]=03h, wait for [4017h].Bit2=0  ;\read databit (repeat 18 times)
databit = [4017h].Bit3                ; (note: the final wait after 18th
[4016h]=01h, wait for [4017h].Bit2=1  ;/bit can/should/must? be ommitted)
```

The separate databits are:

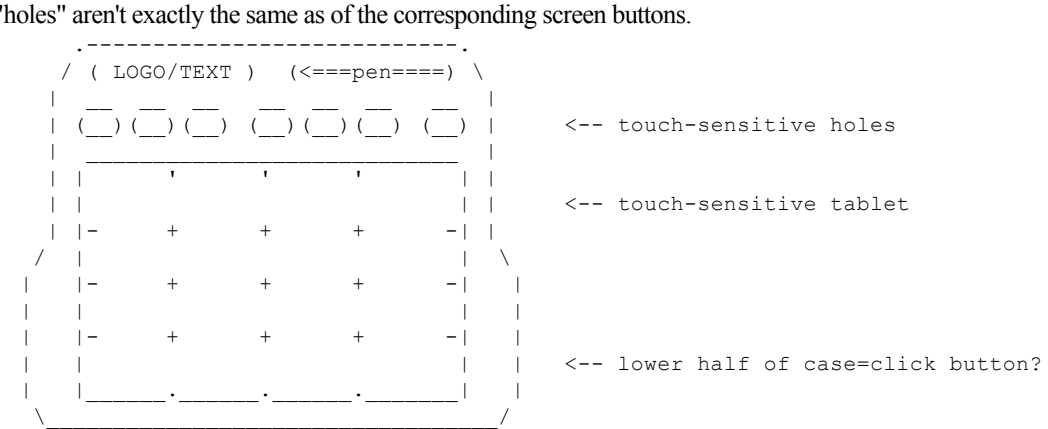
```
1st..8th   X-coordinate, MSB first, inverted  (0=High=One, 1=Low=Zero)
9st..16th  Y-coordinate, MSB first, inverted  (0=High=One, 1=Low=Zero)
17th       Soft-Pressure (Pen touches tablet) (0=High=Yes, 1=Low=No)
18th       Hard-Pressure (Click/Draw) ??????? (0=High=Yes, 1=Low=No)
  XXX or is there a "click" button; the gray bar at front? So one must
  push that button, and the thing can't sense "hard pressure" at all?
```

To convert tablet coordinates to 256x240 pixel screen coordinates:

```
tablet.x = tablet.x XOR FFh                  ;\undo inversion of databits
tablet.y = tablet.y XOR FFh                  ;/
screen.x = tablet.x + (tablet.x/10h) - 08h   ;\scale to screen dimensions
screen.y = tablet.y - (tablet.y/10h) + 0Dh   ;/
```

```
  screen.x = minmax(screen.x,00h..FFh)          ;\clip to min/max range
  screen.y = minmax(screen.x,00h..EDh)          ;/
  (of course, the coordinates are valid only if the pen touches the tablet)
```

The touch area is divided into eight sections: seven small "holes" at the top of the tablet, and the main drawing area. Note: The horizontal coordinates of these "holes" aren't exactly the same as of the corresponding screen buttons.



Famicom Titler

A special NES console from Sharp. The thing has built-in Tablet and Keypad.

Controllers - Trackball and Mouse

Hori Track

```
  Moero Pro Soccer (J) 1988 Jaleco
  Operation Wolf (J) (U) 1989 Taito (also supports Zapper lightgun)
  Putt Putt Golf (FDS) 1989 Pack-In-Video
  US Championship V'Ball (J) 1989 Technos Japan Corp
```

Hori seems to have planned both NES and Famicom versions of the controller; as far as known, the NES version wasn't released, but, most of the existing games (except Putt Putt Golf) do contain both NES and Famicom controller protocol versions:

Version	Input Data from	ID Bits	Connector	Released
Famicom	[4016h/4017h].Bit1	0,1	15pin	Yes
NES	[4016h/4017h].Bit0	1,0	7pin	No

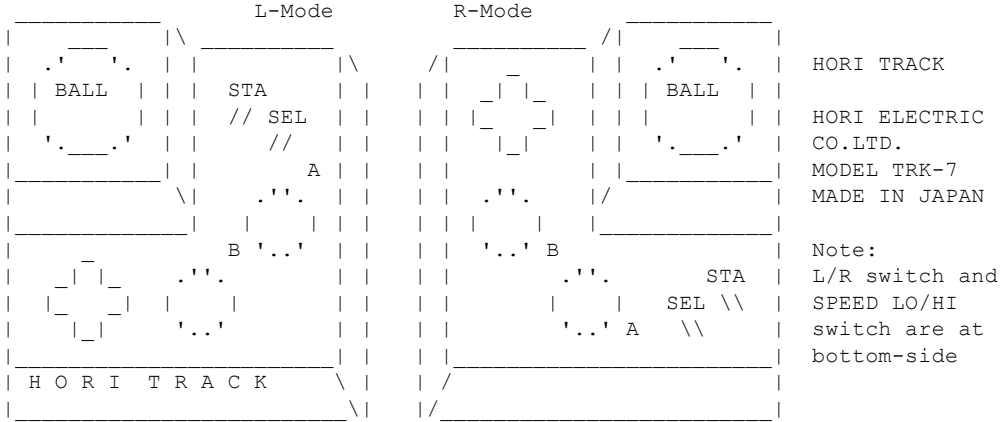
Controller Bits:

```
  (to be preceeded by normal joypad like 1-then-0 strobing on OUT-0)
  1st..8th bit   --> Same as normal joypad data
  9th..12th bit  --> Axis 1, signed 4bit (MSB first, inverted, 1=Low=Zero)
  13th..16th bit --> Axis 2, signed 4bit (MSB first, inverted, 1=Low=Zero)
  17th           --> L/R mode switch (0=High=R-Mode, 1=Low=L-Mode)
  18th           --> Unknown/unused (probably SPEED LO/HI switch) (=?)
  19th           --> ID Bit1 (0=High=Famicom, 1=Low=NES)
  20th           --> ID Bit0 (0=High=NES, 1=Low=Famicom)
  21th..24th     --> Unknown/unused (read by software, but seems to be unused)
  25th and up    --> Unknown/unused (probably whatever padding bits)
```

Trackball Orientation:

```
  When 17th Bit=1:  (L-Mode) (supported by all games)
   Axis 1 is to be treated as Y-axis (POSITIVE = DOWN = towards DPAD)
   Axis 2 is to be treated as X-axis (POSITIVE = RIGHT = towards START/SELECT)
  When 17th Bit=0:  (R-Mode) (not supported by Operation Wolf)
   Axis 1 is to be treated as X-axis (POSITIVE = LEFT = towards DPAD)
   Axis 2 is to be treated as Y-axis (POSITIVE = DOWN = towards START/SELECT)
```

DPAD orientation is unknown (according to manual, it sounds like UP=Towards Ball, RIGHT=Towards B-Button) (in software, this should be probably always kept handled as so, regardless of the L/R switch).



Joyball Note

There's also a roughly similar looking controller called Joyball. However, that thing is just a regular joystick with "ball-shaped" handle, not a trackball.

Subor Mouse

The Subor Mouse is bundled with a keyboard-shaped NES clone; the mouse is supported by only one known cartridge (bundled with the mouse & NES clone):

```
Educational Computer 2000 (RU) (with mimmicked russian Win95 GUI)
```

Reading a mouse byte is done as so:

```
[4016h]=01h
wait 28 clks (14 NOPs)
[4016h]=06h
read 8bits from [4017h].R.Bit0 (MSB first) (bits are NOT inverted!)
```

Mouse response can be a single-byte (max +/-1 mickey), or 3-byte (max +/-31 mickeys). The response type & index is indicated in lower 2bit of the bytes.

Single-Byte Packet:

```
7     Left Button     (1=Pressed)
6     Right Button    (1=Pressed)
5-4   Step X (0=None, 1=Plus 1/Right, 2=Treated same as 1, 3=Minus 1/Left)
3-2   Step Y (0=None, 1=Plus 1/Down,  2=Treated same as 1, 3=Minus 1/Up)
1-0   Must be 0 for single-byte packet
```

1st-byte of Three-byte Packet:

```
7     Left Button     (1=Pressed)
6     Right Button    (1=Pressed)
5     Direction X (0=Plus/Right, 1=Minus/Left)
4     Unsigned Step X, Bit4
3     Direction Y (0=Plus/Down, 1=Minus/Up)
2     Unsigned Step Y, Bit4
1-0   Must be 1 for 1st byte of multi-byte packet
```

2nd-byte of Three-byte Packet:

```
7-6   Unused (maybe copy of buttons)
5-2   Unsigned Step X, Bit3-0
1-0   Must be 2 for 2nd byte of multi-byte packet
```

3rd-byte of Three-byte Packet:

```
7-6   Unused (maybe copy of buttons)
5-2   Unsigned Step Y, Bit3-0
1-0   Must be 3 for 3rd byte of multi-byte packet
```

Threshold must be implemented by software (as done in Educational Computer 2000):

```
00h..0Eh Mickeys ---> Mul 1.0 --> 00h..0Eh Pixels
0Fh..13h Mickeys ---> Mul 1.5 --> 16h..1Ch Pixels
14h..18h Mickeys ---> Mul 2.0 --> 28h..30h Pixels
19h..1Dh Mickeys ---> Mul 2.5 --> 3Eh..48h Pixels
1Eh..1Fh Mickeys ---> Mul 3.5 --> 69h..6Ch Pixels
```

Motion counters are reset to zero after reading. Data should be read per NMI (read at least one byte, and, in case of 3-byte response: read all 3 bytes).

Caution:

The Subor Clone does also contain a keyboard (and joypads), reading the controllers might somehow interfere. Namely, Subor is doing the keyboard reading AFTER and ONLY AFTER single-byte-mouse reads (in case of multi-byte-mouse reads, it's completely omitting keyboard reading in that frame) (though unknown if that kind of handling is really required).

Controllers - Typewriter Keyboards

Controllers - Power Glove

Controllers - Power Glove Transmission Protocol (RX/TX)

Transmit (TX) (Configuration)

Transmission mode is entered by outputting a LONG strobe signal:

```
Set [4016h].W.Bit0=1, then wait around 3330 clks      ;-Enter TX Mode
```

Thereafer, the TX packet can be sent bit-by-bit:

```
Set [4016h].W.Bit0=Databit                           ;-Output Data Bit
Do dummy-read from [4016h].R                         ;-Output Clk Pulse
```

Bytes are sent MSB first. Insert a small delay between each two bytes:

```
1280 clks should be fine; as used by Super Glove Ball ;-Delay between bytes
(the older Bad Street Brawler game used 2304 clks)
```

For the standard Analog mode with 9-byte response, the transmitted packet should be usually following 7 bytes: 06h,C1h,08h,00h,02h,FFh,01h. For details on other TX packets, see:

Controllers - Power Glove TX Packets (Configuration Opcodes)

After the transfer, the value of the lastmost databit may be left on the strobe line; the glove doesn't seem to treat this as new LONG strobe.

Receive (RX) (Read Sensors/Buttons)

Read one Status Byte per frame (ie. via 60Hz NMI handler), just like normal joypad reading:

```
Set [4016h].W.Bit0 to 1-then-0                          ;-Strobe
Read 8bits from [4016h].R.Bit0 (MSB first, inverted)    ;-Read Byte
```
This byte indicates if the glove is ready to send a new data packet:
```
5Fh (received) aka A0h (when XORing by FFh)    --> analog mode, ready
00h? (received) aka FFh? (when XORing by FFh) --> analog mode, not ready (?)
other (joypad data)                            --> joypad emulation mode
```
If the Status Byte indicates Ready, then the next some bytes (usually 9 bytes) are packet data, these can be read as so:
```
wait 100 clks                                           ;-Delay between bytes
Set [4016h].W.Bit0 to 1-then-0                          ;-Strobe (on each byte)
Read 8bits from [4016h].R.Bit0 (MSB first, inverted)    ;-Read Byte
XOR byte by FFh                                         ;-Undo inversion
```
For details on the packet content, see:
Controllers - Power Glove RX Packets (Position/Sensor Data)
After the transfer, a long pause may be required (it seems as if the glove terminates RX mode via a timeout); best wait until next 60Hz frame before trying to receive a new packet.
The conversion time isn't constant (for example, flex A/D conversion time seems to increase when making a tight fist). Normally, status "Ready" should be received around every 3 frames. Generate a timeout if ready isn't seen after 20 frames; that might happen if the glove isn't connected, or not initialized (not in analog mode), or if the user has hit the PROG button.

Caution
Reading [4016h].R.Bit0 does, of course, work on NES ONLY. Unknown how to read data from the japanese PAX Power Glove; [4016h].R.Bit1 should be a good guess.

The existing games expect the glove connected to port 1 (4016h.R), alternately it should also work in port 2 (4017h.R). When trying to connect two gloves, the ultrasonic speakers would disturb each other, so only Flex and Buttons would work.

Controllers - Power Glove TX Packets (Configuration Opcodes)

Overall TX Packet Format
The overall transmit packet format is:
```
1. Length Byte (total number of following bytes) (used range is 05h..32h)
2. 16bit Opcode Area (convert analog positions to temporary flags)
3. 8bit Opcode Area (forward temporary flags to joypad bits; can use logic)
4. Optional 6 extra bytes (whatever purpose, used only in one program)
```

Analog Mode TX Packets
To enter the Analog mode, this should be usually the following 7 bytes:
```
06h      Length, total number of following bytes
C1h      Analog Mode (bit7), one 16bit opcode (bit3-0)
08h,00h  Opcode 0800h (maybe analog request, or maybe just a dummy-opcode)
02h      Two 8bit "opcodes" (in Analog mode, they are "masks", not "opcodes")
FFh,01h  Mask Word 01FFh (bit0-8: request 1st..9th response byte)
```
The 7th..8th response bytes contain unknown/unused values, changing the mask from 01FFh to 013Fh should omit them (and save around 500 clks per received packet); the Error Flags (normally in 9th byte) would then appear in 7th byte. Setting Mask bit9..15 causes the glove to send extra garbage bytes.

Digital Mode TX Packets
Digital mode would be required only if you want to use the joypad emulation mode (ie. cripple the 3D analog position data to 2D digital joypad signals). For details on doing that, read on below:

========== 16bit Opcode Area (Part 1) ==========

Header Byte for 16bit Opcode Area
```
7    Low level mode  (0=Digital Joypad Emulation, 1=Analog Low-Level Mode)
6    Unknown (always 1 in known packets)
5    Configure glove for use with other games (1=Survive POWER-OFF ?)
4    Unknown (always 0 in known packets) "prevent re-flash later"?
3-0  Number of following 16bit commands (used range is 01h..09h)
```

16bit Opcodes
The upper 8bit (first byte) are:
```
15   Unknown (always 0 in known packets)
14   Unknown (maybe only-if-NEWLY?)   ;-can be used together with bit13-8
13   Examine Thumb Finger Flex        ;\
12   Examine Index Finger Flex        ; only one of these
11   Examine Middle Finger Flex       ; bits should be set
10   Examine Ring Finger Flex         ; (usually)
9    Examine Wrist Rotation Angle     ;
8    Examine X/Y/Z Coordinate         ;/
```
The lower 8bit (second byte) can have following uses:
For Finger Flex:
```
7-4  Unknown (usually 0) (except, can be 1 when checking TWO fingers...?)
3-2  Wanted flex or so   (3 or 2=Want Flex, 0=Want NO Flex)
1-0  Unwanted flex or so (0 or 2=Want Flex, 3=Want NO Flex)
```

For Wrist Rotation:

```
  7-4  Max Angle (00h..0Bh)  ;\eg. wanted clock range 3:00 .. 6:00 should be
  3-0  Min Angle (00h..0Bh)  ;/defined as Min=3, Max=6; or maybe Max=7, or so
```

For X/Y/Z Coordinate:

```
  7-3  Unknown, maybe flag(s) and/or boundary value
  2    Select Direction (0=Right/Up/Back, 1=Left/Down/Forward)
  1-0  Select Parameter (0=X, 1=Y, 2=Z, 3=Fourth Axis??)
```

The opcodes are computing if the glove is moved in the specified way, and the result is then stored in a flag array: Input(1..9) = Result of 1st..9th 16bit opcode (used as "Input" to the 8bit Opcodes in Part 2).

Unknown/uncommon 16bit opcodes

Some very strange ones:

```
  1D18           ;   right??       ;\these are used in combination with
  0000           ;   left ??       ; "normal" left/right opcodes...
  1D18           ;   ...  ??       ; might be somehow related to near & far
  0000           ;   ...  ??       ;/transformed to slow/pulsed & fast/normal
  (Maybe flex on 3 fingers, with two 2x2bit flex pairs in LSBs if first word,
  plus bit8=whatever, plus an extra 2x2bit flex pair somewhere in 2nd word?)
```

Some other odd ones:

```
  440C           ;\maybe "true if ring is NEWLY bent"?
  440C           ;/
  040C           ;-looks like "ring", but is undocumented in manual
  6418           ;-two fingers bent?
  011E           ;-looks like hand forward, but is undocumented in manual
```

========== 8bit Opcode Area (Part 2) ==========

Header Byte for 8bit Opcode Area

```
  7    Append EXTRA SIX BYTES after 8bit Opcode Area (0=Normal, 1=Extra)
  6-5  Unknown (always 0 in known packets)
  4-0  Number of following 8bit commands (used range is 01h..18h)
```

Boolean Registers used with 8bit Opcodes

```
  flg            1-bit accumulator
  CondFlag       Conditional Flag (false: skip all but opcode 6nh/7nh or so)
  Input(0)       General purpose flag (can be used to memorize current "mode")
  Input(1..9)    Results from 1st..9th 16bit Opcode (see Part 1)
  Input(Ah..Dh)  Unknown/unused
  Input(Eh)      Another general purpose flag (used as frame-toggle or so)
  Input(Fh)      Another general purpose flag (used as frame-toggle or so)
  Output(0)      Should be BEEP sound (judging from description in manual)
  Output(1..8)   Joypad Shift Register (R,L,D,U,?,?,B,A) ;?,?=probably STA,SEL
  Output(9)      Used in Brawler init and in "Joust" program (... LED?)
  Output(Ah)     Unknown/unused
  Output(Bh)     Maybe abort opcode execution... or LED control?
  Output(Ch..Fh) Unknown/unused
  PulseA         Pulse generator 1 (mainly used for auto-fire on Button A)
  PulseB         Pulse generator 2 (mainly used for auto-fire on Button B)
```

For the first opcode, incoming flg and CondFlag seem to be initially true.

PulseA and PulseB can be also used to pulse DPAD bits, the pulse rate can be changed and enabled/disabled via numeric keypad.

The Joypad DPAD outputs are reportedly also affecting the Direction LEDs on the receiver unit. And, the other two LEDs are reportedly also affected by "autofire", which probably means Joypad A/B buttons, or the PulseA/PulseB generators. And the 9th LED (on the controller pad): unknown if it's software controlled, or just a Power LED.

8bit Opcodes

```
  0nh   Unknown/unused
  1nh   flg = Input(n)
  2nh   CondFlag = flg, and, thereafter, flg = Input(0..9)      ;IF command
  3nh   Used... 33h is related to Input(3) and related to CondFlag?
  4nh   flg = flg AND Input(n)
  5nh   flg = flg OR  Input(n)
  6nh   Used... somehow related to CondFlag stuff...            ;ELSE/ELSEIF?
  7nh   Exchange flg <--> Input(n)     ;or so? or CondFlag?     ;ENDIF?
  8nh   Output(n) = flg
  9nh   Output(n) = flg AND PulseA
  Anh   Output(n) = flg AND PulseB
  Bnh   Input(n) = Input(n) XOR flg    ;or so? or Input(n) = (NOT?) flg
  Cnh   Output(n) = NOT flg
  Dnh   Unknown/unused
  Enh   Unknown/unused
  Fnh   Used... FEh is used, related to Input(n) .. and CondFlag?
```

Extra Six

Finally, "PROGRAM E" has EXTRA SIX BYTES after the 8bit opcode area (indicated in bit7 of the 8bit Area's header). The bytes there are 00h,00h,00h,00h,57h,57h, purpose is unknown.

Controllers - Power Glove RX Packets (Position/Sensor Data)

Packet Summary

The standard packets are 9 bytes in size (not counting the preceeding "ready" byte; and obviously not counting the preceeding/following "not ready" bytes).

```
1st byte: Signed X-Coordinate
2nd byte: Signed Y-Coordinate
3rd byte: Signed Z-Coordinate
4th byte: Wrist Rotation Angle (around Z-axis)
5th byte: Finger Flex Sensors
6th byte: Control Pad Buttons
7th byte: Unknown/unused (00h)
8th byte: Unknown/unused (00h)
9th byte: Error Flags
```

Note: It is possible to disable some of the response bytes (via Mask Word in configuration packet). For example, one may want to disable 7th..8th byte (since they seem to be useless, and waste around 500 clks RX time); when doing so, the numbering of the following bytes changes (ie. in that example, Error Flags in "9th" byte would move to location of "7th" byte).

1st/2nd/3rd byte: Signed X/Y/Z-Coordinates

```
1st byte: X-coordinate (-80h=Left, +7Fh=Right)
2nd byte: Y-coordinate (-80h=Down, +7Fh=Up)
3rd byte: Z-coordinate (-80h=Forward, +7Fh=Back)  ;Forward=Towards Screen
```

All coordinates are relative to the "Center" (the location where one has most recently pushed the Center button).

Warning: There's a software bug in the Super Glove Ball game: When moving the glove too fast, the sprite jumps to random locations in left screen half, and/or stays stuck at botton-most screen coordinate. As a workaround, emulators may limit the distance to max 31 units per packet.

UNKNOWN if the glove is calibrating itself to biggest/smallest coordinates it has seen (similar as for the finger flex calibration).

If so, the calibrated range would "grow" each time when moving towards the boundaries; to avoid that "growing" effect, it may be a good idea to clip values to -40h..3Fh in software (that, being the "used" range, and values beyond that range being a "dead" zone) (if the user gets into the dead zone, nothing bad happens as it's still far to the -80h/+7Fh maximum boundaries where the growing effect might take place).

4th byte: Wrist Rotation Angle (around Z-axis)

```
7-4  Unused (always 0?)
3-0  Clockwise rotation in 30 degree steps (00h..0Bh)
```

Rotation around Z-axis is determined by sensing X and Y coordinates of the two ultrasonic speakers in relation to each other. Examples for the four major rotation angles are:

Val	Clock-face	Degrees	Back-side-of-Hand	Thumb
00h	12:00	0'	Points up	Points left
03h	3:00	+90'	Points right	Points up
06h	6:00	+180'	Points down	Points right
09h	9:00	-90'	Points left	Points down

Mind that there are some body-mechanic restrictions:

```
In general, forearm can be be turned clockwise from around 0:00 to 6:00
When using your shoulder you may also go anti-clockwise from 11:00 to 8:00
When breaking your arm, or maybe doing a salto, you might also get to 7:00
```

Note: Rotation around X-axis or Y-axis won't work: The speakers are no longer aimed at the microphones, so the signals get lost, and the glove cannot compute position & rotation values.

5th byte: Finger Flex Sensors

```
7-6  Thumb (hard to use)    (0..3; 0=Straightest, 3=Most Bent)
5-4  Index Finger           (0..3; 0=Straightest, 3=Most Bent)
3-2  Middle Finger          (0..3; 0=Straightest, 3=Most Bent)
1-0  Ring Finger            (0..3; 0=Straightest, 3=Most Bent)
N/A  Little Finger          (there is no flex sensor for this finger)
```

The glove is automatically calibrating itself to the minimum/maximum flex positions that it has seen; ie. before using it, one should stretch/bend all fingers a few times. When not doing that, it defaults to return all 3's (ie. highest flex seen so far).

Note: The flex sensors seem to be somehow fragile; it appears to be more or less common that only 3 of the 4 sensors are working (unknown what is causing that problem; maybe they are broken, or maybe just dirty).

6th byte: Control Pad Buttons

```
7-0  Button number of currently pressed button
```

Button numbers are:

```
00h      Keypad "0" or Center (glove beeps & centers when pressing Center?)
01h-09h  Keypad "1".."9"
0Ah      Button A
0Bh      Button B
0Ch      DPAD Left   ;XXX or Up ?
0Dh      DPAD Up     ;XXX or Right ?
0Eh      DPAD Down
0Fh      DPAD Right  ;XXX or Left ?
80h      Enter (glove beeps when pressing this button?)
82h      Start
83h      Select
84h (?)  PROG (glove goes into program-mode when pressing this button)
FFh (?)  None (no button pressed)
```

There can be only one button pressed. Priority order is reportedly (starting with highest priority):

```
  Left,Up,Down,Right, Enter,Start,Select,Center, 0,1,2,3,4,5,6,7,8,9, A,B.
```
Center and Enter are to-be-avoided as they beep the glove and wreck (?) one sample or so of data. WHEREAS, "wrecking" sounds unlikely... maybe it does rather stay not-ready for a while during centering? "0" and "Center" are returning the same value according to two sources (but both being unaware of the "ready" status-byte; so maybe they are seeing status=00h=centering/not ready, rather than seeing button=00h=center).

7th byte: Unknown/unused
8th byte: Unknown/unused
These two bytes seem to be always 00h. Purpose unknown.

9th byte: Error Flags
```
  7-6  Unused (always 0)
  5    Right speaker (on little finger) to Sensor 3 (lower-right mic) (1=Okay)
  4    Right speaker (on little finger) to Sensor 2 (upper-right mic) (1=Okay)
  3    Right speaker (on little finger) to Sensor 1 (upper-left mic)  (1=Okay)
  2    Left speaker (on index finger)   to Sensor 3 (lower-right mic) (1=Okay)
  1    Left speaker (on index finger)   to Sensor 2 (upper-right mic) (1=Okay)
  0    Left speaker (on index finger)   to Sensor 1 (upper-left mic)  (1=Okay)
```
Indicates if the speaker pings have reached the microphones. Should be 3Fh when everything is fine, otherwise coordinate & rotation bytes may contain garbage and one should ignore them.
Note: The six LEDs in the receiver unit seem to be wired to the serial joypad data. So, after reading the last byte of a packet, the LEDs should reflect Bit0-3 and Bit6-7 of the Error Flag byte (until the NMI handler reads the following "Not Ready" byte in next frame).

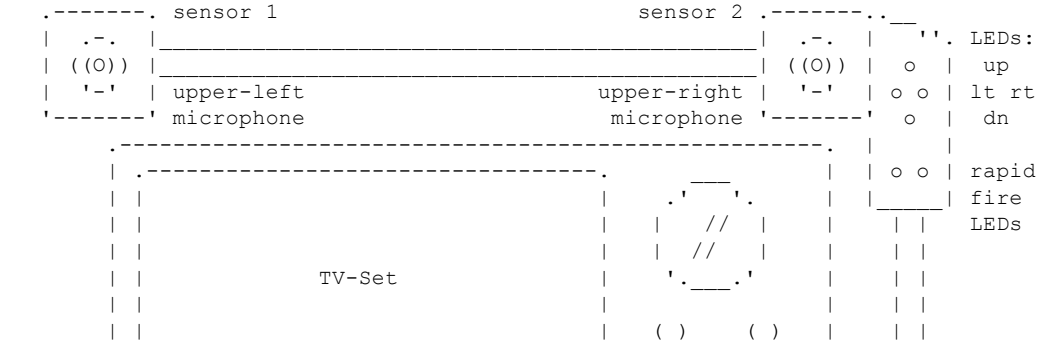
Controllers - Power Glove Games and Compatibilty Modes

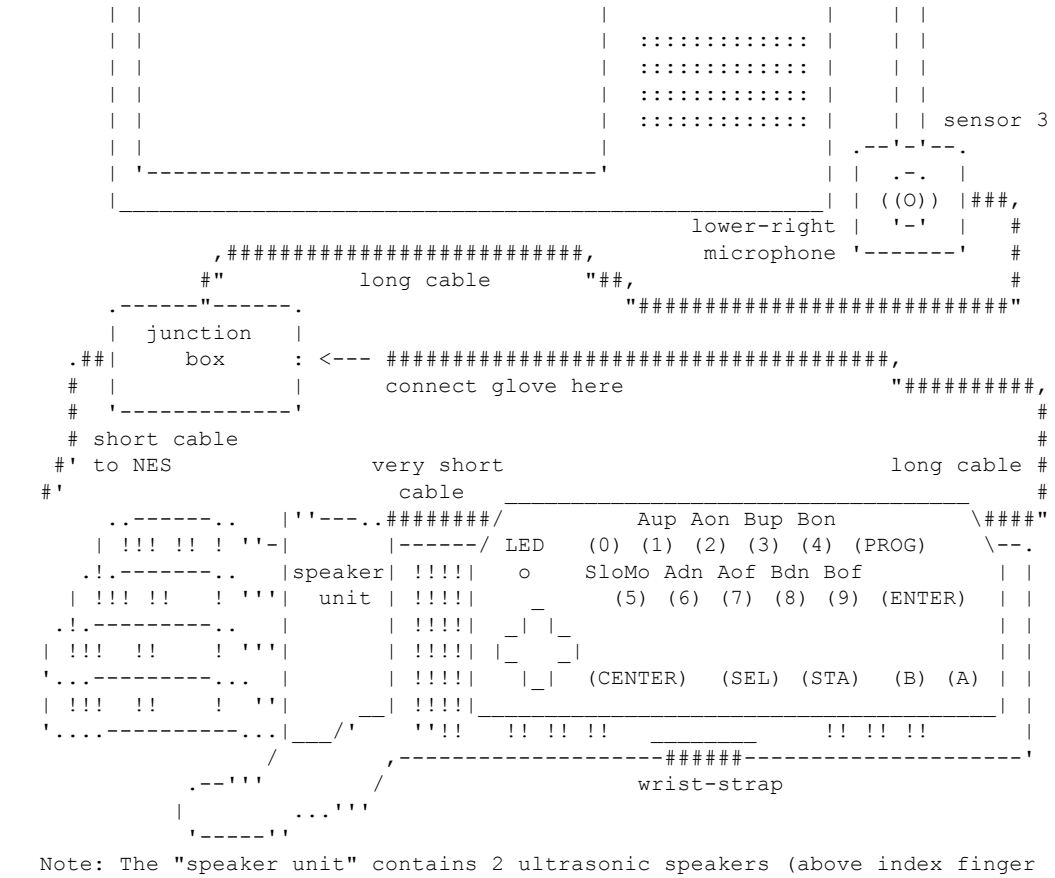
Games with Glove Support
There have been only two games released. The first one with "faked" support (merely configures the glove to simulate a joypad). The second one does actually support analog inputs.
```
  Bad Street Brawler (U) 1989 Mattel/Beam Software (uses digital mode only)
  Super Glove Ball (U) 1990 Mattel/Rare/Novak (uses real analog low-level mode)
```
As far as known, these games have been sold only in USA. So, japanese users have been stuck with the 14 built-in compatibility modes.

Built-in Compatibilty Modes (Program 1-14)
The glove BIOS contains 14 built-in "programs" that are allowing to use it with different existing games.
```
  1.  Put the game into NES and press POWER.  (Glove turns on and beeps.)
  2.  Press PROG on glove.
  3.  Press the number of the program you want to use on the number pad.
  4.  Press ENTR.  (The glove makes a dim beep.)
  5.  Press ENTR again.  (The glove beeps.)
  6.  Press START or SELECT on your glove.  (LED Panel turns on.)
  7.  Make a fist a few times and center before you start playing.
  8.  Repeat steps 2-5 to change programs.
      Always make a fist a few times and re-center after you change programs.
```
Programs 1-13 are simulating joypad buttons when sensing special gestures. Program 14 disables all sensors and allows to use the glove's control pad as normal joypad.

Extended Compatibilty Modes (Program A-I)
Aside from the actual game, the Bad Street Brawler game cartridge additionally contains another 9 compatibility modes (for games like Gyruss and Gunsmoke).
```
  1.  Put a special series game into the NES and press POWER.
  2.  Select special program option as prompted.
  3.  Select special series program as prompted.
  4.  Turn off NES. You have 30 seconds to complete step 5.
  5.  Put game you want to play in the NES and turn it back on.
  6.  Make fists and center as usual, then press START, to start playing game.
```
BUG: The menu selection for the right column doesn't work (trying to select Program F-I will mirror to Program A-D), and, Program C seems to have accidently swapped clockwise & anti-clockwise directions.

Controllers - Power Glove Drawings





Note: The "speaker unit" contains 2 ultrasonic speakers (above index finger and little finger), plus one regular speaker (for producing beep sounds), plus wires going to the flex sensors. The ultrasonic speakers must be aimed towards the microphones for position sensing.
According to the glove manual, the microphones are intended to be hung on the TV-Set. However, the surface of the screen may produce echos, and thus mess-up signal sensing; results can be reportedly improved by depositing the microphones elsewhere, possibly backed with towels for avoiding echos.

Controllers - Power Glove Pinouts

Pinouts for the CPU should be as shown below (info from Tim Deagan; descriptions in middle column are assuming that the CPU has COP888CLMH-style pinouts; but it might actually be a different COP888xxx variant).

Pin	COP888	Power Glove	
1	C2,I/O	INPUT C on 4021	;\serial data
2	C3,I/O	INPUT D on 4021	;/to NES and LEDs?
3	G4,I/O,SO	?	
4	G5,I/O,SK	DATA LATCH	;\serial data
5	G6,I,SI,ME	DATA CLOCK	;/from NES?
6	G7,I/CKO,HALT RESTART	XTAL	
7	CKI	XTAL	
8	Vcc	+5VDC	
9	I0,I	R1 pullup,Button0,Button8,RIGHT	
10	I1,I	R2 pullup,Button1,Button9,LEFT	
11	I2,I	R3 pullup,Button2,ENTER ,DOWN	
12	I3,I	R4 pullup,Button3,PROG ,UP	
13	I4,I	R5 pullup,Button4, ,START	
14	I5,I	R6 pullup,Button5, ,SELECT ;but, where	
15	I6,I	R7 pullup,Button6, ,B ;is Center?	
16	I7,I	R8 pullup,Button7, ,A	
17	L0,I/O,MIWU	R26 gnd,THUMB	
18	L1,I/O,MIWU	R27 gnd,INDEX	
19	L2,I/O,MIWU	R28 gnd,MIDDLE	
20	L3,I/O,MIWU	R29 gnd,RING	
21	C4,I/O	?	;\maybe these are
22	C5,I/O	Button0-7 & CENTER	; outputs to 8x3
23	C6,I/O	ENTER	;/keypad matrix?
24	C7,I/O	GND	
25	L4,I/O,MIWU,T2A	CLK on 4021	
26	L5,I/O,MIWU,T2B	RC net to LBlu,->|- red finger wires	
27	L6,I/O,MIWU	?	
28	L7,I/O,MIWU	GRY from top of glove (XMTR2 ?)	
29	D0,O, I/O BIT 0	YEL from top of glove (XMTR1)	
30	D1,O, I/O BIT 1	GRN from top of glove (XMTR2)	
31	D2,O, I/O BIT 2	BLU from top of glove (BEEPER)	
32	D3,O, I/O BIT 3	PUR from top of glove (XMTR1 ?)	
33	D4,O, I/O BIT 4	INPUT E on 4021	;\
34	D5,O, I/O BIT 5	INPUT F on 4021	; serial data
35	D6,O, I/O BIT 6	INPUT G on 4021	; to NES and LEDs?

```
36      D7,O, I/O BIT 7         INPUT H on 4021       ;/
37      GND                     GND
38      RESET#                  ?
39      G0,I/O,INT,ALE          ?
40      G1,WDOUT                ?
41      G2,I/O,T1B,WR#          BRN to junct box (pin1 LM324 near rcvrs)
42      G3,I/O,T1A,RD#          ORG to junct box (RCs to LM324 near rcvrs)
43      C0,I/O                  INPUT A on 4021       ;\serial data
44      C1,I/O                  INPUT B on 4021       ;/to NES and LEDs?
```

Power Glove Component List

Glove Speaker PCB

```
two ultrasonic speakers (near index finger/little finger knuckles)
one normal speaker (beeper)
three transistors, resistors, capacitors, diodes
wires to control-pad PCB, wires to finger flex sensors
```

Glove Control Pad PCB

```
(components here are mostly unknown)
Seems to consist of a 44pin National Semiconductor COP888-family CPU,
a 4021 shift-register, and possibly whatever other components.
one LED (on solder side), 21 buttons (on solder side)
```

Junction Box PCB

```
two LM324 chips (two quad-amplifiers)
three diodes, and many resistors and capacitors
DB9 connector (to glove), and wire to NES, and wire to microphone units
```

Microphone PCBs (three pieces)

```
TL062 (dual-amplifier)
one microphone, one capacitor, six resistors
```

LED PCB (near upper-right microphone PCB)

```
SN74LS164N (8bit serial-in, parallel-out shift-register)
six LEDs, six resistors, one capacitor
```

Controllers - UForce

[Controllers - UForce I/O]
[Controllers - UForce Drawings]
[Controllers - UForce Games and Game Switches]

How the UForce Works (directly from the manual)

"UForce uses an array of sensors to create a three-dimensional Power Field about 8 to 10 inches above or in front of each sensor. The Power Field senses the position and motion of objects within it by combining information from all its sensors."

Note: The so-called "Power Field" consists of 9 infra-red transmitters, each one bundled with an infra-red receiver, meant to sense reflections from the player's hands.

Controllers - UForce I/O

For reading analog data: Set Game Switches SW1+SW2+SW3 to the "Down" position (otherwise the UForce will simulate a digital joypad). And, the UForce should be connected to 1st controller port (the prototype cartridge supports only 4016h reads, not 4017h).

Byte Reading

Bytes are read the same way as joypad1 (strobe 4016h.W.Bit0 1-then-0, then read 8 bits from 4016h.R.Bit, MSB first). Data appears to be transferred through a shift-register (so no delays should be needed within the byte reading part), however, short delays appear to be required between the separate bytes (exact cycle values are unknown).

Status Byte

```
1st..6th Status Bits   (00h=AllSixBitsHigh=Ready, Other=SomeBitLow=Busy)
7th      Select Button (0=High=Released, 1=Low=Pressed)
8th      Start Button  (0=High=Released, 1=Low=Pressed)
```

The status byte should be usually polled within the NMI handler. If it's indicating "Ready" (which reportedly happens about 10 times per second), then the NMI handler should read 8 more bytes (containing the analog data described below).

The Start/Select bits are usually read from "Ready" bytes (unknown if they can be also read from "Busy" bytes).

Analog Data Bytes

```
1st  Analog Data, Bit4  ;\5bit analog sensor value,
2nd  Analog Data, Bit3  ; bits are inverted (0=High=One, 1=Low=Zero),
3rd  Analog Data, Bit2  ; after undoing that inversion (XOR by 1Fh),
4th  Analog Data, Bit1  ; values are: 00h..1Eh = Distant..Closest
5th  Analog Data, Bit0  ;/(aka least..most IR light reflection)
6th  Analog Data, Bit0 (according to Kevin: duplicated, same as above)
7th  Presence Bit (according to Kevin: 0=High=Low=Sensor covered, heh?)
8th  Unused  (according to Kevin: "0=NOT(0)=0" uh, that can't be true?)
```

The existing prototype software is merely treating the byte as "8bit analog value", without giving special attention to the stuff in lower bits (so, although the content of the LSBs is rather unclear, their exact values don't really matter for emulation purposes). Also note: Kevin claims the analog range to be only 00h..1Eh (not 00h..1Fh), this may be true, but he also claims that the closest value is most distant, so better don't count on it.

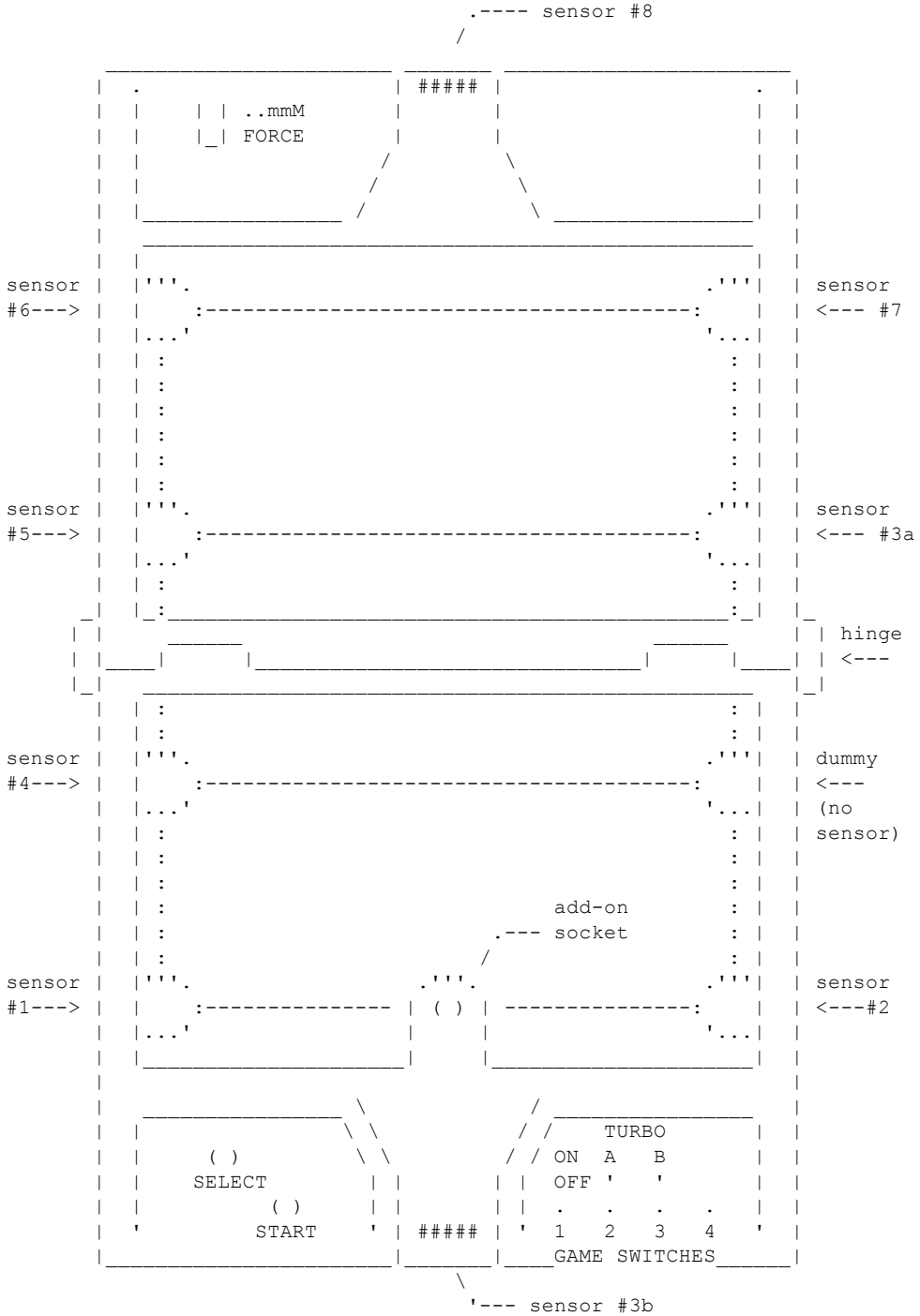
UForce Sensor Locations

Byte Number (within packet)	Official Naming (from manual)	Other Numbering (kevtris doc)
8th	Top	1
6th 7th	1. 2.	2 3
5th 3rd'a	3. 4.	4 5
4th -	5. 6.	6 dummy
1st 2nd	7. 8.	7 8
3rd'b	Bottom	9

The UForce has 9 sensors (plus one dummy location without sensor installed in it). Of that 9 sensors, only 8 can be used at once (the 3rd byte in the 8-byte data packet contains either the 3rd'a or 3rd'b value, depending on the SW4 switch setting).

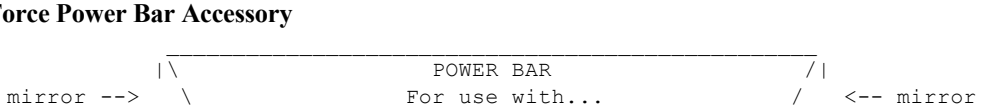
Controllers - UForce Drawings

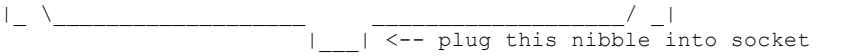
UForce Main Unit (with sensor numbers as ordered in the 8-byte data packet)



Of the 9 sensors, only 8 can be used at once. Game Switch 4 allows to select between using sensor #3a or #3b. The other Game Switches select analog mode, or digital joypad emulation modes.

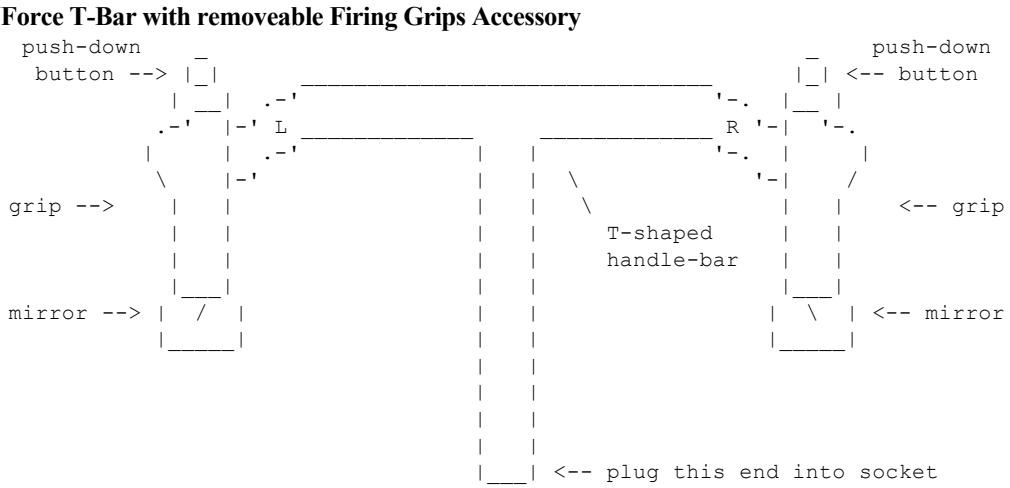
UForce Power Bar Accessory





This thing is badged "UForce Power Bar, For use with Mike Tysons's Punch Out!! (and future games)". The bar can be plugged into the add-on socket between the bottom-most sensor pair, there seem to be mirrors at the bar ends, which do redirect that sensors towards the sides.

UForce T-Bar with removeable Firing Grips Accessory



This thing can be plugged into the add-on socket between bottom-most sensor pair. It can be turned into all three directions. There aren't any electronics/mechanics in the T-bar; it's only intended to keep the player's hands at the height of the top-most sensor pair, which may then eventually react to the player's steering efforts.
The two removeable handles have thumb buttons, which appear to turn mirrors inside of the handles, which do then trigger the bottom-most sensors.

Controllers - UForce Games and Game Switches

Digital Game Modes
The five digital modes (called "Mode A-E") are used for compatibility with existing games; where the UForce is using it's analog inputs to simulate digital joypad signals matched for different game types. Naturally this isn't working too well, as it's best it may be useful for adding some extra difficulty to the games.

Analog Game Mode
The analog mode (called "Mode F") was intended for future games that do directly read the analog sensors. However, there hasn't been any such future game ever released. There's only one prototype cartridge, that has leaked into internet:

```
Uforce Power Games (USA) (Prototype) (1990) Broderbund
```

The cartridge contains four crude mini-games (Hose 'em Down, Nuclear Rat Attack, Rock on Air, and Power Field B-Ball). The usage of the sensors is as so:

```
Hose'em Down:
Use "Top"                       as ANALOG left/right
Use "Lower-Left in Lower Area"  as ANALOG up/down
Nuclear Rat Attack:
Use upper-four sensors          as DIGITAL push buttons
Rock on Air:
Use left four sensors           as DIGITAL push buttons
Use "Top"                       as ANALOG pitch
Use "Bottom"                    as ANALOG volume
Power Field B-Ball
Use "Upper-Left in Upper Area"  as walk left
Use "Lower-Left in Upper Area"  as walk right
Use "Upper-right in Upper Area" as throw
Use whatever                    as jump
```

UForce Game Modes

```
Mode SW1  SW2  SW3  SW4   Hinge/Angle  Add-on   Purpose
A    Up   Up   Up   Down  Upright 85'  T-Bar    Various Games (Forward View)
B    Down Up   Up   Up    Tilt   110'  Pow-Bar  Mike Tyson's Punch-Out!!
C    Up   Down Up   Down  Upright 85'  -        Rad Racer
D    Down Down Up   Up    Upright 85'  -        Excitebike
E    Down Up   Down Down  Flat   180'  -        Various Games (Side Scroll)
F    Down Down Down (Down)   (var)      (var)   Analog (UForce Power Games)
```

Aside from the official 6 mode settings, there are some more possible settings for SW1-SW3:

```
-    Up   Up   Down (any)    (any)      (any)   Reportedly same as Mode A
-    Up   Down Down (any)    (any)      (any)   Unknown
```

SW4 does simply select which of the two "shared" sensors to use (either bottom-most one, or the lower-right in upper-field one), so changing SW4 may allow some desired or undesired variations.

UForce Turbo A/B Switches
The two turbo switches can apply auto-fire to joypad button A/B signals. Accordingly, they are used only in digital joypad emulation modes, not in analog

mode. The auto-fire rate is reportedly around 10Hz.

Controllers - Barcode Readers

Datach - Joint ROM System (Bandai) (1992)
A mini-cartridge adaptor with built-in barcode reader. The device connects to cartridge slot, and can be used (only) with seven special mini-cartridge games (six of them actually supporting the barcode feature):

```
Dragon Ball Z: Gekito Tenkaichi Budokai (December 1992)
Ultraman Club: Spokon Fight!! (April 1993)
SD Gundam: Gundam Wars (April 1993)
Crayon Shin-Chan: Orato Poi Poi (August 1993) (this WITHOUT barcode support)
Yu Yu Hakusho: Bakuto Ankoku Bujutsue (October 1993)
Battle Rush: Build Up Robot Tournament (November 1993)
J League: Super Top Players (April 1994)
```

The barcode reader hardware can see only one pixel at a time:

```
[6000h].R.Bit3 Barcode Sensor (0=Black, 1=White)
```

The data seen during scanning is:

```
All Black         ;no card inserted
61 pixels White   ;leading white space (ca. 61 pixels on included cards)
95 pixels Stripes ;barcode (95 pixels for EAN-13 and UPC-A codes)
61 pixels White   ;ending white space (ca. 61 pixels on included cards)
All Black         ;no card inserted
```

Example: In Dragon Ball Z, barcode 062982-144233 gives HP:51500, BP:32500, DP:25000.
The included cards measure roughly 8.55cm x 5.9cm each. The existing games support UPC-A and EAN-13 barcodes (both have 30 black stripes), EAN-8 barcodes (22 black stripes), and, an unknown variable-length barcode format (with 27 or more black stripes). UPC-E barcodes (17 black stripes) aren't supported.
The scanning software uses 8bit counters (incremented at 73 clk rate), for a reasonable resolution & avoiding overflows (on wide 4pixel stripes), the scanning time should be around 300..4600 clks per pixel.
The games are using a variant of "Mapper 16", but apparently with VRAM instead of VROM. The VRAM has unknown size, and it's probably contained in the "Datach" adaptor.
Mapper 16: Bandai - PRG/16K, VROM/1K, IRQ, EPROM
Note: Bandai also made a similar mini-cartridge device (named Sufami Turbo) for the Super Famicom; that device features two cartridge slots, but doesn't include a barcode reader.

Barcode Battler II (Epoch) (1992)
A handheld console with built-in barcode reader and very simple LCD display. The device can be used as standalone handheld console, or as external barcoder reader for other consoles. Supported by only one Famicom game:

```
Barcode World (1992) Sunsoft (JP) (includes cable with 15pin connector)
```

Note: The Barcode Battler II is also supported by a couple of SNES games.
Controllers - Barcode Battler (barcode reader)
Controllers - Barcode Battler Transmission I/O
Controllers - Barcode Battler Drawings
In short: The Barcode Battler outputs barcodes as 20-byte ASCII string, at 1200 Baud, 8N1. The NES software receives that bitstream via Port 4017h.Bit2.

Barcode Format
Controllers - Barcode Formats

Controllers - Barcode Battler (barcode reader)

The Barcode Battler from Epoch allows to scan barcodes (either from special paper cards, or from daily-life products like food packagings), games can then use the barcode digits as Health Points, or other game attributes.

Standalone-Mode
The device was originally designed as stand-alone gaming console with some push buttons, a very simple LCD screen with 7-segment digits & some predefined LCD symbols, and a built-in game BIOS (ie. without external cartridge slot, and without any bitmap graphics).

Link-Mode
Later versions (with black case) include an "EXT" link port, allowing to link to other Barcode Battler hardware, or to Famicom/Super Famicom consoles. The EXT port is probably bi-directional, but existing Famicom/Super Famicom games seem to be using it only for reading barcodes (without accessing the LCD screen, push buttons, speaker, or EEPROM).

Barcode Battler Famicom (NES) Games

```
Barcode World (1992) Sunsoft (JP) (includes cable with 15pin connector)
```

Barcode Battler Super Famicom (SNES) Games

```
Alice's Paint Adventure (1995)
Amazing Spider-Man, The - Lethal Foes (19xx)
```

```
Barcode Battler Senki Coveni Wars (1993) Epoch
Donald Duck no Mahou no Boushi (19xx)
Doraemon 2: Nobita's Great Adventure Toys Land (1993)
Doraemon 3: Nobita and the Jewel of Time (1994)
Doraemon 4 - Nobita to Tsuki no Oukoku (19xx)
Doroman (canceled)
Dragon Slayer - Legend of Heroes 2 (1993) Epoch
J-League Excite Stage '94 (1994)
J-League Excite Stage '95 (1995)
Lupin Sansei - Densetsu no Hihou wo Oe! (19xx)
Super Warrior Combat (19xx - does this game exist at all?)
```

Barcode Battler Hardware Versions

Region	Case	EXT	Barcode-Reader	Name	Year
Japan	White	None	Yes	Barcode Battler	1991
Japan	Black	1	Yes	Barcode Battler II	1992
Japan	Black	2	None	Barcode Battler II^2	199x
Europe	Black	1	Yes	Barcode Battler	1992/1993

The versions with one EXT socket can be connected to NES/SNES, or to one or more of the "II^2" units (allowing more players to join the game).

Connection to SNES/NES consoles

Connection to Super Famicom or SNES requires a "BBII INTERFACE": a small box with 4 LEDs and two cables attached (with 3pin/7pin connectors), the interface has been sold separetedly, it's needed to add a SNES controller ID code to the transmission protocol.
Connection to Famicom consoles requires a simple cable (without interface box) (with 3pin/15pin connectors), the cable was shipped with the "Barcode World" Famicom cartridge, connection to NES would require to replace the 15pin Famicom connector by 7pin NES connector.
The required 3pin EXT connector is available only on newer Barcode Battlers (with black case), not on the original Barcode Battler (with white case).

```
Unknown if all 3 pins are actually used by NES/SNES cable/interface?
Unknown if NES/SNES software can access LCD/buttons/speaker/EEPROM ?
```

Connectivity

"Connectivity mode is accessible if you plug in a standard 3.5mm mono jack plug into the expansion port on the left hand side of the unit, hold down the R-Battle and R-Power buttons and turn the unit on, the Barcode Battler II goes into scanner mode."

Barcode Battler II Interface

The hardware itself was manufactured by Epoch, and licensed by Nintendo (it says so on the case).
The four lights, from left to right, indicate as follows:

```
"OK"    All is well, the device is operating as normal.
"ER"    Maybe there's something wrong?
"BBII"  The Barcode Battler is sending data to the device.
"SFC"   The SFC/SNES is waiting for a signal from the Barcode Battler.
```

Component List (may be incomplete)

```
80pin NEC uPD75316GF (4bit CPU with on-chip 8Kx8 ROM, 512x4 RAM, LCD driver)
8pin Seiko S2929A (Serial EEPROM, 128x16 = 2Kbit) (same/similar as S29290)
3pin EXT socket (3.5mm "stereo" jack) (only in new versions with black case)
LCD Screen (with 7-segment digits and some predefined words/symbols)
Five LEDs (labelled "L/R-Battle Side")
Seven Push Buttons (L/R-POWER, L/R-Battle, Power on/off, Select, Set)
Speaker with sound on/off switch (both on bottom side)
Barcode reader (requires card-edges to be pulled through a slot)
Batteries (four 1.5V AA batteries) (6V)
```

Controllers - Barcode Battler Transmission I/O

The Barcode Battler outputs barcodes as 20-byte ASCII string, at 1200 Baud, 8N1. The NES software receives that bitstream via Port 4017h.Bit2. The SNES software requires a BBII Interface, which converts the 8bit ASCII digits into 4bit nibbles, and inserts SNES controller ID and status codes, the interface should be usually connected to Controller Port 2 (although the existing SNES games seem to accept it also in Port 1).

Barcode Battler (with BBII Interface) SNES Controller Bits

```
1st..12th   Unknown/unused (probably always 0=High?)
13th..16th  ID Bits3..0          (MSB first, 1=Low=One) (must be 0Eh)
17th..24th  Extended ID Bits7..0 (MSB first, 1=Low=One) (must be 00h..03h)
            (the SNES programs accept extended IDs 00h..03h, unknown
            if/when/why the BBII hardware does that send FOUR values)
25th        Status: Barcode present (1=Low=Yes)
26th        Status: Error Flag 1 ?
27th        Status: Error Flag 2 ?
28th        Status: Unknown      ?
```

Following bits need/should be read ONLY if the "Barcode Present" bit is set.

```
29th-32th   1st Barcode Digit, Bits3..0  (MSB first, 1=Low=One)
33th-36th   2nd Barcode Digit, Bits3..0  (MSB first, 1=Low=One)
37th-40th   3rd Barcode Digit, Bits3..0  (MSB first, 1=Low=One)
41th-44th   4th Barcode Digit, Bits3..0  (MSB first, 1=Low=One)
45th-48th   5th Barcode Digit, Bits3..0  (MSB first, 1=Low=One)
49th-52th   6th Barcode Digit, Bits3..0  (MSB first, 1=Low=One)
```

```
  53th-56th   7th Barcode Digit, Bits3..0  (MSB first, 1=Low=One)
  57th-60th   8th Barcode Digit, Bits3..0  (MSB first, 1=Low=One)
  61th-64th   9th Barcode Digit, Bits3..0  (MSB first, 1=Low=One)
  65th-68th   10th Barcode Digit, Bits3..0 (MSB first, 1=Low=One)
  69th-72th   11th Barcode Digit, Bits3..0 (MSB first, 1=Low=One)
  73th-76th   12th Barcode Digit, Bits3..0 (MSB first, 1=Low=One)
  77th-80th   13th Barcode Digit, Bits3..0 (MSB first, 1=Low=One)
  81th and up Unknown/unused
        Above would be 13-digit EAN-13 codes
        Unknown how 12-digit UPC-A codes are transferred   ;\whatever leading
        Unknown if/how 8-digit EAN-8 codes are transferred ; or ending padding?
        Unknown if/how 8-digit UPC-E codes are transferred ;/
```

For some reason, delays should be inserted after each 8 bits (starting with 24th bit, ie. after 24th, 32th, 40th, 48th, 56th, 64th, 72th bit, and maybe also after 80th bit). Unknown if delays are also needed after 8th and 16th bit (automatic joypad reading does probably imply suitable delays, but errors might occur when reading the ID bits via faster manual reading).

Barcode Battler RAW Data Output

Data is send as 20-byte ASCII string. Bytes are transferred at 1200 Bauds:

```
  1 Start bit (must be 1=LOW)
  8 Data bits (LSB first, 1=LOW=Zero, 0=HIGH=One)
  1 Stop bit  (must be 0=HIGH)
```

The first 13 bytes can contain following strings:

```
  "nnnnnnnnnnnnn"    ;13-digit EAN-13 code (ASCII chars 30h..39h)
  <Unknown>          ;12-digit UPC-A code (with ending/leading padding?)
  "     nnnnnnnn"    ;8-digit EAN-8 code (with leading SPC-padding, ASCII 20h)
  <Unknown>          ;8-digit UPC-E code (with ending/leading padding?)
  "ERROR        "    ;indicates scanning error
```

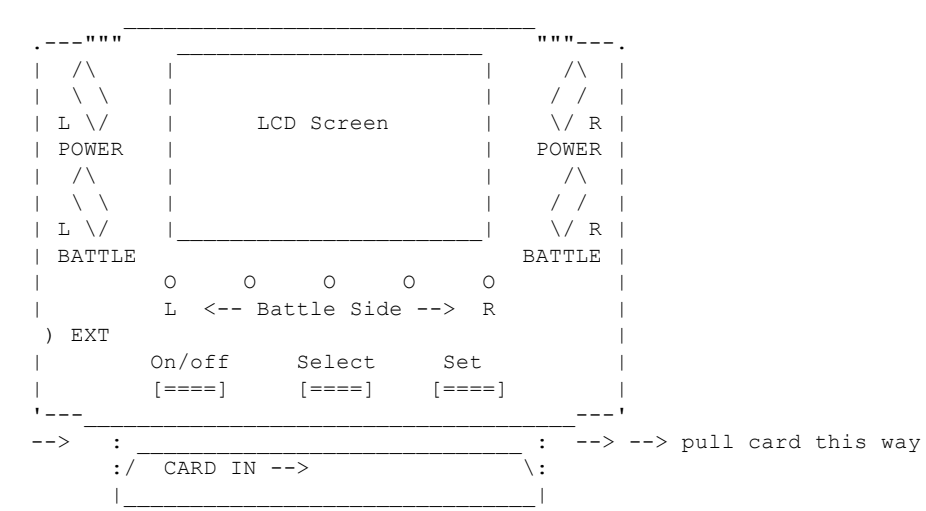
The last 7 bytes must contain either one of following ID strings:

```
  "EPOCH",0Dh,0Ah    ;<-- this is sent/accepted by existing hardware/software
  "SUNSOFT"          ;<-- this would be alternately accepted by the NES game
```

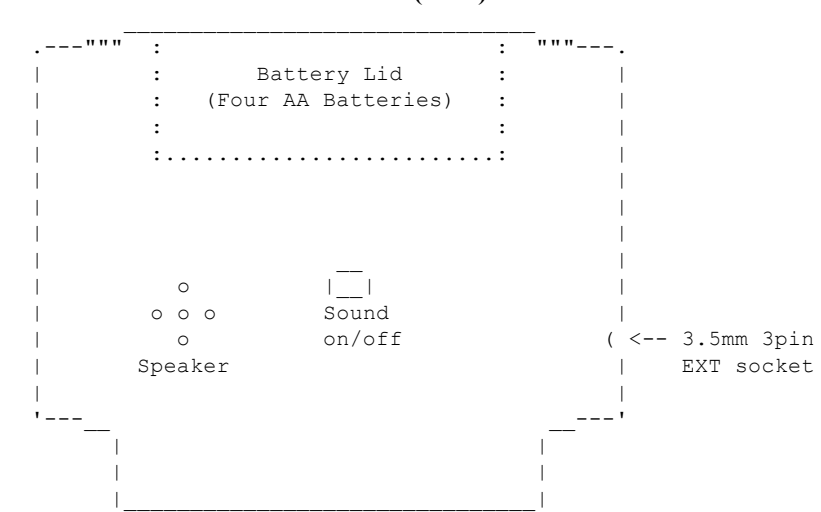
There are rumours that one "must" use a mono 3.5mm plug in order to receive data - that's obviously bullshit, but it might indicate that the middle pin of stereo plugs must be GNDed in order to switch the Barcode Battler into transmit mode(?)

Controllers - Barcode Battler Drawings

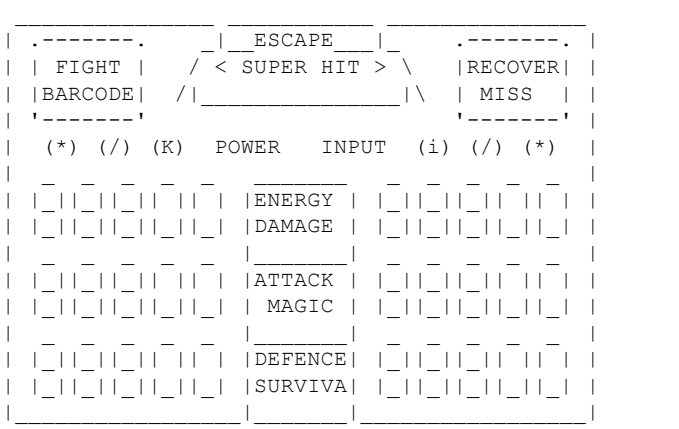
Barcode Battler - Handheld Console (Front)



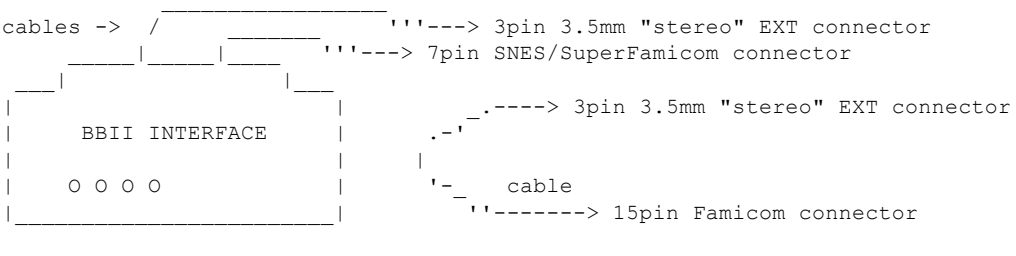
Barcode Battler - Handheld Console (Back)



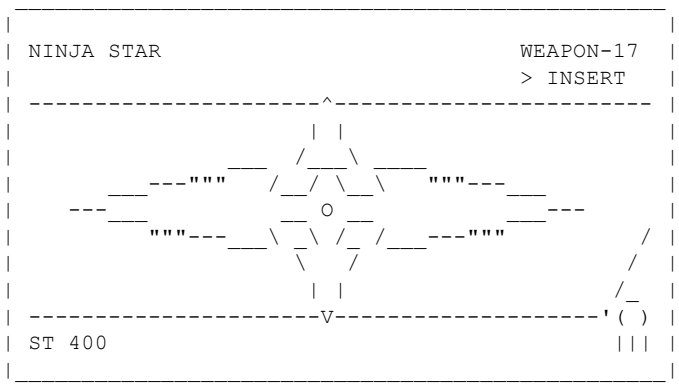
Barcode Battler - LCD Screen Layout



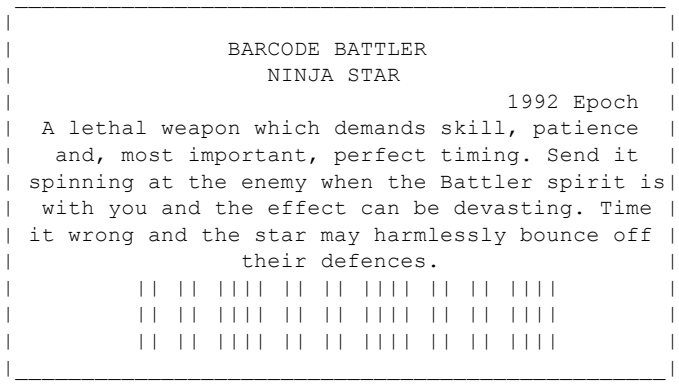
Barcode Batter II Interface (SNES/SuperFamicom) & Simple Cable (Famicom)



Paper-Card Front (Picture Side)



Paper-Card Back (Description & Barcode)



Controllers - Barcode Formats

Common Barcode Formats

```
EAN-13 13-digits in 12 symbols (extra digit encoded in parity)
UPC-A  12-digits in 12 symbols
EAN-8  8-digits in 8 symbols
UPC-E  8-digits in 6 symbols (extra digits encoded in parity)
```

Barcode Symbols

Each symbol consists of 7 pixels with 4 bars: White-Black-White-Black in first half, and vice-versa in second half; this allows to determine the scanning speed per symbol. There are 3 possible symbols (with inverse and/or reverse pixels) for each digit:

```
Digit     Left/Odd  Left/Even  Right/Even
0         ...##.#   .#..###    ###..#.
1         ..##..#   .##..##    ##..##.       "." = White
2         ..#..##   ..##.##    ##.##..       "#" = Black
3         .####.#   .#....#    #....#.
4         .#...##   ..###.#    #.###..
```

```
5         .##...#   .###..#    #..###.
6         .#.####   ....#.#    #.#....       Left Sync Mark:    #.#
7         .###.##   ..#...#    #...#..       Center Sync Mark:  .#.#.
8         .##.###   ...#..#    #..#...       Right Sync Mark:   #.#
9         ...#.##   ..#.###    ###.#..
```

12-digit UPC-A (and 8-digit EAN-8) barcodes use "Left/Odd" for the first 6 (or 4) symbols, and "Right/Even" for the remaining 6 (or 4) symbols. The "Left/Even" symbols are used only for 13-digit EAN-13 and 8-digit UPC-E barcodes (see below).

UPC-A (12-digits in 12 symbols)

This is the original barcode format, used mainly in North America, but also found on products that imported/exported to/from that area (namely, on many Audio CDs).

EAN-13 (13-digits in 12 symbols; extra digit encoded in parity)

This is an extension of the UPC-A format, with an additional leading digit (if the digit is "0" then it's an UPC-A barcode, otherwise an international EAN-13 barcode).

EAN-13 barcodes are having only 12 symbols (as UPC-A), the 13th digit is encoded in the "parity" of the 2nd..6th symbol; the other symbols are fixed: 1st=Odd and 12th=Even allow to determine scanning direction; 7th..11th=Even aren't containing any special information.

The parity (E=Left/Even, O=Left/Odd) for the 1st..6th symbol (aka 2nd..7th digit) depends on the 1st digit:

```
Digit  1st=0  1st=1  1st=2  1st=3  1st=4  1st=5  1st=6  1st=7  1st=8  1st=9
Parity OOOOOO OOEOEE OOEEOE OOEEEO OEOOEE OEEOOE OEEEOO OEOEOE OEOEEO OEEOEO
```

The parity for the 7th..12th symbol (aka 8nd..13th digit) is fixed (always Right/Even).

EAN-8 (8-digits in 8 symbols)

A short barcode for international use. Encoded as UPC-A, but only 4 symbols in each half.

UPC-E (8-digits in 6 symbols; extra digits encoded in parity)

This is a compressed UPC-A barcode; with the "middle zeros" removed from the manufacturer/product number field. Unlike UPC-A, these barcodes are rarely found outside of North America.

```
UPC-E        UPC-A
tMmpppXc --> tMmX0000pppc  ;with X=0..2   ;\for UPC-E, first digit (t)
tMmmpp3c --> tMmm00000ppc                 ; may be 0..1 only),
tMmmmp4c --> tMmmm00000pc                 ; last digit (c) must be checksum
tMmmmmXc --> tMmmmm0000Xc  ;with X=5..9   ;/of the "decompressed" UPC-A code
```

The first digit (0 or 1), and the last digit (checksum, 0..9) are encoded as "parity" of the six symbols:

```
Digit  8th=0  8th=1  8th=2  8th=3  8th=4  8th=5  8th=6  8th=7  8th=8  8th=9
1st=0  EEEOOO EEOEOO EEOOEO EEOOOE EOEEOO EOOEEO EOOOEE EOEOEO EOEOOE EOOEOE
1st=1  OOOEEE OOEOEE OOEEOE OOEEEO OEOOEE OEEOOE OEEEOO OEOEOE OEOEEO OEEOEO
Left Sync Mark: #.#    Center Sync Mark: None    Right Sync Mark: .#.#.#
```

Whereas, E=Left/Even, O=Left/Odd (Right/Even symbols aren't used by UPC-E).

Checksums

The barcode checksum is located in the last digit. This value must be chosen so that the sum of all digits, plus twice the sum of each second digit sums up to a decimal value ending with zero.

```
EAN-13 Checksum Weighting: 1-313131-313131     ;1=counted once
UPC-A  Checksum Weighting:   313131-313131     ;3=counted thrice
EAN-8  Checksum Weighting:     3131-3131
UPC-E  Decompress UPC-E to UPC-A, then do UPC-A checksumming
```

Notes

Scanning direction can be determined by checking "Odd/Even" parity of the first & last symbol.

Error checking can be done by verifying the checksum digit and by rejecting any invalid symbols.

Controllers - Pachinko

Pachinko Controller Games

```
Pachinko Daisakusen (J) 1991 Coconuts/C*Dream
Pachinko Daisakusen 2 (J) 1992 Coconuts/C*Dream
Pachio Kun 4 (J) 1991 Coconuts/C-Dream
Pachio Kun 5 (J) 1993 Coconuts/C-Dream
```

Pachinko is a japanese gambling game; its appearance is resembling pinball, but concerning stupidity it's more resembling one-armed-bandit-style slot machines.

Pachinko Controller Access

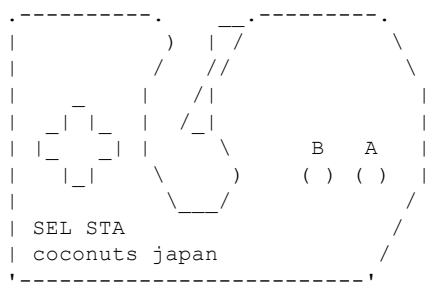
The controller is accessed like normal Joypad 3, but with the 8bit joypad data being followed by additional 8bit ADC data. First do the normal 1-then-0 strobing, then read 16bit from [4016h].Bit1:

```
1st..8th bit  --> same as normal joypad data
9th..16th bit --> analog ADC data (MSB first, inverted, 1=Low=Zero)
17th and up   --> unknown/unused
```

Average analog range returned on real hardware is unknown. In software, the used range is 00h=Stopped through 63h=Fastest.

Coconuts Japan Pachinko Controller

The controller consists of normal joypad buttons, plus an analog trigger (or analog dial?) in the middle of the controller. The joypad is used for menu selections, the analog thing for firing the pachinko balls.



Controllers - Microphones

Standard Famicom Microphone
On older Famicoms, the second control pad (that without Start and Select buttons) has a microphone with volume control built-in. The signal goes to Bit 2 of Port 4016h (simple 1bit input, not an analogue ADC-converted input).
The signal is also merged with the PPUs sound output signals, as such, allowing to use the television set/speaker as amplifier/megaphone.
Games that do use the Standard Microphone

* Atlantis no Nazo (NES = Super Pitfall II) - get the microphone power-up and then yell into the microphone to freeze and kill most enemies.
* Hikari Shinwa: Palutena no Kagami (NES = Kid Icarus) - talk into the microphone to bargain for lower shop prices.
* Raid on Bungeling Bay (NES = Raid on Bungeling Bay) - ?
* SD Kamen Rider -- in one of the mini games, blow air into the microphone to get a windmill to spin.
* Takeshi no Chousenjou - microphone section, where players are required to sing a verse of karaoke or talk whilst playing a pachislo minigame (the microphone section was replaced in later versions of the game once the microphone was dropped from the Famicom).
* Zelda no Densetsu: The Hyrule Fantasy (cart/disk) (NES = The Legend of Zelda) - yell into the microphone to kill Pols Voice enemies.

Note: Many Coconuts games do also contain a Famicom Microphone reading function: I Love Softball, Pachio Kun 1-6, Pachinko Daisakusen 1-2 (though unknown when/if/which games do actually use that function).

Bandai Microphone
Cartridge with attached "stage" microphone, two 2 push buttons (A and B), and ROM expansion slot.
Mapper 188: UNROM-reversed

NES LaserScope / Famicom Gun Sight (Konami)
This is basically a regular "Zapper" lightgun in form of a headset. The thing contains a microphone that replaces the Zapper's trigger button; thus allowing "voice activated" shooting.

Controllers - Reset Button

Reset Button
As an additional "control" the console is equipped with a reset button, which is grounding the 2A03s /RST pin (resets CPU and APU).
On NES consoles (not on Famicom consoles), the reset signal is also connected to PPU /SYNC input, causing the picture to be disabled during reset (the PPU registers are left unchanged though). For curiosity, the NES reset signal is also connected to the power LED, the LED goes off when pressing Reset (or when the lockout chip generates a reset).
Anyways, RAM and VRAM is left unaffected, so that the program may recover from reset by invoking a warmboot rather than complete coldboot. As simple example, it may, if desired, preserve high score values, etc.
The cartridge bus doesn't include a reset signal, so mapper registers would be usually left unaffected as well - unless any mappers figure out any ridiculous ways to detect resets, for example by examining address signals.

SRAM Protection via Reset Button
The Reset button is also important for some games with battery backed SRAM: The consoles cartridge bus becomes unstable during power-off, so that SRAM content may get overwritten randomly.
As workaround, some games prompt the user to hold down the reset button during power-off (eg. Maniac Mansion, MMC1). Other games include mappers that can enable/disable SRAM, and don't need that trick (eg. Kirby's Adventure, MMC3).

Controllers - Arcade Machines

VS Unisystem
Arcade Machine with additional coin-detection and DIP-switch inputs. Joysticks and Lightguns are accessed slightly different as on NES.
VS System

Play Choice 10
Arcade Machine with additional coin-detection, DIP-switch, and game-select inputs. The inputs are controlled by a separate Z80 CPU. More info:
Nintendo Playchoice 10

FamicomBox
For hotel rooms.
FamicomBox

Storage Data Recorder

Nintendo Data Recorder
Used to load/save BASIC programs, game positions, or custom edited levels. The recorder is badged "Nintendo", but actually it's just an off-the-shelf audio tape recorder (with speaker and microphone), connected via two cables (load/save) with mono 3.5mm plugs to the "Family Basic Keyboard".

Connection
The Data Recorder connects to the Famicom BASIC Keyboard (which itself connects to 15pin controller port).
Controllers - Typewriter Keyboards
Note: There is also a device called S.D. Station (from Hori) which seems to have Headphone and Recorder IN/OUT sockets... unknown if the recorder part is identical with the Famicom Keyboard recorder connectors.

Software that supports the Data Recorder
```
Castle Excellent (supports both Data Recorder and Turbo File)
Excitebike
Family Basic
Lode Runner (unknown HOW to access LOAD/SAVE menu... maybe via keyboard?)
Mach Rider
Wrecking Crew
```
Note: Accessing the load/save functions may be a difficult task: In Castle Excellent it involves using 2nd Joypad, and to confirm selections by this or that button (for selecting either Data Recorder, or Turbo File mode).

Storage Turbo File

The Turbo File is an external battery-backed RAM-Disk made by ASCII. The device connects to 15pin Famecom expansion port, accordingly, it's been sold only in japan, and it's mainly supported by ASCII's own games.

Turbofile (AS-TF02)
Original version, contains 8Kbytes battery backed RAM, and a 2-position PROTECT switch, plus a LED (unknown purpose).

Turbo File II (TFII)
Newer version, same as above, but contains 32Kbytes RAM, divided into four 8Kbyte slots, which can be selected with a 4-position SELECT switch.

Turbo File Adapter
Allows to connect a Turbo File or Turbo File II to SNES consoles. Aside from the pin conversion (15pin NES to 7pin SNES), it does additionally contain some electronics (for generating a SNES controller ID, and a more complicated protocol for entering the data-transfer phase). Aside from storing SNES game positions, this can be also used to import NES files to SNES games.

Turbo File NES I/O Ports
```
4016h.Write.Bit0 = Data.Out (must be same as OLD data when READING data)
4016h.Write.Bit1 = Reset Address to Offset 0000h
4016h.Write.Bit2 = Data.Clock
4017h.Read.Bit2  = Data.In (can be ignored when WRITING data)
```
To reset the address to offset 0000h:
```
[4016h]=00h           ;reset address to zero
[4016h]=02h           ;release reset
```
To read a bit (and to write-back the same unchanged value):
```
old=[4017h].bit2      ;get old data
[4016h]=old+06h       ;output data and clk=high
[4016h]=old+02h       ;output data and clk=low
```
To write a bit:
```
[4016h]=new+06h       ;output data and clk=high
[4016h]=new+02h       ;output data and clk=low
```
Bytes are usually transferred LSB first. Except, the oldest game (Castle Excellent from 1986) did use MSB first. The address increments after each 8 bits.
To seek a specific address: Reset address to 0000h, then issue dummy reads until the desired address is reached.

Turbo File Memory
The first byte (at offset 0000h) is unused (possibly because that there is a risk that other games with other controller access functions may destroy it); after resetting the address, one should read one dummy byte to skip the unused byte. The used portion is 8191 bytes (offset 0001h..1FFFh). The "filesystem" is

very simple: Each file is attached after the previous file, an invalid file ID indicates begin of free memory (though SOME games seem to keep searching for valid IDs even after that point?).

Turbo File Fileformat (newer files) (1987 and up)
Normal files are formatted like so:
```
  2   ID "AB" (41h,42h)
  2   Filesize (16+N+2) (including title and checksum)
  16  Title in ASCII (terminated by 00h or 01h)
  N   Data Portion
  2   Checksum (all N bytes in Data Portion added together)
```

Turbo File Fileformat (old version) (1986)
The oldest Turbo File game (Castle Excellent from 1986) doesn't use the above format. Instead, it uses the following format, without filename, and with hardcoded memory offset 0001h..01FFh (511 bytes):
```
  1   Don't care (should be 00h)    ;fixed, at offset 0001h
  2   ID AAh,55h                    ;fixed, at offset 0002h..0003h
  508 Data Portion (Data, end code "BEDEUTUN", followed by some unused bytes)
```
CAUTION:
```
  The early version has transferred all bytes in reversed bit-order,
  so above ID bytes AAh,55h will be seen as 55h,AAh in newer versions!
```
Since the address is hardcoded, Castle Excellent will forcefully destroy any other/newer files that are located at the same address. Most newer NES/SNES games (like NES Fleet Commander from 1988, and SNES Wizardry 5 from 1993) do include support for handling the Castle Excellent file. One exception that doesn't support the file is NES Derby Stallion - Zenkoku Ban from 1992.

Deleting Files
Delting files would require to relocate all following files - since the NES has only 2K WRAM, this would require to do the relocation in smaller blocks - which is simply not supported by most games. So far, the only way for deleting files is to remove the batteries (wheras it may take some minutes until the data is lost).
NES Derby Stallion - Zenkoku Ban (1992) does have some limited delete-support: If there isn't free enough memory, then it does allow to erase the last-most file(s); which can be done without relocations.
SNES Wizardry 5 (1993) does allow to delete individual files (the SNES has 128K WRAM, so relocating the following files is pretty simple).

NES Games that do support the Turbo File / Turbo File II
```
  Best Play Pro Yakyuu (1988) ASCII (J)
  Best Play Pro Yakyuu '90 (1990) (J)
  Best Play Pro Yakyuu II (1990) (J)
  Best Play Pro Yakyuu Special (1992) (J)
  Castle Excellent (1986) ASCII (J) (early access method without filename)
  Derby Stallion - Zenkoku Ban (1992) Sonobe Hiroyuki/ASCII (J)
  Downtown - Nekketsu Monogatari (19xx) Technos Japan Corp (J)
  Dungeon Kid (1990) Quest/Pixel (J)
  Fleet Commander (1988) ASCII (J)
  Haja no Fuuin (19xx) ASCII/KGD (J)
  Itadaki Street - Watashi no Mise ni Yottette (1990) ASCII (J)
  Ninjara Hoi! (J)
  Wizardry - Legacy of Llylgamyn (19xx?) (J)
  Wizardry - Proving Grounds of the Mad Overlord (1987) (J)
  Wizardry - The Knight of Diamonds (1991) (J)
```
NES games that do support Turbo File should have a "TF" logo on the cartridge.

Plus... (?)
Searching for: ad17404a4a290109068d1640 (late_nes_recv_joy2_byte opcodes)
Searching for: ad174029044a4a09068d1640 (newer_nes_recv_joy2_byte opcodes)
```
 !! Best Keiba - Derby Stallion (1991) Sonobe Hiroyuki/ASCII (J)
 !! Famicom Shougi - Ryuuousen (1991) I'MAX/HOME DATA (J)
 !! Money Game 2 - Kabutochou no Kiseki, The (1989) Sofel Ltd (J)
 !  Castlequest (U)           US VERSION of Castle Excellent (FUNCTIONAL???)
 !  Kunio 8-in-1 [p1]         (pirate multicart, probably contains a TF-game?)
```

SNES Games that support Turbo File Adapter & Turbo File Twin in TFII-Mode
```
  Ardy Lightfoot (1993)
  Derby Stallion II (1994)
  Derby Stallion III (1995) (supports both TFII and STF modes)
  Derby Stallion 96 (1996) (supports TFII and STF and Satellaview-FLASH-cards)
  Derby Stallion 98 (NP) (1998) (supports both TFII and STF modes)
  Down the World: Mervil's Ambition (1994)
  Kakinoki Shogi (1995) ASCII Corporation
  Tactics Ogre - Let Us Cling Together (supports both TFII and STF modes)
  Wizardry 5 - Heart of the Maelstrom (1992) Game Studio/ASCII (JP)
  BS Wizardry 5 (JP) (Satellaview BS-X version)
```
Note: The US version of Wizardry 5 (1993) contains 99% of the turbo file functions, but lacks one opcode that makes the hardware detection nonfunctional. Wizardry 5 was announced to be able to import game positions from NES to SNES.

Turbo File Schematic
```
         .--------. 74HC368      .---||--GND     .--------.         D0-----PORT1.2
   GND---|1A   /Y1|---NC         | 680pF         |4024  Q1|---NC
```

```
 OUT2---|2A   /Y2|-------------o-----/OUT2    |      Q2|---NC
/OUT2---|3A   /Y3|--------[100]--o------------|/CLK  Q3|---NC
  GND---|/OE1    |               |1000pF      |      Q4|-------.
        | - - - -|               '--||-GND    |      Q5|---NC  |
 OUT1---|4A   /Y4|------------------------o----|RST   Q6|---NC  |
 OUT0---|5A   /Y5|----------.             |    |      Q7|---NC  |
  GND---|6A   /Y6|---NC     |             |    '--------'       |
/OUT2---|/OE2    |          |             | .-------------o-----'    SRAM 8Kx8
        '--------'          |             | |             | .-------.LH5164L-10
 OUT1-----[10K]---GND       |             | |  .--------. '-|A0   NC|--NC
 OUT2-----[10K]---GND       |             | |  |4040  Q1|---|A1   D0|--D0
        .--------. 74HC573  |             | |  |      Q2|---|A12  D1|--D1
/OUT2---|G       |          |             | '--|/CLK  Q3|---|A7   D2|--D2
/OUT2---|/OE     |          |             |    |      Q4|---|A6   D3|--D3
   D7---|D0    Q0|---D6     |             '----|RST   Q5|---|A4   D4|--D4
   D6---|D1    Q1|---D5     |                  |      Q6|---|A3   D5|--D5
   D5---|D2    Q2|---D4     |                  |      Q7|---|A5   D6|--D6
   D4---|D3    Q3|---D3     |                  |      Q8|---|A11  D7|--D7
   D3---|D4    Q4|---D2     o OFF              |      Q9|---|A10 /OE|--GND
   D2---|D5    Q5|---D1   \    PROTECT        |     Q10|---|A9  /WE|--/OUT2
   D1---|D6    Q6|---D0    \   SWITCH         |     Q11|---|A8  /CE|--/GOOD
   D0---|D7    Q7|-------o  o-----------D7    |     Q12|---|A2  CE2|--OUT1
        '--------'       ON                  '--------'   '-------'
```

Plus, a bunch of transistors, resistors, diodes that control supply and battery, power LED, and power /GOOD signal. Battery standby supply is wired to the SRAM, and also to the 74HC368.
Note: Turbo File II uses 32Kx8 SRAM which lacks CE2 pin, so schematic may be a bit different; and of course, A13 and A14 are somehow wired to the 4-position switch for 8K bank selection.

Storage Battle Box

The Battle Box from IGS is an external storage device with 512 byte capactity.
Storage Battle Box I/O Access
Storage Battle Box Filesystem

Battle Box Games

```
Armadillo (J) 199x
Battle Stadium - Senbatsu Pro Yakyuu (J) 1990 IGS
J-League Fighting Soccer - The King of Ace Strikers (J) 1993 IGS
Seiryaku Simulation - Inbou no Wakusei - Shancara (J) 1992 IGS
```

Battle Box Schematic (BB-10)

```
                        .-------.
VCC--||--GND        GND--|RES   Q|---------------------------.
                    GND--|SET    | 4027         TOSHIBA      |    TOSHIBA
PORT1.CLK--[330]--o------|CLK    | Dual J-K     TC89102P     |    TC89102P
                  |  .---|J      | flip-flop   .---------.   |   .---------.
                  |  | .-|K    /Q|-------------|/CS      |   '---|/CS      |
                  |  | | '-------'       VCC---|ORG      |-------|ORG      |
OUT0------------- | -o-o-----------------------|/CLK     |-------|/CLK     |
                  |      .-------.             |         |       |         |
GND-|>|-PORT1.3   | GND--|RES   Q|-------------|DI       |-------|DI       |
GND-|>|-PORT1.4   | GND--|SET    |         .---|DO       |-------|DO       |
GND-|>|-OUT0      '------|CLK    |         |   '---------'       '---------'
GND-|>|-PORT1.CLK   VCC--|J      |         '----------------PORT1.3
                    VCC--|K    /Q|---------------------------PORT1.4
Components:             '-------'
1x 4027 (dual J-K flip-flops)
2x TOSHIBA TC89102P (two 256x8 bit EEPROMs) (=512 bytes total capacity)
4x Z-diodes 6.2V, 1x resistor 330 ohm, 1x capacitor 100nF, 15pin connector
Note: ORG=VCC selects 128x16bit chip mode (ORG=GND would be 256x8bit mode)
```

Storage Battle Box I/O Access

Battle Box

```
[4016h].W.Bit0 --> CLK to EEPROM (and, when set, enable chipselect toggle)
[4017h].R      --> toggle DATA to EEPROM (always)
[4017h].R      --> toggle 1st/2nd EEPROM chipselect (when [4016h].W.Bit0=1)
[4017h].R.Bit3 <-- DATA from EEPROM
[4017h].R.Bit4 <-- DATA to EEPROM (status of current toggled output value)
```

Commands

```
Binary    Toshiba  NES
10000000  01h      80h --> Read
01100000  06h      60h --> Program
00110000  0Ch      30h --> Chip Erase
```

```
10110000  0Dh      B0h --> Busy Monitor
10010000  09h      90h --> Erase/Write Enable
11010000  0Bh      D0h --> Erase/Write Disable
```

battle_box_command(cmd,addr,data)

```
if cmd<>B0h then battle_box_command(B0h,addr,0)         ;-
[4016h]=01h              ;OUT0=1                        ;\start transfer,
dummy=[4017]             ;toggle chipsel                ; toggle chipsel
if (addr AND 1)<>battle_box_chipsel then dummy=[4017]   ; depending on addr.0
battle_box_chipsel = (addr AND 1)                       ;/
battle_box_send_byte(((addr/2) AND FEh)+(addr AND 01h)) ;\send addr/cmd
battle_box_send_byte(cmd+addr/200h)                     ;/
if cmd=B0h  ;Busy Mode                                  ;\wait busy (if any)
  wait until battle_box_recv_bit=0                      ;/
if cmd=80h  ;Read Word                                  ;\
  data1st=battle_box_recv_byte                          ; read data (if any)
  data2nd=battle_box_recv_byte                          ;/
if cmd=60h  ;Write Word                                 ;\
  battle_box_send_byte(data1st)                         ; write data (if any)
  battle_box_send_byte(data2nd)                         ;/
dummy=[4017h]            ;toggle chipsel                ;\finish transfer
[4016h]=00h              ;OUT0=0                        ;/
```

battle_box_detect_which_chip_is_selected:

```
battle_box_command(80h,000h,data)  ;read ID ("B"=42h, or "X"=58h) from addr 0
battle_box_chipsel=battle_box_chipsel XOR data.bit3
```

battle_box_send_byte(data)

```
data=data XOR FFh                ;-invert
for i=7 downto 0
  [4016h]=00h              ;OUT0=0
  dummy=[4017h]            ;toggle DTA.OUT and get output level
  if dummy.bit4<>data.bit(i) then dummy=[4017h] ;toggle DTA.OUT
  [4016h]=01h              ;OUT0=1
next i
```

battle_box_recv_byte:

```
for i=7 downto 0
  data.bit(i)=battle_box_recv_bit
next i
data=data XOR FFh                ;-invert
```

battle_box_recv_bit:

```
[4016h]=00h              ;OUT0=0
dummy=[4017h]            ;read DATA
bit=dummy.bit3
[4016h]=01h              ;OUT0=1
```

Notes

The Program command seems to be able to rewrite data without needing any prior Erase.
After power-on, wait 1ms before accessing the chip. Then send Erase/Write Enable or Disable as first command.

Storage Battle Box Filesystem

Battle Box Filesystem

Overall Format

```
000h..005h  Root Header (6 bytes)
006h..xxxh  File(s) (variable size) (max 1F9h bytes)
yyyh        First Free byte (1Ah) (1 byte)
zzzh..1FFh  Unused Space (FFh filled or garbage or so)
```

Root Header (6 bytes)

```
00h   ID "B" for Chip 1 (42h)
01h   ID "X" for Chip 2 (58h)
02h   00h      ;\maybe total chip size 0200h ?
03h   02h      ;/
04h   LSB      ;\address of first FREE byte (ie. address of the 1Ah byte)
05h   MSB      ;/
```

File Format (0Dh+N+02h bytes per file)

```
00h   42h "B"    ;-File ID (42h="B"=Used File) (or 1Ah=First Free Byte)
01h   SIZE.LSB   ;\Total Filesize (including File ID and Checksum)
02h   SIZE.MSB   ;/
03h   GAME.LSB   ;\Game ID (0000h=Stadium, 0001h=Armadillo, 0002h=Soccer)
04h   GAME.MSB   ;/
05h   GAME.TITLE ;-Game Title (ASCII, 8 chars) (eg. "F SOCCER" or "RMADILLO")
0Dh.. file body  ;-Body (total_filesize minus 0Fh bytes)
...   CHK.LSB    ;\Checksum (all bytes in BODY added together)
...   CHK.MSB    ;/
```

The existing games seem to allow only two files (if there are already two files, they do erase a file before saving a new file; this is done even if there would be enough free space for a third file).

Battle Box Logical Byte Order

```
  Offset  Physical Content                 (Usage)
  000h    1st byte of 1st Word of Chip 1   (ID Byte: "B" for Chip 1)
  001h    1st byte of 1st Word of Chip 2   (ID Byte: "X" for Chip 2)
  002h    2nd byte of 1st Word of Chip 1   (00h)
  003h    2nd byte of 1st Word of Chip 2   (02h)
  004h    1st byte of 2nd Word of Chip 1   (address LSB of first free byte)
  005h    1st byte of 2nd Word of Chip 2   (address MSB of first free byte)
  006h    2nd byte of 2nd Word of Chip 1   (File ID: "B")  (or 1Ah if no file)
  007h    2nd byte of 2nd Word of Chip 2   (File Size LSB) (or unused)
  ...                                      (...)
  1FCh    1st byte of 128th Word of Chip 1
  1FDh    1st byte of 128th Word of Chip 2
  1FEh    2nd byte of 128th Word of Chip 1
  1FFh    2nd byte of 128th Word of Chip 2
```

Emulators should use the same logical byte order (so that data and ASCII filenames show up correctly in .sav files). However, games may more or less randomly initialize the "B" and "X" ID bytes, so Chip 1 and Chip 2 may be exchanged.

Bit Order

The NES software is doing all transfers in reversed bit-order: Namely, commands are opposite as in datasheet (eg. NES: 80h=Read, Toshiba: 01h=Read). Data is also reversed (read-reversion and write-reversion are compensating themselves). The 7bit addresses are also reversed (data is written to shuffled-addresses, but de-shuffled on reading). The 7bit address should be normally followed by a padding bit (with value "0"), the NES software seems to replace that padding bit by the 1st/2nd chipselect bit (maybe somehow intended to allow to use one chip with 8bit-address intead of two chips with 7bit address). Moreover, the NES software puts additional address bits into unused locations of the command byte (maybe also somehow intended for expansion to chips with more memory).

Hori Game Repeater

The Game Repeater (GR-7) from Hori is a device that can record and playback controller data. The data is stored in 16Kbytes of SRAM (assuming that many games read 1 byte per frame, this allows to record about 5 minutes). The SRAM isn't battery backed, but, for permanent storage, the SRAM content can be loaded/saved to external tape recorder.
The device can be used to create "movies" that play back a whole level, and it may be also useful for recording short button-sequences for getting through difficult sections of a game. The playback function may obviously fail if a game contains random elements (such like starting with different random seeds).
The recorded data is taken from the controller input, so it should be impossible to <output> data from the console (for using the SRAM as expansion memory or writing game positions to tape) (unless the device should happen to have support for doing that via another controller pin?).

Chipset/Components

```
  1x Toshiba N (42pin 4bit single-chip CPU: ROM:2048x8, RAM:128x4)
  2x Toshiba TC5563APL-15 (28pin static ram: RAM:8192x8)
  1x TC4011BP 14pin (quad NAND)
  1x TC4062UBP ? (or is it "8"UBP?) 14pin
  1x TC4078BP 16pin (is that "8"BP?)
  1x TC4021BP 16pin (8bit parallel-in, serial-out shift register)
  1x NEC D4094BC ? 16pin (8bit serial-in, parallel-out shift register)
  3x small buttons (tape recorder LOAD,SAVE,VERIFY)
  1x bigger button (sampling START)
  2x two-position switches (REPEAT/RECORD, and STOP/STANDBY)
  2x small LEDs (tape recorder LOAD,SAVE)
  2x cassette connector (tape recorder LOAD,SAVE)
  1x male dsub 15pin connector (to famicom console)
  1x female dsub 15pin connector (to external joystick)
  1x DC input socket (from power supply)
  1x DC output cable (to famicom console)
  1x 7805 ? voltage converter
  1x unknown thing/unknown purpose (on front side) (connector? dip-switches?)
```

Plus one XTAL, plus resistors/diodes/capacitors. There's no battery on the SRAM chips.

Headphones

Headphones can be connected to the 15pin Famicom Expansion Port.

```
  S.D. Station (Hori) (adaptor for Headphones and Tape Recorder)
  Joycard Sanusui SSS (Hudson) (with adapter for headphones)
  Multi Adapter AX-1 (headphones plus auto-fire or so)
  LaserScope/Gun Sight (Konami) (headset: headphones + voice-activated zapper)
```

R.O.B. (Robotic Operating Buddy)

R.O.B. (Robotic Operating Buddy) is a small wireless plastic robot, about 24 cm in height. Command transmission from console to robot is done via flashing TV frames. There is no direct feedback from the robot to the console (except that, in Gyromite, the robot can be maneuvered to push buttons on joypad 2).

R.O.B. Games

The Robot is used by two games:

```
Gyromite (Robot Gyro) (1985) (JP) (US) (EU)
Stack Up (Robot Block) (1985) (JP) (US)
```

Gyromite is a platform game in which the robot must be used to unlock red/blue barriers (by dropping heavy spintops on the red/blue trays). In Stack-Up, one must stack colored plastic blocks in a specific order.

Hardware Accessories

General Accessories:

```
Sun-glasses (dark transparent cover, for compensating brightness of the TV)
4 AA batteries (for powering the robot)
```

Gyromite Accessories:

```
2 robot-gloves with zig-zagged surface (for holding the spinning gyros)
2 gyros (heavy spintops that can be used to depress red/blue trays)
2 red/blue trays (with levers to push buttons on joypad 2)
2 black trays (for depositing gyros when not using them)
1 spinner (accellerates the gyros; for stabilizing them on red/blue trays)
1 battery (for powering the spinner motor)
```

Stack-Up Accessories:

```
2 robot-gloves with flat rubber-coated surface (for holding blocks)
5 differently colored stack-able blocks (to be arranged in specific orders)
5 gray trays (for depositing the blocks on them)
```

Drawings

R.O.B. Technical Drawings

Robot Motors

```
Left/Right Motor  five stopping points (60'/step, 240' total range)
Up/Down Motor     six stopping points (1.4 cm/step, 7 cm total range)
Open/Close Motor  two stopping points (7 cm/step, 7 cm total range)
```

CRT-Signals

```
Command         <------- Command Transfer Phase (13 Frames) -------->
Left     xxx.xxx.___.___.___.GGG.___.GGG.___.GGG.GGG.GGG.___.GGG.___.xxx.xxx
Right    xxx.xxx.___.___.___.GGG.___.GGG.GGG.GGG.___.GGG.___.GGG.___.xxx.xxx
Up       xxx.xxx.___.___.___.GGG.___.GGG.GGG.GGG.GGG.GGG.___.GGG.___.xxx.xxx
Down     xxx.xxx.___.___.___.GGG.___.GGG.___.GGG.___.GGG.GGG.GGG.___.xxx.xxx
Open     xxx.xxx.___.___.___.GGG.___.GGG.GGG.GGG.___.GGG.GGG.GGG.___.xxx.xxx
Close    xxx.xxx.___.___.___.GGG.___.GGG.___.GGG.GGG.GGG.GGG.GGG.___.xxx.xxx
Test     ___.GGG.___.GGG.___.GGG.___.GGG.___.GGG.___.GGG.___.GGG.___.GGG.___
Picture  xxx.xxx.xxx.xxx.xxx.xxx.xxx.xxx.xxx.xxx.xxx.xxx.xxx.xxx.xxx.xxx.xxx
```

Whereas, "xxx"=Picture Frame, "___"=Black Frame, "GGG"=Green Frame "."=Vblank.

Joypad 1 (user input)

In Gyromite, the joypad buttons are assigned as follows (in one game mode, the joypad is also used to move the player character on the screen, in that mode, Start-button must be pressed prior to each robot-command):

```
Left/Right  --> Turn Shoulders & Arms & Shaft Left/Right (in 60' steps)
Up/Down     --> Move Shoulders & Arms Up/Down (in 1.4cm steps)
Button A    --> Open Arms (Release)
Button B    --> Close Arms (Grab)
```

Stack-Up uses some sort of GUI to enter commands, so there's no direct relation between joypad-buttons and motors in that game.

Joypad 2 (attached to Gyromite Trays, for feedback from robot to console)

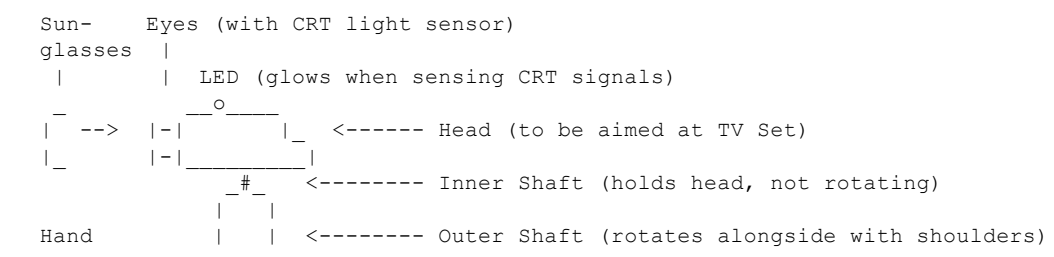
In Gyromite, dropping spintops on trays results in following feedback:

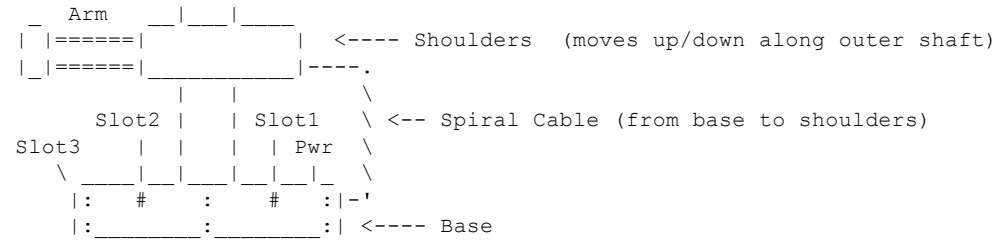
```
Red-Tray (Slot 2)  ---> lever pushes B-Button on Joypad 2
Blue-Tray (Slot 3) ---> lever pushes A-Button on Joypad 2
```

There's no automatic feedback in Stack-Up (the user must manually push Start-Button on Joypad 1 once when having finished a level).

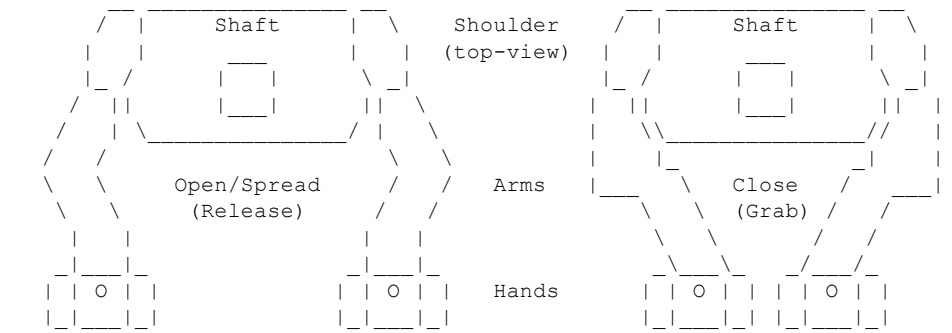
R.O.B. Technical Drawings

Robot (Side View)

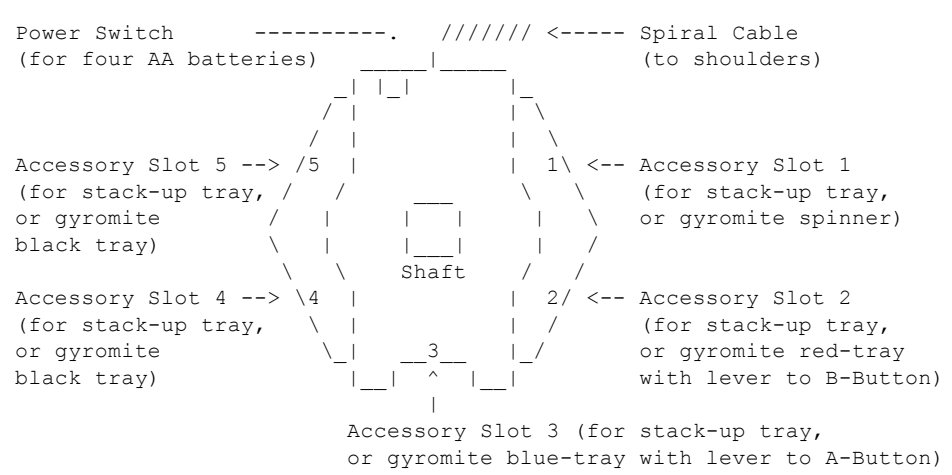




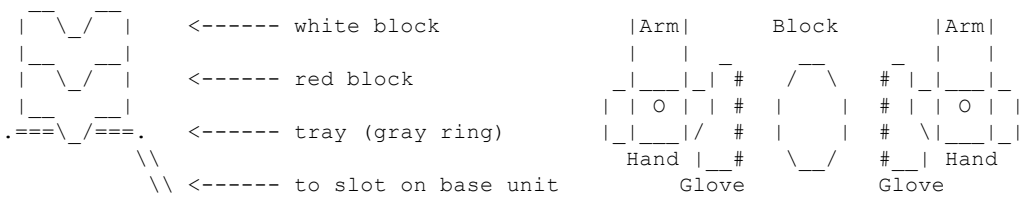
Robot Shoulders/Arms (Top View)



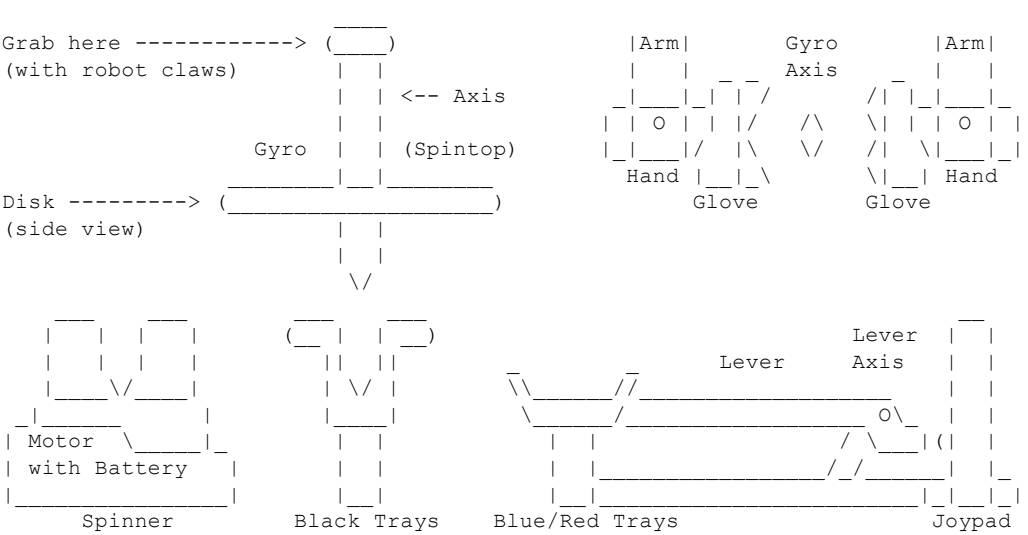
Robot Base (Top View)



Stack-Up Accessories (five Blocks, five Trays, two Gloves)



Gyromite Accessories (two gyros, two gloves, one spinner, four trays)



Cartridges and Mappers

General Cartridge Info

Cartridge Overview
Cartridge ROM-Image File Formats
Cartridge IRQ Counters
Cartridge Bus Conflicts
Cartridge Cicurity Chip (CIC) (Lockout Chip)
Cartridge Cheat Devices
Cartridge Pin-Outs
Cartridge Shell Dimensions

NES Mappers (Numbers as used in .NES fileformat)

Mapper 0: NROM - No Mapper (or unknown mapper)
Mapper 1: MMC1 - PRG/32K/16K, VROM/8K/4K, NT
Mapper 2: UNROM - PRG/16K
Mapper 3: CNROM - VROM/8K
Mapper 4: MMC3 - PRG/8K, VROM/2K/1K, NT, SRAM, IRQ
Mapper 5: MMC5 - BANKING, IRQ, SOUND, VIDEO, MULTIPLY, etc.
Mapper 6: FFE F4xxx - PRG/16K, VROM/8K, NT, IRQ
Mapper 7: AOROM - PRG/32K, Name Table Select
Mapper 8: FFE F3xxx - PRG/32K, VROM/8K, NT, IRQ
Mapper 9: MMC2 - PRG/24K/8K, VROM/4K, NT, LATCH
Mapper 10: MMC4 - PRG/16K, VROM/4K, NT, LATCH
Mapper 11: Color Dreams - PRG/32K, VROM/8K
Mapper 12: FFE F6xxx - Not specified, NT, IRQ
Mapper 13: CPROM - 16K VRAM
Mapper 14: Reportedly SL1632
Mapper 15: X-in-1 - PRG/32K/16K, NT
Mapper 16: Bandai - PRG/16K, VROM/1K, IRQ, EPROM
(above with 24C02)
Mapper 17: FFE F8xxx - PRG/8K, VROM/1K, NT, IRQ
Mapper 18: Jaleco SS8806 - PRG/8K, VROM/1K, NT, IRQ, EXT
Mapper 19: Namcot 106 - PRG/8K, VROM/1K/VRAM, IRQ, SOUND
Mapper 20: Disk System - PRG RAM, BIOS, DISK, IRQ, SOUND
Mapper 21: Konami VRC4A/VRC4C - PRG/8K, VROM/1K, NT, IRQ
Mapper 22: Konami VRC2A - PRG/8K, VROM/1K, NT
Mapper 23: Konami VRC2B/VRC4E - PRG/8K, VROM/1K, NT, (IRQ)
Mapper 24: Konami VRC6A - PRG/16K/8K, VROM/1K, NT, IRQ, SOUND
Mapper 25: Konami VRC4B/VRC4D - PRG/8K, VROM/1K, NT, IRQ
Mapper 26: Konami VRC6B - PRG/16K/8K, VROM/1K, NT, IRQ, SOUND
Mapper 28: Action 53 homebrew X-in-1
Mapper 32: Irem G-101 - PRG/8K, VROM/1K, NT
Mapper 33: Taito TC0190/TC0350 - PRG/8K, VROM/1K/2K, NT, IRQ
Mapper 34: Nina-1 - PRG/32K, VROM/4K
(above and/or BNROM ?)
Mapper 37: Reportedly ZZ
Mapper 39: Reportedly BMC Study & Game 32-in-1
Mapper 40: FDS-Port - Lost Levels
Mapper 41: Caltron 6-in-1
Mapper 42: FDS-Port - Mario Baby
Mapper 43: X-in-1
Mapper 44: 7-in-1 MMC3 Port A001h
Mapper 45: X-in-1 MMC3 Port 6000hx4
Mapper 46: 15-in-1 Color Dreams
(above called "Rumble Station"?)
Mapper 47: 2-in-1 MMC3 Port 6000h
Mapper 48: Taito TC190V
Mapper 49: 4-in-1 MMC3 Port 6xxxh
Mapper 50: FDS-Port - Alt. Levels
Mapper 51: 11-in-1
Mapper 52: 7-in-1 MMC3 Port 6800h with SRAM
Mapper 53: Reportedly Supervision 16-in-1
Mapper 54: Reportedly BMC Noveldiamond 9999999-in-1
Mapper 55: Reportedly BTL Mario1-Malee2
Mapper 56: Pirate SMB3
Mapper 57: 6-in-1
Mapper 58: X-in-1
Mapper 59: Reportedly ..

Mapper 60: Reportedly ..
Mapper 61: 20-in-1
Mapper 62: X-in-1
Mapper 64: Tengen RAMBO-1 - PRG/8K, VROM/2K/1K, NT, IRQ
Mapper 65: Irem H-3001 - PRG/8K, VROM/1K, NT, IRQ
Mapper 66: GNROM - PRG/32K, VROM/8K
Mapper 67: Sunsoft3 - PRG/16K, VROM/2K, IRQ
Mapper 68: Sunsoft4 - PRG/16K, VROM/2K, NT-VROM
(Nantettatte Baseball Double Cassette System for mapper 68)?
Mapper 69: Sunsoft5 FME-7 - PRG/8K, VROM/1K, NT ctrl, SRAM, IRQ
Mapper 70: Bandai - PRG/16K, VROM/8K, NT
Mapper 71: Camerica - PRG/16K
Mapper 72: Jaleco Early Mapper 0 - PRG-LO, VROM/8K
Mapper 73: Konami VRC3 - PRG/16K, IRQ
Mapper 74: Whatever MMC3-style
Mapper 75: Jaleco SS8805/Konami VRC1 - PRG/8K, VROM/4K, NT
Mapper 76: Namco 109 - PRG/8K, VROM/2K
Mapper 77: Irem - PRG/32K, VROM/2K, VRAM 6K+2K
Mapper 78: Irem 74HC161/32 - PRG/16K, VROM/8K
Mapper 79: AVE Nina-3 - VROM/8K
Mapper 80: Taito X-005 - PRG/8K, VROM/2K/1K, NT
Mapper 81: AVE Nina-6
Mapper 82: Taito X1-17 - PRG/8K, VROM/2K/1K
Mapper 83: Cony
Mapper 84: Whatever
Mapper 85: Konami VRC7A/B - PRG/16K/8K, VROM/1K, NT, IRQ, SOUND
Mapper 86: Jaleco Early Mapper 2 - PRG/32K, VROM/8K
Mapper 87: Jaleco/Konami 16K VROM - VROM/8K
Mapper 88: Namco 118
Mapper 89: Sunsoft Early - PRG/16K, VROM/8K
Mapper 90: Pirate MMC5-style
Mapper 91: HK-SF3 - PRG/8K, VROM/2K, IRQ
Mapper 92: Jaleco Early Mapper 1 - PRG-HI, VROM/8K
Mapper 93: 74161/32 - PRG/16K
Mapper 94: 74161/32 - PRG/16K
Mapper 95: Namcot MMC3-Style
Mapper 96: 74161/32 - PRG/32K, CHR/16K/4K, LATCH
Mapper 97: Irem - PRG HI
Mapper 99: VS Unisystem Port 4016h - VROM/8K, (PRG/8K)
Mapper 100: Whatever
Mapper 101: Reportedly ...
Mapper 105: X-in-1 MMC1
Mapper 107: Reportedly Magicseries
Mapper 112: Asder - PRG/8K, VROM/2K/1K
Mapper 113: Sachen/Hacker/Nina
Mapper 114: Super Games
Mapper 115: MMC3 Cart Saint
Mapper 116: Whatever
Mapper 117: Future
Mapper 118: MMC3 TLSROM - PRG/8K, VROM/2K/1K, Banked-NT, SRAM, IRQ
Mapper 119: MMC3 TQROM - PRG/8K, VROM/VRAM/2K/1K, NT, SRAM, IRQ
Mapper 122: Whatever
Mapper 123: Reportedly H2288
Mapper 132: Reportedly TXC 22211
Mapper 133: Sachen
Mapper 137: Reportedly S8259D
Mapper 138: Reportedly S8259C
Mapper 139: Reportedly S8259B
Mapper 140: Reportedly Jaleco JF-xx
Mapper 141: Reportedly S8259A
Mapper 142: Reportedly KS 202
Mapper 143: Reportedly TCA01
Mapper 144: Reportedly AGCI 50282
Mapper 145: Reportedly SA72007
Mapper 146: Reportedly SA0161M
Mapper 147: Reportedly TCU01
Mapper 148: Reportedly SA0037

Mapper 149: Reportedly SA0036
Mapper 150: Reportedly S74LS37AN
Mapper 151: VS Unisystem VRC1 or MMC3 Daughterboards
Mapper 152: Whatever
(above Bandai 74161/32+MIRR)
Mapper 153: No info (reportedly a variant of Mapper 16) BANDAI+WRAM
Mapper 154: Reportedly NAMCOT 118 +A0.D6.MIRR
Mapper 155: Reportedly MMC1A
Mapper 156: Reportedly DAOU 306
Mapper 157: No info (reportedly a variant of Mapper 16) BANDAI+BARCODE (Datach)
Mapper 159: No info (reportedly a variant of Mapper 16) BANDAI+24C01
Mapper 160: Same as Mapper 90
Mapper 161: Same as Mapper 1
Mapper 163: Reportedly NANJING
Mapper 164: Reportedly WAIXING / MARS PRODUCTION
Mapper 165: Reportedly WAIXING SHENGHUO HUIZHANG 2
Mapper 166: Reportedly SUBOR (Russian)
Mapper 167: Reportedly SUBOR (Chinese)
Mapper 168: RacerMate PRG/16K, VRAM/4K, IRQ
Mapper 169: Reportedly N625092
Mapper 170: Reportedly FUJIYA NROM +SECURITY
Mapper 171: Reportedly KAISER KS7058
Mapper 172: Reportedly IDEA-TEK CNROM +SECURITY
Mapper 180: Nihon Bussan - PRG HI
(above UNROM M5)
Mapper 182: Same as Mapper 114
(Mapper 182: Reportedly HOSENKAN ELECTRONICS)
Mapper 183: Reportedly BTL SHUI GUAN PIPE
Mapper 184: Sunsoft - VROM/4K
Mapper 185: VROM-disable
Mapper 186: Reportedly SBX
Mapper 187: No Info --> Mapper 187: Reportedly BTL SFZ297/KOF96/S3DB6
Mapper 188: UNROM-reversed
(Mapper 188: Reportedly BANDAI KARAOKE STUDIO)
Mapper 189: MMC3 Variant
(Mapper 189: Reportedly YOKOSOFT / TXC)
Mapper 191: Reportedly WAIXING MMC3 +XRAM.4K +CRAM.2K
Mapper 192: Reportedly WAIXING MMC3 +XRAM.4K +CRAM.4K
Mapper 193: Reportedly MEGA SOFT (NTDEC)
Mapper 194: Reportedly WAIXING MMC3 +XRAM.4K +CRAM.2K (alt)
Mapper 195: Reportedly WAIXING MMC3 +XRAM.4K +CRAM.4K (alt)
Mapper 196: Reportedly MMC3 +A0/A2
Mapper 197: Reportedly SUPER FIGHTER III
Mapper 198: Reportedly WAIXING MMC3 +XRAM.4K
Mapper 199: Reportedly WAIXING MMC3 +XRAM.4K +CRAM.8K
Mapper 200: Reportedly BMC 1200/36-IN-1
Mapper 201: Reportedly BMC 21/8-IN-1
Mapper 202: Reportedly BMC 150-IN-1
Mapper 203: Reportedly BMC 35-IN-1
Mapper 204: Reportedly BMC 64-IN-1
Mapper 205: Reportedly BMC 15/3-IN-1
Mapper 206: Reportedly DE1ROM (aka DxROM, aka pre-MMC3)
Mapper 207: Reportedly TAITO X-005 +MIRR
Mapper 208: Reportedly GOUDER BTL SF4
Mapper 209: Reportedly J.Y.COMPANY +EXT.MIRR.CTRL
Mapper 210: Reportedly NAMCOT
Mapper 211: Reportedly J.Y.COMPANY +EXT.MIRR.ON
Mapper 212: Reportedly BMC SUPER HIK 300-IN-1
Mapper 213: Reportedly BMC 9999999-IN-1
Mapper 214: Reportedly BMC SUPER GUN 20-IN-1
Mapper 215: Reportedly BMC SUPER 308 3-IN-1 / M-E3
Mapper 216: Reportedly RCM MAGIC JEWELRY 2
Mapper 217: Reportedly BMC SPC009
Mapper 218: Nocash Single-Chip
Mapper 222: Dragon Ninja
Mapper 225: X-in-1
Mapper 226: X-in-1

Mapper 227: X-in-1
Mapper 228: X-in-1 Homebrewn
Mapper 229: 31-in-1
Mapper 230: X-in-1 plus Contra
Mapper 231: 20-in-1
Mapper 232: 4-in-1 Quattro Camerica
Mapper 233: X-in-1 plus Reset
Mapper 234: Maxi-15
(Mapper 234: Reportedly AVE D-1012)
Mapper 235: Reportedly BMC GOLDEN GAME 150/260-IN-1
Mapper 236: Reportedly BMC 800/70-IN-1
Mapper 240: C&E/Supertone - PRG/32K, VROM/8K
Mapper 241: X-in-1 Education
(Mapper 241: Reportedly MXMDHTWO / TXC)
Mapper 242: Waixing - PRG/32K, NT
Mapper 243: Sachen Poker - PRG/32K, VROM/8K
(Mapper 243: Reportedly SACHEN 74LS374N)
Mapper 244: C&E - PRG/32K, VROM/8K
(Mapper 244: Reportedly C&E DECATHLON)
Mapper 245: No Info (seems to be some sort of MMC3 variant)
(Mapper 245: Reportedly WAIXING MMC3 +EX.PRG)
Mapper 246: C&E - PRG/8K, VROM/2K, SRAM
(Mapper 246: Reportedly C&E PHONE SERM BERM)
Mapper 248: No Info
Mapper 249: No Info --> Mapper 249: Reportedly WAIXING MMC3 +EX.PRG/CHR
Mapper 250: No Info --> Mapper 250: Reportedly NITRA MMC3
Mapper 251: No Info
Mapper 252: No Info --> Mapper 252: Reportedly WAIXING SAN GUO ZHI
Mapper 254: No Info --> Mapper 254: Reportedly BTL PIKACHU Y2K
Mapper 255: X-in-1 - (Same as Mapper 225)
XXX NROM-368 variant of NROM (and similar CPROM, CNROM variants)
Note: Mapper numbers in range 0..255 seem to be ALL used by now.
Mapper numbers 256..4095 can be defined in the "NES 2.0" file format.

More Info
A still very incomplete (but growing) mapper list can be found here:
```
http://wiki.nesdev.com/w/index.php/Category:INES_Mappers
(when inventing new mapper numbers, best include them in above list)
```
There is some very detailed info on Kevin Horton's kevtris pages, which are, unfortunately very messy. I think, most of the info is lost in chaos.
A lot of mappers are hiding here:
```
http://kevtris.org/mappers/mappers.html   ;<-- secret undocumented link!?
```
For some reason, this page also hides some mappers:
```
http://kevtris.org/nes/mappers.html       ;<-- official link on NES page!?
```
Plus some completely unlinked pages for konami mappers, eg. for VRC 7:
```
http://kevtris.org/nes/vrcvii.txt         ;<-- or vrcvi for VRC 6
```
And, there might be a couple of further mapper pages hiding elsewhere.

Cartridge Overview

Standard Mappers
There are more than hundred different mappers, though most are unimportant, the standard types are Mapper 0,1,2,3,4 for ROM-cartridges. And Mapper 20 for Floppy disks.

Type	Games	Percent
Mapper 0 (NROM)	446	12.5%
Mapper 1 (MMC1)	723	20.3%
Mapper 2 (UNROM)	397	11.2%
Mapper 3 (CNROM)	273	7.7%
Mapper 4 (MMC3)	784	22.1%
Mapper 20 (FDS)	?	x.x%
Other Mappers	932	26.2%
Total	3555	100.0%

Other mapper types are used by less than 2% per type, though together they make up 26.2%.

Non-standard Mappers
Some mappers like MMC5 have been used only in a few newer cartridges. Several third-party companies (Konami, Irem, Jaleco, Bandai, Sunsoft, etc.) have developed their own mappers which are used only for their own games. That mappers may be important to play specific games, though they are often used only by 1-2 titles.
Also, there have been various pirate / multi-game cartridges manufactured, containing modified ROM-images with custom mapper circuits, these mappers are completely unimportant since the original games used standard mappers.

Maximum manufactured ROM Size
The largest single NES game that I know of is Dragonquest 4 / Dragon Warrior 4 (XXX that's wrong). It has 1 megabyte of program ROM. Also, the Japanese game Metal Slader Glory has 512K of PRG and 512K of CHR ROM, making it also a full megabyte. Several pirate/unlicenced Famicom games are also pretty large.

Minimum manufactured ROM Size
Although the .NES fileformat deems 16K PRG ROM games as the minimum, there have been some 8K games manufactured, such as Galaxian. Later on, skilled programmers have learned to squeeze better code into even less memory, but nowadays most are probably dead.

Banking Granularity
PRG ROM is usually split into banks of 8K, 16K, or 32K, a few mappers like MMC5 also have smaller 1K SRAM banks. VROM is usually split into banks of 1K, 2K, 4K, or 8K.

Note on Mapper Descriptions
In this document Bank Selections are sometimes described as 4K, or as 4x1K.
```
The linear address for a 4K bank   is (N*4096).
The linear address for a 4x1K bank is ((N AND (NOT 3))*1024).
```
Ie. in the latter case, the bank value is specified in 1K-steps, with lower bits ignored, and rounded down to a 4K boundary.

Mapper Reset
In general, mapper registers are uninitialized on reset/power-up, and should be initialized by software; if memory at FFFCh is mappable, then valid reset vectors (and reset handlers) should be contained in all banks.
There is no reset signal available on cartridge bus, possible ways to detect reset are to sense inactivity on A0 or PHI2 lines, to sense reads from the reset vector at FFFCh, or to use power-up capacitors for coldboot detection (though that not for warmboot).
Caution: Several mappers in this document are described to have initial settings on reset or power-up. Most of that info has been taken from other documents, in most cases that is unconfirmed, and probably incorrect. A few mappers seem to be actually containing reset circuits though.

Cartridge ROM-Image File Formats

iNES Format (.NES)
This fileformat and mapper-numbers have been designed/assigned by Marat Fayzullin (author of iNES emulator), please contact him if you want to make any changes to the format or numbers. The file header is 16 bytes:
```
00h  File ID ("NES",1Ah) (aka 4Eh,45h,53h,1Ah)
04h  Number of 16K PRG-ROM pages
05h  Number of 8K CHR-ROM pages (00h=None / VRAM)
06h  Cartridge Type LSB
 Bit7-4  Mapper Number (lower 4bits)
 Bit3    1=Four-screen VRAM layout
 Bit2    1=512-byte trainer/patch at 7000h-71FFh
 Bit1    1=Battery-backed SRAM at 6000h-7FFFh, set only if battery-backed
 Bit0    0=Horizontal mirroring, 1=Vertical mirroring
07h  Cartridge Type MSB (ignore this and further bytes if Byte 0Fh nonzero)
 Bit7-4  Mapper Number (upper 4bits)
 Bit3-2  Reserved (zero)
 Bit1    1=PC10 game (arcade machine with additional 8K Z80-ROM) (*)
 Bit0    1=VS Unisystem game (arcade machine with different palette)
08h  Number of 8K RAM (SRAM?) pages (usually 00h=None-or-not-specified)
09h  Reserved (zero)
0Ah  Reserved (zero) (sometimes 03h,10h,13h,30h,33h purpose unknown) (*)
0Bh  Reserved (zero)
0Ch  Reserved (zero)
0Dh  Reserved (zero)
0Eh  Reserved (zero)
0Fh  Nonzero if [07h..0Fh]=GARBAGE, if so, assume [07h..0Fh]=ALL ZERO (*)
```
The overall file structure is, in following order:
```
16 byte      Header
512 byte     Trainer             ;-if any, see Byte 6, Bit2, mainly FFE games
N*16K        PRG-ROM             ;-see Byte 4
N*8K         CHR-ROM             ;-if any, see Byte 5
8K       (*) PC10 INST-ROM       ;-if any, see Byte 7, Bit1
16 byte  (*) PC10 PROM Data      ;-if any, see Byte 7, Bit1 ;\required, but
16 byte  (*) PC10 PROM CounterOut;-if any, see Byte 7, Bit1 ;/often missing
128 byte (*) Title               ;-if any (rarely used)
```
Items marked as (*) are regulary used, but not offical part of the format.
Many PC10 files declare Z80-ROM as additional VROM bank (instead Byte7/Bit1).

"NES v2.0"
This is an extension to the above iNES format, specs are at:
```
http://wiki.nesdev.com/w/index.php/NES_2.0
```
according to that webpage, it's supported by only 1 emulator.

.UNF - Universal NES Image File Format (UNIF) by Tennessee Carmel-Veilleux

A "newer" fileformat dated back to 2000, the relation between iNES mapper numbers and UNIF MAPR names is still undocumented, and of course nobody uses files with .UNF extension. Still, it's having one or two useful features, and may become more popular if somebody dares to fix the MAPR problem, and to rename it from .UNF to .NES extension.

File Header (32 bytes)

```
00h-03h: "UNIF" tag identifier
04h-07h: Revision number ("currently 4, for REV 7b, Revision 6 of UNIF" Huh!)
08h-1Fh: Reserved for future usage
```

The header is followed by whatever chunks, all chunks are optional, and may or may not be included in the file, only the PRG0 one is obviously required. Software may skip any chunks which are uninteresting or unrecognized, each chunk formatted as such:

```
00h-03h: Chunk ID string (4-letter ASCII, described below)
04h-07h: Length of Data Block in bytes (excluding above ID and length entry)
08h... : Data
```

MAPR - Board Name (aka Mapper) (ASCIZ, suggested max: 32 chars)

```
This uses ASCIZ strings to describe the board names (instead of iNES
mapper numbers), it's meant to be more specific than mapper numbers,
for example, it's using different names for different MMC1-boards.
http://www.parodius.com/~veilleux/boardtable.txt
http://www.parodius.com/~veilleux/boardnames
```

PRG0..PRGF - Binary data of the PRG ROM

CHR0..CHRF - Binary data of the CHR ROM (aka VROM in general)

```
Normally using only PRG0 (and CHR0, if VROM used).
In rare cases, if the cart contains more than 1 PRG (or CHR) ROM chip,
then PRG1-F and CHR1-F may be used for the additional chips.
```

TVCI - Television Standards Compatability Information (One Byte)

```
00h  60Hz/NTSC (USA, Japan, etc.)
01h  50Hz/PAL  (Germany, etc.)
02h  Compatible with both 50Hz and 60Hz refresh rates
```

CTRL - Controllers used by the cartridge (currently only 1 Byte / 8bit)

```
Bit0   Regular Joypad
Bit1   Zapper
Bit2   R.O.B.
Bit3   Arkanoid Controller (Paddle)
Bit4   Power Pad
Bit5   Four-Score adapter (NES 4-player adapter) (Not Famicom adapter!)
Bit6-7 Reserved
```

MIRR - Name Table Mirroring (1 Byte)

```
00h  Two-Screen Horizontal Mirroring (Hard Wired)
01h  Two-Screen Vertical Mirroring (Hard Wired)
02h  Single-Screen BLK0 (Hard Wired)
03h  Single-Screen BLK1 (Hard Wired)
04h  Four-Screens of VRAM (Hard Wired)
05h  Mirroring Controlled By Mapper Hardware
```

BATR - Battery installed on Board (1 dummy byte)

```
Presence of this chunk means yes, absence means no.
```

NAME - Game Title, ASCIZ String

```
Game Title
```

READ - Readme/Comments/Notes/Credits

```
Probably some sort of ASCII text of unspecified formatting
```

VROR - Allow homebrewn games to over-write VROM (1 dummy byte)

```
Presence of this chunk means yes, absence means no.
```

PCK0..PCKF - 32-bit CRCs for PRG0..PRGF blocks (4 bytes, each)

CCK0..CCKF - 32-bit CRCs for CHR0..CHRF blocks (4 bytes, each)

```
Intended "to make sth sure on EPROMs" ;-) Checksum algorythm not specified.
```

DINF - Dumper information block (204 Bytes)

```
100 bytes  ASCIZ name of the person who dumped the cart
4   bytes  day, month, year-lsb, year-msb when cartridge was dumped
100 bytes  ASCIZ agent "name of the ROM-dumping means used"
```

Note: All words and dwords in header/chunks stored LSB first.

Cartridge IRQ Counters

The MMC3's scanline counter

The MMC3 bases it's scanline counter on PPU address line A13 (which is why IRQ's can be fired off manually by toggling A13 a bunch of times via $2006).

A13 cycles (0 -> 1) exactly 42 times per scanline, whereas the CPU count of cycles per scanline is not an exact integer (113.67).

Konami IRQ counters

Running at 113.75 cycles, including during VBlank.

Famicom Disk System IRQ counters

Allows to count clock cycles, rather than scanlines.

Cartridge Bus Conflicts

/PRG Pin - Indicates CPU Memory Access to 8000h-FFFFh (LOW=Read or Write)
R/W Pin - Indicates CPU Direction (LOW=Write, HIGH=Read)

The /PRG Pin indicates read-or-write access to the PRG ROM memory area at 8000h-FFFFh, the read/write direction could be determined by R/W Pin. Most cartridges are ignoring the R/W signal, and are assuming all memory accesses to be read-requests (which makes sense since ROM is read-only). However, many cartridges have write-only mapper ports at 8000h-FFFFh, activated when /PRG=LOW and R/W=LOW. Many of these carts (especially simple TTL circuits like UNROM, CNROM, etc.) still activate ROM on any /PRG signal without checking R/W, so that ROM outputs data simultaneously with the CPU writing data to the mapper port.
Common workaround is to write to a ROM address that contains a value equal to the written value. Also one could probably write to an address that contains FFh (low signals are stronger than high signals, so the values would be logically, or 'forcefully' ANDed. Don't know about any games using that method though). Another workaround is to interprete the lower address bits instead of the data bits (eg. Mapper 225), that's of course still producing a bus-conflict (shortcut), but without disturbing the program flow.

Cartridge Cicurity Chip (CIC) (Lockout Chip)

Lockout chips are contained in most NES consoles, and in all NES cartridges. Both chips are generating an identical serial data stream, and, everything works fine if the streams match. Otherwise the chip in the console issues a reset signal to the NES.

That mechanism is intended both to prevent software piracy, and to prevent third party developers from distributing (unlicensed) games. Also, an US cartridge won't work on a UK console and vice versa, because of different lockout chip versions used in different countries.

The chips have been invented when releasing the NES in 1985, the original Famicom didn't have lockout chips (nor do newer Famicoms; for backwards compatibility reasons). There have been also some lockout chip revisions to make newer NES consoles incompatible with some "faked" lockout chips from other manufacturers.

The lockout chips are 4bit microprocessors in DIL16 package with a built-in program called 10NES, and are connected to S0,S1,S2,4MHz pins of the 72pin NES cartridge slot. Pin 4 of the chip is used to configure the chip:

```
HIGH (+5V)  Lock, used in console
LOW  (GND)  Key, used in cartridge
```

To disable the chip in the console, wire that pin to GND instead of 5V, the NES will then work with any cartridges with or without lockout chip, and with cartridges from other countries - though some NTSC (60Hz) games may be incompatible with PAL (50Hz) refresh rates, and vice versa.

NES cartridges are required to contain a CIC chip (security chip aka lockout chip). The CIC is a small 4bit CPU with built-in ROM. An identical CIC is located in the NES console. The same 4bit CPU (but with slightly different code in ROM) is also used in SNES consoles/cartridges.
The CIC in the console is acting as "lock", and that in the cartridge is acting as "key". The two chips are sending random-like bitstreams to each other, if the data (or transmission timing) doesn't match the expected values, then the "lock" issues a RESET signal to the console. Thereby rejecting cartridges without CIC chip (or such with CICs for wrong regions).

CIC Details
Cartridge CIC Pseudo Code
Cartridge CIC Instruction Set
Cartridge CIC Notes
Cartridge CIC Versions
Cartridge CIC Pinouts
And, for Tengen CIC clone's instruction set:
Cartridge CIC Tengen Clone

Cartridge CIC Pseudo Code

CicMain

```
  CicInitFirst, CicInitTiming, CicRandomSeed, CicInitStreams
  time=data_start, a=1, noswap=1, if snes then noswap=0
 mainloop:
  for x=a to 0Fh
    if nes then Wait(time-5), else if snes then (time-7)     ;\verify idle
    if (nes_6113=0) and (P0.0=1 or P0.1=1) then Shutdown     ;/
    Wait(time+0)                                             ;\
    if (console xor snes) then a=[00h+x].0, else a=[10h+x].0 ; output data
    if noswap then P0.0=a, else P0.1=a                       ;/
    Wait(time+2-data_rx_error)                               ;\
    if (console xor snes) then a=[10h+x].0, else a=[00h+x].0 ; verify input
    if noswap then a=(a xor P0.1), else a=(a xor P0.0)       ;
    if a=1 then Shutdown                                     ;/
    Wait(time+3)                                             ;\output idle
    if noswap then P0.0=0, else P0.1=0                       ;/
```

```
  if snes then time=time+92, else if nes then time=time+79
next x
CicMangle(00h), CicMangle(10h)                      ;\mangle
if snes then CicMangle(00h), CicMangle(10h)         ; (thrice on SNES)
if snes then CicMangle(00h), CicMangle(10h)         ;/
if snes then noswap=[17h].0   ;eventually swap input/output pins (SNES only)
a=[17h]
if a=0 then a=1, time=time+2
if snes then time=time+44, else if nes then time=time+29
goto mainloop
```

CicMangle(buf)

```
for i=[buf+0Fh]+1 downto 1
  a=[buf+2]+[buf+3h]+1
  if a<10h then x=[buf+3], [buf+3]=a, a=x, x=1, else x=0
  [buf+3+x]=[buf+3+x]+a
  for a=x+6 to 0Fh, [buf+a]=[buf+a]+[buf+a-1]+1, next a
  a=[buf+4+x]+8, if a<10h then [buf+5+x]=[buf+5+x]+a, else [buf+5+x]=a
  [buf+4+x]=[buf+4+x]+[buf+3+x]
  [buf+1]=[buf+1]+i
  [buf+2]=NOT([buf+2]+[buf+1]+1)
  time=time+84-(x*6)
next i
```

Note: All values in [buf] are 4bit wide (aka ANDed with 0Fh).

CicInitFirst

```
timer=0                     ;reset timer (since reset released)
P0=00h
console=P0.3                ;get console/cartridge flag
if console
  while P0.2=1, r=r+1       ;get 4bit random seed (capacitor charge time)
  P1.1=1, P1.1=0            ;issue reset to CIC in cartridge
  timer=0                   ;reset timer (since reset released)
if nes_6113 and (console=1)
  Wait(3), nes_6113_in_console=1, P0.0=1      ;request special 6113 mode
if nes_6113 and (console=0)
  Wait(6), nes_6113_in_console=P0.1           ;check if 6113 mode requested
```

CicRandomSeed

```
time=seed_start
for i=0 to 3                ;send/receive 4bit random seed (r)
  bit=((i+3) and 3)         ;bit order is 3,0,1,2 (!)
  if console=1 Wait(time+0+i*15), P0.0=r.bit, Wait(time+3+i*15), P0.0=0 ;send
  if console=0 Wait(time+2+i*15), r.bit=P0.1                           ;recv
next i
```

CicInitStreams

```
if snes
  if ntsc then x=9, else if pal then x=6
  [01h..0Fh]=B,1,4,F,4,B,5,7,F,D,6,1,E,9,8   ;init stream from cartridge (!)
  [11h..1Fh]=r,x,A,1,8,5,F,1,1,E,1,0,D,E,C   ;init stream from console   (!)
if nes_usa                ;3193A
  [01h..0Fh]=1,9,5,2,F,8,2,7,1,9,8,1,1,1,5   ;init stream from console
  [11h..1Fh]=r,9,5,2,1,2,1,7,1,9,8,5,7,1,5   ;init stream from cartridge
  if nes_6113_in_console then overwrite [01h]=5 or so ???   ;special-case
if nes_europe             ;3195A
  [01h..0Fh]=F,7,B,E,F,8,2,7,D,7,8,E,E,1,5   ;init stream from console
  [11h..1Fh]=r,7,B,D,1,2,1,7,E,6,7,A,7,1,5   ;init stream from cartridge
if nes_hongkong_asia      ;3196A
  [01h..0Fh]=E,6,A,D,F,8,2,7,E,6,7,E,E,E,A   ;init stream from console
  [11h..1Fh]=r,6,A,D,E,D,E,8,E,6,7,A,7,1,5   ;init stream from cartridge
if nes_uk_italy_australia ;3197A
  [01h..0Fh]=3,5,8,9,3,7,2,8,8,6,8,5,E,E,B   ;init stream from console
  [11h..1Fh]=r,7,9,A,A,1,6,8,5,8,9,1,5,1,7   ;init stream from cartridge
if_nes_famicombox         ;3198A
  (unknown)
```

Note: In most cases, the PAL region changes are simply inverted or negated NTSC values (not/neg), except, one NES-EUR value, and most of the NES-UK values are somehow different. The rev-engineered NES-UK values may not match the exact original NES-UK values (but they should be working anyways).

CicInitTiming

```
if snes_d411           -> seed_start=630, data_start=817   ;snes/ntsc
if snes_d413           -> (unknown?) (same as d411?)       ;snes/pal
if nes_3193            -> (seems to be same as nes_3195?)  ;nes/usa (v1)
if nes_3195            -> seed_start=32, data_start=200    ;nes/europe
if nes_3196            -> (unknown?)                       ;nes/asia
if nes_3197            -> (unknown?) ("burns five")        ;nes/uk
if nes_6113            -> seed_start=32, data_start=201    ;nes/usa (v2)
if nes_6113_in_console -> seed_start=33, data_start=216    ;nes/special
if nes_tengen          -> seed_start=32, data_start=201    ;nes/cic-clone
;now timing errors...
```

```
  data_rx_error=0  ;default
  if console=0 and nes_3193a -> randomly add 0 or 0.25 to seed_start/data_start
  if console=0 and snes_d413 -> always add 1.33 to seed_start/data_start (bug)
  if console=0 and nes_6113  -> data_rx_error=1 (and maybe +1.25 on seed/data?)
  if other_chips & chip_revisions -> (unknown?)
```

Note: 3197 reportedly "burns five extra cycles before initialization", but unknown if that is relative to 3193 <or> 3195 timings, and unknown if it applies to <both> seed_start and data_start, and unknown if it means 1MHz <or> 4MHz cycles.

Note: The "data_rx_error" looks totally wrong, but it is somewhat done intentionally, so there might be a purpose (maybe some rounding, in case 6113 and 3193 are off-sync by a half clock cycle, or maybe an improper bugfix in case they are off-sync by 1 or more cycles).

Wait(time)

Wait until "timer=time", whereas "timer" runs at 1MHz (NES) or 1.024MHz (SNES). The "time" values are showing the <completion> of the I/O opcodes (ie. the I/O opcodes <begin> at "time-1").

Shutdown (should never happen, unless cartridge is missing or wrong region)

```
  a=0, if nes then time=830142, else if snes then time=1037682
 endless_loop:          ;timings here aren't 100.000% accurate
  if nes_3195 then time=xlat[P1/4]*174785  ;whereas, xlat[0..3]=(3,2,4,5)
  if (console=0) and (snes or nes_6113) then P0=03h, P1=01h
  if (console=1) then P1=a, Wait(timer+time), a=a xor 4  ;toggle reset on/off
  goto endless_loop
```

Cartridge CIC Instruction Set

CIC Registers

```
  A  4bit Accumulator
  X  4bit General Purpose Register
  L  4bit Pointer Register (lower 4bit of 6bit HL)
  H  2bit Pointer Register (upper 2bit of 6bit HL)
  C  1bit Carry Flag (changed ONLY by "set/clr c", not by "add/adc" or so)
  PC 10bit Program Counter (3bit bank, plus 7bit polynomial counter)
```

CIC Memory

```
  ROM   512x8bit (program ROM) (NES/EUR=768x8) (max 1024x8 addressable)
  RAM   32x4bit  (data RAM) (max 64x4 addressable)
  STACK 4x10bit  (stack for call/ret opcodes)
  PORTS 4x4bit   (external I/O ports & internal RAM-like ports) (max 16x4)
```

Newer CIC Opcodes (6113, D411) (and probably F411,D413,F413)

```
  00      nop             no operation (aka "addsk A,0" opcode)
  00+n    addsk  A,n      add, A=A+n, skip if result>0Fh
  10+n    cmpsk  A,n      compare, skip if A=n
  20+n    mov    L,n      set L=n
  30+n    mov    A,n      set A=n
  40      mov    A,[HL]   set A=RAM[HL]
  41      xchg   A,[HL]   exchange A <--> RAM[HL]
  42      xchgsk A,[HL+]  exchange A <--> RAM[HL], L=L+1, skip if result>0Fh
  43      xchgsk A,[HL-]  exchange A <--> RAM[HL], L=L-1, skip if result<00h
  44      neg    A        negate, A=0-A                ;(used by 6113 mode)
  45      ?
  46      out    [L],A    output, PORT[L]=A
  47      out    [L],0    output, PORT[L]=0
  48      set    C        set carry, C=1
  49      clr    C        reset carry, C=0
  4A      mov    [HL],A   set RAM[HL]=A
  4B      ?
  4C      ret             return, pop PC from stack
  4D      retsk           return, pop PC from stack, skip
  4E+n    ?
  52      movsk  A,[HL+]  set A=RAM[HL], L=L+1, skip if result>0Fh
  53      ?                    (guess: movsk  A,[HL-])
  54      not    A        complement, A=A XOR 0Fh
  55      in     A,[L]    input, A=PORT[L]
  56      ?
  57      xchg   A,L      exchange A <--> L
  58+n    ?
  5C      mov    X,A      set X=A
  5D      xchg   X,A      exchange X <--> A
  5E      ???             "SPECIAL MYSTERY INSTRUCTION" ;(used by 6113 mode)
  5F      ?
  60+n    testsk [HL].n   skip if RAM[HL].Bit(n)=1
  64+n    testsk A.n      skip if A.Bit(n)=1
  68+n    clr    [HL].n   set RAM[HL].Bit(n)=0
  6C+n    set    [HL].n   set RAM[HL].Bit(n)=1
  70      add    A,[HL]   add, A=A+RAM[HL]
  71      ?                    (guess: addsk  A,[HL])
  72      adc    A,[HL]   add with carry, A=A+RAM[HL]+C
  73      adcsk  A,[HL]   add with carry, A=A+RAM[HL]+C, skip if result>0Fh
```

```
74+n    mov    H,n      set H=n  ;2bit range, n=0..3 only (used: 0..1 only)
78+n mm jmp    nmm      long jump, PC=nmm
7C+n mm call   nmm      long call, push PC+2, PC=nmm
80+nn   jmp    nn       short jump, PC=(PC AND 380h)+nn
-       reset           PC=000h
```

Note: "skip" means "do not execute next instruction"

Older CIC Opcodes (3195) (and probably 3193,3196,3197,etc.)

```
Exchanged opcodes 48 <--> 49 (set/clr C)
Exchanged opcodes 44 <--> 54 (neg/not A)
ROM Size is 768x8 (although only 512x8 are actually used)
```

Note

The CIC is a 4bit Sharp CPU (maybe a Sharp SM4, but no datasheet exists) (the instruction seems to be an older version of that in the Sharp SM5K1..SM5K7 datasheets).

Cartridge CIC Notes

Program Counter (PC)

The 10bit PC register consists of a 3bit bank (which gets changed only by call/jmp/ret opcodes), and a 7bit polynomial counter (ie. not a linear counter). After fetching opcode bytes, PC is "incremented" as so:

```
PC = (PC AND 380h) + (PC.Bit0 XOR PC.Bit1)*40h + (PC AND 7Eh)/2
```

Ie. the lower 7bit will "increment" through 127 different values (and wrap to 00h thereafter). Address 7Fh is unused (unless one issues a JMP 7Fh opcode, which would cause the CPU to hang on that address).

```
Format     <------------- Valid Address Area ---------->      <--Stuck-->
Linear     00 01 02 03 04 05 06 07 08 09 0A ... 7C 7D 7E  or  7F 7F 7F 7F
Polynomial 00 40 60 70 78 7C 7E 3F 5F 6F 77 ... 05 02 01  or  7F 7F 7F 7F
```

To simplify things, programming tools like assemblers/disassemblers may use "normal" linear addresses (and translate linear/polynomial addressses when needed - the polynomial addresses are relevant only for encoding bits in jmp/call opcodes, and for how the data is physically arranged in the chip ROMs and in ROM-images).

ROM-Images

The existing ROM-images are .txt files, containing "0" and "1" BITS in ASCII format, arranged as a 64x64 (or 96x64) matrix (as seen in decapped chips).

```
Line 1..32   --->   Address X+9Fh..80h            ;\Lines (Y)
Line 33..64  --->   Address X+1Fh..00h            ;/
Column  1+(n*W) --> Data Bit(n) of Address 000h+Y ;\  ;\
Column  2+(n*W) --> Data Bit(n) of Address 020h+Y ;   ; Columns (X)
Column  3+(n*W) --> Data Bit(n) of Address 040h+Y ;   ;
Column  4+(n*W) --> Data Bit(n) of Address 060h+Y ;   ; chips with 200h-byte
Column  5+(n*W) --> Data Bit(n) of Address 100h+Y ;   ; (W=8) (64x64 bits)
Column  6+(n*W) --> Data Bit(n) of Address 120h+Y ;   ;
Column  7+(n*W) --> Data Bit(n) of Address 140h+Y ;   ;
Column  8+(n*W) --> Data Bit(n) of Address 160h+Y ;   ;/
Column  9+(n*W) --> Data Bit(n) of Address 200h+Y ;
Column 10+(n*W) --> Data Bit(n) of Address 220h+Y ;  chips with 300h-byte
Column 11+(n*W) --> Data Bit(n) of Address 240h+Y ;  (W=12) (96x64 bits)
Column 12+(n*W) --> Data Bit(n) of Address 260h+Y ;/
```

Cautions: The bits are inverted (0=1, 1=0) in some (not all) dumps. Mind that the bytes are arranged in non-linear polynomial fashion (see PC register). Recommended format for binary ROM-images would be to undo the inversion (if present), and to maintain the polynomial byte-order.
Note: Known decapped/dumped CICs are D411 and 3195A, and... somebody decapped/dumped a CIC without writing down its part number (probably=6113).

CIC Timings

The NES CICs are driven by a 4.000MHz CIC oscillator (located in the console, and divided by 4 in the NES CIC). The SNES CICs are driven by the 24.576MHz APU oscillator (located and divided by 8 in the console's audio circuit, and further divided by 3 in the SNES CIC).
Ie. internally, the CICs are clocked at 1.000MHz (NES) or 1.024MHz (SNES). All opcodes are executed within 1 clock cycles, except for the 2-byte long jumps (opcodes 78h-7Fh) which take 2 clock cycles. The "skip" opcodes are forcing the follwing opcode to be executed as a "nop" (ie. the skipped opcode still takes 1 clock cycle; or possibly 2 cycles when skipping long jump opcodes, in case the CPU supports skipping 2-byte opcodes at all).
After Reset gets released, the CICs execute the first opcode after a short delay (3195A: randomly 1.0 or 1.25 cycles, D413A: constantly 1.33 cycles) (whereas, portions of that delay may rely on a poorly falling edge of the incoming Reset signal).

CIC Ports

```
Name  Pin  Dir  Expl
P0.0  1    Out  Data Out    ;\SNES version occassionally swaps these
P0.1  2    In   Data In     ;/pins by software (ie. Pin1=In, Pin2=Out)
P0.2  3    In   Random Seed (0=Charged/Ready, 1=Charging/Busy)
P0.3  4    In   Lock/Key    (0=Cartridge/Key, 1=Console/Lock)
P1.0  9    Out  Reset SNES  (0=Reset Console, 1=No)
P1.1  10   Out  Reset Key   (0=No, 1=Reset Key)
P1.2  11   In   Unused, or Reset Speed A (in 3195A) ;\blink speed of reset
P1.3  12   In   Unused, or Reset Speed B (in 3195A) ;/signal (and Power LED)
P2.0  13   -    Unused
P2.1  14   -    Unused
```

```
  P2.2  15   -    Unused
  P2.3  -    -    Unused
  P3.0  -    RAM  Unused, or used as "noswap" flag (in SNES CIC)
  P3.1  -    -    Unused
  P3.2  -    -    Unused
  P3.3  -    -    Unused
```

P0.0-P2.2 are 11 external I/O lines (probably all bidirectional, above directions just indicates how they are normally used). P2.3-P3.3 are 5 internal bits (which seem to be useable as "RAM"). Pin numbers are for 16pin NES/SNES DIP chips (Pin numbers on 18pin SNES SMD chips are slightly rearranged). P4,P5,P6,P7,P8,P9,PA,PB,PC,PD,PF are unknown/unused (maybe 12x4 further bits, or mirrors of P0..P3).

CIC Stream Seeds

There are different seeds used for different regions. And, confusingly, there is a NES-CIC clone made Tengen, which uses different seeds than the real CIC (some of the differences automatically compensated when summing up values, eg. 8+8 gives same 4bit result as 0+0, other differences are manually adjusted by Tengen's program code).

Many of the reverse-engineered NES seeds found in the internet are based on the Tengen design (the USA-seeds extracted from the decapped Tengen chip, the EUR/ASIA/UK-seeds based on sampled Nintendo-CIC data-streams, and then converted to a Tengen-compatible seed format). To convert them to real CIC seeds:

```
  Nintendo[1..F] = Tengen[1..F] - (2,0,0,0,0,0,8,8,8,8,8,8,8,8,2)
```

There are other (working) variations possible, for example:

```
  Nintendo[1..F] = Tengen[1..F] - (2,0,0,0,0,A,E,8,8,8,8,8,8,8,2)
  (That, for Tengen-USA seeds. The Tengen-style-EUR/ASIA/UK seeds may differ)
```

Whereas, the random seed in TengenKEY[1] is meant to be "r+2" (so subtracting 2 restores "r").

CIC Stream Logs

There are some stream logs with filename "XXXX-N.b" where XXXX is the chip name, and N is the random seed, and bytes in the file are as so:

```
  Byte 000h, bit0-7 = 1st-8th bit on Pin 1 (DTA.OUT on NES)(DTA.OUT/IN on SNES)
  Byte 001h, bit0-7 = 1st-8th bit on Pin 2 (DTA.IN on NES) (DTA.IN/OUT on SNES)
  Byte 002h, bit0-7 = 9th-16th bit on Pin 1
  Byte 003h, bit0-7 = 9th-16th bit on Pin 2
  etc.
```

Caution: The "N" in the filename is taken as if the seed were transferred in order Bit 3,2,1,0 (actually it is Bit 3,0,1,2). Ie. file "3195-1.b" would refer to a NES-EUR-CIC with seed r=4. The signals in the files are sampled at 1MHz (ie. only each fourth 4MHz cycle).

The 6113 Chip

The 6113 chip was invented in 1987, and it replaced the 3193 chip in US/Canadian cartridges (while US/Canadian consoles kept using 3193 chips). When used in cartridges, the 6113 does usually "emulate" a 3193 chip. But, for whatever reason, it can do more:

```
  Console  Cartridge  Notes
  3193     3193       Works (the "old" way)        ;\used combinations
  3193     6113       Works (the "new" way)        ;/
  6113     6113       Works (special seed/timing)  ;\
  6113     3193       Doesn't work                 ; not used as far as known
  6113     ??         Might work (??=unknown chip) ;/
```

When used in consoles, the 6113 uses slightly different timings and seed values (and does request cartridges with 6113 chips to use the 6113-mode, too, rather than emulating the 3193).

One guess: Maybe Nintendo originally used different CICs for NTSC regions (like 3193/3194 for USA/Canada/SouthKorea), and later combined them to one region (if so, all NES consoles in Canada or SouthKorea should contain 3194/6113 chips, unlike US consoles which have 3193 chips).

3195A Signals (NES, Europe)

The I/O ports are HIGH (for Output "1"), or LOW-Z (for Output "0" or Input). Raising edges take circa 0.5us, falling edges take circa 3us.

```
  4MHz Clock Units     ..............................
  1MHz Clock Units       .   .   .   .   .   .   .   .
                          ___________                    ;\Console+Cartridge
  Data Should-be       __|           |________________  ;/should be 3us High
                          ___________                    ;\actually 2.5us High
  Data From Console    __.'           ''----.......____  ;/and 3us falling
                          ___________                    ;\
  Data From Cartridge  __.'           ''----.......____  ; either same as console
    or, delayed:           ___________                  ; or 0.25us later
  Data From Cartridge  ___.'           ''----.......___  ;/
```

After Power-up, the Cartridge CIC does randomly start with good timing, or with all signals delayed by 0.25us. In other words, the Cartridge CIC executes the first opcode 1.0us or 1.25us (four or five 4MHz cycles) after Reset gets released. However, for some reason, pushing the Reset Button doesn't alter the timing, the random-decision occurs only on Power-up.

D413A Signals (SNES, Europe)

The D413A signals are looking strange. First, the software switches signals High for 3us, but the actual signals are 3.33us High. Second, the signals on one pin are constantly jumping back'n'forth by 1.33us (in relation to the other pin).

```
  3.072MHz Clock Units ..............................
  1.024MHz Clock Units   .  .  .  .  .  .  .  .  .  .
                            ________                    ;\Console+Cartridge
  Data Should-be       ____|        |_____________     ;/should be 3us High
                            _________                   ;\actually 3.33us high
  Data From/To Console ____|         '--...________     ;/and 2us falling
                          __________                    ;\
  Data From/To Cart    ___|          '--...___________  ; 1.33us earlier
   or, delayed                      _________          ; or 1.33us later
```

```
  Data From/To Cart     _____________|          '--..___ ;/
```

The earlier/later effect occurs because the SNES CICs are occassionally reversing the data-direction of the pins. Ie. in practice, Data from Cartridge is constantly 1.33us LATER than from Console.
Software-wise, the D411 (and probably D413A) is programmed as if the Cartridge CIC would start "immediately", but in practice, it starts 1.33us (four 3.072MHz cycles) after releasing Reset (that offset seems to be constant, unlike as on the 3195A where it randomly changes between 1.0us and 1.25us).

Cartridge CIC Versions

NES CIC Versions

```
  3193,3193A        NES NTSC Cartridges and Consoles       ;\USA,Canada
  6113,6113A,6113B1 NES NTSC Cartridges (not consoles)     ;/(and Korea?)
  3194              Unknown/doesn't exist?
  3195,3193A        NES PAL Cartridges and Consoles "PAL-B";-Europe
  3196(A?)          NES PAL Cartridges and Consoles        ;-Hong Kong,Asia
  3197(A?)          NES PAL Cartridges and Consoles "PAL-A";-UK,Italy,Australia
  3198(A?)          FamicomBox CIC Cartridges and Consoles ;\
  3199(A?)          FamicomBox Coin Timer (not a CIC)      ; Japan
  N/A               Famicom Cartridges and Consoles        ;/
  RFC-CPU10 (?)     NES R.O.B. robot (no CIC, but maybe a 4bit Sharp CPU, too?)
```

SNES CIC Versions

```
  F411,F411A,F411B   SNES NTSC Cartridges-with-SMD-Chipset and Consoles
  D411,D411A,D411B   SNES NTSC Cartridges-with-DIP-Chipset
  F413,F413A,F413B   SNES PAL Cartridges-with-SMD-Chipset and Consoles
  D413,D413A,D413B   SNES PAL Cartridges-with-DIP-Chipset
  SA-1,S-DD1,MCC-BSC SNES Cartridges (coprocessors/mappers with on-chip CIC)
```

NES CIC Clones

```
  23C1033
  337002   ;Tengen's 16pin "Rabbit" CIC clone
  337006   ;Tengen's 40pin "RAMBO-1" mapper with built-in CIC clone
  4051
  7660
  KC5373B
  MX8018
  NINA
  Ciclone  ;homebrew multi-region CIC clone (based on Tengen design)
```

Aside from using cloned CICs, many unlicensed NES cartridges used a different approach: injecting "wrong" voltages to the console, and "stunning" its CIC.

SNES CIC Clones

```
  10198    - CIC clone
  noname   - CIC clone (black chip without any part number)
  ST10198S - NTSC CIC clone
  ST10198P - PAL CIC clone
  265111   - maybe also a CIC clone (used in Bung Game Doctor SF6)
  D1       - maybe also a CIC clone (used in Super UFO Pro8)
  74LS112  - reportedly also a CIC clone (with fake part number) (UFO Pro6)
  CIVIC 74LS13   16pin - CIC/D411 clone (used in a 8-in-1 pirate cart)
  CIVIC CT6911   16pin - CIC      clone (used in a 7-in-1 pirate cart)
  93C26          16pin - CIC      clone (used in a 8-in-1 pirate cart)
  D1             16pin - CIC? (used in Super VG pirate)
  STS9311A 52583 16pin - CIC clone (used in Donkey King Country 3 pirate)
  black blob     16pin - CIC/D411 clone (used in Sonic the Hedgehog pirate)
```

CIC Chip Year/Week Date Codes

```
  Name   YYWW-YYWW
  3193   8539-8642
  3193A  8547-8733 (in cartridges) (but should be in consoles for more years)
  3195   8627-8638
  3195A  8647-9512
  3197A  8647-9227
  6113   8734-8823
  6113A  8823-8933
  6113B1 8847-9344
```

Cartridge CIC Pinouts

XXX below "Cart.nn" pin-numbers are for SNES carts (not NES carts).

CIC Pinouts

F411/F413: 18pin SMD-chip (used in console and in some carts)
D411/D413: 16pin DIP-chip (used in most carts)

```
  SMD DIP Pin Dir Usage  In Console              In Cartridge
  1   1   P00 Out DTA0   Cart.55 CIC1            Cart.24 CIC0
```

```
 2   2   P01 In  DTA1   Cart.24 CIC0             Cart.55 CIC1
 3   3   P02 In  RANDOM Via capacitor to VCC     NC
 4   4   P03 In  MODE   VCC=Console (Lock)       GND=Cartridge (Key)
 5       NC  -   (NC)   NC                       NC
 6   5   CL2 -   (NC)   NC                       SMD:NC or DIP:GND
 7   6   CL1 In  CLK    3.072MHz (from APU)      Cart.56 CIC3 (3.072MHz)
 8   7   RES In  RESET  From Reset button        Cart.25 CIC2 (START)
 9   8   GND -   GND    Supply                   Supply
 10  9   P10 Out /RESET To PPU (and CPU/APU/etc) NC (or to ROM, eg. in SGB)
 11  10  P11 Out START  Cart.25 CIC2             NC
 12  11  P12 -   (NC)   NC                       NC (or SlotID in FamicomBox)
 13  12  P13 -   (NC)   NC                       NC (or SlotID in FamicomBox)
 14      NC  -   (NC)   NC                       NC
 15  13  P20 -   (NC)   NC                       NC
 16  14  P21 -   (NC)   NC                       NC (or SlotID in FamicomBox)
 17  15  P22 -   (NC)   NC                       NC (or SlotID in FamicomBox)
 18  16  VCC -   VCC    Supply                   Supply
```

P00=Out,P01=In are the initial directions (for the Random Seed transfer), later on the directions are randomly swapped (ie. P00=In,P01=Out or P00=Out,P01=In).
START: short HIGH pulse on power-up or when releasing reset button.
/RESET: in console: to PPU, and from there to CPU,APU,Cart,Expansion.

Cartridge CIC Tengen Clone

Tengen Registers

```
PC       8bit Program Counter (00h on Reset)
Acc      4bit Accumulator (aka A)
RamDta   4bit RAM Data-Read Register (aka X)
L        4bit RAM Address Register, LSB
H        1bit RAM Address Register, MSB (0 on Reset)
Cy       1bit Carry Flag
DATA_IN  1bit Input from master CIC
DATA_OUT 1bit Output to master CIC
```

Tengen Memory

```
Program ROM = 256x12 bit
Data RAM = 32x4 bit
```

Type 0 Opcodes - ADD/XNOR with optional Memory Load

```
11-10  Must be 00b for Type 0 Opcodes
9-6    ALU Source 1 and Destination (see list)
5      ALU Source 2 (0=Zero, 1=RamDta)
4      ALU Update Carry (0=No, 1=Yes)       ;<-- done AFTER ALU
3      --> RamDta=[H:L]                     ;<-- done BEFORE ALU
2      ALU Type     (0=ADD, 1=XNOR)
1-0    ALU Source 3 (0..3 = 0,Cy,1,1-Cy) (should be 0 for XNOR?)
```

Type 1 Opcodes - ADD with Immediate

```
11-10  Must be 01b for Type 1 Opcodes
9-6    ALU Source 1 and Destination (see list)
5      Unknown/Unused (should be 0)
4      ALU Update Carry (0=No, 1=Yes)       ;<-- done AFTER ALU
3-0    ALU Source 2 (Immediate 00h..0Fh)
```

Type 2 Opcodes - ADD with optional Special Actions

```
11-10  Must be 10b for Type 2 Opcodes
9-6    ALU Source 1 and Destination (see list, optionally H:0 instead H:L)
5      ALU Source 2 (0=Zero, 1=RamDta)
4      ALU Source 3 (0=Zero, 1=DATA_IN)
3      --> RamDta=[H:L] (or [H:0])          ;<-- done BEFORE ALU
2      --> Force address H:0 instead H:L ;applies to BOTH read/write [H:L]
1      --> DATA_OUT=Cy
0      --> H=Cy            ;<-- done AFTER reading/writing [H:L]
```

Type 3 Opcodes - Jumps

```
11-10  Must be 11b for Type 3 Opcodes
9      Jump Condition (0=Always, 1=If Cy=0)
8      Unknown/Unused (should be 0)
7-0    Jump Target    (PC=00h..FFh)
```

ALU Source 1 and Destination (Bit9-6 of Type 0-2 Opcodes)

```
00h Src=None, Dest=None        08h Src=L,   Dest=[H:L]=Acc
01h Src=None, Dest=None *      09h Src=Acc, Dest=[H:L]=Acc
02h Src=Zero, Dest=Acc         0Ah Src=L,   Dest=Acc
03h Src=Acc,  Dest=Acc *       0Bh Src=Acc, Dest=Acc
04h Src=Zero, Dest=[H:L]       0Ch Src=L,   Dest=[H:L]
05h Src=Acc,  Dest=[H:L] *     0Dh Src=Acc, Dest=[H:L]
```

```
  06h Src=Zero, Dest=L           0Eh Src=L,   Dest=L
  07h Src=Acc,  Dest=L *         0Fh Src=Acc, Dest=L
```
Note: 01h,03h,05h,07h are (occassionally used) dupes of 00h,0Bh,0Dh,0Fh.
Mind that Type 2 Opcodes can force RAM address H:0 instead of H:L.

Update Carry (when Bit4=1 in Type 0 or 1 opcodes)
```
  when ALU Dest = None  ---> Cy = bit0 or (src2)
  when ALU Type = ADD   ---> Cy = carry-out of addition
  when ALU Type = XNOR  ---> Cy = bit3 of (src1 OR src2) (??)
```

Timings
All opcodes take 1 clock cycle (clock is 1MHz).

Tengen ROM-image
There is a "ROM-image" called "tengen2rom.txt" containing "0" and "1" bits in ASCII format (mixed with some english text), the "binary" part contains the ROM as seen under a microscope:
There are 2x6 blocks (mapped to the 12bit databus):
```
  D11 D5
  D10 D4
  D9  D3
  D8  D2
  D7  D1
  D6  D0
```
Each block contains 8x32 bits (for 8bit address bus):
```
  Line   1..32 --> address X+(1Fh..00h)
  Column 1..8  --> address (00h,20h,40h,60h,80h,A0h,C0h,E0h)+Y for D0..D5
  Column 1..8  --> address (E0h,C0h,A0h,80h,60h,40h,20h,00h)+Y for D6..D11
```
Recommended format for a binary ROM-image would be padding the 12bit opcodes to 16bit (bit15-12 zero), and then storing it as a 512-byte file (with the 16bit values in little-endian format).
Warning: The Tengen CIC stream initialization isn't done by "L=imm, [HL]=imm" pairs, but rather by cryptic stuff like "L=L+X+cy, [HL]=A XNOR X".

Cartridge Cheat Devices

Game Genie (19xx)
The Game Genie is an adapter to be connected between the console and game cartridge, it includes a BIOS ROM which prompts the user to enter 6-letter or 8-letter cheat-codes, and then starts the actual game.
The adapter compares the CPU address bus (PRG ROM area 8000h-FFFFh), and optionally also the CPU data bus (reduces the risk to mess-up values in other banks in cartridges with Memory Mappers), if the comparision matches, then the value on data bus will be replaced.
Game Genie Code Format
The letters are translated into 4bit Hex-digits:

Hex	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Letter	A	P	Z	L	G	I	T	Y	E	O	X	U	K	S	V	N

Address/Data/Compare bits A14-A0, D7-D0, C7-C0 are scrambled as such:

Char	Bit3	Bit2	Bit1	Bit0	Char	Bit3	Bit2	Bit1	Bit0
1st	D7	D2	D1	D0	2nd	A7	D6	D5	D4
3rd	LEN	A6	A5	A4	4th	A3	A14	A13	A12
5th	A11	A2	A1	A0	6th	CD3	A10	A9	A8
7th	C7	C2	C1	C0	8th	D3	C6	C5	C4

6-Letter code: LEN=0, CD3 used as D3, acts as "[A]=D"
8-letter code: LEN=1, CD3 used as C3, acts as "If [A]=C then [A]=D"
Example: Code "SXIOPO" changes [91D9h]=ADh (Infinite lives in smb1).

Pro Action Replay (PAR) (Datel) (1992)
```
  XXX...
```

Pro Action Rocky (by Cyber Gadget) (2003) (Japan/Famicom)
This thing seems to be the first and only cheat device released for the japanese Famicom (although it came out much later than the NES cheat devices).
The strange name is apparently inspired on Elvis Presley or Sylvester Stallone & on Datel's Pro Action Replay. The hardware uses ROM patches (and thus works more like Game Genie than Pro Action Replay). Codes are 8-digit hex values in following format.
```
  DDCCAAAA  ;Data,Compare,Address, ie. "if [AAAA]=CC then [AAAA]=DD"
```
And, of course, there's some annoying encryption...
Pro Action Rocky Encode (raw to code):
```
  raw=raw AND FFFF7FFFh    ;strip unused address MSB from the DDCCAAAAh value
  for i=0 to 31
    code=code SHR 1
    if raw AND (1 SHL xlat[i]) then code=code XOR B8309722h
  next i
  code=code XOR FCBDD275h
```
Pro Action Rocky Decode (code to raw):
```
  code=code XOR FCBDD275h, raw=00000000h
  for i=31 to 0
    if code AND 80000000h then raw=raw+(1 SHL xlat[i]), code=code XOR B8309722h
```

```
    code=code SHL 1
  next i
  raw=raw OR 00008000h    ;add unused/fixed address MSB to the DDCCAAAAh value
```

Pro Action Rocky Bit Translation (xlat[0..31]):

```
  15, 3,13,14, 1, 6, 9, 5, 0,12, 7, 2, 8,10,11, 4
  19,21,23,22,20,17,16,18,29,31,24,26,25,30,27,28
```

Cartridge Pin-Outs

60-Pin Famicom Cartridge connector

```
  1     GND            19-25 PPU A6-A0        45    EXP SND_IN
  2-13  CPU A11-A0     26-29 PPU D0-D3        45,46 EXP SND_OUT
  14    CPU R/W        30-31 +5VDC            47    PPU /WR
  15    CPU /IRQ       32    PHI2 CLK         48    PPU NT /CS
  16    GND            33,35 CPU A12-A14      49    PPU NT /A13
  17    PPU /RD        36-43 CPU D7-D0        49-56 PPU A7-A12,A13
  18    PPU NT A10     44    CPU /PRG         57-60 PPU D7-D4
```

The Famicom connector uses standard 2.54mm pitch.

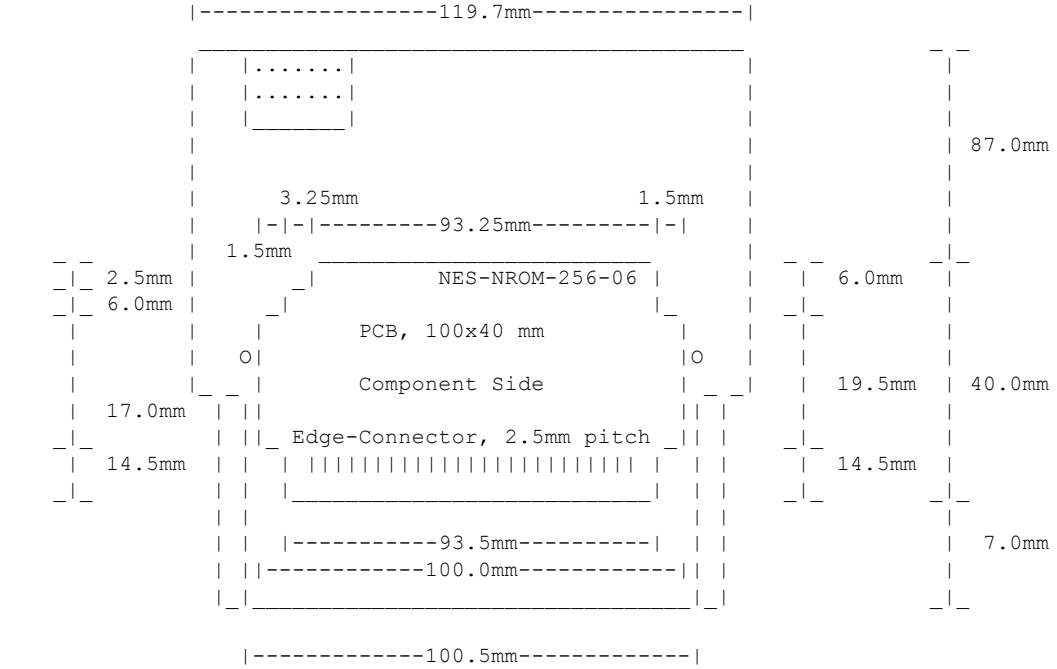
72-Pin NES Cart connector

```
  Pin   Dir Use Expl.                    _______________________________
  1     Out VEE GND                 1|                               |36
  2-13  Out CPU A11-A0             37|_______________________________|72
  14    Out CPU R/W
  15    I/O CPU /IRQ
  16-20 I/O EXP Expansion Port Pins 42-38 (not used by the console itself)
  21    Out PPU /R
  22    In  PPU VA10 (A10 of internal 2K VRAM, ie. select BLK0 or BLK1)
  23-29 Out PPU A6-A0
  30-33 I/O PPU D0-D3
  34-35 I/O CIC S0-S1 (cicurity/lockup chip protocol signals)
  36    Out VCC +5VDC
  37    Out CPU 21.47727MHz (NTSC), 26.601712MHz (PAL)
  38    Out CPU PHI2
  39-41 Out CPU A12-A14
  42-49 I/O CPU D7-D0
  50    Out CPU /PRG (PRG-ROM access, logical NAND of PHI2 and CPU A15)
  51-55 I/O EXP Expansion Port Pins 06-10 (not used by the console itself)
  56    Out PPU /W
  57    In  PPU /VCS (internal 2K VRAM Chip-Select)
  58    Out PPU /A13 (inverted A13, wired to /VCS when used as name table)
  59-65 Out PPU A7-A9,A11,A10,A12-A13
  66-69 I/O PPU D7-D4
  70    I/O CIC S2 (cicurity/lockup chip protocol signals)
  71    Out CIC 4Mhz (cicurity/lockout chip clock line)
  72    Out VEE GND
```

Caution: The NES edge connector uses a nonstandard 2.5mm pitch (not 2.54mm). And, the PCBs (including copper layers) are only 1.2mm thick (not 1.5mm). There's no tolerance there (2.54mm pitch would produce short-cuts, and inserting or removing 1.5mm PCBs works only with extreme force).

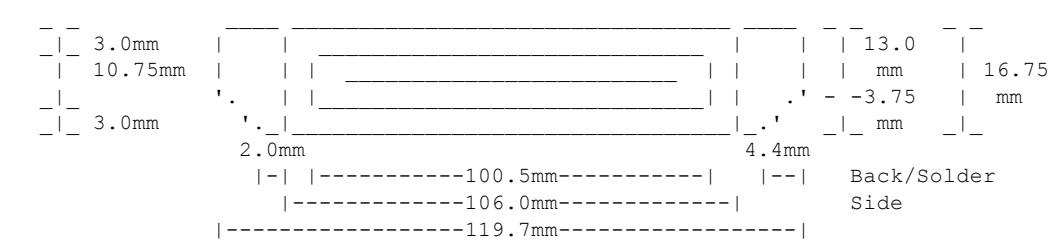
Cartridge Shell Dimensions

NES Cart-Shell with NES-NROM PCB Dimensions

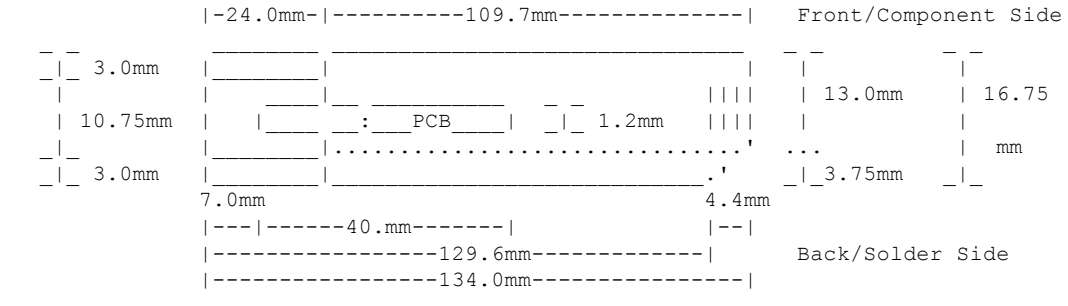


```
              |---------------106.0mm---------------|
```

NES Cart-Shell, Connector Side



NES Cart-Shell, Side View



Caution
Above values aren't 100% correct.

Mapper 0: NROM - No Mapper (or unknown mapper)

NROM (No Mapper)
Used in games with (max) 32K ROM + 8K VROM, ie. games that do not require any bank-switching hardware.
Name Table can be hardwired either to Horizontal or Vertical Mirroring.

Unknown Mappers
ROM-Images with unknown mapping hardware are also often assigned as "Mapper 0". Namely, if the ROM is bigger than 32K+8K, then it's obviously some unknown stuff rather than NROM.

Mapper 1: MMC1 - PRG/32K/16K, VROM/8K/4K, NT

This mapper is used on numerous U.S. and Japanese games, including Legend of Zelda, Metroid, Rad Racer, Mega Man 2, and many others.

```
  8000h-FFFFh
    Bit 0  Serial data loaded to 5bit shift register (LSB=1st write)
    Bit 7  Clear 5bit shift register (1=Reset, next write will be "1st write")
```

On fifth write, data in shift register is copied to Register 0..3 (depending on upper address bits), and the shift register is automatically cleared.

```
  8000h-9FFFh  Register 0 - Configuration Register
    Bit0-1 Name Table Mirroring
      0    Single-Screen BLK0
      1    Single-Screen BLK1
      2    Two-Screen Vertical Mirroring
      3    Two-Screen Horizontal Mirroring
    Bit2-3 PRG-Switching Mode (usually 3)
      0,1  Switchable 32K Area at 8000h-FFFFh (via Register 3)
      2    Switchable 16K Area at C000h-FFFFh (via Register 3)
           And Fixed  16K Area at 8000h-BFFFh (always 1st 16K)
      3    Switchable 16K Area at 8000h-BFFFh (via Register 3)
           And Fixed  16K Area at C000h-FFFFh (always last 16K)
    Bit4   VROM Switching Size (for carts with VROM)
      0    Swap 8K of VROM at PPU 0000h
      1    Swap 4K of VROM at PPU 0000h and 1000h
  A000h-BFFFh  Register 1
    Bit4-0 Select 4K or 8K VROM bank at 0000h (4K and 8K Mode, see Reg0/Bit4)
  C000h-DFFFh  Register 2
    Bit4-0 Select 4K VROM bank at 1000h (used in 4K Mode only, see Reg0/Bit4)
  E000h-FFFFh  Register 3
    Bit3-0 Select 16K or 2x16K ROM bank (see Reg0/Bit3-2)
    Bit4   RAM Disable (newer MMC1 revisions only)
```

Initially 1st and last 16K are mapped to 8000h and C000h.
In 32K PRG and 8K VROM mode, bank numbers specified in steps of two.

Register 3 is restricted to sixteen 16K banks, cartridges with more than 256K PRG ROM use Bit4 of Register 0-2 to expand the available memory area:

```
 Register 0, Bit 4
  <1024K carts>
   0 = Ignore 256K selection register 1
   1 = Acknowledge 256K selection register 1
 Register 1, Bit4 - 256K ROM Selection Register 0
  <512K carts>
   0 = Swap banks from first 256K of PRG
   1 = Swap banks from second 256K of PRG
  <1024K carts with bit 4 of register 0 off>
   0 = Swap banks from first 256K of PRG
   1 = Swap banks from third 256K of PRG
  <1024K carts with bit 4 of register 0 on>
   Low bit of 256K PRG bank selection
 Register 2, Bit4 - 256K ROM Selection Register 1
  <1024K carts with bit 4 of register 0 off>
   Store but ignore this bit (base 256K selection on 256K selection Reg 0)
  <1024K carts with bit 4 of register 0 on>
   High bit of 256K PRG bank selection
```

Reportedly some MMC1 carts have 16K SRAM, of which only 8K are battery backed, no idea how/where the additionally 8K are accessed, and no idea which 8K are battery backed and which are not (?).

Mapper 2: UNROM - PRG/16K

This mapper is used on many older U.S. and Japanese games, such as Castlevania, Mega Man, Ghosts & Goblins, and Amagon.

```
  8000h-FFFFh  Select 16K ROM bank at 8000h-BFFFh (initially 1st bank)
  N/A          Fixed  16K ROM at C000h-FFFFh (always last bank)
```

All carts using it have 8K of VRAM at PPU 0000h. Most carts with this mapper are 128K. A few, mostly Japanese carts, such as Final Fantasy 2 and Dragon Quest 3, are 256K.
Bus-conflicts. Uses a 74LS161 chip (connection: /PRG-CLK, R/W-/LOAD), and its outputs are each ORed with A14 by a 74LS32 chip.
Board NES-UN-ROM-05 and Konami 531320 (both using only 3bits / 8banks)

Mapper 3: CNROM - VROM/8K

This mapper is used on many older U.S. and Japanese games, such as Solomon's Key, Gradius, Cybernoid, and Hudson's Adventure Island.

```
  8000h-FFFFh
   Bit 0-1  Select 8K VROM bank at PPU 0000h (initially 1st bank)
   Bit 4-5  Security Diodes (some crude copy protection)
```

Bus-conflicts. Contains a 74LS161 counter chip mis-used as 4bit latch (connection: /PRG-CLK, R/W-/LOAD).
Cybernoid supports both above Port 8000h-FFFFh, and alternately Port 6000h.

Security Diodes
Some CNROM boards can be fitted with "Security Diodes", wired as so:

```
  Mapper.Bit5 ---- D1 ---- PPU.A10
  Mapper.Bit4 ---- D2 ---- PPU.A12
```

The diodes can be pointed in this or that direction (to or from PPU), the game program code (or cartridge dumping device) must configure the two mapper outputs according to the diode directions (so that no current will flow through the diodes) (otherwise the diodes would cause shortcuts on the PPU address lines, causing the VROM to output garbage).
There is also another board with Security Diodes and additional VROM-disable feature:
Mapper 185: VROM-disable

Mapper 4: MMC3 - PRG/8K, VROM/2K/1K, NT, SRAM, IRQ

A great majority of newer NES games (early 90's) use this mapper, both U.S. and Japanese. Among the better-known MMC3 titles are Super Mario Bros. 2 and 3, Mega Man 3, 4, 5, and 6, and Crystalis.

```
  8000h  Index/Control (5bit)
         Bit7   CHR Address Select  (0=Normal, 1=Address Areas XOR 1000h)
         Bit6   PRG Register 6 Area (0=8000h-9FFFh, 1=C000h-DFFFh)
         Bit2-0 Command Number
           0 - Select 2x1K VROM at PPU 0000h-07FFh (or 1000h-17FFh, if Bit7=1)
           1 - Select 2x1K VROM at PPU 0800h-0FFFh (or 1800h-1FFFh, if Bit7=1)
           2 - Select 1K VROM at PPU 1000h-13FFh   (or 0000h-03FFh, if Bit7=1)
           3 - Select 1K VROM at PPU 1400h-17FFh   (or 0400h-07FFh, if Bit7=1)
           4 - Select 1K VROM at PPU 1800h-1BFFh   (or 0800h-0BFFh, if Bit7=1)
           5 - Select 1K VROM at PPU 1C00h-1FFFh   (or 0C00h-0FFFh, if Bit7=1)
           6 - Select 8K ROM at 8000h-9FFFh (or C000h-DFFFh, if Bit6=1)
           7 - Select 8K ROM at A000h-BFFFh
         N/A - Fixed  8K ROM at C000h-DFFFh (or 8000h-9FFFh, if Bit6=1)
         N/A - Fixed  8K ROM at E000h-FFFFh (always last 8K bank)
  8001h  Data Register    (Indexed via Port 8000h)
  A000h  Mirroring Select (Bit0: 0=Vertical, 1=Horizontal Mirroring)
```

```
  A001h  Save RAM Control for RAM at 6000h-7FFFh:
         Bit7: SRAM Chip Enable   (0=Disable both read/write, 1=Enable)
         Bit6: SRAM Write Protect (0=Read/Write-able, 1=Read-only)
  C000h  IRQ Reload Value (loaded into IRQ Counter on Counter underflow)
  C001h  IRQ Force Reload (writing any value will set the Counter to zero,
         so that underflow/reload will occur at next rising edge of A13)
  E000h  IRQ Disable/Acknowledge (write any value) ;doesn't change/stop counter
  E001h  IRQ Enable (write any value) ;doesn't change/reload counter
```

The fixed PRG banks are always the LAST two 8K banks in the cart.
On carts with VROM, the first 8K of VROM is swapped into PPU $0000 on reset.
On carts without VROM, as always, there is 8K of VRAM at PPU $0000.

The IRQ counter is decremented each scanline, based on PPU address line A13 which toggles between Pattern Tables (LOW) and Name Tables (HIGH) 42 times per scanline. The counter is paused during VBlank, which allows to use the same settings for PAL and NTSC timings. Note that the counter gets clocked when toggling A13 a bunch of times during VBlank by software via Port 2006h.

MMC3 Boards
NES-TEROM - Max. 64K PRG, 64K CHR, optionally hardwired mirroring
NES-TQROM - Max. 128K PRG, 64K CHR, 8K CHR-RAM (see Mapper 119)
NES-TVROM - Max. 128K PRG, 64K CHR, 4-screen mirroring (Rad Racer II)
NES-TFROM - Max. 512K PRG, 64K CHR, optionally hardwired mirroring
NES-TGROM - Max. 512K PRG, 8K CHR-RAM
NES-TSROM - Max. 512K PRG, 256K CHR, 8K WRAM, Non-battery-backed.
NES-TKROM - Max. 512K PRG, 256K CHR, 8K WRAM, Battery-backed.
NES-TLROM - Max. 512K PRG, 256K CHR
NES-TLSROM - Max. ?K PRG, 128K CHR, NT-bank-select (see Mapper 118)
NES-TR1ROM - Max. 512K PRG, 64K CHR, 4-screen mirroring (Gauntlet)

MMC3 Variants
Mapper 118: MMC3 TLSROM - PRG/8K, VROM/2K/1K, Banked-NT, SRAM, IRQ
Mapper 119: MMC3 TQROM - PRG/8K, VROM/VRAM/2K/1K, NT, SRAM, IRQ

MMC6 (same as MMC3 but with on-chip PRG-RAM) (this is also named "Mapper 4")
NES-HKROM - used only by StarTropics and StarTropics 2

Multicarts with MMC3 and additional Game-Select Ports
Mapper 44: 7-in-1 MMC3 Port A001h
Mapper 45: X-in-1 MMC3 Port 6000hx4
Mapper 47: 2-in-1 MMC3 Port 6000h
Mapper 49: 4-in-1 MMC3 Port 6xxxh
Mapper 52: 7-in-1 MMC3 Port 6800h with SRAM

Tengen MIMIC-1 (MMC3 Variant)
Tengen MIMIC-1 uses only Port 8000h (lower 3bit only) and Port 8001h.
The ROM-images are typically declared as "Mapper 4" even though it supports only a subset of the MMC3 functions. Tengen's Gauntlet additionally contains 2K SRAM to expand Name Table memory to 4K.

IRQ Notes form Kevin Horton...
[Observe that the A12-stuff conflicts with above 42-step-A13-prescale] [?]
[According below, IRQ when counter=00h, but reload when counter<00h] [?]

```
 * The IRQ counter WILL NOT STOP.  It will continue to decrement and reload
   as long as A12 on the PPU bus toggles.
 * Whenever the IRQ counter changes from a non-zero value to 00h, the IRQ
   flag will be set if it is enabled.
 * The exact number of scanlines before the interrupt fires is (N+1),
   where N = the IRQ reload value.  2 to 256 scanlines are supported.
 * Writing 00h to C000h will result in a SINGLE interrupt being generated
   on the next rising edge of A12.  No more interrupts will be generated
   until C000h is changed to a non-zero value.  The counter is still being
   reloaded, however, because writing a non-zero value to C000h results in
   it firing an interrupt after the new count expires.

 * The IRQ counter WILL NOT DECREMENT AT ALL unless bit 3 OR bit 4 of 2000h
   on the PPU are set! If both of these bits are clear, the IRQ counter will
   not count no way no how!!!
   If both are set, the counter decrements twice per frame on my MMC3, but
   it may act erratically on your MMC3.  Don't count on this effect occuring.

 * For some reason, yet to be determined, if both bits 3 and 4 of PPU
   register 2000h are clear, the IRQ counter will not decrement, even if
   the PPU address is manually manipulated (with 2001h set to 00h to disable
   rendering) through 2006h. If either or both bits are set, the counter will
   decrement properly if the PPU address is manually manipulated.
```

```
Various notes and effects of the IRQ counter reloading stuff:
The IRQ counter reloading has some minor consequences that should be made
clear.  Some games like Megaman 6 and Pinbot use this so emulating properly
is important for these games to work.

MM6 will write to E000h to clear the flag, E001h to re-enable interrupts,
and finally it will write to C000h to set up a new time period.  This is
legal SO LONG AS C000h is written to before the next scanline (remember:
the counter is reloaded on the NEXT rising edge of A12 after the counter
reaches 00h, and thus fires).

Kevin did a little test to determine the effect of C001h on the IRQ counter
like so:
First, C000h had 02h written to it. Toggling A12 3 times resulted in an IRQ
being generated. The IRQ was cleared, then A12 was toggled 2 more times.
Next, 03h was written to C000h, C001h was written to, and finally 04h was
written to C000h. A12 was toggled again and it took *5* counts to flag an
interrupt, proving that the value of C000h is only checked at the time of
reloading on the rising edge of A12. Kevin performed other tests to
corroborate  this, and they all passed (i.e. checking to see if the reload
happened on  the FALLING edge of A12, etc.)
```

Mapper 5: MMC5 - BANKING, IRQ, SOUND, VIDEO, MULTIPLY, etc.

Used by Gun Sight (Laser Invasion), Uchuu Keibitai SDF, Bandit Kings (Suikoden), Castlevania 3, Nobunaga Sengoku (Nobunaga's Ambition 2), Nobunaga Bushou, Shin 4 Nin Uchi Mahjong, Ishin no Arashi, L'Empereur, Ganbare Goemon Gaiden (bugged hack?), Romance of the Three Kingdoms 2 (Sangokushi 2), Gemfire (Royal Blood), Uncharted Waters (Daikoukai Jidai), Aoki Ookami, Just Breed, Metal Slader Glory.

Mapper 5: MMC5 - I/O Map

Summary of all MMC5 Registers

```
  5000h        Sound Channel 1 Pulse Control  ;\
  5002h        Sound Channel 1 Frequency LSB  ;
  5003h        Sound Channel 1 Frequency MSB  ;/
  5004h        Sound Channel 2 Pulse Control  ;\
  5006h        Sound Channel 2 Frequency LSB  ;
  5007h        Sound Channel 2 Frequency MSB  ;/
  5010h        Sound Channel 3 PCM Control/Status (R/W)
  5011h        Sound Channel 3 PCM Data
  5015h        Sound Channel 1 and 2 Enable/Status (R/W)
  5100h        PRG Bank Size (Mode for Port 5114h-5117h)
  5101h        CHR Bank Size
  5102h        RAM Write Protect Key 1
  5103h        RAM Write Protect Key 2
  5104h        EXRAM Mode Setting
  5105h        Name Table Select
  5106h        Name Table Fill-Mode Tile Number
  5107h        Name Table Fill-Mode Palette Number
  5113h-5117h  PRG Bank Selection Registers
  5120h-5127h  CHR Bank Selection for Sprites and for CPU Access
  5128h-512Bh  CHR Bank Selection for Background
  5130h        CHR Bank MSBs
  5200h        Horizontal Split Control
  5201h        Horizontal Split Scroll Position
  5202h        Horizontal Split CHR Bank Selection
  5203h        Vertical IRQ Counter
  5204h        Vertical IRQ Control/Status (R/W)
  5205h        Multiply Unit input/output (R/W)
  5206h        Multiply Unit input/output (R/W)
  5800h        Unknown... (Just Breed writes 0xh to this address)
  5C00h-5FFFh  EXRAM (1K) (R/W)
```

Ports 5102h-5103h, 5105h-5107h, 5200h-5203h are Write Only.
Ports 5204h-5206h are Read/Write. Other Ports unknown.

Mapper 5: MMC5 - CPU Memory Control

5100h - PRG Bank Size Control (Mode for Port 5114h-5117h)

```
Bit7-2  Not used
Bit1-0  PRG Bank Size (0=32K, 1=16K, 2=Mixed, 3=8K)
```

5102h-5103h - RAM Write Protect Keys

```
5102h  RAM Write Protect Key 0 (Lower 2bit must be 02h for write-enable)
5103h  RAM Write Protect Key 1 (Lower 2bit must be 01h for write-enable)
```

RAM is always read-able, but is write-able only with above settings.

5113h-5117h - PRG Bank Selection Registers

```
Port   Type     Mode3/8K     Mode2/Mixed    Mode1/16K      Mode0/32K
5113h  RAM      8K at 6000h  8K at 6000h    8K at 6000h    8K at 6000h
5114h  ROM/RAM  8K at 8000h, N/A          , N/A          , N/A
5115h  ROM/RAM  8K at A000h, 2x8K at 8000h, 2x8K at 8000h, N/A
5116h  ROM/RAM  8K at C000h, 8K   at C000h, N/A          , N/A
5117h  ROM      8K at E000h, 8K   at E000h, 2x8K at C000h, 4x8K at 8000h
```

Lower one or two bits of 2x8K or 4x8K bank numbers are ignored.

RAM bank selection via Port 5113h-5116h (not 5117h):

```
Bit7   ROM/RAM Mode (0=RAM, 1=ROM) (Port 5114h-5116h only, not 5113h,5117h)
Bit6-3 Not used
Bit2   RAM Chip Select (0=1st chip, 1=2nd chip, or open bus if single chip)
Bit1-0 Select 8K RAM Bank in currently selected RAM chip (32K chips only)
```

Existing RAM configurations are: 8K (single 8K chip), 16K (two 8K chips),
and 32K (single 32K chip).

On reset, 8K mode is activated, and all ROM banks are set to the LAST 8K bank in the cartridge.

Mapper 5: MMC5 - Video Name Table

5105h - Name Table Select

```
Bit1-0 Select NT0 VRAM at 2000h-23FFh (0=BLK0, 1=BLK1, 2=EXRAM, 3=FILLMODE)
Bit3-2 Select NT1 VRAM at 2400h-27FFh (0=BLK0, 1=BLK1, 2=EXRAM, 3=FILLMODE)
Bit5-4 Select NT2 VRAM at 2800h-2BFFh (0=BLK0, 1=BLK1, 2=EXRAM, 3=FILLMODE)
Bit7-6 Select NT3 VRAM at 2C00h-2FFFh (0=BLK0, 1=BLK1, 2=EXRAM, 3=FILLMODE)
```

If it isn't used for other purpose, then EXRAM can be used as 3rd Name-table.

5106h - Name Table Fill-Mode Tile Number (Bit7-0)
5107h - Name Table Fill-Mode Palette Number (only Bit1-0 used)
In FILLMODE, the entire Name-table is filled by Port 5106h/5107h settings.

Mapper 5: MMC5 - Video Pattern Table

5101h - CHR Page Size

```
Bit7-6 Not used
Bit1-0 CHR Bank Size (0=8K, 1=4K, 2=2K, 3=1K)
```

Bank selection registers below are 8bit, so 1K mode can address only 256K VROM, 8K mode could address up to 2MB VROM.

5120h-5127h - CHR Bank Selection for Sprites and for CPU Access

```
Port   Mode3/1K     Mode2/2K     Mode1/4K     Mode1/8K
5120h  1K at 0000h  N/A          N/A          N/A
5121h  1K at 0400h  2K at 0000h  N/A          N/A
5122h  1K at 0800h  N/A          N/A          N/A
5123h  1K at 0C00h  2K at 0800h  4K at 0000h  N/A
5124h  1K at 1000h  N/A          N/A          N/A
5125h  1K at 1400h  2K at 1000h  N/A          N/A
5126h  1K at 1800h  N/A          N/A          N/A
5127h  1K at 1C00h  2K at 1800h  4K at 1000h  8K at 0000h
```

Used for Sprite Tiles (not for Background Tiles), and also used for CPU VRAM Access via Port 2006h/2007h.

5128h-512Bh - CHR Bank Selection for Background

```
5128h  1K at X000h  N/A          N/A          N/A
5129h  1K at X400h  2K at X000h  N/A          N/A
512Ah  1K at X800h  N/A          N/A          N/A
512Bh  1K at XC00h  2K at X800h  4K at X000h  8K at 0000h
```

Used for Background Tiles, the "XN00h" addresses in 1K,2K,4K are shared for both Pattern Tables at 0N00h and 1N00h, ie. BG Pattern Table selection in Port 2000h/Bit4 doesn't matter (except in 8K mode).

5130h - CHR Bank MSBs

Just Breed, Gun Sight, and Uchuu Keibitai SDF write 00h to this address.

```
Bit7-6 Not used
Bit1-0 Upper 2bit for 8bit Port 5120h-512Bh, and for EXRAM 6bit Tile Banks
```

Writing to Port 5120h-512Bh copies the written 8bit value plus the 2bit port 5130h value to internal 10bit registers (subsequent writes to 5130h do not affect those memorized 10bit bank numbers).

Other Background CHR Bank Selection Modes (which do not use 5128h-512Bh)

In Horizontal Split Mode, left or right BG Tiles use 4K CHR bank in Port 5202h.
In ExGrafix Mode, 4K CHR banks are specified for each single BG Tile in EXRAM.

Mapper 5: MMC5 - Video Split and IRQ and Multiply Unit

MMC5 allows to split the screen horizontally and vertically.
Horizontal Split is handled by hardware (automatically mid-scanline).
Vertical Split is to be handled by software (upon IRQ during HBlank).

5200h - Horizontal Split Control

```
Bit7   For the E function (0=Don't use, 1=Use)
Bit6   Boundary's side is for using Split Mode extension of graphics
         (0=Left side, 1=Right side)
Bit5   Not used
Bit4-0 Left boundary is designated with the char. # to count places
```

Used by Uchuu Keibitai SDF, most or all other games don't use H-Split.

```
Examples for 5200h Settings:
00h (not?) used yet
82h Used for SplitMode GFX extension from left 1-2 character
C2h Used for SplitMode GFX extension from the right side 3 chars.
C0h Used for SplitMode GFX extension on the whole screen
D0h Used for SplitMode GFX extension on the right side of the screen
90h Used for SplitMode GFX extension on the left side of the screen
```

5201h - Horizontal Split Scroll Position

"$2005 determines the vertical movement; it can also delay ext. gfx's vert. movement if necessary. It's written 2 times in bulk in the same way as it would slip off a grade in $2005."

5202h - Horizontal Split CHR Bank Selection

```
Bit7-6 Not used
Bit5-0 Select 4K VROM at both 0000h-0FFFh and 1000h-1FFFh
```

Presumably used for BG Tiles in the Horizontal Split area, instead of the normal BG-CHR Bank Selection via 5128h-512Bh.

5203h - Vertical IRQ Counter

MMC3-style, decremented each scanline, paused during VBlank.
A setting of 00h seems to disable the counter (or, maybe sets it to 256 lines).

5204h - Vertical IRQ Control/Status (R/W)

```
Bit7/Write IRQ Enable (0=Disable, 1=Enable; forward IRQ Flag to CPU)
Bit7/Read  IRQ Flag   (0=No, 1=Interrupt Request; gets set even if disabled)
Bit6/Read  In Frame Rendering Flag (0=Vblank, 1=Rendering/Non-Vblank)
```

The IRQ flag is cleared/acknowledged when reading 5204h, and also automatically during Vblank.

5205h - Unsigned Multiply unit Factor A (W) / Result LSB (R)
5206h - Unsigned Multiply unit Factor B (W) / Result MSB (R)

Multiplication is unsigned. There is no noticable delay (the 16bit result can be read back right after writing).

Mapper 5: MMC5 - Video EXRAM

5C00h-5FFFh - EXRAM

Built-in 1K RAM, can be used in different modes, as VRAM or as WRAM.

5104h - EXRAM Mode Setting

```
Bit7-6 Not used
Bit1-0 Select EXRAM Mode
  0  VRAM Extra Name Table (via Port 5105h)
  1  VRAM ExGrafix Color Expansion (see below)
  2  General purpose WRAM (read/write)
  3  General purpose WRAM (write protected)
```

In VRAM modes, EXRAM can be probably accessed during VBlank only (just as normal VRAM). In WRAM modes, EXRAM can be probably accessed at any time, for use as general purpose Work RAM, instead of (or additionally to) normal SRAM.

ExGrafix Mode (used by most MMC5 titles, except Castlevania 3)
5C00h-5FBFh - Tile Number banks and Palettes for 32x30 Tiles
```
Bit7-6  Palette Number for each Tile
Bit5-0  4K Bank Number for each Tile (upper 2bit in port 5130h)
```
The 6bit+2bit Bank Numbers expand each of the 8bit Tile Numbers in Name Table entries 000h-3BFh to 16bit Tile Numbers (max 1MB VROM addressable). The Palette Numbers allow to specify different palettes for each single Tile, instead of the normal Name Table palettes which share one palette entry for each 4 Tiles.
Name Table entries 3C0h-3FFh and EXRAM 5FC0h-5FFFh are not used in this mode.
Also BG-CHR Bank Selection Ports 5128h-512Bh are not used.

Mapper 5: MMC5 - Sound Control

MMC5 Sound, Japanese 60pin Famicom carts only, not NES 72pin carts.
The MMC5 sound registers (port 50xxh) are working very similar to NES APU registers (port 40xxh), but without sweep support, without triangle/noise channels, without real DMA, and with a bunch of other differences.

5000h - MMC5 Sound Volume/Decay Channel 1 (Rectangle)
5004h - MMC5 Sound Volume/Decay Channel 2 (Rectangle)
```
0-3    Volume / Envelope decay rate
        When Bit4=1: Volume (0=Silent/None..F=Loud/Max)
        When Bit4=0: Envelope decay rate, 240Hz/(N+1) for MMC5
4      Envelope decay disable (0=Envelope/Decay, 1=Fixed Volume)
5      Length counter clock disable / Envelope decay looping enable
        When Bit4=1: length counter clock disable
        When Bit4=0: envelope decay looping enable
         0: Disable Looping, stay at 0 on end of decay [ \_____ ]
         1: Enable Looping, restart decay at F         [ \\\\\\ ]
         (Does this still affect Length counter clock disable ?)
6-7    Duty cycle type (MMC5 duty is inverse of NES APU duty)
         0  [__--------------] 87.5%   Whereas,
         1  [____------------] 75.0%   [_] = LOW  (zero) (0)
         2  [________--------] 50.0%   [-] = HIGH (volume/decay) (0..F)
         3  [______________----] 25.0%
```
The Duty Cycle counter is reset when the length counter of the same channel is written to (via 5003h/5007h).

Initial Decay Volume:
Only a write out to 5003h/5007h will reset the current envelope decay counter to a known state (to 0Fh=max) for the appropriate channel's envelope decay hardware. Otherwise, the envelope decay counter is always counting down (by 1) at the frequency currently contained in the volume / envelope decay rate bits (even when envelope decays are disabled by setting bit 4), except when the envelope decay counter contains a value of 0, and envelope decay looping (bit 5) is disabled (0).

5002h - MMC5 Sound Frequency Channel 1 (Rectangle)
5006h - MMC5 Sound Frequency Channel 2 (Rectangle)
```
0-7    Lower 8 bits of wavelength (upper 3 bits in Register 3)
```
F = 1.79MHz/(N+1)/16 for Rectangle channels

5003h - MMC5 Sound Length Channel 1 (Rectangle)
5007h - MMC5 Sound Length Channel 2 (Rectangle)
Writing to the length registers restarts the length (obviously), and also restarts the duty cycle (channel 1,2 only), and restarts the decay volume.
```
0-2   Upper 3 bits of wavelength (unused on noise channel)
3-7   Length counter load register (5bit value, see below)
```
The above 5bit value is translated to the actual 7bit counter value as such:
```
Bit3=0 and Bit7=0 (Dividers matched for use with PAL/50Hz)
  Bit6-4  (0..7 = 05h,0Ah,14h,28h,50h,1Eh,07h,0Dh)
Bit3=0 and Bit7=1 (Dividers matched for use with NTSC/60Hz)
  Bit6-4  (0..7 = 06h,0Ch,18h,30h,60h,24h,08h,10h)
Bit3=1 (General Fixed Dividers)
  Bit7-4  (0..F = 7Fh,01h..0Fh)
```
The 7bit counter value is decremented at 120Hz rate (ie. MMC5 decrements faster than NES APU), the counter and sound output are stopped when reaching a value of zero. The counter can be paused (and restarted at current location) by Length Counter Clock Disabled bits in Register 5000h/5004h.

5010h MMC5 Sound Channel 3 PCM Control/Status (R/W)
```
Bit7/Write PCM IRQ upon Data=00h Enable (0=Disable, 1=Enable)
Bit7/Read  PCM IRQ upon Data=00h Flag   (0=None, 1=Interrupt Request)
Bit6-1     Not used
Bit0/Write Wave Output (0=Manual Write to 5011h, 1=Capture 8000h-BFFFh Reads)
```

5011h MMC5 Sound Channel 3 PCM Data
```
Bit7-0 Unsigned 8bit PCM Data (01h..FFh) (or 00h=trigger IRQ)
```
Used only in "Manual" mode (writes to 5011h are ignored in "Capture" mode).

MMC5 PCM Notes

Naturally, the cartridge cannot take control of the CPU bus (and thus can't perform real DMA transfers), instead, it's supporting a semi-automatic "Capture" mode, in that mode the CPU must read data from 8000h-BFFFh (eg. via LDA 8xxxh), and the data is then automatically forwarded to the DAC (without needing a STA 5011h opcode); this can save a few clock cycles.
Writing 00h to the DAC (either via Manual or Capture mode) leaves the DAC unchanged, and does instead trigger an IRQ; which can be used to handle end of data blocks, this can also save a few clock cycles.

5015h MMC5 Sound Channel 1 and 2 Enable/Status (R/W)

```
Bit7-2 Not used
Bit1   Channel 2  (0=Disable, 1=Enable)
Bit0   Channel 1  (0=Disable, 1=Enable)
```

Note
MMC5 Audio is used by Just Breed, Shin 4-Nin Uchi Mahjong, Metal Slader Glory.

Mapper 6,8,12,17: Front Far East (FFE) Configuration, IRQs, Patches

Front Far East (FFE) disk drive "backup unit" connects to the cartridge slot, allows to load copies of games from floppy/hdd/cdrom into RAM (max 512K) and VRAM (max 256K).

FFE Mapper Modes
Mapper 2: UNROM - PRG/16K
Mapper 6: FFE F4xxx - PRG/16K, VROM/8K, NT, IRQ
Mapper ?: FFE F4xxx - PRG/16K, VROM/8K, NT, IRQ
Mapper 8: FFE F3xxx - PRG/32K, VROM/8K, NT, IRQ
Mapper 12: FFE F6xxx - Not specified, NT, IRQ
Mapper 17: FFE F8xxx - PRG/8K, VROM/1K, NT, IRQ

FFE IRQ Registers

```
4501h  IRQ Disable/Acknowledge (write any value, usually 00h)
4502h  IRQ set lower 8bit of 16bit counter
4503h  IRQ set upper 8bit of 16bit counter and Start/Enable IRQs
```

IRQ counter is incremented each clock cycle, and produces IRQ on overflow.

FFE Trainers/Patches
All FFE games are patched to work with the "FFE" mappers. In case that the patches don't fit into normal ROM area, additional 512-byte patches are often located in SRAM area at 7000h-71FFh (or, in a few cases, reportedly at 5D00h), that patch-area may be used to handle FFE memory mapping, or for cheats/trainers. Some MMC games also contain similar trainers (maybe working on FFE device, if it supports MMC mappers?, or otherwise working on emulators only).

FFE Configuration Registers
Configuration Registers are initialized before the game is started, so most games don't need to access these registers, except for changing Name Table / Mirroring bits. A few games might also change the mode bits.

```
42FCh-42FFh  Configuration Register 1
  A0     Name Table Mode  (0=One-Screen, 1=Two-Screen) (with D4 below)
  A1     Unknown          (0=WE, 1=SW) (usually 1)
  D7-D5  Memory Mode (0-7) "*MODE"
           1      Mapper 6 F4xxx
           2      Mapper 2 UNROM
           3      Mapper ? F4xxx
           4      Mapper 8 F3xxx/GNROM
           0,5-7  unknown (Great Tank uses settings 1 and 6)
           ?      unknown how to select Mapper 12 and Mapper 17

           0      Mapper 17 (Kaiketsu, Saiyuuki)
           7      Mapper 17 (Wing of Madoola)
  D4     When A0=0: Select VRAM Page (?=BLK0, ?=BLK1)
         When A0=1: ?Mirroring (0=Vertical, 1=Horizontal Mirroring)
  D3-D0  Unknown (usually zero)
43FEh  Memory Control (apparently independendly of current Mode) (?)
  D7-D2  Select ?K ROM at 8000h-?
  D1-D0  Select 8K VROM at PPU 0000h-1FFFh
43FFh  Memory Control (as for current mode, ie. mirror of 8000h-FFFFh) (?)
4500h  Configuration Register 2
  D7-D6  FDS Mode (0=Disk/Load, 1=Reserved, 2=Cartridge, 3=Disk/Execute)
  D5-D4  SRAM 6000h-7FFFh BANK "Present or Not" (0-3=?)
  D3     SW Pin (maybe something related with above WE/SW selection)
  D2-D0  PPU Mode Select (1or2?="*MODE" (32K), 5=256K, VRAM EXT, 7=256K)
```

Mapper 6: FFE F4xxx - PRG/16K, VROM/8K, NT, IRQ

Several hacked Japanese titles use this mapper, such as the hacked version of Wai Wai World. The unhacked versions of these games seem to use a Konami VRC mapper, and it's better to use them if possible.

```
8000h-FFFFh  Memory Control (6bit)
  Bit1-0  Select 8K VRAM (read/write-able) at PPU 0000h-1FFFh
  Bit5-2  Select 16K ROM at 8000h-BFFFh (bank 0-0Fh)
  N/A     Fixed  16K ROM at C000h-FFFFh (always bank 7) (!)
```

Additional FFE registers:

Mapper 6,8,12,17: Front Far East (FFE) Configuration, IRQs, Patches

Mapper ?: FFE F4xxx - PRG/16K, VROM/8K, NT, IRQ

10% of games declared as "Mapper 6" are this (and not Mapper 6).

```
8000h-FFFFh  Memory Control (6bit)
  Bit3-0  Select 16K ROM at 8000h-BFFFh
  Bit5-4  Select 8K VROM at PPU 0000h-1FFFh
```

Lower bits are ROM bank, upper bits VROM bank, ie. vice-versa as Mapper 6.

Additional FFE registers:

Mapper 6,8,12,17: Front Far East (FFE) Configuration, IRQs, Patches

Mapper 7: AOROM - PRG/32K, Name Table Select

Numerous games released by Rare Ltd. use this mapper, such as Battletoads, Wizards & Warriors, and Solar Jetman.

```
8000h-FFFFh Memory Control
  Bit2-0   Select 32K ROM bank at 8000h-FFFFh (initially 1st bank)
  Bit4     One-Screen Name Table Select (0=BLK0, 1=BLK1)
  Bit3,5-7 Not used
```

Uses a single 74LS161 chip (connection: /PRG-CLK, R/W-/LOAD). ANROM additionally uses a 74LS02 to enable ROM only when R/W=HIGH and /PRG=LOW.

Board NES-AOROM-03: 256K PRG ROM (8 banks) (32pin ROM) (with bus-conflicts)

Board NES-ANROM-03: 128K PRG ROM (4 banks) (28pin ROM) (without bus-conflicts)

Board NES-AMROM: ?

All carts using it have 8K of VRAM at PPU 0000h.

BNROM

Board NES-BNROM-01: 128K PRG ROM (4 banks) (28pin ROM) (with bus-conflicts)

Deadly Towers uses BNROM (with horizontal mirroring, instead of Bit4), the game is typically marked as mapper 7 or mapper 34, although it's NOT AOROM nor NINA-001.

Mapper 8: FFE F3xxx - PRG/32K, VROM/8K, NT, IRQ

Several hacked Japanese titles use this mapper, such as the hacked version of Doraemon.

```
8000h-FFFFh  Memory Control (same as GNROM, Mapper 66)
  Bit1-0 Select 8K VROM (usually read-only) at PPU 0000h-1FFFh
  Bit5-4 Select 32K ROM at 8000h-FFFFh (initially 1st bank)
```

Additional FFE registers:

Mapper 6,8,12,17: Front Far East (FFE) Configuration, IRQs, Patches

Mapper 9: MMC2 - PRG/24K/8K, VROM/4K, NT, LATCH

Used only by Punch-Out, and Mike Tyson's Punch-Out.

```
A000h-AFFFh  Select 8K ROM at 8000h-9FFFh (initially 1st bank)
N/A          Fixed 24K ROM at A000h-FFFFh (always last three 8K banks)
B000h-CFFFh  Select 4K VROM at PPU 0000h-0FFFh
D000h-DFFFh  Select 4K VROM at PPU 1000h-1FFFh (used when latch=FDh)
E000h-EFFFh  Select 4K VROM at PPU 1000h-1FFFh (used when latch=FEh)
F000h-FFFFh  Mirroring Select (Bit0: 0=Vertical, 1=Horizontal mirroring)
PPU 1FD0h-1FDFh  Access to Pattern Table 0, Tile FDh --> sets latch=FDh
PPU 1FE0h-1FEFh  Access to Pattern Table 0, Tile FEh --> sets latch=FEh
```

The latch contains FEh on reset. The latch is automatically written to on any access to PPU 1FD0h-1FEFh, which does usually happen when the PPU fetches bitmap data for Tile FDh or FEh from Pattern Table 1.

The latches might also get changed on access to PPU 0FD0h-0FEFh (?)

Mapper 10: MMC4 - PRG/16K, VROM/4K, NT, LATCH

Used only by Fire Emblem, Fire Emblem Gaiden, and Family War.

```
A000h-AFFFh  Select 16K ROM bank at 8000h-BFFFh (initially 1st bank)
N/A          Fixed  16K ROM bank at C000h-FFFFh (always last bank)
B000h-BFFFh  Select 4K VROM bank at PPU 0000h-0FFFh (used when latch0=FDh)
C000h-CFFFh  Select 4K VROM bank at PPU 0000h-0FFFh (used when latch0=FEh)
D000h-DFFFh  Select 4K VROM bank at PPU 1000h-1FFFh (used when latch1=FDh)
E000h-EFFFh  Select 4K VROM bank at PPU 1000h-1FFFh (used when latch1=FEh)
F000h-FFFFh  Mirroring Select (Bit0: 0=Vertical, 1=Horizontal mirroring)
PPU 0FD0h-0FDFh  Access to Pattern Table 0, Tile FDh --> sets latch0=FDh
PPU 0FE0h-0FEFh  Access to Pattern Table 0, Tile FEh --> sets latch0=FEh
PPU 1FD0h-1FDFh  Access to Pattern Table 1, Tile FDh --> sets latch1=FDh
PPU 1FE0h-1FEFh  Access to Pattern Table 1, Tile FEh --> sets latch1=FEh
```

The latches contain FEh on reset. Latches are automatically written to on any access to PPU 0FD0h-0FEFh or 1FD0h-1FEFh, which does usually happen when the PPU fetches bitmap data for Tile FDh or FEh. The new latch setting is then used for all <further> tiles (tiles FDh/FEh are still fetched from the <old> latch setting).

Mapper 11: Color Dreams - PRG/32K, VROM/8K

This mapper is used on several unlicensed Color Dreams titles, including Crystal Mines and Pesterminator. Not sure if their religious ("Wisdom Tree") games use the same mapper or not.

```
8000h-FFFFh  Memory Control
  Bit3-0 Select 32K ROM bank at 8000h-FFFFh (initially 1st bank)
  Bit7-4 Select 8K VROM bank at PPU 0000h-1FFFh (initially 1st bank)
```

Many games using this mapper are somewhat glitchy. Bus-conflicts.
Uses a single 74LS377 (8bit D flip-flop with clock enable).

Mapper 12: FFE F6xxx - Not specified, NT, IRQ

No info. Don't have a ROM-image.
Maybe this meant to be "Mapper ?",
Mapper ?: FFE F4xxx - PRG/16K, VROM/8K, NT, IRQ
Additional FFE registers:
Mapper 6,8,12,17: Front Far East (FFE) Configuration, IRQs, Patches

Mapper 13: CPROM - 16K VRAM

Used by Videomation (a bitmap drawing program).

```
N/A          Fixed  4K VRAM at PPU 0000h-0FFFh (always Bank 0)
8000h-FFFFh  Select 4K VRAM at PPU 1000h-1FFFh (Bank 0-3)
```

16K VRAM for 32x30 different BG tiles (plus 64 sprites). Bus-conflicts.

Mapper 15: X-in-1 - PRG/32K/16K, NT

Used by Contra 100-in-1 (fake multicart with less than 100 different games), and hacked versions of Crazy Climber, Dragon Ball, and Mobile Suit (single game carts).

```
8000h-FFFFh  Memory Control (Decoded by address AND data lines)
  D5-D0 Select 16K ROM Bank (X)
  D6    Mirroring Control   (0=Vertical, 1=Horizontal Mirroring)
  D7    Select 8K ROM Bank  (Y) (should be zero in non-8K-modes)
  A1-A0 ROM Bank Mode       (0=32K, 1=128K, 2=8K, 3=16K)
Mapping in different modes is:
  8K Mode   - Bank (X*2+Y) at each 8000h, A000h, C000h, E000h
  16K Mode  - Bank (X) at 8000h-BFFFh and (X) at C000h-FFFFh
  32K Mode  - Bank (X) at 8000h-BFFFh and (X OR 1) at C000h-FFFFh
  128K Mode - Bank (X) at 8000h-BFFFh and LAST bank at C000h-FFFFh
```

Initially first 32K ROM selected. The cartridge contains 8K VRAM.

Mapper 16: Bandai - PRG/16K, VROM/1K, IRQ, EPROM

This mapper is used on several Japanese titles by Bandai, such as the DragonBall Z series and the SD Gundam Knight series.

```
6000h,7FF0h,8000h  Select 1K VROM at PPU 0000h-03FFh
6001h,7FF1h,8001h  Select 1K VROM at PPU 0400h-07FFh
6002h,7FF2h,8002h  Select 1K VROM at PPU 0800h-0BFFh
6003h,7FF3h,8003h  Select 1K VROM at PPU 0C00h-0FFFh
6004h,7FF4h,8004h  Select 1K VROM at PPU 1000h-13FFh
6005h,7FF5h,8005h  Select 1K VROM at PPU 1400h-17FFh
6006h,7FF6h,8006h  Select 1K VROM at PPU 1800h-1BFFh
6007h,7FF7h,8007h  Select 1K VROM at PPU 1C00h-1FFFh
```

```
6008h,7FF8h,8008h  Select 16K ROM at 8000h-BFFFh (initially 1st bank)
N/A                Fixed  16K ROM at C000h-FFFFh (always last bank)
6009h,7FF9h,8009h  Mirroring/Page Select (Bit1-0)
  0 Two-Screen Vertical mirroring
  1 Two-Screen Horizontal mirroring
  2 Single-Screen BLK0
  3 Single-Screen BLK1
600Ah,7FFAh,800Ah  IRQ Control Register (Bit 0)
  0 Disable/Acknowledge IRQ
  1 Enable IRQ
600Bh,7FFBh,800Bh  Low byte of IRQ counter
600Ch,7FFCh,800Ch  High byte of IRQ counter
600Dh,7FFDh,800Dh  EPROM I/O Port - unknown how this works.
```

The IRQ counter is decremented each clock cycle if active, and set off when it reaches zero. An IRQ interrupt is executed at that point.

Datach Variant

Bandai's "Datach" games are using a Mapper 16 variant (with VRAM instead VROM), and with built-in barcode reader. This seems to be occassionally referred to as "Mapper 157" (though older Datach ROM-images are typically declared as "Mapper 16").

Controllers - Barcode Readers

Mapper 17: FFE F8xxx - PRG/8K, VROM/1K, NT, IRQ

Several hacked Japanese titles use this mapper, such as the hacked versions of Parodius and DragonBall Z 3.

```
4504h  Select 8K ROM at 8000h-9FFFh (initially 1st half of 1st 16K)
4505h  Select 8K ROM at A000h-BFFFh (initially 2nd half of 1st 16K)
4506h  Select 8K ROM at C000h-DFFFh (initially 1st half of last 16K)
4507h  Select 8K ROM at E000h-FFFFh (initially 2nd half of last 16K)
4510h  Select 1K VROM at PPU 0000h-03FFh
4511h  Select 1K VROM at PPU 0400h-07FFh
4512h  Select 1K VROM at PPU 0800h-0BFFh
4513h  Select 1K VROM at PPU 0C00h-0FFFh
4514h  Select 1K VROM at PPU 1000h-13FFh
4515h  Select 1K VROM at PPU 1400h-17FFh
4516h  Select 1K VROM at PPU 1800h-1BFFh
4517h  Select 1K VROM at PPU 1C00h-1FFFh
```

Additional FFE registers:

Mapper 6,8,12,17: Front Far East (FFE) Configuration, IRQs, Patches

Mapper 18: Jaleco SS8806 - PRG/8K, VROM/1K, NT, IRQ, EXT

This mapper is used on several Japanese titles by Jaleco, such as Baseball 3, Lord of Kings, etc.

```
8000h/8001h  Select 8K ROM at 8000h-9FFFh (Lower/Upper 4bits)
8002h/8003h  Select 8K ROM at A000h-BFFFh (Lower/Upper 4bits)
9000h/9001h  Select 8K ROM at C000h-DFFFh (Lower/Upper 4bits)
N/A          Fixed  8K ROM at E000h-FFFFh (always last bank)
9002h        Battery Back SRAM (Bit0: 0=Enable, 1=Disable)
             (unused by Lord of Kings)
9003h        Unknown
             (used by Lord of Kings)
A000h/A001h  Select 1K VROM at PPU 0000h-03FFh (Lower/Upper 4bits)
A002h/A003h  Select 1K VROM at PPU 0400h-07FFh (Lower/Upper 4bits)
B000h/A001h  Select 1K VROM at PPU 0800h-0BFFh (Lower/Upper 4bits)
B002h/A003h  Select 1K VROM at PPU 0C00h-0FFFh (Lower/Upper 4bits)
C000h/C001h  Select 1K VROM at PPU 1000h-13FFh (Lower/Upper 4bits)
C002h/C003h  Select 1K VROM at PPU 1400h-17FFh (Lower/Upper 4bits)
D000h/D001h  Select 1K VROM at PPU 1800h-1BFFh (Lower/Upper 4bits)
D002h/D003h  Select 1K VROM at PPU 1C00h-1FFFh (Lower/Upper 4bits)
E000h/E001h  Lower 8bit of decrementing 16bit IRQ counter (Lower/Upper 4bits)
E002h/E003h  Upper 8bit of decrementing 16bit IRQ counter (Lower/Upper 4bits)
F000h  IRQ Control Register 0
 Bit0    Maybe 1=Load Counter?
F001h  IRQ Control Register 1
 Bit0    IRQ Enable        (0=Disabled, 1=Enable)
 Bit1-3  IRQ Counter Width (0=16bit, 1=12bit, 2-3=8bit, 4-7=4bit)
 With widths less than 16bit, underflows recurse only lower counter bits.
F002h  Name Table Select (2bit)
             0   Two-Screen, Horizontal Mirroring
             1   Two-Screen, Vertical Mirroring
             2-3 Single-Screen BLK0
F003h  Unused (or an External I/O Port which is unused?)
```

Mapper 19: Namcot 106 - PRG/8K, VROM/1K/VRAM, IRQ, SOUND

This mapper is used on several Japanese titles by Namcot, such as Splatterhouse and Family Stadium '90.

Pattern Table Control

```
  8000h-87FFh  Select 1K VROM at PPU 0000h-03FFh (with E800h/Bit6)
  8800h-8FFFh  Select 1K VROM at PPU 0400h-07FFh ("")
  9000h-97FFh  Select 1K VROM at PPU 0800h-0BFFh ("")
  9800h-9FFFh  Select 1K VROM at PPU 0C00h-0FFFh ("")
  A000h-A7FFh  Select 1K VROM at PPU 1000h-13FFh (with E800h/Bit7)
  A800h-AFFFh  Select 1K VROM at PPU 1400h-17FFh ("")
  B000h-B7FFh  Select 1K VROM at PPU 1800h-1BFFh ("")
  B800h-BFFFh  Select 1K VROM at PPU 1C00h-1FFFh ("")
```

The upper two bits Port E800h-EFFFh are used to select VROM/VRAM mode (mind that the lower six bits of that port Select 8K ROM at A000h-BFFFh).

```
  E800h, Bit6  VROM/VRAM Mode for PPU 0000h-0FFFh (0=VROM+VRAM, 1=VROM-Only)
  E800h, Bit7  VROM/VRAM Mode for PPU 1000h-1FFFh (0=VROM+VRAM, 1=VROM-Only)
```

In VROM-Only mode, VROM banks 0-FFh can be selected. In VROM+VRAM mode only VROM banks 0-DFh can be selected, and values E0h-FFh select VRAM.

Name Table Control

```
  C000h-C7FFh  Select 1K VROM/VRAM at PPU 2000h-23FFh (E0h and up = VRAM)
  C800h-CFFFh  Select 1K VROM/VRAM at PPU 2400h-27FFh (E0h and up = VRAM)
  D000h-D7FFh  Select 1K VROM/VRAM at PPU 2800h-2BFFh (E0h and up = VRAM)
  D800h-DFFFh  Select 1K VROM/VRAM at PPU 2C00h-2FFFh (E0h and up = VRAM)
```

Only VROM banks 0-DFh can be selected, and values E0h-FFh activate internal VRAM, Bit0 of the bank number is then used to select BLK0 or BLK1.

CPU Memory Control

```
  E000h-E7FFh  Select 8K ROM at 8000h-9FFFh (initially 1st half of 1st 16K)
               Bit5-0  Page_number
  E800h-EFFFh  Select 8K ROM at A000h-BFFFh (initially 2nd half of 1st 16K)
               Bit5-0  Page_number
               Bit6  Select at CHR_address $0000-$0FFF 0:ROM&RAM 1:ROM
               Bit7  Select at CHR_address $1000-$1FFF 0:ROM&RAM 1:ROM
  F000h-F7FFh  Select 8K ROM at C000h-DFFFh (initially 1st half of last 16K)
               Bit5-0  Page_number
  N/A          Fixed  8K ROM at E000h-FFFFh (always    2nd half of last 16K)
```

The lower 6bit of these registers specify ROM bank numbers. Caution: The upper 2bit of E800h-EFFFh are used to select Pattern Table VROM/VRAM Mode.

IRQ Control

```
  5000h-57FFh  Bit7-0: Lower 8bit of 15bit IRQ counter (R/W) (!)
  5800h-5FFFh  Bit6-0: Upper 7bit of 15bit IRQ counter (R/W) (!)
               Bit7:   0=Disable IRQs, 1=Enable IRQs
```

The IRQ counter is incremented each clock cycle, an IRQ is generated when it overflows (at 7FFFh, since it's a 15bit value). Sangokushi 2 uses IRQs, but many other Namcot games don't use IRQs.

Sound Control

```
  4800h Expand I/O Data Register
  F800h Expand I/O Address Register
    Bit7   Auto Increment (0=Disable, 1=Enable)
    Bit6-0 Address        (00h-7Fh)
```

Index Addresses: (Dots "..." = Japanese text, not translated)

```
  00h-3Fh                         See NAMCO.TXT, Japanese (.............)
  40h,48h,50h,58h,60h,68h,70h,78h Channel 1-8, Frequency Lower 8bit
  41h,49h,51h,59h,61h,69h,71h,79h See NAMCO.TXT, Japanese (......)
  42h,4Ah,52h,5Ah,62h,6Ah,72h,7Ah Channel 1-8, Frequency Middle 8bit
  43h,4Bh,53h,5Bh,63h,6Bh,73h,7Bh See NAMCO.TXT, Japanese (......)
  44h,4Ch,54h,5Ch,64h,6Ch,74h,7Ch Channel 1-8, Frequency Upper 2bit & Option
    Bit7-5 Not used
    Bit4-2 VVV: 8-(....)(... 2byte) ..: VVV=000... 16byte,VVV=100..8byte....
    Bit1-0 Frequency Upper 2bit
  45h,4Dh,55h,5Dh,65h,6Dh,75h,7Dh See NAMCO.TXT, Japanese (......)
  46h,4Eh,56h,5Eh,66h,6Eh,76h,7Eh Channel 1-8, Offset Address (00h-3Fh)
    Bit7-1 AAAAAAA [6bit address stored in a 7bit value?]
    Bit0   Not used
  47h,4Fh,57h,5Fh,67h,6Fh,77h,7Fh Channel 1-8
    Bit7-4 ????: 7...(kingofkings),3...
    Bit3-0 VVVV: ....
```

Frequency: 0=Lowest, 3FFFFh=Highest.
According to Gorohs frequency table, Tone "C" of Octave "1-8" is: 1:47Eh, 2:8FBh, 3:11F6h, 4:23ECh, 5:47DAh, 6:8FB3h, 7:11F66h, 8:23ECCh.
According to my CPC frequency table, middle "C" of octave "0" should be 261.626Hz. No idea how to match that into a formula.

Mapper 20: Disk System - PRG RAM, BIOS, DISK, IRQ, SOUND

Used by Famicom Disk System only.
Famicom Disk System (FDS)

Mapper 21: Konami VRC4A/VRC4C - PRG/8K, VROM/1K, NT, IRQ

Mapper 21-26,73,75,85: Konami VRC Mappers

Mapper 22: Konami VRC2A - PRG/8K, VROM/1K, NT

Mapper 21-26,73,75,85: Konami VRC Mappers

Mapper 23: Konami VRC2B/VRC4E - PRG/8K, VROM/1K, NT, (IRQ)

Mapper 21-26,73,75,85: Konami VRC Mappers

Mapper 24: Konami VRC6A - PRG/16K/8K, VROM/1K, NT, IRQ, SOUND

Mapper 21-26,73,75,85: Konami VRC Mappers

Mapper 25: Konami VRC4B/VRC4D - PRG/8K, VROM/1K, NT, IRQ

Mapper 21-26,73,75,85: Konami VRC Mappers

Mapper 26: Konami VRC6B - PRG/16K/8K, VROM/1K, NT, IRQ, SOUND

Mapper 21-26,73,75,85: Konami VRC Mappers

VRC6 Sound Registers

Two Rectangle channels with 4bit volume levels each, one Saw channel with 5bit volume level. These are added into a 6bit output level, and then merged with normal Famicom sound signal.

9000h/A000h - Channel 1/2 - Square Volume/Duty

```
Bit7-4 Duty Cycle bits:
   0000 - 1/16  "-_______________-_______________" ( 6.25%)
   0001 - 2/16  "--______________--______________" (12.50%)
   0010 - 3/16  "---_____________---_____________" (18.75%)
   0011 - 4/16  "----____________----____________" (25.00%)
   0100 - 5/16  "-----___________-----___________" (31.25%)
   0101 - 6/16  "------__________------__________" (37.50%)
   0110 - 7/16  "-------_________-------_________" (43.75%)
   0111 - 8/16  "--------________--------________" (50.00%)
   1xxx - 16/16 "--------------------------------" (100.00%)
Bit3-0 Linear Volume  (0=Silence, 0Fh=Loudest)
```

100% Duty can be used as digitized mode, the channel permanently outputs high level, ie. the current volume setting, and isn't affected by the frequency registers.

B000h - Channel 3 - Saw Volume Step

```
Bit7-6 Not used
Bit5-0 Volume Step (V) (0..2Ah=Silent..Loudest) (2Bh..3Fh=Wraps/Garbage)
```

The overall output looks like "//////" whereas each "/" is split into 7 steps, the output level on step 1-7 is calculated as such:

```
FOR I=1 to 7                ;step 1-7
  IF I=1 THEN X=0           ;reset to 0 in 1st step
         ELSE X=(X+V) AND FFh ;add accumulator
  Output=(X/8)              ;output upper 5bit of X
NEXT
```

Note: X is an 8bit value, and wraps on overflow, ie. if V>2Ah.

9001h/A001h/B001h - Channel 1/2/3 - Frequency LSB

```
Bit7-0 Lower 8 bits of frequency data
```

9002h/A002h/B002h - Channel 1/2/3 - Frequency MSB

```
Bit7   Channel disable  (0=Disable, 1=Enable)
Bit6-4 Not used
Bit3-0 Upper 4 bits of frequency data
```

To calculate output frequency:

```
Channel 1/2: F=1.79MHz/16/(N+1)   ;16-step duty cycles
Channel 3:   F=1.79MHz/14/(N+1)   ;7-step phases
```

Mapper 21-26,73,75,85: Konami VRC Mappers

This chapter describes all known VRC variants. The different Port addresses are specified as X.Y.Z which may look a bit abstract at the first glance, at second glance it should be easier to understand as than using separate chapters for each of the 13 variants.

VRC Chip Versions

Type	PRG Bank	VROM Banks	NT	IRQ	Sound
VRC1	PRG/8K	VROM/4K	NT	-	-
VRC2	PRG/8K	VROM/1K	NT	-	-
VRC3	PRG/16K	VRAM		IRQ	-
VRC4	PRG/8K	VROM/1K	NT	IRQ	-
VRC6	PRG/16K/8K	VROM/1K	NT	IRQ	SOUND
VRC7	PRG/16K/8K	VROM/1K	NT	IRQ	SOUND

For most chips, there are different connection variants, VRC2a, VRC2b, etc.

VRC Data Bus

VRC1, VRC2, and VRC3 use a 4bit data bus, connected to D3-D0, any 8bit registers are thus split into two 4bit ports. VRC4 seems to have an additional D4 pin, which is used only for 5bit PRG ROM banks. VRC6 and VRC7 have a full 8bit data bus.

VRC Address Bus

Most VRCs have a 6bit address bus, described here as X.Y.Z - the upper four address bits are always A15-A12 (X), however, the connection of the lower two address bits (Y and Z) varies. VRC7 normally uses only X.Y bits (only the Sound registers use X.Y.Z). VRC1/VRC3 uses only the X address bits. A15 serves as chip select, and must be HIGH for all VRC registers.

VRC Connection Variants of Lower two bits of X.Y.Z addresses

Mapper		Y	Z	Used by
75	VRC1	-	-	Ganbare Goemon 1, Junior Basket - Two on Two, King Kong 2, Exciting Boxing, Jajamaru Ninpou Chou, Tetsuwan Atom
22	VRC2a	A0	A1	Twin Bee 3, Ganbare Pennant Race
23	VRC2b	A1	A0	Wai Wai World 1, Getsufuu Maden, Kaiketsu Yanchamaru 2 Dragon Scroll, Gryzor/Contra, Jarinko Chie, Ganbare Goemon
73	VRC3	-	-	Salamander
21	VRC4a	A2	A1	Wai Wai World 2
21	VRC4c	A7	A6	Ganbare Goemon Gaiden 2
25	VRC4b	A0	A1	Bio Miracle Bokutte Upa, Ganbare Goemon Gaiden, Gradius 2, Racer Mini Yonku
25	VRC4d	A2	A3	Teenage Mutant Hero Turtles 1+2, Goal!!
23	VRC4e	A3	A2	Parodius da!, Akumajou Special, Crisis Force, Tiny Toon Adventures 1, Moe Pro!
24	VRC6a	A1	A0	Akumajou Densetsu (Castlevania 3)
26	VRC6b	A0	A1	Esper Dream 2, Mouryou Senki Madara
85	VRC7	A4	(A5)	Lagrange Point (Z=A5 used for Sound only)
85	VRC7b	A3	(?)	Tiny Toon Adventures 2 (no Sound - maybe not a VRC7 ?)

Note that most mapper numbers are shared for two different connection variants, mapper 23 is even shared for different chip versions, the unused address bits are usually zero, so that software and hardware could, for example, reproduce Z=(A0 OR A2) for mapper 23.
VRC2a variant uses VROM bank outputs Bit1-7, all other VRCs use Bit0 and up.

PRG ROM Bank Registers (decoded by X or X.Y parts of the X.Y.Z address)

VRC1	VRC2	VRC3	VRC4	VRC6	VRC7	Expl.
-	-	F	-	8	-	Select 16K ROM at 8000h-BFFFh
8	8	-	8	-	8.0	Select 8K ROM at 8000h-9FFFh
A	A	-	A	-	8.1	Select 8K ROM at A000h-BFFFh
C	-	-	-	C	9.0	Select 8K ROM at C000h-DFFFh
-	FIX	FIX	FIX	-	-	Fixed 8K ROM at C000h-DFFFh (last-1 8K)
FIX	FIX	FIX	FIX	FIX	FIX	Fixed 8K ROM at E000h-FFFFh (last-0 8K)

The 16K banks of VRC3/VRC6 are Linear Addresses divided by 16K (not by 8K).
VRC4 can swap 8000h-9FFFh and C000h-DFFFh, see VRC4 Memory Control below.

VROM Bank Registers (VRC2,VRC4=2x4bit LSB/MSB, VRC6,VRC7=8bit)

VRC2,4	VRC6	VRC7	Expl.
B.0.0/1	D.0.0	A.0	Select 1K VROM bank at PPU 0000h-03FFh
B.1.0/1	D.0.1	A.1	Select 1K VROM bank at PPU 0400h-07FFh
C.0.0/1	D.1.0	B.0	Select 1K VROM bank at PPU 0800h-0BFFh
C.1.0/1	D.1.1	B.1	Select 1K VROM bank at PPU 0C00h-0FFFh
D.0.0/1	E.0.0	C.0	Select 1K VROM bank at PPU 1000h-13FFh
D.1.0/1	E.0.1	C.1	Select 1K VROM bank at PPU 1400h-17FFh
E.0.0/1	E.1.0	D.0	Select 1K VROM bank at PPU 1800h-1BFFh
E.1.0/1	E.1.1	D.1	Select 1K VROM bank at PPU 1C00h-1FFFh

Note that VRC2A uses Bit7-1 of the 2x4bit register, VRC2B uses Bit6-0, VRC4 and up use Bit6-0 (or all bits, Bit7-0, for large VROMs).
Lagrange Point contains VRAM instead VROM, the VRAM <is> map-able.
Salamander (VRC3) contains VRAM, which appears to be <not> map-able.

VRC1 VROM Bank Registers (5bit values, split into 4+1 bits)

9	Bit0: Mirroring, Bit1-2: MSBs of VROM banks, Bit3: Unused/zero

```
E   Lower 4bit of 4K VROM bank at PPU 0000h-0FFFh (MSB in Bit1 of Register 9)
F   Lower 4bit of 4K VROM bank at PPU 1000h-1FFFh (MSB in Bit2 of Register 9)
```

IRQ Registers (VRC1,VRC2=N/A, VRC3,VRC4=2x4bit, VRC6,VRC7=8bit)

```
VRC4    VRC6  VRC7 VRC3 Expl.
F.0.0/1 F.0.0 E.1  A/B  IRQ Reload value
F.1.0   F.0.1 F.0  C    IRQ Control (Bit0: 0=Disable, Bit1: 0=One-Shot)
F.1.1   F.1.0 F.1  D    IRQ Acknowledge (write any value to this address)
```

IRQ Reload is loaded to actual counter on write to IRQ Control Register, and on Counter overflow. If IRQ Control Regiser Bit1 is set, then the counter is automatically Restarted and Reloaded on Overflow.
The IRQ counter is incremented each 113.75 cycles (or each 114 cycles?), which is almost exactly once per NTSC-scanline, including for "hidden" scanlines during VBlank (only exception is VRC3, which is incremented every 256 cycles).
Mind that PAL/NTSC have different VBlank/Hblank times, and so, need different counter values. The VRC4 games Goal! and Moe Pro! appear to be bugged pirate ports from original Jaleco mapper to Konami-style mapper, these games do incorrectly acknowledge IRQs by writing zero to the IRQ Control register rather than by writing any value to the IRQ Acknowledge register, not sure if that works on real VRC4 hardware.

VRC4 Memory Control (VRC4 only - not VRC2,6,7)

```
9.0.1 (or 9.1.0?)  Memory Control (2bit)
```

Bit1: PRG ROM Swap (0=Normal, 1=Swap) When swapped: Port 8.0.0 controls ROM at C000h-DFFFh, and ROM at 8000h-9FFFh becomes fixed, containing the 1st half of <last> 16K.
Bit0: Enable SRAM at 6000h-7FFFh (0=Disable, 1=Enable), VRC6 is having an equivalent function Bit7 of Name Table Control register.

Name Table Control

```
VRC1 VRC2,4  VRC6   VRC7  Expl.
9    9.0.0   B.1.1  E.0   Mirroring/Page Select
```

Mirroring selection VRC2,VRC4,VRC7: Bit1-0, VRC6: Bit3-2, VRC1: Bit0:

```
0  Two-Screen Vertical mirroring   (VRC1: Register 9, Bit0=0)
1  Two-Screen Horizontal mirroring (VRC1: Register 9, Bit0=1)
2  Single-Screen BLK1              (VRC1: N/A)
3  Single-Screen BLK0              (VRC1: N/A)
```

On VRC6, bit5 additionally inverts the BLK-outputs, BLK0 then becomes BLK1, and vice versa, that also applies in Two-Screen modes, ie. the upper-left name table may be either BLK1 or BLK0.
And, on VRC6, Bit7 controls SRAM (0=Disable, 1=Enable).
Note: VRC1 is also used by some VS System games. The VS System has 4K VRAM for Four-Screen nametables (so, the VRC1's mirroring flag is probably ignored on the VS System).

VRC6 Sound Registers

Mapper 26: Konami VRC6B - PRG/16K/8K, VROM/1K, NT, IRQ, SOUND

VRC7 OPL2 Sound Registers

```
9.1.0   Index Register
9.1.1   Data Register
```

Mapper 85: Konami VRC7A/B - PRG/16K/8K, VROM/1K, NT, IRQ, SOUND

Unknown Registers

VRC3 - Salamander writes 00h to Port 8000h and 9000h.

Mapper 28: Action 53 homebrew X-in-1

Used by the Action 53 homebrew multicart series. Supports NROM (1x16K/1x32K PRG), UNROM (max 16x16K PRG), AOROM/ANROM/BNROM (max 8x32K PRG), CNROM (max 4x8K CHR).
The menu should initialize the four registers per game (for CHR ROM games it should also relocate data to CHR RAM), and should finally set {5000h]=00h/01h to select CHR/PRG mapping. The games can then simply use [8000h] to change CHR ot PRG mapping (and usually don't need to touch [5000h] themselves; unless they want to use both CHR and PRG banking, or want to change Hori/Vert mirroring).

Registers

```
4444h        Unknown (used by menu software)
5000h-5FFFh  Select index (00h,01h,80h,81h) (bit1-6=Reserved)
8000h-FFFFh  Write data to selected register
```

Some games ported from SGROM (=what?) may rewrite Reg[80h], usually to change mirroring.

Reg[00h]: CHR bank (CNROM style)

```
0-1  Select 8K VRAM at PPU 0000h-1FFFh (max 32Kbyte addressable)
2-3  Reserved (0)
4    One-Screen Name Table Select (0=BLK0, 1=BLK1)
5-7  Reserved (0)
```

Bit4 is copied to Reg[80h].Bit0 when Reg[80h].Bit1=0 (otherwise bit4 is ignored).

Reg[01h]: PRG bank (inner bank) (AOROM/ANROM/BNROM/UNROM style)

```
0-3  Select 16K/32K ROM bank at 8000h-HFFFh or C000h-FFFFh or 8000h-FFFFh
4    One-Screen Name Table Select (0=BLK0, 1=BLK1)
```

```
  5-7  Reserved (0)
```
Bit4 is copied to Reg[80h].Bit0 when Reg[80h].Bit1=0 (otherwise bit4 is ignored).

Reg[80h]: Mode
```
  0-1  Nametable mode      (0=BLK0, 1=BLK1, 2=VertMirroring, 3=HoriMirroring)
  2-3  PRG bank mode       (0=32K, 1=Same as 0, 2=16K at C000h, 3=16K at 8000h)
  4-5  PRG outer bank size (0=32K, 1=64K, 2=128K, 3=256K)
  6    Reserved (0)
  7    Reserved (0 or 1 used by menu, but has no function)
```
If Reg[80h].Bit1=0 then Reg[80h].Bit0 can be also changed via writes to Reg[00h/01h].Bit4.
```
  PRG Bank Mode  8000h-BFFFh              C000h-FFFFh
  32K at 8000h   Var: outer*2+inner*2+0   Var: outer*2+inner*2+1   ;AOROM etc
  16K at 8000h   Var: outer*2+inner*1     Fixed: outer*2+1         ;UNROM
  16K at C000h   Fixed: outer*2+0         Var: outer*2+inner*1
```
Merging "outer+inner" is masking off outer.LSBs and inner.MSBs depending on outer bank size (ie. it's no real addition with overlapping bits). However, the fixed banks are NOT masking off outer.LSBs.

Reg[81h]: PRG bank (outer bank) (max 8Mbyte)
```
  0-7  Select 32K ROM bank at 8000h-FFFFh (initially last 16K at C000h)
```
When the outer bank size is set greater than 32K, the bank number is not barrel shifted. Instead, the least significant bits are ignored. Bits 5 through 3, for example, always control PRG ROM A20 through A18.

Reset

Power-up reset maps last 16K of ROM to C000h-FFFFh (and leaves everything else uninitialized). Soft reset does not affect any hardware registers, however, games should contain some stub for redirecting their reset vector(s) to restart the menu.

Mapper 32: Irem G-101 - PRG/8K, VROM/1K, NT

This mapper is used on several Japanese titles by Irem, such as ImageFight 2.
```
  9FFFh  Control Register (Bit1,0)
          Bit0 - Name Table ?Mirroring (0=Horizontal, 1=Vertical Mirroring)
          Bit1 - Port 8FFFh Switching Mode (see above)
  8FFFh  When 9FFFh/Bit1=0:
          Select 8K ROM bank at 8000h-9FFFh (initially 1st 8K bank)
          Fixed  8K ROM bank at C000h-DFFFh (always 1st half of last 16K)
         When 9FFFh/Bit1=1:
          Fixed  8K ROM bank at 8000h-9FFFh (always 1st 8K bank)
          Select 8K ROM bank at C000h-DFFFh (initially probably 9FFFh/Bit1=0)
  AFFFh  Select 8K ROM bank at A000h-BFFFh (initially 2nd 8K bank)
  N/A    Fixed  8K ROM bank at E000h-FFFFh (always last 8K bank)
  BFF0h  Select 1K VROM bank at PPU 0000h-03FFh
  BFF1h  Select 1K VROM bank at PPU 0400h-07FFh
  BFF2h  Select 1K VROM bank at PPU 0800h-0BFFh
  BFF3h  Select 1K VROM bank at PPU 0C00h-0FFFh
  BFF4h  Select 1K VROM bank at PPU 1000h-13FFh
  BFF5h  Select 1K VROM bank at PPU 1400h-17FFh
  BFF6h  Select 1K VROM bank at PPU 1800h-1BFFh
  BFF7h  Select 1K VROM bank at PPU 1C00h-1FFFh
```

Mapper 33: Taito TC0190/TC0350 - PRG/8K, VROM/1K/2K, NT, IRQ

Used by Don Doko Don I, II, Flintstones - Rescue of Dino & Hoppy, Bakushou Jinsei Gekijou I, II, III, Insector X, Operation Wolf, Power Blazer, Golf Ko Open, Akira, Takeshi no Sengoku Fuuunji, Jetsons - Cogswell's Caper, Bubble Bobble 2, Captain Saver.

TC0190 and TC0350 are slightly different, one has IRQs, one doesn't. No idea which is which, so they'll be referenced as Type I and II.

Type I and II - Memory Banking Registers
```
  8000h  Select 8K ROM bank at 8000h-9FFFh (Type I: Bit6=Mirroring, see below)
  8001h  Select 8K ROM bank at A000h-BFFFh
  N/A    Fixed 16K ROM bank at C000h-FFFFh (always last 16K)
  8002h  Select 2K VROM bank at PPU 0000h-07FFh
  8003h  Select 2K VROM bank at PPU 0800h-0FFFh
  A000h  Select 1K VROM bank at PPU 1000h-13FFh
  A001h  Select 1K VROM bank at PPU 1400h-17FFh
  A002h  Select 1K VROM bank at PPU 1800h-1BFFh
  A003h  Select 1K VROM bank at PPU 1C00h-1FFFh
```

Type I - Mirroring
```
  8000h  Bit4-0:See above, Bit6:Mirroring (0=Vertical, 1=Horizontal Mirroring)
```
Ignore this bit if Type II registers are used.

Type II - Mirroring and IRQ
```
  C000h  IRQ Counter (incremented every scanline, paused during VBlank)
```

```
C001h  IRQ Related (write same value as to C000h)
C002h  IRQ Start/Enable     (write any value)
C003h  IRQ Acknowledge/Stop (write any value)
E000h  Mirroring (Bit6) (0=Vertical, 1=Horizontal Mirroring)
E001h,E002h,E003h  Unknown
```

Mapper 34: Nina-1 - PRG/32K, VROM/4K

Used by Impossible Mission II.

```
7FFEh  Select 4K VROM bank at PPU 0000h-0FFFh (4bit)
7FFFh  Select 4K VROM bank at PPU 1000h-1FFFh (4bit)
7FFDh  Select 32K ROM bank at 8000h-FFFFh (1bit) (initially 1st bank)
```

Contains 8K WRAM at 6000h-7FFFh (not sure if last three bytes can be used).
Uses six TTL chips, 2x74LS173, 74LS139, 74LS133, 74LS74, and 74LS00, and a faux-lockout chip labelled 'NINA'.

Note: BNROM is accidentally also marked Mapper 34, although it's actually a AOROM variant. For details, see
Mapper 7: AOROM - PRG/32K, Name Table Select

Mapper 40: FDS-Port - Lost Levels

Used by Super Mario Bros 2 - Lost Levels.

```
8000h-9FFFh  Disable/Reset IRQ counter (by writing any value)
A000h-BFFFh  Enable/Start IRQ counter (by writing any value)
C000h-DFFFh  Not Used
N/A          Fixed 8K ROM at 6000h-7FFFh (always bank 6)
N/A          Fixed 8K ROM at 8000h-9FFFh (always bank 4)
N/A          Fixed 8K ROM at A000h-BFFFh (always bank 5)
E000h-FFFFh  Select 8K ROM at C000h-DFFFh
N/A          Fixed 8K ROM at E000h-FFFFh (always bank 7, ie. last bank)
```

When enabled, IRQ generated after 4096 clock cycles (about 36 scanlines).

Uses different ports, but the features are about same as:
Mapper 50: FDS-Port - Alt. Levels

Mapper 41: Caltron 6-in-1

Used by Caltron 6-in-1 cartridge only.

```
6000h-67FFh  Main Control Register (decoded by ADDRESS lines A0-A5)
  A2-A0  Select 32K ROM at 8000h-FFFFh
  A2     MSB of above bank number - also enables second register
  A4-A3  Upper two bits of 8K VROM bank at 0000h-1FFFh
  A5     Name Table   (0=Vertical, 1=Horizontal Mirroring)
8000h-FFFFh  Auxilary CHR control (decoded by DATA lines D0-D1)
  This register is write-protected when above A2=0 (!)
  D1-D0  Lower two bits of 8K VROM bank at 0000h-1FFFh
```

When the NES is switched on, or the reset button is pressed, both registers are cleared to 00h, done by a cool little diode / RC circuit on the PHI2 line.

Mapper 42: FDS-Port - Mario Baby

Used only by one game: A pirate copy of Bio Miracle Boukette Upa, renamed to Mario Baby, and modified to work as cartridge (instead FDS floppy disk).

```
E000h-FFFCh  Select 8K ROM at 6000h-7FFFh
N/A          Fixed 32K ROM at 8000h-FFFFh (always last 32K)
E001h-FFFDh  Select mirroring (Bit3: 0=Vertical, 1=Horizontal Mirroring)
E002h-FFFEh  IRQ Control      (Bit1: 0=Disable/Reset, 1=Enable/Start)
E003h-FFFFh  Not used
```

When enabled, IRQ generated after 24576 clock cycles (about 216 scanlines).
These ports are mirrored from E000h-FFFFh, every 4 bytes.
Consists of a whopping 11 chips - 9 TTL/CMOS, 8K RAM, and 128K ROM.

Mapper number 42 is also assigned to Ai Senshi Nicol. In short, doing this:

```
8000h        Select 8K VROM at PPU 0000h-1FFFh
F000h        Select 8K ROM at 6000h-7FFFh
N/A          Fixed 32K ROM at 8000h-FFFFh (always last 32K)
```

However, it is doing some odd initialization, writing to Port E000h (index, 1-5 used), and Port F000h (data for index 1-5, looks like 8K ROM banks at 8000h, A000h, C000h, 6000h, E000h). Note that: A) the last 32K seem to be always mapped to 8000h-FFFFh. B) each final write to F000h seems to be done with index 4. If that behaviour doesn't change later on in the game, then Port F000h (or even E000h) could be interpreted exactly as for Mario Baby.

Mapper 43: X-in-1

Used by 150-in-1 (a fake containing 56 games, plus some cheat modes, chip 0 is 1024K, chip 1 is 512K, chip 2-3 are not installed).

```
8000h-FFFFh  Memory Control (Write any data, port decoded by address lines)
  A7-A0  Select 32K ROM Bank (From currently selected Chip)
  A9-A8  Select ROM Chip     (Empty bus if selected chip not installed)
  A10    Not used            (Always zero)
  A11    Bank Mode           (0=32K, 1=16K; Lower/Upper half via A12)
  A12    Select 16K ROM Bank (0=Lower, 1=Upper) (Should be zero in 32K mode)
  A13    Mirroring           (0=Vertical, 1=Horizontal Mirroring)
  A14    Not used            (Always zero)
```

Initially 1st 32K selected. The cartridge includes 8K VRAM.

Mapper 44: 7-in-1 MMC3 Port A001h

Used by Super Big 7-in-1.

```
A001h - Select 128K ROM/VROM Block (0..5) or last 256K ROM/VROM Block (6)
```

Block 0 seems to be required on power-up.

The rest of both mappers is same as MMC3 (for the selected block of memory),
Mapper 4: MMC3 - PRG/8K, VROM/2K/1K, NT, SRAM, IRQ

Mapper 45: X-in-1 MMC3 Port 6000hx4

Used in Super 3-in-1, Super 4-in-1, Super 8-in-1, Hero 8-in-1, Super 13-in-1, and 1000000-in-1 (a fake with 1000000 duplicated/nonsense titles).
Configuration value initialized by each FOUR writes to Port 6000h.

```
6000h 1st write - Configuration Bits 0-7
6000h 2nd write - Configuration Bits 8-15
6000h 3nd write - Configuration Bits 16-23
6000h 4th write - Configuration Bits 24-31
```

The meaning of the 32 configuration bits is:

```
Bit7-0   VROM base in 1K steps
Bit15-8  ROM base in 8K steps
Bit19-16 VROM mask in 1K steps, Mask=(2 SHL (X AND 7))+(X AND 8)/8
Bit23-20 VROM base in 256K steps
Bit29-24 ROM mask in 8K steps, Mask=(3Fh AND (NOT X))
Bit30    LOCK (set when menu selection completed, probably locks Port 6000h)
Bit31    ???
```

Memory selections are Bank=((Mmc3Bank AND ConfigMask) OR ConfigBase). For MMC3 Registers 0 and 1 (map 2x1K banks), above formula applies to both banks, ie. VROM Mask=0 maps same 1K bank twice (instead two continous 1K banks).
128K Block 3 seems to be required on power-up, ie. the entry point is at the end of 512K ROMs, or in the middle of 1024K ROMs. The MMC3 Port A001h apparently can write-protect Port 6000h (just like it disables SRAM at 6000h-7FFFh). At least some carts have at least 2K SRAM (Mario 3 in Super 8-in-1 uses RAM at 7800h-7FFFh).

The rest of the mapper is same as MMC3 (for the selected block of memory),
Mapper 4: MMC3 - PRG/8K, VROM/2K/1K, NT, SRAM, IRQ

Mapper 46: 15-in-1 Color Dreams

Used by Rumble Station 15-in-1, which contains 15 Color Dreams games.

```
6000h-7FFFh  Multicart Memory Control
  Bit0-3 Select 64K ROM Block (initially 1st bank) (always same as below)
  Bit4-7 Select 64K VROM Block (initially 1st bank) (always same as above)
8000h-FFFFh  Memory Control (selection within current 64K block)
  Bit1   Select 32K ROM bank at 8000h-FFFFh (initially 1st bank)
  Bit6-4 Select 8K VROM bank at PPU 0000h-1FFFh (initially 1st bank)
```

Port 8000h-FFFFh is same as Mapper 11, except that it is limited to 64K, and except that it doesn't seem to have bus-conflicts (the menu always uses Port 8888h, regardless of underlaying ROM content).
In some games ROM/VROM is less than 64K, so some memory areas are unused.
Mapper 11: Color Dreams - PRG/32K, VROM/8K

Note: The Rumble Station is not a cartridge - it is a complete NES with built-in games, assembled in a "wing-shaped" gamepad case (ie. it consists of a CPU, PPU, APU, ROM, mapper, and joypad).

There's also a "Rumble Station Mark 2" - claiming to contain 29 games - actually containing only 16 games - which may use different mapper?

Mapper 47: 2-in-1 MMC3 Port 6000h

Used by 2-in-1 cart "Super Spike V'Ball + Nintendo World Cup".

```
6000h  Select 1st or 2nd half of ROM/VROM (0 or 1)
```

The rest of the mapper is same as MMC3 (for the selected block of memory),
Mapper 4: MMC3 - PRG/8K, VROM/2K/1K, NT, SRAM, IRQ
The cartridge doesn't have SRAM, but the MMC3 Port A001h can apparently write-protect Port 6000h (just like it could disable SRAM at 6000h-7FFFh).

Mapper 48: Taito TC190V

Reportedly "Tatio TC190V" used by "FlintStone".
Sounds like the Type II (or Type I?) variant of Mapper 33:
Mapper 33: Taito TC0190/TC0350 - PRG/8K, VROM/1K/2K, NT, IRQ

Mapper 49: 4-in-1 MMC3 Port 6xxxh

Used by Super HIK 4-in-1.
ROM/VROM are split into 128K blocks each, done as such:

```
[6000h]=01h ;init
[6800h]=00h ;game 0 + [6808h]=08h crashes ?
[6841h]=41h ;game 1
[6881h]=81h ;game 2
[68C1h]=C1h ;game 3
```

The rest of the mapper is same as MMC3 (for the selected block of memory),
Mapper 4: MMC3 - PRG/8K, VROM/2K/1K, NT, SRAM, IRQ
The cartridge doesn't have SRAM, but the MMC3 Port A001h apparently can write-protect Ports at 6000h-7FFFh (just like it could disable SRAM at 6000h-7FFFh).

Mapper 50: FDS-Port - Alt. Levels

Used by Super Mario Bros 2 - Alt. Levels.

```
4022h  Select 8K ROM at C000h-DFFFh
         Bit0 and/or Bit3 ZERO  Bank 0
         Bit0 and/or Bit3 SET   Bank 0Ch (or maybe INCREMENT bank number?)
         Other Bits             Unknown
4122h  IRQ Control
         Bit0 and/or Bit1 ZERO  Disable/Acknowledge
         Bit0 and/or Bit1 SET   Enable/Start
N/A    Fixed 8K ROM at 6000h-7FFFh (always bank 0Fh, ie. last bank)
N/A    Fixed 8K ROM at 8000h-9FFFh (always bank 08h)
N/A    Fixed 8K ROM at A000h-BFFFh (always bank 09h)
N/A    Fixed 8K ROM at E000h-FFFFh (always bank 0Bh)
```

When enabled, IRQ generated after 4096 clock cycles (about 36 scanlines).

Uses different ports, but the features are about same as:
Mapper 40: FDS-Port - Lost Levels

Mapper 51: 11-in-1

Used by 11-in-1, containing ball games, like "soccer ball", and "golf ball".

```
6000h  Mode Register
  Bit1  ROM Block Size (0=128K Mode, 1=32K Mode)
  Bit4  Unknown
8000h  Base Address in 32K Steps (X) (0-0Fh)
  32K Mode:  Select 32K Bank (X) at 8000h-FFFFh (initially 1st 32K bank)
  128K Mode: Select 16K Bank (X*2 OR 07h) at C000h-FFFFh
  And:       Select  8K Bank (X*4 OR 23h) at 6000h-7FFFh (for FDS ports)
C000h  Lower 16K Select (Y) (0-1Fh) (128K Mode only, UNROM-style)
  128K Mode: Select 16K Bank (Y*2 OR Y/10h) at 8000h-BFFFh
```

The cartridge does have 8K VRAM, even though there's a ROM-image around in which somebody has incorrectly included a copy of above 8K VRAM as "8K VROM".

Mapper 52: 7-in-1 MMC3 Port 6800h with SRAM

Used by Mario Party 7-in-1 (or short, Mari7in1).
"It's MMC3 and an extra bank control register. There is 1Mbyte of PRG ROM and 1Mbyte of CHR ROM on this cart. Interestingly, all the games appear to be NTSC, except SMB2. For some reason, this is the PAL version! It consists of 1 6264 8K RAM chip (for WRAM), and 3 glop-tops. 2 are 1Mbyte ROMs while the remaining chip is the mapper."

```
6800h  Bank Control Byte
  Bit7      Not used
  Bit6      VROM Bank Size    (0=256K, 1=128K)
  Bit5,2,4  VROM 128K Bank    (Bit4 not used in 256K CHR mode)
  Bit3      PRG ROM Bank Size (0=256K, 1=128K)
  Bit2,1,0  PRG ROM 128K Bank (Bit0 not used in 256K PRG mode)
```

Note: Bit2 is both MOST significant PRG bit, and MIDDLE significant CHR bit.
After a reset, this register is 00h. It can only be written once.
To reset it and allow another write, you must reset the console.

Mapper 4: MMC3 - PRG/8K, VROM/2K/1K, NT, SRAM, IRQ

Mapper 56: Pirate SMB3

Used only by a pirate copy of Super Mario Bros. 3.

```
8000h  Unknown (always 08h) (maybe counter LSBs, if any)
9000h  Bit7-4 of 16bit IRQ counter
A000h  Bit11-8 of 16bit IRQ counter
B000h  Bit15-12 of 16bit IRQ counter
C000h  IRQ Anable      (FFh=Enable, 00h=Disable)
D000h  IRQ Acknowledge (Always write FFh, or EFh)
E000h  Ignore - MMC3-index (Port 8000h) relicts redirected to E000h
F000h  Select 8K ROM at 8000h-9FFFh
F001h  Select 8K ROM at A000h-BFFFh
F002h  Select 8K ROM at C000h-DFFFh
N/A    Fixed  8K ROM at E000h-FFFFh (always last bank)
F003h  Unknown (always 10h)
F400h  Select 1K VROM at PPU 0000h-03FFh
F401h  Select 1K VROM at PPU 0400h-07FFh
F402h  Select 1K VROM at PPU 0800h-0BFFh
F403h  Select 1K VROM at PPU 0C00h-0FFFh
F404h  Select 1K VROM at PPU 1000h-13FFh
F405h  Select 1K VROM at PPU 1400h-17FFh
F406h  Select 1K VROM at PPU 1800h-1BFFh
F407h  Select 1K VROM at PPU 1C00h-1FFFh
```

The cartridge often uses silly mirrors like C5A7h, mask 8000h-EFFFh by F000h, and F000h-FFFFh by F407h. The IRQ counter is incremented every clock cycle.

Mapper 57: 6-in-1

Used by Game Star 6-in-1 GK-L01A, 6-in-1 GK-L02A, and 54-in-1 GK-54.

```
8000h  Extra Port for CNROM Games in 2nd 64K of VROM
  Bit2-0 Select 8K VROM at PPU 0000h-1FFFh (ORed with value in Port 8800h)
  Bit5-3 Not used  (zero)
  Bit6   Must be set for Second 64K Block of VROM
  Bit7   Must be set for First 64K Block of VROM
8800h  Main Port
  Bit2-0 Select 8K VROM at PPU 0000h-1FFFh (ORed with value in Port 8000h)
  Bit3   Mirroring (0=Vertical, 1=Horizontal Mirroring)
  Bit4   ROM Size  (0=16K; Bank X twice, 1=32K; Bank X and X+1)
  Bit7-5 Select 16K ROM at 8000h-BFFFh and C000h-FFFFh (X)
```

All carts have 128K ROM and 128K VROM and contain 6 games (even "54-in-1").

Mapper 58: X-in-1

Used by 68-in-1 (a fake containing only 8 games).

```
C000h-FFFFh  Memory Control (Write any data, port decoded by address lines)
  A2-A0  Select 16K ROM Bank at 8000h-BFFFh and C000h-FFFFh (X)
  A5-A3  Select 8K VROM Bank at PPU 0000h-1FFFh
  A6     ROM Size  (0=32K; Bank X and X+1, 1=16K; Bank X twice)
  A7     Mirroring (0=Vertical, 1=Horizontal Mirroring)
  A12-A8 Unknown   (Usually 0,/A6,0,/A6,A5,A3, except in yie-ar-kung-fu)
```

Note: Study and Game 32-in-1 declared as "Mapper 58" should be Mapper 241,
Mapper 241: X-in-1 Education

Mapper 61: 20-in-1

Used by 20-in-1.

```
8000h-FFFFh  Memory Control (Write any data, port decoded by address lines)
  A3-0 Select 32K ROM Bank at 8000h-FFFFh
  A4   Bank Size (0=32K, 1=16K; only lower/upper half via Bit5)
  A5   Select lower/upper half of selected 32K bank (in 16K mode)
  A7   Mirroring (0=Vertical, 1=Horizontal Mirroring)
  A6,A8-A14   Not used (always 0)
```

There's also a version of the same cartridge with slightly different mapper:
Mapper 231: 20-in-1

Mapper 62: X-in-1

Used by 700-in-1 (a fake containing only somewhat 50-100 games).

```
8000h-BFFFh  Memory Control (Decoded by address AND data lines)
  A4-A0,D1-D0  Select 8K VROM at PPU 0000h-1FFFh
  A5           ROM Size  (0=32K; Bank X-1 and X, 1=16K; Bank X twice)
  A7           Mirroring (0=Vertical, 1=Horizontal Mirroring)
  A6,A13-A8    Select 16K ROM at 8000h-BFFFh and C000h-FFFFh (X)
  A14          Always 0 ?
```

ROM Bank lower bit (A8) should be always SET in 32K Mode. Initially 1st 32K.

Mapper 64: Tengen RAMBO-1 - PRG/8K, VROM/2K/1K, NT, IRQ

Used by Shinobi, Klax, and Skull & Crossbones.

```
8000h  Index/Control (6bit)
       Bit7   CHR Address Select (0=Normal, 1=Address Areas XOR 1000h)
       Bit6   PRG Address Select (0=Normal, 1=Address Areas plus 2000h)
       Bit5   CHR Bank Size      (0=Normal, 1=Use 1K-resolution instead 2x1K)
       Bit3-0 Command Number (Note: Index 0-7 same as for MMC3)
         0 - Select 2x1K VROM at PPU 0000h-07FFh (or 1000h-17FFh, if Bit7=1)
         1 - Select 2x1K VROM at PPU 0800h-0FFFh (or 1800h-1FFFh, if Bit7=1)
         2 - Select 1K VROM at PPU 1000h-13FFh   (or 0000h-03FFh, if Bit7=1)
         3 - Select 1K VROM at PPU 1400h-17FFh   (or 0400h-07FFh, if Bit7=1)
         4 - Select 1K VROM at PPU 1800h-1BFFh   (or 0800h-0BFFh, if Bit7=1)
         5 - Select 1K VROM at PPU 1C00h-1FFFh   (or 0C00h-0FFFh, if Bit7=1)
         6 - Select 8K ROM at 8000h-9FFFh (or A000h-BFFFh, if Bit6=1)
         7 - Select 8K ROM at A000h-BFFFh (or C000h-DFFFh, if Bit6=1)
         F - Select 8K ROM at C000h-DFFFh (or 8000h-9FFFh, if Bit6=1)
       N/A - Fixed  8K ROM at E000h-FFFFh (always last 8K bank)
         8 - Select 1K VROM page at PPU 0400h (used only if Bit5=1)
         9 - Select 1K VROM page at PPU 0C00h (used only if Bit5=1)
8001h  Data Register (indexed via Port 8000h)
A000h  Mirroring Select (Bit0: 0=Vertical, 1=Horizontal Mirroring)
A001h  Not used
C000h  IRQ Reload Value (for decrementing Scanline-/clock-cycle-counter)
C001h  IRQ Clock Source (Bit0: 0=Scanline Counter, 1=System Clock div 4)
E000h  IRQ Disable/Acknowledge (write any value)
E001h  IRQ Enable              (write any value)
```

Write to C000h: After N+1 clocks have occured, the IRQ flag will be set. N is the value written to C000h. Writing a value of 00h will cause NO interrupts to be generated at all. (Note: if C001h is written to, then a value of 00h can be used- it will cause an interrupt to be generated in 2 clocks after C001h is written to. No more interrupts will be generated unless C001h is written to again).
Write to C001h: Will trigger a reload of the IRQ counter (and select the desired clock source). One thing of note, after writing, the IRQ counter will expire after a count of N+2 instead of N+1. In this case, a value of 00h in C000h WILL generate an interrupt after 2 clocks. A value of FFh likewise generates an interrupt after 257d clocks.

Mapper 65: Irem H-3001 - PRG/8K, VROM/1K, NT, IRQ

Used by Daiku no Gensan 2, Kaiketsu Yanchamaru 3, and Spartan X 2.
Note: Ai Sensei no Oshiete declared as "Mapper 65" is maybe a Konami mapper?

```
9000h  Unknown
9001h  Unknown
9003h,9004h  IRQ Control (not sure about difference between 9003h/9004h)
 (00h=Disable IRQ, C0h=Enable IRQ, other values unknown)
9005h  IRQ Counter MSB of decrementing 16bit counter
9006h  IRQ Counter LSB of decrementing 16bit counter
B000h  Select 1K VROM bank at PPU 0000h-03FFh
B001h  Select 1K VROM bank at PPU 0400h-07FFh
B002h  Select 1K VROM bank at PPU 0800h-0BFFh
B003h  Select 1K VROM bank at PPU 0C00h-0FFFh
```

```
B004h  Select 1K VROM bank at PPU 1000h-13FFh
B005h  Select 1K VROM bank at PPU 1400h-17FFh
B006h  Select 1K VROM bank at PPU 1800h-1BFFh
B007h  Select 1K VROM bank at PPU 1C00h-1FFFh
8000h  Select 8K ROM bank at 8000h-9FFFh (initially 1st half of 1st 16K)
A000h  Select 8K ROM bank at A000h-BFFFh (initially 2nd half of 1st 16K)
C000h  Select 8K ROM bank at C000h-DFFFh (initially 1st half of last 16K)
N/A    Fixed  8K ROM bank at E000h-FFFFh (always 2nd half of last 16K)
```

Mapper 66: GNROM - PRG/32K, VROM/8K

This mapper is used on several Japanese titles, such as Dragon Ball, and on U.S. titles such as Gumshoe and Dragon Power.

```
8000h-FFFFh  Memory Control (2x2bits)
  Bit1-0 Select 8K VROM bank at PPU 0000h-1FFFh (initially 1st bank)
  Bit5-4 Select 32K ROM bank at 8000h-FFFFh (initially 1st bank)
```

This mapper is used on the DragonBall (NOT DragonBallZ) NES game.

Mapper 67: Sunsoft3 - PRG/16K, VROM/2K, IRQ

Used by Fantasy Zone 2.

```
8000h        IRQ Acknowledge (write any data to this address)
8800h-8FFFh  Select 2K VROM bank at PPU 0000h-07FFh
9800h-9FFFh  Select 2K VROM bank at PPU 0800h-0FFFh
A800h-AFFFh  Select 2K VROM bank at PPU 1000h-17FFh
B800h-BFFFh  Select 2K VROM bank at PPU 1800h-1FFFh
C800h-CFFFh  IRQ Counter (two writes: 1st=MSB, 2nd=LSB)
             (16bit decrementing clock cycle counter)
D800h-DFFFh  IRQ Control (Bit4: 0=Disable, 1=Enable)
E800h-EFFFh  No info - maybe Mirroring control ?
F800h-FFFFh  Select 16K ROM bank at 8000h-BFFFh (initially 1st bank)
N/A          Fixed  16K ROM bank at C000h-FFFFh (always last bank)
```

Mapper 68: Sunsoft4 - PRG/16K, VROM/2K, NT-VROM

This mapper is used by After Burner I and II, and by Maharaja.

```
8000h  Select 2K VROM bank at PPU 0000h-07FFh
9000h  Select 2K VROM bank at PPU 0800h-0FFFh
A000h  Select 2K VROM bank at PPU 1000h-17FFh
B000h  Select 2K VROM bank at PPU 1800h-1FFFh
C000h  Select 1K VROM bank as BLK0 (in VROM Mode) (from LAST 128 banks)
D000h  Select 1K VROM bank as BLK1 (in VROM Mode) (from LAST 128 banks)
E000h  Name Table Control
        Bit4  Name Table VROM Mode  (0=VRAM, 1=VROM via Port C000h/D000h)
        Bit0  Name Table Mirroring  (0=Horizontal, 1=Vertical Mirroring)
F000h  Select 16K ROM bank at 8000h-BFFFh (initially 1st bank)
N/A    Fixed  16K ROM bank at C000h-FFFFh (always last bank)
```

Mapper 69: Sunsoft5 FME-7 - PRG/8K, VROM/1K, NT ctrl, SRAM, IRQ

This mapper is used by Batman, Hebereke, Pyokotan no Da Meiro, Barcode World, Gremlin 2, Honoo no Doukyuuji 1 and 2, and Gimmick.

```
8000h  Index Register (4bit)
         0 - Select 1K VROM at PPU 0000h-03FFh
         1 - Select 1K VROM at PPU 0400h-07FFh
         2 - Select 1K VROM at PPU 0800h-0BFFh
         3 - Select 1K VROM at PPU 0C00h-0FFFh
         4 - Select 1K VROM at PPU 1000h-13FFh
         5 - Select 1K VROM at PPU 1400h-17FFh
         6 - Select 1K VROM at PPU 1800h-1BFFh
         7 - Select 1K VROM at PPU 1C00h-1FFFh
         8 - Select 8K ROM/RAM at 6000h-7FFFh
              Bit6=0        --> Select 8K ROM (Page number in bit5-0)
              Bit6=1,Bit7=1 --> Select 8K SRAM
              Bit6=1,Bit7=0 --> Select 8K "pseudo-random numbers?"
         9 - Select 8K ROM at 8000h-9FFFh
         A - Select 8K ROM at A000h-BFFFh
         B - Select 8K ROM at C000h-DFFFh
         C - Select Mirroring
              0  Two-Screen, Vertical Mirroring
              1  Two-Screen, Horizontal Mirroring
              2  One-Screen, BLK0
              3  One-Screen, BLK1
         D - IRQ control (00h=Disable, 81h=Enable, other values?)
         E - IRQ LSB of decrementing clock cycle counter
```

```
          F - IRQ MSB of decrementing clock cycle counter
        N/A - Fixed  8K ROM at E000h-FFFFh (always last 8K bank)
  A000h  Data Register (indexed via Port 8000h)
```

Mapper 70: Bandai - PRG/16K, VROM/8K, NT

Used by Taito's Arkanoid 2, and various Bandai games: Space Shadow, Kamen Rider Club, Saint Seiya, Pocket Zaurus, Gegege no Kitarou 2, and two Family Trainer games.

```
  C000h-C0FFh  Memory Control
   Bit7    Name Table Select (0/1 = BLK0/BLK1) (One-Screen Mode only)
   Bit6-4  Select 16K ROM at 8000h-BFFFh
   Bit3-0  Select 8K VROM at PPU 0000h-1FFFh
```

Bus-Conflicts, memory at C000h-C0FFh should be filled by 00h-FFh.

Mapper 71: Camerica - PRG/16K

This mapper is used on Camerica's unlicensed NES carts, including Firehawk and Linus Spacehead.

```
  8000h-BFFFh  Unknown
  C000h-FFFFh  Select 16K ROM bank at 8000h-BFFFh (initially 1st bank)
  N/A          Fixed  16K ROM bank at C000h-FFFFh (always last bank)
```

All carts using it have 8K of VRAM at PPU 0000h-1FFFh.
Many ROMs from these games are incorrectly defined as mapper 2.

Mapper 72: Jaleco Early Mapper 0 - PRG-LO, VROM/8K

Used by Pinball Quest, Pro Tennis, Pro Judo.

```
  8000h-FFFFh  Memory Control
    Bit7-6  Function Select
     0  Confirm Selection
     1  Select 8K VROM bank at PPU 0000h-1FFFh
     2  Select 16K ROM bank at 8000h-BFFFh (lower half of PRG memory)
     3  Reserved (would probably select both PRG+VROM)
    Bit5-4  Not used
    Bit0-3  ROM or VROM Bank Number for above Selection
```

Bus-conflicts. Example: To select PRG Bank 7, first write 87h, then 07h.
Same as Mapper 92, except that this one maps the LOWER half of PRG memory.

Not sure if that makes sense, but it appears to consist of three latches, latch1 directly accessed from CPU, the other latches loaded from the latch1 on high-to-low transitions in bit6/7.

Mapper 73: Konami VRC3 - PRG/16K, IRQ

Mapper 21-26,73,75,85: Konami VRC Mappers

Mapper 74: Whatever MMC3-style

Reportedly "Taiwan MMC3 -Varient Mapper#0" used by "KidNiKi3J(hacked)". Personally, I have a ROM-image named "Ji Jia Zhan Shi", which may or may not be same as "KidNiKi3J(hacked)".

Anyways, that ROM-image seems to be basically MMC3-compatible,
Mapper 4: MMC3 - PRG/8K, VROM/2K/1K, NT, SRAM, IRQ
except that it gets wrong anytime when selecting ROM bank number 14h via Register 7, don't know what is/should be happenening there (?)

Mapper 75: Jaleco SS8805/Konami VRC1 - PRG/8K, VROM/4K, NT

Mapper 21-26,73,75,85: Konami VRC Mappers

Mapper 76: Namco 109 - PRG/8K, VROM/2K

Used by Digital Devil / Megami Tensei.

```
  8000h  Index/Control (3bit)
```

```
        Bit2-0 Command Number
          0 - Not used
          1 - Not used
          2 - Select 2K VROM at PPU 0000h-07FFh
          3 - Select 2K VROM at PPU 0800h-0FFFh
          4 - Select 2K VROM at PPU 1000h-17FFh
          5 - Select 2K VROM at PPU 1800h-1FFFh
          6 - Select 8K ROM at 8000h-9FFFh
          7 - Select 8K ROM at A000h-BFFFh
        N/A - Fixed 16K ROM at C000h-FFFFh (always last bank)
  8001h  Data Register (Indexed via Port 8000h)
```

Mapper 77: Irem - PRG/32K, VROM/2K, VRAM 6K+2K

Used by Napoleon Senki.

```
  8000h-FFFFh  Memory Control
    Bit0-1  Select 32K ROM bank at 8000h-FFFFh
    Bit2-3  Not used
    Bit4-7  Select 2K VROM bank at PPU 0000h-07FFh
  6K VRAM at PPU 0800h-1FFFh (ie. upper 6K of Pattern Tables are VRAM)
  2K VRAM at PPU 2800h-2FFFh (ie. uses Four-Screen Name Tables)
```

Bus-conflicts.

Mapper 78: Irem 74HC161/32 - PRG/16K, VROM/8K

Used by Holy Diver, and Cosmo Carrier (Uchuusen). The two games seem to be using different/incompatible mapper circuits for name table control?

```
  8000h-FFFFh  Memory Control
    Bit2-0 Select 16K ROM bank at 8000h-BFFFh (initially 1st bank)
    N/A    Fixed  16K ROM bank at C000h-FFFFh (always last bank)
    Bit3   Name Table Control
           Jaleco/Cosmo Carrier: One-Screen (0=BLK0, 1=BLK1)
           Irem/Holy Diver: Two-Screen (0=Horizontal, 1=Vertical Mirroring)
    Bit7-4 Select 8K VROM bank at PPU 0000h-1FFFh
```

Mapper 79: AVE Nina-3 - VROM/8K

[See also]

Mapper 113: Sachen/Hacker/Nina

Made by American Video Entertainment (AVE), used by Krazy Kreatures, Double Strike, etc.

```
  4100h  Bit1-0 Select 8K VROM bank at PPU 0000h-1FFFh
```

Port decoded as A14=A8=HIGH, A15=A13=LOW, ie. with mirrors at 4100h-41FFh, 4300h-43FFh ... 5F00h-5FFFh. Contains 74LS175, 74LS138, and "Nina"-lockout chip.

Mapper 80: Taito X-005 - PRG/8K, VROM/2K/1K, NT

Used by Fudou Myouou Den (Demon Sword), Kyonshiizu 2, Mirai Shinwa Jarvas, Taito Grand Prix - Eikou heno License, Yamamura Misa Suspense - Kyouto Ryuu no Tera Satsujin, Minelvaton Saga.

```
  7EF0h  Select 2x1K VROM at PPU 0000h-07FFh (Bit7: Name Table, see below)
  7EF1h  Select 2x1K VROM at PPU 0800h-0FFFh (Bit7: Name Table, see below)
  7EF2h  Select 1K VROM at PPU 1000h-13FFh
  7EF3h  Select 1K VROM at PPU 1400h-17FFh
  7EF4h  Select 1K VROM at PPU 1800h-1BFFh
  7EF5h  Select 1K VROM at PPU 1C00h-1FFFh
  7EF6h  Unknown     (usually FFh, 01h, or 00h)
  7EF8h  SRAM Enable (A3h=Enable, FFh=Disable)
  7EFAh  Select 8K ROM 8000h-9FFFh
  7EFCh  Select 8K ROM A000h-BFFFh
  7EFEh  Select 8K ROM C000h-DFFFh
  N/A    Fixed  8K ROM E000h-FFFFh (always last bank)
  7EF7h,7EF9h        Not used
  7EFBh,7EFDh,7EFFh  Dupes of 7EFAh,7EFCh,7EFEh used by Kyonshiizu 2 only
  7F00h-7FFFh        SRAM Area (seems to be only 256 bytes or less used)
```

Bit7 of 7EF0h and Bit7 of 7EF1h are somehow related to Name Tables: Most carts use Vertical Mirroring, these carts always have both bits cleared. Demon Sword uses One-Screen mode, either both bits cleared (BLK0), or both bits set (BLK1).

Mapper 81: AVE Nina-6

Made by American Video Entertainment (AVE), used by Deathbots, Mermaids of Atlantis, etc.

No info. For Deathbots, see:
Mapper 113: Sachen/Hacker/Nina

Also, presumably mis-numbered "Mapper 81 - Taito C075"
Reportedly "Tatio C075" used by "(many Japanese title from tatio)".
Don't have a ROM-image that is assigned as Mapper 81. Don't know which titles are using it. Maybe meant to be Taito's Arkanoid 2, ie. this mapper:
Mapper 70: Bandai - PRG/16K, VROM/8K, NT

Mapper 82: Taito X1-17 - PRG/8K, VROM/2K/1K

Used by SD Keiji Blader, and Kyuukyoku Harikiri 1, 2, 3.

```
7EF0h  Select 2x1K VROM at PPU 0000h-07FFh (or 1000h-17FFh if swapped)
7EF1h  Select 2x1K VROM at PPU 0800h-0FFFh (or 1800h-1FFFh if swapped)
7EF2h  Select 1K VROM at PPU 1000h-13FFh   (or 0000h-03FFh if swapped)
7EF3h  Select 1K VROM at PPU 1400h-17FFh   (or 0400h-07FFh if swapped)
7EF4h  Select 1K VROM at PPU 1800h-1BFFh   (or 0800h-0BFFh if swapped)
7EF5h  Select 1K VROM at PPU 1C00h-1FFFh   (or 0C00h-0FFFh if swapped)
7EF6h  Swap PPU 0000h-0FFFh / 1000h-1FFFh  (Bit1: 0=Normal, 1=Swap)
7EF7h  SRAM .... CAh,00h,01h,40h
7EF8h  SRAM .... 69h,00h,40h
7EF9h  SRAM .... 84h,00h,40h
7EFAh  Select 8K ROM 8000h-9FFFh (Bit7-2)
7EFBh  Select 8K ROM A000h-BFFFh (Bit7-2)
7EFCh  Select 8K ROM C000h-DFFFh (Bit7-2)
N/A    Fixed  8K ROM E000h-FFFFh (unknown?)
7EFDh  SRAM .... FFh
7EFEh  SRAM .... FFh,07h
7EFFh  SRAM .... FFh
6000h-7FFFh  SRAM Area (probably 8K size, at least 6000h-73xxh used)
```

Mapper 83: Cony

There are different Cony variants for cartridges of different size:

```
Cony (A) 128K+256K Fatal Fury 2
Cony (B) 256K+512K World Heroes 2
Cony (C) 4x256K+4x256K Dragon Ball Z 4-in-1
Also used by Garou Densetsu Special?
```

Cony (A) and (C) only - 8bit bank numbers (256x1K=256K):

```
8310h  Select 1K VROM at PPU 0000h-03FFh (in current 256K block)
8311h  Select 1K VROM at PPU 0400h-07FFh (in current 256K block)
8312h  Select 1K VROM at PPU 0800h-0BFFh (in current 256K block)
8313h  Select 1K VROM at PPU 0C00h-0FFFh (in current 256K block)
8314h  Select 1K VROM at PPU 1000h-13FFh (in current 256K block)
8315h  Select 1K VROM at PPU 1400h-17FFh (in current 256K block)
8316h  Select 1K VROM at PPU 1800h-1BFFh (in current 256K block)
8317h  Select 1K VROM at PPU 1C00h-1FFFh (in current 256K block)
```

Cony (B) only - 8bit bank numbers (256x2K=512K):

```
8310h  Select 2K VROM at PPU 0000h-07FFh
8311h  Select 2K VROM at PPU 0800h-0FFFh
8316h  Select 2K VROM at PPU 1000h-17FFh
8317h  Select 2K VROM at PPU 1800h-1FFFh
```

Cony (A) only - 4bit bank numbers (16x8K=128K):

```
8300h  Select 8K ROM at 8000h-9FFFh
8301h  Select 8K ROM at A000h-BFFFh
8302h  Select 8K ROM at C000h-DFFFh
N/A    Fixed  8K ROM at E000h-FFFFh (always last bank)
```

Cony (B) and (C) only - 4bit bank numbers (16x16K=256K):

```
8000h  Select 16K ROM at 8000h-BFFFh (in current 256K block)
N/A    Fixed  16K ROM at C000h-FFFFh (last bank in current 256K block)
```

Cony (A) and (B) and (C):

```
8200h  IRQ Counter LSB, writing to this address acknowledges IRQs
8201h  IRQ Counter MSB, writing to this address starts counting
```

Cony (A) and (B) only:

```
8100h  IRQ Control (Bit7=Enable IRQs) (other bits unknown) Bit7, unlike C
5000h  Unknown, program reads from this address
```

Cony (C) only:

```
8100h  IRQ Control (Bit1=Enable IRQs) (other bits unknown) Bit1, unlike A/B
B000h  Select 256K ROM/VROM Windows (upper two address bits)
  Bit0-3  Unknown
  Bit4,6  Bit0 of 256K Block Number
  Bit5,7  Bit1 of 256K Block Number
  Used values are 00h,50h,A0h,F0h. Other values could probably select
  separate 256K banks for ROM/VROM. The ROM selection also affects
  the "fixed" 16K at C000h-FFFFh (last bank in current 256K block).
B0FFh  Probably same as B000h
```

```
B1FFh  Probably same as B000h
510Xh  Unknown, program reads/writes to/from this address
430Xh  Unknown, program reads from this address
```

Mapper 84: Whatever

No info. Reportedly "PC-SMB2J" used by "SMBJ2".
Don't have a ROM-image, unless it is meant to be same as Mapper 40 or 50:
Mapper 40: FDS-Port - Lost Levels
Mapper 50: FDS-Port - Alt. Levels

Mapper 85: Konami VRC7A/B - PRG/16K/8K, VROM/1K, NT, IRQ, SOUND

VRC7 General Memory and IRQ Registers
Mapper 21-26,73,75,85: Konami VRC Mappers

VRC7 Sound Registers
The sound generation is done using FM synthesis, so the music sounds like "Adlib" OPL2 music. All sound registers are accessed through only two physical registers.

```
9010h (aka 9.1.0)  Index register
9030h (aka 9.1.1)  Data Register
```

There are 6 channels, each containing three registers, and 8 custom instrument control registers.

Index 10h-15h - Channel 0-5, Frequency LSB

```
Bit7-0  Lower 8bit of 9bit Frequency (f; 0-1FFh)
```

Index 20h-25h - Channel 0-5, Frequency MSB, Octave, Trigger

```
Bit7-5  Unknown
Bit4    Channel trigger
Bit3-1  Octave Select                (o; 0-7)
Bit0    Upper 1bit of 9bit Frequency (f; 0-1FFh)
```

Frequency is calculated as: F = 49722Hz * f / 2^(19-o)

Index 30h-35h - Channel 0-5, Instrument and Volume

```
Bit7-4 Instrument number (0=Custom, 1-0Fh=Fixed Instruments)
Bit3-0 Volume
```

Index 00h-07h - Custom Instrument (Instrument 0)
Note: I will not provide too extensive documentation of the instrument registers since their functions are identical to those of the OPL2 chip, commonly found on Adlib/Soundblaster/compatible cards, and there is alot of information out on how to program these. I will use terminology similar to that found in said documents. My VRC7 "emulator" test program I wrote simply re-arranged and tweaked the register writes to correspond with the OPL2 registers.

```
Here's a link to a good document about this chip:
http://www.ccms.net/~aomit/oplx/
```

The tremolo depth is set to 4.3db and the vibrato depth is set to 14 cent (in reguards to OPL2 settings; to achieve this you would write 0C0h to OPL register 0BDh). All operator connections are fixed in FM mode. (Where Modulator modulates the Carrier).

Index 00h (Modulator)
Index 01h (Carrier)

```
Bit7   Tremolo Enable
Bit6   Vibrato Enable
Bit5   Sustain Enable
Bit4   KSR
Bit3-0 Multiplier
```

Index 02h

```
Bit7-6 Key Scale Level
Bit5-0 Output Level
```

Index 03h

```
Bit7-5 Not used (Write 0's)
Bit4   Carrier Waveform
Bit3   Modulator Waveform
    There are only two waveforms available.  Sine and rectified sine (only
    the positive cycle of the sine; negative cycle "chopped off".)
```

```
  Bit2-0 Feedback Control
```

Index 04h (Modulator)
Index 05h (Carrier)
```
  Bit7-4 Attack
  Bit3-0 Decay
```

Index 06h (Modulator)
Index 07h (Carrier)
```
  Bit7-4 Sustain
  Bit3-0 Release
```

Register Settings for the 15 fixed instruments
These instruments are not 100% correct! There is no way to extract the register settings from the chip short of an electron microscope.
I have "tuned" these instruments best I could, though I know a couple
are not exactly right.
Table shows Register 0-7 settings for Instrument 1-0Fh
```
  1 - 05 03 10 06 74 A1 13 F4
  2 - 05 01 16 00 F9 A2 15 F5
  3 - 01 41 11 00 A0 A0 83 95
  4 - 01 41 17 00 60 F0 83 95
  5 - 24 41 1F 00 50 B0 94 94
  6 - 05 01 0B 04 65 A0 54 95
  7 - 11 41 0E 04 70 C7 13 10
  8 - 02 44 16 06 E0 E0 31 35
  9 - 48 22 22 07 50 A1 A5 F4
  A - 05 A1 18 00 A2 A2 F5 F5
  B - 07 81 2B 05 A5 A5 03 03
  C - 01 41 08 08 A0 A0 83 95
  D - 21 61 12 00 93 92 74 75
  E - 21 62 21 00 84 85 34 15
  F - 21 62 0E 00 A1 A0 34 15
```

Mapper 86: Jaleco Early Mapper 2 - PRG/32K, VROM/8K

Used only by Moero Pro Baseball (Red/Black).
```
  6000h  Memory Control
    Bit6,1,0  Select 8K VROM bank at PPU 0000h-1FFFh
    Bit5,4    Select 32K ROM bank at 8000h-FFFFh
    Bit7,3,2  Not used (always zero)
  7000h  Unknown
```
Also used by Lum no Wedding Bell though that does use only VROM banking.
Functional same as Mapper 87, though that does as well use only VROM banking.

Mapper 87: Jaleco/Konami 16K VROM - VROM/8K

Used only by Hyper Olympic, Goonies, Choplifter, Argus, Ninja Jajamaru Kun, City Connection.
```
  6000h  Select 8K VROM bank at PPU 0000h-1FFFh (Bit 1 used only)
```

Mapper 88: Namco 118

Used by Devil Man, Dragon Spirit, Namcot Mahjong 3, Quinty.
```
  8000h  Index/Control (3bit)
         Bit2-0 Command Number
           0 - Select 2x1K VROM at PPU 0000h-07FFh (Banks 0-63)
           1 - Select 2x1K VROM at PPU 0800h-0FFFh (Banks 0-63)
           2 - Select 1K VROM at PPU 1000h-13FFh   (Banks 64-127)
           3 - Select 1K VROM at PPU 1400h-17FFh   (Banks 64-127)
           4 - Select 1K VROM at PPU 1800h-1BFFh   (Banks 64-127)
           5 - Select 1K VROM at PPU 1C00h-1FFFh   (Banks 64-127)
           6 - Select 8K ROM at 8000h-9FFFh
           7 - Select 8K ROM at A000h-BFFFh
         N/A - Fixed 16K ROM at C000h-FFFFh (always last bank)
  8001h  Data Register (Indexed via Port 8000h)
```
The carts have 128K VROM, of which the lower 64K can be mapped only to Pattern Table 0, the upper 64K only to Pattern Table 1.
Devil Man additionally writes 00h to Port C000h, purpose unknown.
Devil Man ROM-image is declared as 4-screen-vertical-mirror, that is nonsense.

Mapper 89: Sunsoft Early - PRG/16K, VROM/8K

Used by only by Mito Koumon.

```
8000h-FFFFh  Memory Control
  Bit7   Unknown - maybe Name Table related, maybe not.
  Bit6-4 Select 16K ROM bank at 8000h-BFFFh
  N/A    Fixed  16K ROM bank at C000h-FFFFh (always last bank)
  Bit3-0 Select 8K VROM bank at PPU 0000h-1FFFh
```

The program seems to attempt (unsuccesfully) to resolve bus-conflicts.

Mapper 90: Pirate MMC5-style

Used by some pirate titles (Taiwan) such as Super Mario World, Tekken2 and Mortal Kombat. Features similar functions as MMC5, though the Port addresses are completely different.

```
5000h(W)     Maths Coprocessor Parameter A
5001h(W)     Maths Coprocessor Parameter B
5000h(R)     Maths Coprocessor 8bit Result of A*B
8000h        PRG 8K at 8000h-9FFFh
8001h        PRG 8K at A000h-BFFFh or 16k PRG bank at $A000-?
8002h        PRG 8K at C000h-DFFFh
8003h        PRG 8K at E000h-FFFFh
9000h/A000h  LSB/MSB of VROM bank at PPU 0000h (1K,2K,4K,8K)
9001h/A001h  LSB/MSB of VROM bank at PPU 0400h (1K)
9002h/A002h  LSB/MSB of VROM bank at PPU 0800h (1K,2K)
9003h/A003h  LSB/MSB of VROM bank at PPU 0C00h (1K)
9004h/A004h  LSB/MSB of VROM bank at PPU 1000h (1K,2K,4K)
9005h/A005h  LSB/MSB of VROM bank at PPU 1400h (1K)
9006h/A006h  LSB/MSB of VROM bank at PPU 1800h (1K,2K)
9007h/A007h  LSB/MSB of VROM bank at PPU 1C00h (1K)
B000h/B004h  LSB/MSB of 1K VROM bank at PPU 2000h (Name Table VROM mode)
B001h/B005h  LSB/MSB of 1K VROM bank at PPU 2400h (Name Table VROM mode)
B002h/B006h  LSB/MSB of 1K VROM bank at PPU 2800h (Name Table VROM mode)
B003h/B007h  LSB/MSB of 1K VROM bank at PPU 2C00h (Name Table VROM mode)

$C000       irq registers     Unknown
$C001       irq registers     Unknown
$C006       irq registers     Unknown
$C007       irq registers     Unknown
$C002       irq clear         irq_flag=0 and INT signal is clear
$C003       irq reset         if $C005=0, irq_flag=0
                              else, irq_flag=1 and irq_counter=irq_latch
$C004       irq reset         It seems same of $C003
$C005       irq counter       irq_flag=1, irq_latch = irq_counter = value
```

IRQs work like MMC3 does.
IRQ counter is decremented at every scanline, always while not blanking (scanline < 240), and background or sprites are enabled. When it reaches zero (or a negative value), IRQ is triggered _IF_ the irq_flag is set, clearing the irq_flag and irq_latch.

```
D000h  Bank Mode
         Bit1-0  PRG Bank Size
                  0  Fixed last 32K at 8000h-FFFFh (initial setting)
                  1  16K Banks, and Fixed last 16K at C000h-FFFFh
                  2  8K Banks, via Bits 2,7, and Ports 8000h-8003h
                  3  8K in reverse mode?
         Bit2    PRG Bank at E000h in 8K Mode (0=Last 8K, 1=Port 8003h)
         Bit4-3  VROM Bank Size    (0=8K, 1=4K, 2=2K, 3=1K)
         Bit5    Name Table Source (0=VRAM via D001h, 1=VROM via B00Xh)
         Bit6    Not used
         Bit7    PRG Bank at 6000h  (1=enabled) (Similiar/Instead E000h?)
D001h  Name Table Control (in VRAM mode) (only lower 2bit used)
         0   Two-Screen, Vertical mirroring
         1   Two-Screen, Horizontal mirroring
         2,3 One-Screen, BLK0

$D002       unknown           Unused?
$D003       bank page         Only used by larger carts
```

if (bankmode.bit5), map CHR data in the nametable area using $B00x values,
But, if (high byte, low byte) != (0,0) or (0,1) or (0,2) or (0,3),
so bankmode.bit5 is cleared, and mirroring does not change.

```
for(i=0;i<4;i++) {
  if(!nam_high_byte[i] && (nam_low_byte[i] == i)) {
    bankmode &= 0xdf; //clear bit5 --> use VRAM with mirroring
    return;
next
```

If you ignore it, a lot of crappy gfx might be displayed.

Mapper 91: HK-SF3 - PRG/8K, VROM/2K, IRQ

This mapper is used on the pirate cart with a title screen reading "Street Fighter 3". It may or may not have been used in other bootleg games.

```
6000h  Select 2K VROM bank at PPU 0000h-07FFh
6001h  Select 2K VROM bank at PPU 0800h-0FFFh
6002h  Select 2K VROM bank at PPU 1000h-17FFh
6003h  Select 2K VROM bank at PPU 1800h-1FFFh
7000h  Select 8K ROM bank at 8000h-9FFFh
7001h  Select 8K ROM bank at A000h-BFFFh
N/A    Fixed 16K ROM bank at C000h-FFFFh (always last 16K)
7006h  IRQ Disable/Acknowledge (write any value)
7007h  IRQ Enable              (write any value)
```

When enabled, IRQs are requested every 8 scanlines, except during VBlank.
Vertical mirroring is always active.

Mapper 92: Jaleco Early Mapper 1 - PRG-HI, VROM/8K

Used by Moero Pro Soccer, Moero Pro Baseball'88.

```
8000h-FFFFh  Memory Control
  Bit7-6  Function Select
   0  Confirm Selection
   1  Select 8K VROM bank at PPU 0000h-1FFFh
   2  Select 16K ROM bank at C000h-FFFFh (upper half of PRG memory)
   3  Reserved (would probably select both PRG+VROM)
  Bit5-4  Not used
  Bit0-3  ROM or VROM Bank Number for above Selection
```

Bus-conflicts. Example: To select PRG Bank 7, first write 87h, then 07h.
Same as Mapper 72, except that this one maps the UPPER half of PRG memory.

Mapper 93: 74161/32 - PRG/16K

Used only by Fantasy Zone.

```
8000h-FFFFh  Memory Control
  Bit0    Unknown, seems to be always set.
  Bit1-3  Always zero
  Bit4-6  Select 16K ROM bank at 8000h-BFFFh
  Bit7    Always zero
```

Bus-conflicts. Uses VRAM.

Mapper 94: 74161/32 - PRG/16K

Used only by Senjou no Okami (Capcom's Commando, japanese version).

```
8000h-FFFFh  Memory Control
  Bit0-1  Always zero
  Bit2-4  Select 16K ROM bank at 8000h-BFFFh
  Bit5-7  Always zero
```

Bus-conflicts. Uses VRAM.

Mapper 95: Namcot MMC3-Style

Looks like MMC3, but doesn't seem to have IRQs and various other functions.
Used by Dragon Buster 1, and maybe many other "MMC3" games.

```
8000h  Index/Control (3bit)
       Bit2-0 Command Number
         0 - Select 2x1K VROM at PPU 0000h-07FFh
         1 - Select 2x1K VROM at PPU 0800h-0FFFh
         2 - Select 1K VROM at PPU 1000h-13FFh
         3 - Select 1K VROM at PPU 1400h-17FFh
         4 - Select 1K VROM at PPU 1800h-1BFFh
         5 - Select 1K VROM at PPU 1C00h-1FFFh
         6 - Select 8K ROM at 8000h-9FFFh
         7 - Select 8K ROM at A000h-BFFFh
       N/A - Fixed 16K ROM at C000h-FFFFh (always last bank)
8001h  Data Register (Indexed via Port 8000h)
```

Mapper 96: 74161/32 - PRG/32K, CHR/16K/4K, LATCH

Used by Oeka Kids Anpanman no Hiragana Daisuki, and Oeka Kids Anpanman to Oekaki Shiyou (note: these games do require the Oeka Kids Tablet controller, see Controllers chapter for details).

```
8000h-FFFFh
 Bit0-1  Select 32K PRG-ROM bank at 8000h-FFFFh (2bit) (AOROM-style)
 Bit2    Select 16K CHR-RAM bank at PPU:0000h-1FFFh (via below 4K banks)
PPU:2000h-20FFh Select 1st 4K (of above 16K) at PPU:0000h-0FFFh (Latch=0)
PPU:2100h-21FFh Select 2nd 4K (of above 16K) at PPU:0000h-0FFFh (Latch=1)
PPU:2200h-22FFh Select 3rd 4K (of above 16K) at PPU:0000h-0FFFh (Latch=2)
PPU:2300h-23FFh Select 4th 4K (of above 16K) at PPU:0000h-0FFFh (Latch=3)
N/A             Fixed Last 4K (of above 16K) at PPU:1000h-1FFFh
PPU:2400h-2FFFh Other Nametables (same 4K mapping as PPU:2000h-23FFh)
```

Name table seems to be mapped in One-Screen mode (though maybe one of the four bits of the 74HC161 chip allows to select Two-Screen mode, or to swap BLK0 and BLK1)?
Reading (and probably writing) PPU name table entries (PPU:2000h-23FFh) does automatically change the 4K bank at for lower half of 8K CHR-RAM (this allows to display full-screen 16K bitmaps).
For manually setting the 4K bank, it's enough to set the PPU address to 2000h-23FFh via Port 2006h (without actually needing to read/write anything from/to that address via Port 2007h) (ie. apparently, the PPU does output the selected address immediately, even though no reading/prefetching occurs at that time yet).
Bus-conflicts. Chipset: 128K PRG-ROM, 32K CHR-RAM, 74HC02 (quad NOR), 74HC161 (4bit-counter; probably used as simple 4bit latch; without counting), 74HC74 (dual flipflop).

Mapper 97: Irem - PRG HI

Used by Kaiketsu Yanchamaru.

```
8000h-FFFFh  Memory Control
  Bit7-6  Unknown (used values are 1,2 - values 0,3 unused)
            (Maybe Name Table Mirroring)
  Bit5-4  Not used (always zero)
  Bit3-0  Select 16K ROM bank at C000h-FFFFh (upper block!)
  N/A     Fixed  16K ROM bank at 8000h-BFFFh (always LAST 16K bank)
```

Bus-conflicts.

Mapper 99: VS Unisystem Port 4016h - VROM/8K, (PRG/8K)

This is the standard "on-board" mapper used by most VS System games.
VS System

Mapping is done via "controller" port,

```
Port 4016h/Write:
  Bit2  VS Unisystem Select 8K VROM bank at PPU 0000h-1FFFh
  Bit1  VS Dualsystem: Send IRQ to other CPU (0=No, 1=IRQ)
  Bit0  Joypad Strobe (as usually)
```

For games with 40K PRG-ROM (VS Gumshoe has 40K PRGROM+16K CHRROM):

```
  Bit2 additionally selects 8K PRG-ROM (bank 0 or 4) at 8000h-9FFFh,
  Note: Due to .NES fileformat, the 40K PRG-ROM is zeropadded to 48K.
```

Above bank selection uses an expansion port output, which does not show up on NES/Famicom cartridge bus, and thus works only with VS Unisystem arcade machines.
Most (or all?) of these games use 4-screen mirroring.

The standard VS System games are consisting of EPROM Sets rather than of Cartridges. Typically consisting of up to six EPROMs:

```
1A  PRG-ROM E000h-FFFFh         2A  CHR-ROM Bank 1
1B  PRG-ROM C000h-DFFFh         2B  CHR-ROM Bank 0
1C  PRG-ROM A000h-BFFFh
1D  PRG-ROM 8000h-9FFFh
```

For the VS Dualsystem's second CPU, the PRG-ROMs are 6A..6D and CHR-ROMs are 8A-8B accordingly. The names are based on the mainboard coordinates, eg. "1A" is located somewhere near "row 1, column A".

Mapper 100: Whatever

No info. Don't have any such ROM-images.
Reportedly "MMC3/Nestice/Trainer/Buugy Mode Used in hacked roms !!"
Sounds like homebrew hacks that work only on certain emulators, but not on real MMC3 hardware. Or it is just meant to be corrupted .NES files with garbage in reserved header entries at [07h..0Fh], thus changing mapper number 04h (MMC3) to 64h (100 decimal).

Mapper 105: X-in-1 MMC1

Used by Nintendo World Championships 1990. Works similar like normal MMC1:
Mapper 1: MMC1 - PRG/32K/16K, VROM/8K/4K, NT
However, the four registers are used like this:

```
Register 0  Configuration Register (same as MMC1)
Register 1  ROM Bank Base (Bit4 unknown)
Register 2  Not used
Register 3  ROM Bank (same as MMC1, but ORed with Base)
```

And, accessing Register 3 via Port FFF0h (instead normal FFFFh) appears to mask (zero) the new Register 3 bank number, until writing to Register 1. Initially first 32K.

NB. The Championships 3-in-1 multicart doesn't have a game selection menu, it runs game 1 until reaching a certain score, and then switches to game 2, and so on. The controls are strange: it appears one can start the cartridge only when connecting a zapper to port 1, or a joypad to port 2, whilst gameplay works only with joypad at port 1.

Mapper 112: Asder - PRG/8K, VROM/2K/1K

Used by Huang Di, and San Guo Zhi - Qun Xiong Zheng Ba.

```
8000h  Index (0-7)
        0  Select 8K ROM at 8000h-9FFFh
        1  Select 8K ROM at A000h-BFFFh
        2  Select 2x1K VROM at PPU 0000h-07FFh
        3  Select 2x1K VROM at PPU 0800h-0FFFh
        4  Select 1K VROM at PPU 1000h-13FFh
        5  Select 1K VROM at PPU 1400h-17FFh
        6  Select 1K VROM at PPU 1800h-1BFFh
        7  Select 1K VROM at PPU 1C00h-1FFFh
       N/A Fixed 16K ROM at C000h-FFFFh (always last 16K)
A000h  Data (indexed via Port 8000h)
C000h  Unknown, always 00h
E000h  Unknown, always 00h
```

Mapper 113: Sachen/Hacker/Nina

[Seems to be same as Nina-3 and/or Nina-6]
Mapper 79: AVE Nina-3 - VROM/8K
Mapper 81: AVE Nina-6

Used by Metal Fighter, Side Winder, Rad Racket - Deluxe Tennis II, AV Hanafadu Club, AV Soccer, Papillion, Deathbots, Mahjong Companion, 4-in-1 Total Funpack.

```
4100h-41FFh  Memory Control (commonly used addresses: 4100h, 4101h, 4120h)
  Bit0-2  Select 8K VROM bank at PPU 0000h-1FFFh
  Bit3-4  Select 32K ROM bank at 8000h-FFFFh (bigger carts only)
```

Many of these games seem to have been originally desiged for mappers at 8000h-FFFFh, and do still write to these addresses. Also, "16 Mahjong" is declared as Mapper 113, though it uses 8000h-FFFFh only?

There's also a variant in which ROM selection is moved to Bit2:
Mapper 133: Sachen

Mapper 114: Super Games

Used by Lion King, Super Donkey Kong, and Pocohontos. Mask addresses by E001h.

```
6000h  Unknown (usually zero, except Lion King before crashing?)
8000h  Unknown (see notes below)
A000h  Memory Control Index (see list below)
C000h  Memory Control Data (indexed via A000h)
E000h  IRQ Acknowledge (write any value)
6001h  Unknown (always zero)
8001h  Unknown (see notes below)
A001h  IRQ Counter (MMC3-style, decremented per scanline, paused in VBlank)
C001h  IRQ Counter
E001h  IRQ Start
```

Memory Control Indexes are:

```
0  Select 2x1K VROM at PPU 0000h-07FFh
1  Select 1K VROM at PPU 1400h-17FFh
2  Select 2x1K VROM at PPU 0800h-0FFFh
3  Select 1K VROM at PPU 1C00h-1FFFh
4  Select 8K ROM at 8000h-9FFFh
5  Select 8K ROM at A000h-BFFFh
```

```
6  Select 1K VROM at PPU 1000h-13FFh
7  Select 1K VROM at PPU 1800h-1BFFh
```

Port 8001h, Bit0 seems to be used as IRQ disable in Lion King. All games seem to use Vertical Mirroring, but Pocohontos seems to toggle Name Tables via Port 8000h and 8001h during initialization.

Mapper 115: MMC3 Cart Saint

Used by Yuu Yuu Hakusho Final - Makai Saikyou Retsuden.

```
6000h  Unknown (used values 00h, A0h, A4h)
6001h  Unknown (always 00h)
```

No idea how it does <really> work, but the game appears to work when selecting 8K ROM bank number 8 at 8000h-9FFFh when [6000h]=A4h.
Otherwise the mapper works like MMC3:
Mapper 4: MMC3 - PRG/8K, VROM/2K/1K, NT, SRAM, IRQ
Note: MMC3 Register 7 <must> be initialized to 01h on reset.

Mapper 116: Whatever

No info. Don't have a ROM-image.
Reportedly "PC-Reserved" used by "AV beautiy fighting(not playable yet)".

Mapper 117: Future

Used by Sangokushi 4 (a clone of Warrior of Fate).
Appears related with Mapper 90.

```
8000h  Select 8K ROM at 8000h-9FFFh
8001h  Select 8K ROM at A000h-BFFFh
8002h  Select 8K ROM at C000h-DFFFh
N/A    Fixed  8K ROM at E000h-FFFFh (last bank)
9000h  Unknown (always FFh)
9001h  Unknown (always 08h)
9003h  Unknown (always 00h)
A000h  Select 1K VROM at PPU 0000h-03FFh
A001h  Select 1K VROM at PPU 0400h-07FFh
A002h  Select 1K VROM at PPU 0800h-0BFFh
A003h  Select 1K VROM at PPU 0C00h-0FFFh
A004h  Select 1K VROM at PPU 1000h-13FFh
A005h  Select 1K VROM at PPU 1400h-17FFh
A006h  Select 1K VROM at PPU 1800h-1BFFh
A007h  Select 1K VROM at PPU 1C00h-1FFFh
A008h-A00Fh  Unknown (always 01h, probably VROM bank related)
C001h  IRQ Counter/Start (MMC3, decremented per scanline, paused in VBlank)
C002h  IRQ Acknowledge (write any value)
C003h  IRQ Counter/Start (always write same value as to C001h)
D000h  Unknown (always 00h)
E000h  IRQ Enable (Bit0), upper 7bit unknown (always 0000011b)
F000h  Unknown (always 00h)
```

Mapper 118: MMC3 TLSROM - PRG/8K, VROM/2K/1K, Banked-NT, SRAM, IRQ

Used by Goal 2, Pro Sport Hockey, Armadillo, and Ys III.
MMC3 variant with special Name Tables connnection. The CHR ROM bank numbers are limited to 7bit values (128 1K-banks), the normal MMC3 Name Table Mirroring Register (Port A000h) is not used. Instead, Bit7 of the CHR ROM bank(s) is connected to VA10 line (selecting name table BLK0 or BLK1). The relation between the separate CHR bank registers and VA10 is unknown? Probably, NT0 at 2000h-23FFh is controlled by CHR bank for 0000h-03FFh, NT1 at 2400h-27FFh by CHR bank for 0400h-07FFh, and so on.
Otherwise same as MMC3:
Mapper 4: MMC3 - PRG/8K, VROM/2K/1K, NT, SRAM, IRQ

Mapper 119: MMC3 TQROM - PRG/8K, VROM/VRAM/2K/1K, NT, SRAM, IRQ

Used by Pinbot and High Speed.
Contains 64K CHR ROM plus 8K CHR-RAM, selected by Bit6 of the CHR bank numbers (0=ROM, 1=RAM). The CHR-RAM can be banked and mapped like the CHR ROM. Mapping hardware consists of MMC3B plus 74HC32 (used to disable ROM when RAM enabled by CE=A16=HIGH).
Otherwise same as MMC3:
Mapper 4: MMC3 - PRG/8K, VROM/2K/1K, NT, SRAM, IRQ

Mapper 122: Whatever

No info. Don't have a ROM-image.
Reportedly "74161/32" used by "Madoola No Tsubasa".
Maybe "Madoola No Tsubasa" is same as "Wing of Madoola" (?)
Mapper 184: Sunsoft - VROM/4K

Mapper 133: Sachen

Used by Jovial Race.

```
4120h  Memory Control
  Bit1-0  Select 8K VROM at PPU 0000h-1FFFh
  Bit2    Select 32K ROM at 8000h-FFFFh
```

Appears to be a variant of Mapper 113, with PRG ROM selection moved to Bit2.

Mapper 151: VS Unisystem VRC1 or MMC3 Daughterboards

This is a nonsense mapper number for VS Unisystem games with daughterboards.
VS System
Namely, this mapper number is used for VS Gradius (VRC1), VS Goonies (VRC1), and VS TKO Boxing (MMC3-style). The correct mappers for that games should be:
Mapper 4: MMC3 - PRG/8K, VROM/2K/1K, NT, SRAM, IRQ
Mapper 75: Jaleco SS8805/Konami VRC1 - PRG/8K, VROM/4K, NT

Mapper 152: Whatever

The readme of Pocketnes (for Gameboy Advance) mentions existance Mappper 152.
No info. Don't have any ROM-images.

Mapper 160: Same as Mapper 90

Seems to be duplicate/nonsense, same as Mapper 90, used by Aladdin.
Mapper 90: Pirate MMC5-style

Mapper 161: Same as Mapper 1

Seems to be duplicate/nonsense, same as Mapper 1, used by Hanjuku Eiyuu.
Mapper 1: MMC1 - PRG/32K/16K, VROM/8K/4K, NT

Mapper 168: RacerMate PRG/16K, VRAM/4K, IRQ

Used only by RacerMate Challenge II.

```
6000h..600Fh  Unknown (looks like Debug LED Control or so)
B000h         Memory Banking (74LS174; 6bit D flip-flop)
                N/A   Fixed  4K CHR-RAM at PPU:0000h-0FFFh (bank 00h)
                D0-D3 Select 4K CHR-RAM at PPU:1000h-1FFFh (bank 00h..0Fh)
                D4-D5 Unused
                D6-D7 Select 16K PRG-ROM at 8000h-BFFFh (bank 00h..03h)
                N/A   Fixed  16K PRG-ROM at C000h-FFFFh (bank 03h)
F000h         IRQ/counter control or so ;\maybe IRQ ack,reset,enable,disable
F080h         IRQ/counter control or so ;/or whatever (maybe 7474 flipflop)
```

The game does also require a special controller:
Controllers - RacerMate Bicycle Training System
Note: The IRQ function is used as baudrate timer for the controller data transfers (not as video/scanline interrupt). Used ports/values are:

```
[F080h]=FFh   disable IRQ and/or reset IRQ-counter or so
[F000h]=00h   enable IRQ and/or start IRQ-counter or so
```

Above two ports are mirrored to C000h-FFFFh, they are actually only ONE port, using either A7 (old PCB design) or D2 (revised PCB design) as input to one of the 7474's flipflops. The output from 1st flipflop is somehow forwarded to the 2nd flipflop (eventually unlocking SRAM writes at some point).

Unknown Details
Unknown if both SRAM chips are battery backed. Unknown what the 7474 is used for (maybe IRQ/counter control related). Port 6000h-600Fh seems to

have no function; there are no latches or RAM in the cartridge that could be mapped to that addresses (except maybe the 7474), and there should be nothing special in the "modified NES" console (the game does reportedly work on any normal NES with "CIC disabled").

Racer-Mate PCB R982-073

```
U1   28pin  HY52256A SRAM 32Kx8 (CHR-RAM)
U2   28pin  HY52256A SRAM 32Kx8 (CHR-RAM)
U3   28pin  27C512 EPROM 64Kx8 (badged V903_128) (PRG-ROM)
U4   16pin  74LS174 (6bit D flip-flop with reset) ;memory banking
U5   14pin  74HCT32 (quad OR gates)
U6   14pin  74LS00  (quad NAND gates)
U7   14pin  74HCT32 (quad OR gates)
U8   14pin  74LS00  (quad NAND gates)
U9   14pin  74HCT74 (dual flipflop)               ;maybe IRQ status/control?
U10  16pin  74HCT4040N (12bit counter with reset) ;IRQ counter (input=PHI2)
U11  16pin  Unknown, maybe CIC (not installed)
B1   2pin   Maxell Battery
J1J2 3pin   Jumpers (J2 installed)
J3J4 3pin   Jumpers (J3 installed)
J1   72pin  Cart-edge connector (for NES consoles) (not Famicom)
```

Plus 2 transistors, 2 diodes (near battery), resistors, capacitors.
Early cartridges used Tengen CIC clones. Later cartridges didn't include any CICs for legal reasons (and instead, beared a sticker saying says that it'll work only with special modified NES consoles; ie. such with CIC disabled).

Cautions/Confusions

Existing ROM-images may include 64K VROM (ie. the video RAM dumped as if it were ROM); this is wrong, ignore that garbage, and instead, allocate 64K VRAM.
There's some info that claims memory to be controlled by Port 8000h, with D0,D1,D2=PRG-ROM/16K, and D3,D6,D7=CHR-RAM/8K; this is wrong on about every single detail, ignore that.
The cartridge does reportedly use horizontal mirroring (but, that info seems to come from same person who dumped RAM as ROM, and who talked about Port 8000h, so chances are around 80% that this info is wrong, too).

Mapper 180: Nihon Bussan - PRG HI

Used by Crazy Climber. And MGC 2011?

```
8000h-FFFFh  Memory Control
  Bit7-3  Not used (always zero)
  Bit2-0  Select 16K ROM bank at C000h-FFFFh (upper block!)
  N/A     Fixed  16K ROM bank at 8000h-BFFFh (always FIRST 16K bank)
```

Bus-conflicts.
Same as UNROM, but with fixed bank at 8000h-BFFFh instead of C000h-FFFFh.
Note: Crazy Climber uses two standard joypads, but wants them to be held rotated by 90 degrees.

Mapper 182: Same as Mapper 114

Mapper 114: Super Games

Mapper 184: Sunsoft - VROM/4K

Used by Atlantis no Nazo, Kanshakudama, and Wing of Madoola.

```
6000h  Select VROM Banks
  Bit2-0  Select 4K VROM at PPU 0000h-0FFFh
  Bit6-4  Select 4K VROM at PPU 1000h-1FFFh
```

Mapper 185: VROM-disable

Used for copy-protected "NROM" games with (max) 32K PRG-ROM and 8K VROM.

```
8000h-FFFFh
 Bit 0-1  Select 8K VROM or Open Bus at PPU 0000h
 Bit 4-5  Security Diodes (some crude copy protection)
```

The actual board is a CNROM variant, with the VROM bank-selection bits misused to deselect VROM (the games do verify that feature and refuse to run if the deselection doesn't work), plus CNROM-style Security Diodes. For details see:
Mapper 3: CNROM - VROM/8K

Example Values

Values used to switch VROM on/off are:

```
Off  On   Title
F0h  0Fh  Bird Week
```

```
  00h  33h  B-Wings
  00h  11h  Mighty Bomb Jack
  20h  22h  Sansuu 1 Nen - Keisan Game
  20h  22h  Sansuu 2 Nen - Keisan Game
  00h  FFh  Sansuu 3 Nen - Keisan Game
  13h  21h  Spy vs Spy
```
Above games are working when mapping an empty VROM bank (FFh-filled) either when (X)=13h, or when (X AND 0Fh)=0.

Mapper 188: UNROM-reversed

Used by Karaoke Studio. Appears to be same as UNROM, but the first/second 128K are exchanged in the ROM-image (for unknown reason), ie. all bank numbers (including the fixed "last" bank) are XORed by 08h.
```
      ... aka bit3 = chipselect (1=Main ROM, 0=Expansion ROM)
```
Mapper 2: UNROM - PRG/16K
Or, maybe only the first 8 banks are used, and the further banks are garbage?
Bus-conflicts.

Microphone
The cartridge does have an external microphone attached to it.
As far as known, the microphone signal is passed to the Audio In pin on the cartridge slot (and thus forwarded to the speaker in the TV Set).
Reportedly, the game can "judge" good/bad singers, so there must be some way to read the microphone level by software; I/O ports and bit-depth are unknown.
The game does read whatever 3bit value from Port 6000h.

```
  Karaoke Studio (main cartridge with microphone and 128K main ROM) (Jul 1987)
  Karaoke Studio Senyou Cassette Vol. 1 (128K expansion ROM) (Oct 1987)
  Karaoke Studio Senyou Cassette Vol. 2 (128K expansion ROM) (Feb 1988)
```
The expansion ROMs are small cartridges that can be plugged into the main cartridge; ROM-images for expansion carts should be 256K in size (the 128K main ROM plus 128K expansion ROM badged together).

See also
Controllers - Microphones

Mapper 189: MMC3 Variant

Used by Master Figher 2, and Street Fighter 2. There are three Master Fighter 2 versions, the 1st works as described below, the 2nd works but has distorted background, the 3rd doesn't work - ROM addresses appear corrupted (?). And, Street Fighter 2 works completely different - uses Ports 4132h for ROM, and 4122h/4123h for VROM (?) Anyways, the one working one works as such:
```
  610xh  Select 32K ROM Block (D7-D0 should match A7-A0, eg. [6103h]=03h)
```
The rest of the mapper is same as MMC3 (for the selected block of memory),
Mapper 4: MMC3 - PRG/8K, VROM/2K/1K, NT, SRAM, IRQ

Mapper 218: Nocash Single-Chip

Used by Magic Floor (2012) and Starfight (2018). Contains only one single PRG-ROM chip (and a CIC, if required) (and, in case of the bewitched Magic Floor cartridge: it's planned to contain a hair, a finger nail, and a drop of blood; the existing prototype PCB lacks these components though).
There's no CHR-ROM or CHR-RAM. Instead, the console's internal 2Kbyte Name Table RAM is mapped as CHR-RAM. The 2K RAM is permanently selected (/VCS wired to GND), and can be used in four modes by wiring VA10 to one of the PPU.A10..A13 address lines:
```
  VA10     Effect on                       iNES Byte 6   UNIF "MIRR"
  to       Name Tables                     Bit3.Bit0     Bit7-0
  PPU.A10  Two-Screen, Vertical Mirroring  0.1           01h
  PPU.A11  Two-Screen, Horizontal Mirroring 0.0          00h
  PPU.A12  One-Screen, BLK0                1.0           02h
  PPU.A13  One-Screen, BLK1                1.1           03h
  Note: Bit 3 in Byte 6 of iNES header would be usually Four-Screen flag,
  but, for this mapper it is used as One-Screen flag.
```
The VA10 connection does, of course, also affect the CHR-RAM mapping at 0000h-1FFFh. BLK1 would be the most common case (1K NT plus 1K CHR-RAM). BLK0 would allow to swap CHR RAM via Port 2000h.Bit3-4. Two-Screen would allow to use two NTs (and to squeeze CHR data into unused NT areas). Two-Screen would also allow to use 2K OBJ tiles (when leaving BG unused).
Note: 1K CHR-RAM allows to use as much as 64 tiles of 2bpp (or, with suitable color attributes, 128 monochrome tiles of 1bpp).

Mapper 222: Dragon Ninja

Used by a pirate copy of Dragon Ninja.
```
  8000h        Select 8K ROM at 8000h-9FFFh
  A000h        Select 8K ROM at A000h-BFFFh
```

```
N/A          Fixed 16K ROM at C000h-FFFFh (last 16K)
9000h        Unknown (always E0h = Vertical Mirroring)
B000h/B001h  Lower/upper 4bit of 1K VROM bank at PPU 0000h-03FFh
B002h/B003h  Lower/upper 4bit of 1K VROM bank at PPU 0400h-07FFh
C000h/C001h  Lower/upper 4bit of 1K VROM bank at PPU 0800h-0BFFh
C002h/C003h  Lower/upper 4bit of 1K VROM bank at PPU 0C00h-0FFFh
D000h/D001h  Lower/upper 4bit of 1K VROM bank at PPU 1000h-13FFh
D002h/D003h  Lower/upper 4bit of 1K VROM bank at PPU 1400h-17FFh
E000h/E001h  Lower/upper 4bit of 1K VROM bank at PPU 1800h-1BFFh
E002h/E003h  Lower/upper 4bit of 1K VROM bank at PPU 1C00h-1FFFh
F000h        IRQ Counter/Stop/Set/Ack
F001h        IRQ Counter/Stop/Set/Ack
F002h        IRQ Counter/Start (incrementing approx every 120 (?) cycles)
```

Mapper 225: X-in-1

Used by 52-in-1, 58-in-1, 64-in-1, 72-in-1, 110-in-1, 115-in-1 carts. The reset vectors in all games are redirected to the game selection menu, and all copyright messages have been shamelessly removed.

```
8000h-FFFFh  Memory Control (Write any data, port decoded by address lines)
  A14,A5-0   Select 8K VROM bank at PPU 0000h-1FFFh
  A14,A11-A6 Select PRG 2x16K ROM bank at 8000h-FFFFh
  A12        Select PRG page size (0=32K, 1=16K)
              0  32K page at 8000h-FFFFh (LSB/A6 of bank number ignored)
              1  16K page mirrored to 8000h-BFFFh and C000h-FFFFh
  A13        ?Mirroring select (0=Vertical, 1=Horizontal Mirroring)
  A15        Must be "1"
5800h-5FFFh  4x4bit Register File (D0-D3 data bits, addressed via A0-A1)
```

The 4x4bit latch is used as 16bit "RAM", used to restore the old menu selection when re-entering the menu by pushing the Reset button.
A14 is shared for both PRG/VROM in larger 2048K+1024K carts (those with more than 100 games), smaller 1024K+512K carts don't use A14.

Mapper 226: X-in-1

Used by Super 42-in-1 (1024K), and 76-in-1 (2048K).
Typically booted with opcode 8E 8E 8E 00 - MOV [8E8E],X; BRK in RAM.

```
8000h,8E8Eh  Memory Control
  Bit4-0 Bank Number Bit4-0
  Bit5   Mode
          0  Map 32K ROM at 8000h-FFFFh (bank bits 6-1 used, bit0 ignored)
          1  Map the same 16K ROM bank at both 8000h-BFFFh and C000h-FFFFh
  Bit6   Name Table (0=Horizontal, 1=Vertical Mirroring)
  Bit7   Bank Number Bit5
8001h  Upper Bit of bank selection (2048K carts only)
  Bit0   Bank Number Bit6
```

The "VROM" banks of the original games are contained in PRG ROM, and are copied to 8K VRAM at PPU 0000h-1FFFh when starting a game.
See also:
Mapper 233: X-in-1 plus Reset
Mapper 230: X-in-1 plus Contra

FROM 226.TXT
Mapper hardware is provided by five 74-series ICs; LS74A, LS273, LS139, LS02
and LS153. A diode and capacitor are arranged to reset the mapping when the
Reset button is pressed.

```
Register 1, Bit 1 - controls whether the CHR-RAM is write-protected:
  0 - not write-protected
  1 - write-protected
```

When the Reset button is pressed, both registers are reset to all zero bits.

Mapper 227: X-in-1

Used by 1200-in-1 (a fake containing only 14 different games).

```
8000h-FFFFh  Memory Control (Write any data, port decoded by address lines)
  A6-A2      Select 16K ROM at 8000h-BFFFh (X)
  A1         Mirroring (0=Vertical, 1=Horizontal Mirroring)
  A14-A13    Menu mode (00b=Menu, 11b=Other)
  A9         128K Mode (1=128K, 0=Other)
  A0         32K Mode  (1=32K, 0=Other)
  A11-A10,A8 Always 0
  A12        Usually 1 (except when initializing VRAM for game)
  A7         Usually 1 (except menu/contra/galaxian)
```

16K ROM at C000h-FFFFh is Bank 0 in Menu Mode (and on reset), Bank (X OR 1) in 32K Mode, Bank (X OR 7) in 128K Mode, or otherwise Bank (X) in 16K mode.

Mapper 228: X-in-1 Homebrewn

Used in two carts with incredible crude homebrewn games: Action 52 (multicart 52-in-1) and Cheetah Men 2 (single game cart).

```
8000h-FFFFh  Memory Control (Decoded by address AND data lines)
  A3-A0,D1-D0          Select 8K VROM at PPU 0000h-1FFFh
  A12-A7               Select 32K ROM at 8000h-FFFFh
  A14-A13,A6-A4,D7-D2  Not used (always zero)
5FF0h-5FF3h  4x4bit Register File (D0-D3 data bits, addressed via A0-A1)
```

The 4x4bit latch is used as 16bit "RAM", used to restore the old menu selection when re-entering the main menu (used in Action52 multicart only).

Notes

Action52 has an odd ROM-size of 1.5MB, Banks 30h-3Fh are probably mirrors.
There seems to be no Name Table control bit (unless it is shared with bank-selection bits), most games look better at Vertical Mirroring (eg. cheetahmen, silversword), though some look better at Horizontal Mirroring (eg. criticalbp).

Mapper 229: 31-in-1

Used by 31-in-1 (512K+256K).

```
8000h-FFFFh  Memory Control (Write any data, port decoded by address lines)
  A4-A0  Bank Selection, shared for PRG and VROM:
    Select 8K VROM bank at PPU 0000h-1FFFh
    Select 16K ROM bank at 8000h-BFFFh and same bank at C000h-FFFFh
    A selection of 01h works special, it maps 16K ROM banks 0 and 1,
    and bank 1 VROM, used for Super Mario which has 32K PRG ROM.
  A6-A5  Name Table
    0  Two-Screen, Vertical Mirroring
    1  Two-Screen, Horizontal Mirroring
    2  Probably one-screen, used on boot
    3  Probably one-screen, used in menu
  A14    The menu sets this bit when accessing bank 0
```

Mapper 230: X-in-1 plus Contra

Used in a 640K ROM-image. The 1st 128K contain a single game (Contra), the remaining 512K contain a 22-in-1 multicart. NB. the multicart menu also tries to detect further ROM, and if any such found, displays a 63-in-one menu.
No idea if/how selection between Contra and 22-in-1 works. The 22-in-1 part is identical with Mapper 226 (banks 0..31 located AFTER the 1st 128K):
Mapper 226: X-in-1

Mapper 231: 20-in-1

Used by 20-in-1.

```
8000h-FFFFh  Memory Control (Write any data, port decoded by address lines)
  A7-A6  Name Table Setting (0-3)
    0  Probably one-screen (used by menu only)
    1  Two-Screen Vertical Mirroring
    2  Two-Screen Horizontal Mirroring
    3  Not used
  A5     Always opposite of A1, ie. A5=(A0 XOR 1), probably 2nd chip-select
  A4-A1  Select 32K ROM bank at 8000h-FFFFh
  A0     Mode (0=Normal, 1=Mirror 1st half selected 32K bank to C000h-FFFFh)
```

The "VROM" banks of the original games are contained in PRG ROM, and are copied to 8K VRAM at PPU 0000h-1FFFh when starting a game.
There's also a version of the same cartridge with slightly different mapper:
Mapper 61: 20-in-1

Mapper 232: 4-in-1 Quattro Camerica

Used by Camerica 4-in-1 games with 256K ROM and 8K VRAM - Quattro Adventure, Quattro Arcade, and Quattro Sports.

```
9000h        Select 64K block for 8000h-FFFFh   (block number in Bit4-3)
C000h-Fxxxh  Select 16K ROM bank at 8000h-BFFFh (within current 64K)
N/A          High   16K ROM bank at C000h-FFFFh (last 16K of current 64K)
FFF0h,FFF1h  Unknown - Write any value at proper timing (maybe lockout)
```

In some (not all) ROM-images, 2nd/3rd game seem to be mis-exchanged.

Mapper 233: X-in-1 plus Reset

Used by "42 Games" which is almost the same as "Super 42-in-1" (Mapper 226, which comes with separate 22 and 20 games menues, prompting the user to press Select to switch to the next menu).

42 Games (1024K ROM) comes with a "Level 1-4" main menu, each "Level" allows to select from 10 or 11 games. That main menu shows up only if the cartridge detects a special reset function, if that detection fails, then it assumes to be a 512K ROM, and enters a 22 or 20 games menu. That means, the cartridge can be split into two fully functional 512K ROMs when removing the reset function. That reset function appears to work like this:

```
FFFDh  Reading from this address (the MSB of reset vector) destroys the
       current Bank selection, probably setting it to a value of FFh, at
       least anything different than 00h or 80h
```

Aside from the extra reset function, it is same as Mapper 226,
Mapper 226: X-in-1

There's also ROM named "Unknown" numbered as Mapper 233 with other functionality?

Mapper 234: Maxi-15

Used by the AVE Maxi 15 Game Cartridge.

```
Registers are set by writing *or reading* certain locations. In the case
of writing, the programmer would need to ensure that the written value and
that put on the data bus by the program ROM do not conflict.
```

On power-up, and when the Reset button is pressed, registers R1 and R2 are cleared. R3 is not cleared when Reset is pressed. After R1 has been set to a non-zero value, it cannot be changed until the Reset button is pressed.

```
FF80h-FF9Fh  Configuration Register (R1)
  Bit7   Name Table Control (0=Vertical, 1=Horizontal Mirroring)
  Bit6   Page Mode ROM/VROM Size    (0=32K, 1=64K)
  Bit5-0 Select 32K ROM/VROM bank (LSB ignored in 64K Page Mode)
         Bit5 is wired to /CS or /OE of the ROM chips, ie. both ROM
         and VROM are disabled when bit5 is set (unless additional ROMs
         would be connected to inverted Bit5, in larger carts).
FFE8h-FFF7h  Memory Banking Register (R2)
  Bit7   Not Used
  Bit6-4 Select 8K VROM at PPU 0000h-1FFFh (Bit6 not used in 32K Page mode)
  Bit3-1 Not Used
  Bit0   Select 32K ROM at 8000h-FFFFh     (Bit0 not used in 32K Page mode)
FFC0h-FFDFh  Lockout Register (R3)
 Initially it is not possible to access R3. This is only possible
 after R1 has been set to a non-zero value.
  Bit7-2 Not Used
  Bit1   CIC RST
  Bit0   CIC OUT
```

Memory-mapping hardware consists of eleven chips: 74LS273, 2x74LS322, 2x74LS175, 2x74LS138, 74LS30, 74HC08, 74HC04 and a 4053. There are several discrete components, mostly related to the CIC-defeating function.

Mapper 240: C&E/Supertone - PRG/32K, VROM/8K

Used by Jing Ke Xin Zhuan (via Port 4800h), and Sheng Huo Lie Zhuan (via Port 4120h, or alternately GNROM-style via Port 8000h-FFFFh with bus-conflicts).

```
4120h,4800h,8000h-FFFFh  Memory Control
  Bit7-6  Not used (always zero)
  Bit5-4  Select 32K ROM at 8000h-FFFFh (initially any 32K bank)
  Bit3-0  Select 8K VROM at PPU 0000h-1FFFh
```

Mapper 241: X-in-1 Education

Used by Education Games 18-in-1, and Study and Game 32-in-1.

```
8000h-FFFFh        Select 32K ROM at 8000h-FFFFh (initially 1st 32K bank)
5FF0h-5FFFh/Write  Unknown (No info)
5FF0h-5FFFh/Read   Unknown (somewhat Bit6: 1=Ready/Okay)
```

Both cartridges require a special keyboard controller, similar as the Famicom keyboard, but with different keyboard matrix.
Controllers - Typewriter Keyboards

Mapper 242: Waixing - PRG/32K, NT

Used by Wai Xing Zhan Shi.

```
8000h-FFFFh  Memory Control (Write any data, port decoded by address lines)
 A6-A3      Select 32K ROM at 8000h-FFFFh (initially 1st 32K bank)
 A1         Mirroring (0=Vertical, 1=Horizontal Mirroring)
 A7,A0      Always 1
```

```
  A14-A8,A2  Always 0
```
Similar as Mapper 227 (without using the various memory modes though),
Mapper 227: X-in-1

Mapper 243: Sachen Poker - PRG/32K, VROM/8K

Used by Mei Nu Quan (Honey Peach), and Poker III 5-in-1.
```
  4100h  Index (0-7)
  4101h  Data for above index
```
The separate register indexes are:
```
  0  Unknown, always 00h
  1  Unknown, always 00h
  2  Bit3 of 8K VROM at PPU 0000h-1FFFh
  3  Unknown, always 00h
  4  Bit0 of 8K VROM at PPU 0000h-1FFFh
  5  Select 32K ROM at 8000h-FFFFh
  6  Bit2,1 of 8K VROM at PPU 0000h-1FFFh
  7  Unknown, always 05h
```
Formula for VROM Registers 2,4,6: Bank=(R2*8)+(R4 AND 1)+(R6 AND 3)*2

Mapper 244: C&E - PRG/32K, VROM/8K

Used by Decathlon only. Note: Cat Ninden Teyandee translations declared as "Mapper 244" seem to be MMC3 games.
```
  8000h-FFFFh Memory Control
    Bit7   Not used (zero)
    Bit6-4 Swap bits (some sort of confusion / copy protection, see below)
    Bit3   Set ROM or VROM bank (0=ROM, 1=VROM)
    Bit2-0 Select 32K ROM at 8000h-FFFFh or 8K VROM at PPU 0000h-1FFFh
```
Swap bits for ROM Bank Number:
```
  Bit4=1: XOR bank number by 03h
  Bit5=1: Exchange bank number Bit0,1
  Bit6=1: Not used
```
Swap bits for VROM Bank Number:
```
  Bit4=1, Bit5=1, Bit6=1: XOR bank number by 07h (without further exchanges)
  Bit4=0, Bit5=1, Bit6=1: Not used
  Bit4=1: Exchange bank number Bit 0,1 (processed first)
  Bit5=1: Exchange bank number Bit 1,2
  Bit6=1; Exchange bank number Bit 2,0 (processed last)
```
Bus-conflicts.

Mapper 246: C&E - PRG/8K, VROM/2K, SRAM

Used by Fong Shen Bang - Zhu Lu Zhi Zhan.
```
  6000h  Select 8K ROM at 8000h-9FFFh
  6001h  Select 8K ROM at A000h-BFFFh
  6002h  Select 8K ROM at C000h-DFFFh
  6003h  Select 8K ROM at E000h-FFFFh (initially probably last bank, or bank 3)
  6004h  Select 2K VROM at PPU 0000h-07FFh
  6005h  Select 2K VROM at PPU 0800h-0FFFh
  6006h  Select 2K VROM at PPU 1000h-17FFh
  6007h  Select 2K VROM at PPU 1800h-1FFFh
```
The cartridge also contains SRAM at 6000h-7FFFh.
Not sure if SRAM at 6000h-6007h can be used.

Mapper 255: X-in-1 - (Same as Mapper 225)

Duplicated/nonsense mapper number, 255 is functional same as 225.

Mapper 225: X-in-1

Famicom Disk System (FDS)

Famicom Disk System (FDS) is a Famicom extension unit which was produced by Nintendo and only sold in Asian countries. It consists of a disk drive accepting 2.5" or 3" (?) floppies, 32K of RAM to load programs into, 8K of VRAM, and some other hardware described below.

FDS Memory and I/O Maps
FDS I/O Ports - Timer

FDS Memory and I/O Maps

FDS Memory Map

```
4020h-40FFh  I/O Ports (2C33) (Disk, Sound, Timer)
6000h-DFFFh  32K WRAM
E000h-FFFFh  8K FDS BIOS ROM
```

Caution: Parts of the FDS 32K WRAM, and of the built-in 2K WRAM are reserved for use by the FDS BIOS: 0000h-000Eh, 00F9h-0103h, and DFF6h-DFFFh.

FDS VRAM Map

```
0000h-1FFFh  Pattern Tables - 8K VRAM
```

Note: Horizontal/Vertical Name Table Mirroring can be selected via Port 4025h.

FDS I/O Map (2C33 Registers)

```
4020h Timer IRQ Counter Reload value LSB (W)
4021h Timer IRQ Counter Reload value MSB (W)
4022h Timer IRQ Enable/Disable (W)
4023h 2C33 I/O Control Port
4024h Disk Data Write Register (W)
4025h Disk Control Register (W)
4026h Disk External Connector Output (W)
4030h Disk Status Register 0 (R)
4031h Disk Data Read Register (R)
4032h Disk Status Register 1 (R)
4033h Disk External Connector Input (R)
4040h..407Fh Sound Wave RAM - 64 x 6bit sample data (R/W)
4080h Sound Volume Envelope (W)
4082h Sound Wave RAM Sample Rate LSB (W)
4083h Sound Wave RAM Sample Rate MSB and Control (W)
4084h Sound Sweep Envelope (W)
4085h Sound Sweep Bias (W)
4086h Sound Modulation Frequency LSB (W)
4087h Sound Modulation Frequency MSB (W)
4088h Sound Modulation Table (W)
4089h Sound Wave RAM Control (W)
408Ah Sound Envelope Base Frequency (W)
4090h Sound Current Volume Gain Level (6bit) (R)
4092h Sound Current Sweep Gain Level (6bit) (R)
```

FDS I/O Ports - Timer

Interrupt Timer intended to produce mid-screen scanline interrupts.

4020h - Timer IRQ Counter Reload value LSB (W)
4021h - Timer IRQ Counter Reload value MSB (W)

```
Reload value loaded to actual 16bit counter register on write to 4022h,
and on counter underflow. Counter is decremented once per CPU clock cycle.
```

4022h - Timer IRQ Enable/Disable (W)

```
Bit1  Enable  (0=Stop/Acknowledge? Timer IRQ, 1=Start/Enable Timer IRQ)
```

Note: Timer IRQ Flag is found in Bit0 of Port 4030h (Disk Status Register 0); reading that status register does (also?) acknowledge IRQs.

The IRQ Vector is controlled by the BIOS via [0101h] and [DFFEh],
FDS BIOS Data Areas in WRAM

FDS I/O Ports - Disk

Disk access can be handled by using the FDS BIOS, so there's usually no need to write to below ports directly, the only exception would be the Screen Mirroring flag, to change it: Read the current value from [FAh], change Bit3, then write the new value to both [FAh] and [4025h].

4025h - Disk Control Register (W) (Read-able copy in WRAM at 00FAh)

```
Bit0   Drive Motor         (0=On, 1=Off)
 When active (0), causes disk drive motor to stop. During this time,
 $4025.1 has no effect. Uh, Active=0=Stop ?
Bit1   \ = Set drive head to the start of the first track.
 When active (0), causes disk drive motor to turn on. This bit must stay
 active throughout a disk transfer, otherwise $4032.1 will always return 1.
 When deactivated, disk drive motor stays on until disk head reaches most
 inner track of disk.
Bit2   Disk Data Direction (0=Write, 1=Read)
Bit3   Screen Mirroring    (0=Vertical, 1=Horizontal Mirroring)
Bit4   Enable CRC Phase    (0=Read/Write Data, 1=Verify/Write CRC)
Bit5   Unknown             (Should be always 1)
Bit6   GAP Control, Read Mode: 1=Reset CRC, and wait for end of GAP.
       Write Mode: 1=Reset CRC, and start writing data. 0=Write GAP (zeros)
Bit7   Disk IRQs on every byte transfer  (0=Disable, 1=Enable)
```

4030h - Disk Status Register 0 (R)

```
Bit0  Timer IRQ Flag  (0=None, 1=IRQ: Timer Underflow)
Bit1  Disk IRQ Flag   (0=None, 1=IRQ: Request Data Transfer via 4024h/4031h)
      Reset when $4024, $4031, or $4030 has been serviced.
Bit4  CRC Status  (0=Okay, 1=Error, Checksum at end of block not matching)
Bit6  Lost Data   (0=Okay, 1=Error, CPU didn't process 4024h/4031h in time)
Bit7  Unknown
```

Bits in this register seem to be reset to zero after reading; namely, IRQs are acknowledge after reading, and, error flags are probably cleared, too.

4032h - Disk Status Register 1 (R)

```
Bit0   Disk Presence     (0=Inserted, 1=Not inserted)
Bit1   Disk Rewind Flag  (0=Ready/Playback, 1=Rewind Active)
Bit2   Write Protection  (0=Writeable, 1=Read-only, or Disk not inserted)
Bit6   Usually 1         (probably relict of recent opcode byte)
```

4031h - Disk Data Read Register (R)

4024h - Disk Data Write Register (W)

```
8bit data received from / to be written to disk (least significant first).
```

Note: Disk IRQ Flag indicates when next byte is to be transferred.

4026h - External Connector Output (W) (Read-able copy in WRAM at 00F9h)

4033h - External Connector Input (R) (Inputs work only if Outputs=High)

```
Bit0-6  External Connector Pins 3-9 (0=Low, 1=High/Input)
Bit7    Power Good                  (0=Okay, 1=Battery power low)
```

Port 4026h should output High to any input pins, especially Bit7 should be always set to configure Power Good as input.

4023h - 2C33 I/O Control Port

```
Bit0  Disk I/O   (0=Disable, 1=Enable)
Bit1  Sound      (0=Disable, 1=Enable)
```

FDS I/O Ports - Sound

4040h..407Fh - Wave RAM - 64 x 6bit sample data (Read/Write)

Writes to these registers are ignored unless Write Mode is turned on (see register 4089h).

4089h - Wave RAM Control (Write Only)

```
Bit7    Wave Write Mode (1=Stop Sound output & Allow to write to Wave RAM)
Bit6-2  Not used
Bit1-0  Master Volume   (0-3 = 100%,66%,50%,40% = 30/30,20/30,15/30,12/30)
```

4082h - Wave RAM Sample Rate LSB (Write Only)

```
Bit7-0  Lower 8 bits of the main unit's frequency (upper 4 bits in 4083h)
```

4083h - Wave RAM Sample Rate MSB and Control (Write Only)

```
Bit7    Main Unit disable  (0=Enable, 1=Disable Sound Output)
Bit6    Envelope disable   (0=Normal, 1=Disable Volume/Sweep Envelopes)
Bit5-4  Not used
Bit3-0  Upper 4 bits of the main unit's frequency
```

Main Unit / Sample Rate: (per entry of the 64-entry wave ram)

```
F = 1.79MHz * (Freq + Mod) / 65536
Mod = Frequency change based on the Modulation unit
```

If the 12bit frequency is zero, the Main unit is disabled (channel silent).

408Ah - Envelope Base Frequency (Write Only)

```
Bit7-0  Envelope Base Frequency, Fbase=1.79MHz/8/N
```

Fbase used by 4080h and 4084h. Volume/Sweep Envelope are disabled if N=0.

4080h - Volume Envelope (Write Only)

```
Bit7    Volume Envelope Mode      (0=Volume Envelope, 1=Fixed Volume)
```

```
  Bit6    Volume Envelope Direction (When enabled / at specified rate)
           0=Decrease Volume by 1 (only if Volume>00h)
           1=Increase Volume by 1 (only if Volume<20h)
  Bit5-0  When Bit7=1: Volume Level (0-20h=Muted-Loudest, 21h-3Fh=Same as 20h)
  Bit5-0  When Bit7=0: Volume Envelope Rate, F=Fbase/(N+1)
```
The volume level can be set to 00h-3Fh by write with Bit7=1, this level is also used as initial volume when switching to envelope mode by setting Bit7=0.
In decrease mode, initial values 21h-3Fh are resulting delayed decrease; volume stays at maximum level until the value gets smaller than 20h.

4084h - Sweep Envelope (Write Only)
```
  Bit7    Sweep Envelope Disable (1=Disable)
  Bit6    Sweep Envelope Mode    (0=Decrease, 1=Increase sweep gain)
  Bit5-0  When Bit7=1: Sweep Gain
  Bit5-0  When Bit7=0: Sweep Envelope Rate, F=Fbase/(N+1)
```

4085h - Sweep Bias (Write Only)
```
  Bit7    Not used
  Bit6-0  Sweep Bias (signed 7bit; -40h..+3Fh)
```
Sweep Bias is a used by the Modulation unit in calculating frequency bend.
Sweep Bias negative: Modulation unit will be bending frequency down.
Sweep Bias positive: Modulation unit will be bending frequency up.
Any write to Sweep Bias register resets Modulation Unit's address to zero. This address is used by the Modulation Unit when looking up entries written to the Modulation table (via $4088).

4086h - Modulation Frequency LSB (Write Only)
```
  Bit7-0  Lower 8bit of 12bit Modulation frequency
```
4087h - Modulation Frequency MSB (Write Only)
```
  Bit7    Modulation Enable/Disable (0=Enable, 1=Disable)
  Bit6-4  Not used
  Bit3-0  Upper 4bit of 12bit Modulation frequency
```
Modulation Unit: Modulation Rate (per entry of the 64-entry modulation table)
```
   F = 1.79MHz * ModFreq / 65536
```
If the 12bit frequency is zero, the Modulation unit is disabled.

4088h - Modulation Table (Write Only)
```
  Bit7-3  Not used
  Bit2-0  Modulation input
```
Writing to this register puts the value written at the END of the modulation table **twice**, and shifts each entry already in the table 2 places to the front.
The first 2 entries of the Modulation table are shifted out and lost.
old, old <-- ModTable_0 <-- ModTable_1 <-- ... <-- ModTable_63 <-- new, new

4090h - Current Volume Gain Level (6bit) (Read Only)
4092h - Current Sweep Gain Level (6bit) (Read Only)

4023h - 2C33 I/O Control Port
```
  Bit0  Disk I/O   (0=Disable, 1=Enable)
  Bit1  Sound      (0=Disable, 1=Enable)
```

FDS Sound by Disch, Release 1, 07/14/2004, based on info from Nori.

---- Sound Notes ----

Sweep Envelope and Modulation Units
Sweep Envelope unit behaves just like the Volume Envelope, only it alters Sweep Gain instead of Volume Gain. The Envelope Unit never pushes Sweep Gain above $20, but it still can get above $20 if set that way via $4084.
```
    Increase/Decrease mode is determined by bit 6 of $4084
```
Sweep Gain is used when calculating the Frequency change in the Modulation Unit...
The Modulation Unit, when clocked, takes 1 step through the Modulation Table (set by writes to $4088). The Sweep Bias is adjusted based on the 3-bit value in the table:
```
  0:Bias=Bias+0  1:Bias=Bias+1  2:Bias=Bias+2  3:Bias=Bias+4
  4:Bias=0       5:Bias=Bias-4  6:Bias=Bias-2  7:Bias=Bias-1
```
The address of the Modulation unit is incremented so that next clock it will use the next 3-bit value in the table. This address wraps at 64 and can be reset to zero by any write to $4085.
Sweep Bias wraps to fit within a signed 7-bit value, if it goes greater than 63, it wraps around to -64, and if it goes below -64, it wraps to 63.
The Modulation Unit works by altering the Frequency of the Main Unit by a value calculated from the Sweep Gain and Sweep Bias values:
```
  temp = Sweep_Bias * Sweep_Gain;
  if temp AND 0Fh then
    if Sweep_Bias<0 then temp=temp-10h else temp=temp+20h
  temp=temp/10h
  if temp>193 then temp -= 258;  // not a typo... for some reason the wraps
  if temp<-64 then temp += 256;  //    are inconsistent
  Mod = Freq * temp / 64;
```
In this code, Freq is the 12-bit MAIN UNIT frequency, and Mod is the amount that frequency is altered. This generated 'Mod' value is used in the frequency calculation of the main unit (given earlier):

```
Hz = NES * (Freq + Mod) / 65536
```

If at any time the Modulation unit is off, 'Mod' is zero. Otherwise 'Mod' is the above calculated value. If Freq + Mod produces a number less than or equal to zero, the channel is presumably silenced.

Unit Activity

There are many factors that could disable a unit, here's an overview section to cover all the needed requirements for the channel to be active.
Remember that each unit can be active regardless of the activitiy of other units. For example... even though the main unit is off and the channel is silent, this does not mean the Volume Envelope or Modulation units are inactive.
If any of the supplied conditions are false... the unit is inactive and will not be clocked. All conditions must be true for the unit to be active.

Volume Envelope Unit:

```
- Volume Envelope must be enabled (bit 7 of $4080 must be off)
- Envelope Speed must be nonzero (set by $408A)
- Envelope must be enabled (bit 6 of $4083 must be off)
```

Sweep Envelope Unit:

```
- Sweep Envelope must be enabled (bit 7 of $4084 must be off)
- Envelope Speed must be nonzero (set by $408A)
- Envelope must be enabled (bit 6 of $4083 must be off)
```

Modulation Unit:

```
- Modulation must be enabled (bit 7 of $4087 must be off)
- Modulation frequency must be non-zero (set by $4086/$4087)
```

Main Unit (Wave RAM Sound Output):

```
- Main Unit must be enabled (bit 7 of $4083 must be off)
- Main Unit Frequency must be non-zero (set by $4082/$4083)
- 'Freq + Mod' must be greater than zero (see Frequency Calculation section)
- Write Mode must be off (bit 7 of $4089 must be off)
```

FDS BIOS Disk Format

Each disk has two sides, each side having a capacity of circa 64K, typically less than 64K data because some space is used for gaps and headers, also the exact capacity may vary depending on the transfer rate/rotation speed of the drive that has recorded the disk. To change an active side, the disk has to be removed, flipped, and inserted back into the drive.

The drive doesn't support random access, and the disk is NOT split into tracks and sectors. The data is stored sequentially on a single "track" which is wound in a spiral, starting at the outer edge, towards the center of the disk.

Side Header Block (56 bytes) (1st block on disk)

```
00h      Block Type     (01h)
01h-0Eh  Disk ID        (Must be ASCII string "*NINTENDO-HVC*")
0Fh      Maker ID
10h-13h  Game Name      (usually 4 letter ASCII)
14h      Version Number (usually 00h)
15h      Side Number    (00h=Side A, 01h=Side B)      (00h=bootable)
16h      Disk Number    (00h=First, 01h=Second, etc.) (00h=bootable)
17h-18h  Extra Disk ID Field
19h      Highest File ID for Boot files (all files with File ID's less
         or equal than this value are loaded automatically on power-up)
1Ah-37h  Reserved Space (30 bytes, ignored by BIOS)
```

File Number Block (2 bytes) (2nd block on disk)

```
00h      Block Type     (02h)
01h      Number of Files on this side
```

File Header Block (16 bytes) (for each file, 3rd,5th,7th... block on disk)

```
00h      Block Type     (03h)
01h      File Number    (00h=First file on this side, 01h=Second, etc.)
02h      File ID        (used to access files by Load Files function)
03h-0Ah  File Name      (not used, the BIOS access files by above File ID)
0Bh-0Ch  Target Address (LSB, MSB)
0Dh-0Eh  File Size      (LSB, MSB)
0Fh      Target Area    (00h=WRAM, Other=VRAM)
```

File Data Block (1+LEN bytes) (for each file, 4th,6th,8th... block on disk)

```
00h      Block Type     (04h)
01h-LEN  Data           (LEN=File Size in File Header Block)
```

Gaps, Start Bits, CRC Values

Each block is preceded by a GAP (a stream of "0" bits), followed by a Start Bit ("1"), followed by the actual bytes contained in the block, followed by a 16bit CRC value.

FDS Disk Images

Disk Images should have extension ".fds" and are having a 16-byte header:

```
00h 4  File ID  ("FDS",1Ah)
04h 1  Number of Sides (usually 1 or 2; or more, eg. in Gunfight)
05h 11 Reserved (00h-filled)
```

Followed by the Disk Image (65500 bytes per side, padded with 00h if less bytes are used). The image contains raw data (without low level information like GAP-lengths, leading start bits, or trailing checksums), it should be usually starting with the 56-byte Side Header Block (ie. 01h,"*NINTENDO-HVC*", etc.).

FDS BIOS Disk Functions

Disk Boot

On power-up, the FDS does call the Load Files function to load the boot files. The DiskID is FFh-filled (wildcards), except the Side Number and Disk Number entries which must be both 00h on boot-able disks. LoadList is empty, indicating to load all boot files, ie. all files with File IDs less or equal than the Disk Header's Boot ID value.

There must be at least two boot files on the disk, one containing the program with entrypoint (16bit pointer, which must be loaded to DFFCh), the other file containing the Nintendo License string (E0h bytes, which must be loaded to PPU 2800h, and which is verified against a copy in BIOS at ED37h).

DiskID

Used by most BIOS functions to ensure that the correct disk/side is inserted. DiskID consists of 10 bytes which are compared against Disk Header Block [0Fh..18h]. Each DiskID byte may be set to FFh, which is used as wildcard, comparision always passes okay for that bytes. If the comparision does not match then error codes 04h..10h are returned, indicating which entry didn't match.

E1F8h - Load Files

The function scans <all> files on disk, respectively, the execution time is the same, no matter how many/how large files are loaded. On the contrary, multiple calls to LoadFiles would be unneccessarily slow.

```
RETaddr:        pointer to DiskID
RETaddr+2:      pointer to LoadList
A on return:    error code
Y on return:    count of files actually found
```

LoadList is a list of up to 20 File IDs, if the list contains less than 20 IDs then it must be terminated by FFh. If LoadList is empty (FFh in the first byte) then the boot files are loaded, that are all files with File IDs less or equal than the Disk Header's Boot ID value.

E32Ah - Get Disk Information

```
RETaddr:        pointer to DiskInfo
A on return:    error code
```

The DiskInfo pointer should point to a free memory location, which will receive the following data:

```
0Ah bytes   Disk Header Block [0Fh..18h], manufacturer, disk name, etc.
1   byte    File Number Block [01h], number of files on disk (N)
N*9 bytes   File Header Block [02h..0Ah], File ID and Filename, for each file
2   bytes   Disk Size (MSB,LSB)
```

Disk size is equal to the sum of each file's size entry, plus an extra 261 per file.

E237h - Append File

Sets A=FFh (append after last file), ie. same as A=FileCount, then continues at E239h (Write File).

E239h - Write File

Register A specifies how many old files are to be kept preserved on disk, these files are read/skipped, and the new file is then written to disk after those files, and the disks FileCount is set to A+1, making the new file to be the last file on disk, any further files are deleted/hidden.

In a second cycle, the written data is verified, if the verification fails (error 26h), then the file count is decremented, ie. the new file is deleted.

```
RETaddr:        pointer to DiskID
RETaddr+2:      pointer to FileInfo
A on call:      File Number  (00h=First) (FFh=Append after last file)
A on return:    error code
```

FileInfo occupies 17 bytes, the first 14 bytes contain File Header entries [02h..0Fh], ie. File ID, Filename, Load Address, Filesize, and Load Area.

The last 3 bytes contain the Source Address and Source Area (which may or may not be same as Load Address and Load Area).

E2BBh - Adjust File count

```
RETaddr:        pointer to DiskID
A on call:      number to reduce current file count by
A on return:    error code
Special error:  #$31 if A is less than the disk's file count
```

Reads in disk's file count, decrements it by A, then writes the new value back.

E2B7h - Check File count

```
RETaddr:        pointer to DiskID
A on call:      number to set file count to
A on return:    error code
Special error:  #$31 if A is less than the disk's file count
```

Reads in disk's file count, compares it to A, then sets the disk's file count to A.

E305h - Set File count (alt. 1)
```
 RETaddr:        pointer to DiskID
 A on call:      number to set file count to
 A on return:    error code
```
Sets the disk's file count to A.

E301h - Set File count (alt. 2)
```
 RETaddr:        pointer to DiskID
 A on call:      number to set file count to minus 1
 A on return:    error code
```
Sets the disk's file count to A+1.

Don't expect disk calls to return quick; it may take several seconds to complete. The ROM BIOS always uses disk IRQ's to transfer data between the disk, so programs must surrender IRQ control to the ROM BIOS during these disk calls. The value at [$0101] however, is preserved on entry, and restored on exit.

WRAM Target Addresses
Target Addresses should be in range 0200h-07FFh or 6000h-DFFFh. The BIOS rejects (silently ignores) most attempts to load data to 0000h-01FFh. In particular, it rejects Target Addresses at 0-1FFh (including mirrors at 800h,1000h,1800h), it also rejects wraps from FFFFh to 0000h.
However, it does not reject wraps to mirrors (eg. from 7FFh to 800h), clever use of this feature might allow to modify values on stack, and to bypass the Boot License.
Furthermore, target address 2000h can be used to enable NMIs during loading, the games Bislot, Bisyosya, and Bishojo Control are using this trick to abort the boot process and to start the game - without Boot License - via NMI vector at [DFFAh].

VRAM Source/Target Addresses
Mind that the screen should be disabled when loading/writing VRAM data, Port 2001h/Bit3-4 should be zero, otherwise VRAM could be accessed only in VBlank. Mind that physical content of VRAM addresses 2400h-2BFFh changes depending on current mirroring. The BIOS re-initializes parts of VRAM after loading the boot files on power up, overwriting any boot-files loaded to that areas.

Boot License String
32x7 characters in VRAM 2800h-28DFh (and copy in BIOS at ED37h)
```
  "           NINTENDO r           "
  "       FAMILY COMPUTER TM       "
  "                                "
  "  THIS PRODUCT IS MANUFACTURED  "
  "  AND SOLD BY NINTENDO CO;LDT.  "
  "  OR BY OTHER COMPANY UNDER     "
  "  LICENSE OF NINTENDO CO;LTD..  "
```
By using Non-ASCII BIOS Tile Numbers: A..Z=0Ah..23h SPC=24h .=26h ;=27h r=28h.
See WRAM Target Addresses above for methods to bypass the Boot License.

FDS BIOS Disk Errors

Error codes are returned in both A and X registers (plus zero flag, Z=okay)
```
  00h Okay (no error) (zero flag set)
  01h No disk inserted (Port 4032h, Bit0)
  02h No battery/power (Port 4033h, Bit7)
  03h Disk write-protected (Port 4032h, Bit2)
  04h Bad Side Header [0Fh], Maker ID
  05h Bad Side Header [10h..13h], Game name
  06h Bad Side Header [14h], Game version
  07h Bad Side Header [15h], Side number (flip the disk)
  08h Bad Side Header [16h], Disk number
  09h Bad Side Header [17h], Extra ID Value 1
  10h Bad Side Header [18h], Extra ID Value 2
  20h Bad Nintendo License String (must be loaded to PPU 2800h-28DFh on boot)
  21h Bad Side Header [01h..0Eh], Disk ID (must be "*NINTENDO-HVC*")
  22h Bad Side Header [00h], Block ID must be 01h
  23h Bad File Number [00h], Block ID must be 02h
  24h Bad File Header [00h], Block ID must be 03h
  25h Bad File Data   [00h], Block ID must be 04h
  26h Write-Verify Error (verification of written data failed)
  27h Block CRC Read Failure (Port 4030h, Bit 4)
  28h Lost Data (Port 4030h, Bit 6), CPU didn't read from 4031h in time
  29h Lost Data (Port 4030h, Bit 6), CPU didn't write to 4024h in time
  30h Disk Full (Port 4032h, Bit 1), Disk head has reached most inner track
  31h Data number of a disk doesn't match up (?)
```

FDS BIOS Data Areas in WRAM

The BIOS uses several places in memory, but only some of them are expected to be maintained by game code.

```
  Addr  Size  Expl.
```
Scratch Area (destroyed by any calls to BIOS disk functions)
```
  0000h  2    first 16bit parameter
  0002h  2    second 16bit parameter
  0004h  1    previous stack frame
  0005h  1    error retry count
  0006h  1    file counter
  0007h  1    current block type
  0008h  1    boot ID code
  0009h  1    dummy read flag
  000Ah  2    16bit destination address
  000Ch  2    16bit transfer length count
  000Eh  1    file found counter
```
Copies of I/O Ports (used to READ content of Write-Only I/O Ports)

The BIOS does (and the game should) keep these bytes in sync with the ports.
```
  00F9h  1    value last written to [$4026]   $FF on reset (disk ext connector)
  00FAh  1    value last written to [$4025]   $2E on reset (disk control)
  00FBh  1    value last written to [$4016]   0'd on reset (joypad)
  00FCh  1    value last written to [$2005]#2 0'd on reset (ppu scrolling)
  00FDh  1    value last written to [$2005]#1 0'd on reset (ppu scrolling)
  00FEh  1    value last written to [$2001]   $06 on reset (ppu control)
  00FFh  1    value last written to [$2000]   $80 on reset (ppu control)
```
IRQ/NMI/Reset Control
```
  0100h  1    Action on NMI   (set to C0h on reset)
  0101h  1    Action on IRQ   (set to 80h on reset)
  0102h  2    Action on Reset (AC35h after disk-boot, 5335h after warm-boot)
```
IRQ/NMI/Reset Vectors
```
  DFF6h  2    Game NMI vector 1, used if [0100h]=01xxxxxxb
  DFF8h  2    Game NMI vector 2, used if [0100h]=10xxxxxxb
  DFFAh  2    Game NMI vector 3, used if [0100h]=11xxxxxxb
  DFFCh  2    Game Reset vector, used if [0102h]=5335h or =AC35h
  DFFEh  2    Game IRQ vector,   used if [0101h]=11xxxxxxb
```
If [0100h..0102h] don't match then the IRQ/NMI/Reset is handled internally by the BIOS, without using (and without changing) the Game vectors.
Address DFFCh contains the initial entrypoint at disk boot, and is also used as warm-boot vector when pushing the Reset button at a later time.

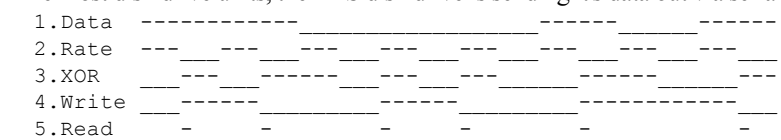
There may be more structured data areas in the zero page (for example, the BIOS joypad routines use $F5..$F8 for storing controller reads), but only the listed ones are used by the disk call subroutines.

FDS Disk Drive Operation

When the head reaches the end of the disk (most inner track), it returns to the beginning of the disk (most outer track) and the cycle repeats, upon request from the RAM adaptor. This means that on every scan, the entire disk is read (which takes about 6 seconds). The disk drive signals the RAM adaptor when the head has been positioned to the outer most track, and is starting a new scan.

FDS data transfer protocol

Like most disk drive units, the FDS disk drive is sending it's data out via serial connection.



The 1st row shows a 8bit data value, in this example A3h, or 10100011b, transferred LSB first. The 2nd row shows the transfer rate clock. The data/rate signals are XORed, as shown in the 3rd row. The actual signal written to disk is shown in the 4th row, magnetic polarity changes on any Low-to-High transitions of the XOR-signal. When reading from disk, spikes of one microsecond length are received on any polarity changes, as shown in the 4th row.

The RAM adaptor expects a transfer rate of 96.4kHz, although the tolerance it has for this rate is +/- 10%. This tolerance is neccessary since, the disk drive can NOT turn the disk at a constant speed.

First GAP

The length of the first GAP period present on typical FDS disks (relative to the instant the disk drive's "-ready" signal is activated) is about 40000 bits, after which the first block start mark (indicating the beginning of the first file) will appear.
The disk drive unit signals the RAM adaptor when the head has moved to the beginning of the disk via the "-ready" signal it sends out (more on this later).
The "-ready" signal is based on a mechanical switch inside the drive which is activated when the head is brought back to the outer most edge of the disk (the beginning). Because the switch will usually be triggered prematurely, the first 13000 bits (approx.) of data the drive will send out immediately after this switch is activated will be invalid. To compensate for this, the RAM adaptor purposely ignores the first 26100 bits (approx.) sent to it since it recieves the "-ready" signal from the disk drive.

Further GAPs

The typical GAP period size used between files on FDS disks is roughly 976 bits (this includes the bits that are ignored by the RAM adaptor).
the RAM adaptor always ignores the first 488 bits (approx.) to follow after the immediate end of any file. This period allows the RAM adaptor (or the game rather) an oppertunity to make the switch from reading from the disk to writing or vice-versa.

Final "GAP"
- The rest of the disk is filled with 0's after the last file is recorded (although it really shouldn't matter what exists on the disk after this).

CRC calculation
CRC appended to the immediate end of every file. The CRC is 16-bits, and is generated with a 17 bit poly. The poly used is 10001000000100001b (the X25 standard). Right shift operations are used to calculate the CRC (this effectively reverses the bit order of the polynomial, resulting in the 16-bit poly of 8408h). The file this algorithm is designed to work on has no block start mark in it ($80), and has 2 extra bytes at the end (where a CRC calculation would normally reside) which are 0'd. While the block start mark is actually used in the calculation of a FDS file CRC, you'll see in the algo below that the block start mark ($80) is moved directly into a register.

```
  // ax is used as CRC accumulator
  // si is the array element counter
  // di is a temp reg
  // Size is the size of the file + 2 (with the last 2 bytes as 0)
  // Buf points to the file data (with the 2 appended bytes)
   mov  ax,8000h        // this is the block start mark
   sub  si,si           // zero out file byte index ptr
  @@lop1:
   mov  dl,byte ptr Buf[si]
   inc  si
   REPT 8
     shr  dl,1; rcr ax,1; sbb di,di; and di,8408h; xor ax,di
   ENDM
   cmp  si,Size
   jc   @@lop1
```

// ax now contains the CRC.
Special thanks to Val Blant for assistance in cracking the CRC algorithm used by the FDS.

- Some unlicenced FDS games activate the "-stop motor" signal (and possibly even "-write", even though the storage media is not intended to be written to) when a media transfer is to be discontinued, while "-scan media" is still active. While this is an unorthodoxed method of doing this, the best way to handle this situation is to give the "-stop motor" signal priority over any others, and force data transfer termination during it's activation.

- Check out the FDS loader project (which uploads *.FDS files to the RAM adaptor) for source code to my working disk drive emulator.

Hardware disk copy protection
Apparently, Nintendo had designed FDS disk drive units so that they cannot reprogram entire disks, while still somehow being able to write the contents of individual files to the end of disks. Now, there's alot of undocumented things going on inside the disk drive unit, so I'm just going to say that there are two evil IC's you've got to watch out for inside the FDS disk drive- the 3213, and the 3206. There is a collection of 6 "FDS-COPY" jpegs over at NESdev which (pg. 4 right side, and pg. 5) give a pretty graphic overview of the steps involved in modding a stock FDS disk drive, so that it may reprogram disks. Although I haven't built the specific circuit described in the jpegs, I had designed & built a similar working circuit to defeat the FDS's evil copy protection circuitry, with excellent results.

Software disk copy protection
Special thanks to Chris Covell for bringing this to attention.
Apparently, some FDS disks implement a very simple copy protection scheme, which the game relies on in order for the game to refuse to work on the copied disk. Normally, the number of files that exist on an FDS disk is stored in the second block recorded on it. However, some games maintain "invisible" files, which are basically files that exist beyond what the file count number in the file count block indicates. This poses somewhat of a problem for copy software like FDSLOADR, since these tools rely on the file count block, and don't assume that there is any valid data past the last file found on the disk. This means that when these types of disks are copied, the invisible files will be lost, and when the game loads the files that do exist, the game's going to give the user heat about there being a file missing or somthing, gumming up the works. However in practice, when an FDS disk is programmed, the unused end of the disk is usually completely zeroed out, and this makes detecting the end of the disk simple: just wait to find a GAP period of extreme length. Except in rare cases, this model for detecting the true end of an FDS disk should generally provide the best results for copying the complete contents for all types of FDS disks.
[That may be as well a trick to improve disk boot speed, not necessarily a copy protection.]

Physical disk lockout mechanism
Ever wonder why Nintendo engraved their company's name along the handle edge of all FDS disks? Inside the FDS disk drive bay, sitting just behind the lower part of the front black plastic faceplate, is a little plastic block with the letters "Nintendo" carved out of a hard plastic block. This basically forces disks that don't have holes in those locations from completely loading into the drive, circumventing usage. Now while many companies made FDS disks with those holes cut out, I'm sure there must be some disks out there that are compatable with the FDS, but don't have the holes. So, the solution is to simply disassemble the FDS disk drive, remove the disk cage, and remove the two screws securing the "Nintendo" letterblock.

VS System

Nintendo's VS Systems are arcade machines with same chipset as NES consoles. There are two versions:

```
  VS UniSystem   --> 1-2 players (1 Monitor,  1 CPU,  1 PPU,  1 ROM-Set)
  VS DualSystem  --> 2-4 players (2 Monitors, 2 CPUs, 2 PPUs, 2 ROM-Sets)
```

The DualSystem essentially contains two NES consoles that can be linked together (or optionally can be used as two independend consoles, each one running a different game).

VS System Controllers

Controller Ports

Port 4016h/Write:

```
Bit0    Joypad Strobe (as usually)
Bit1    VS Dualsystem: Send IRQ to other CPU (0=No, 1=IRQ)
Bit2    Select 8K VROM bank at PPU 0000h-1FFFh (Mapper 99 games only)
```

Port 4016h/Read:

```
Bit2    Credit Service Button        (0=Released, 1=Service Credit)
Bit3-4  DIP Switch 1-2               (0=Off, 1=On)
Bit5-6  Credit Left/Right Coin Slot  (0=None, 1=Coin) (Acknowledge via 4020h)
Bit7    VS Dualsystem: Master/Slave ID (0=Slave CPU, 1=Master CPU)
```

Port 4017h/Read:

```
Bit2-7  DIP Switch 3-8               (0=Off, 1=On)
```

Port 4020h/Write:

```
Bit0    Acknowledge Coin Slot Signal (0=Normal, 1=Acknowledge Coin)
```

Memory at 6000h-67FFh:

```
2Kbyte of shared RAM for VS Dualsystem CPU-to-CPU communications.
Access, according to schematic:
  Owner depends on OUT-1 output from MASTER(?) CPU ;<--that is: "2J" CPU
  aka IRQ input from SLAVE(?) CPU (or vice-versa?) ;<--that is: "8J" CPU
Typically handshake involves IRQ from Master CPU, followed by a
response-IRQ from Slave CPU.
```

Joysticks

The Joysticks are working like normal Joypads, but with different bit assignments, Start/Select are renamed to Button 1-4, and controls for Player 1 and 2 are exchanged:

Read	NES/4016h	VS/4016h	NES/4017h	VS/4017h
1st	Button A (1)	Button A (2)	Button A (2)	Button A (1)
2nd	Button B (1)	Button B (2)	Button B (2)	Button B (1)
3rd	Select (1)	Button 1	Select (2)	Button 2
4th	Start (1)	Button 3	Start (2)	Button 4
5th	Up (1)	Up (2)	Up (2)	Up (1)
6th	Down (1)	Down (2)	Down (2)	Down (1)
7th	Left (1)	Left (2)	Left (2)	Left (1)
8th	Right (1)	Right (2)	Right (2)	Right (1)

Reportedly, some VS games have those inputs assigned differently?

Controllers - Joypads

Lightguns

VS Lightguns are more or less same as NES Zappers, but the signals are injected to the "joypad" shift registers (the normal NES-like connection would conflict with VS DIP-Switches), for details on both VS and NES guns, see:

Controllers - Lightguns (Zapper)

Coin Slots

There are two coin slots (maybe for different coins?). Some games allow to change the number of coins per game via DIP-Switches; ie. in some cases one needs to insert more than one coin before the game starts (or push the coin button more than once in emulators). Credit Service Button (inside the cabinet) typically adds a free game.

Exact purpose of the Coin Acknowledge output is unknown.

At least some VS games (eg. Clu Clu Land) do accept Coin signals only if they fall within whatever min/max lengths; there is no such timing restriction for the service button.

DIP-Switches

Allow to configure the games. Such like: Skipping the title screen, muting sound effects, changing difficulty, changing PPU palette (on third-party games), changing coins per game.

Some Daughterboards do contain additional DIP-Switches (used to change capacity of the ROMs).

VS System Games

VS System Games

```
  PPU          Mapper   Title
 *RP2C04-0001 MMC3+?   Atari RBI Baseball
  RP2C04-0003 DUAL     Balloon Fight (... Dualsystem only?)
  RP2C04-0001 DUAL     Baseball (... Dualsystem only?)
```

```
 *RP2C04-0001 -         Battle City
  RP2C04-0002 UNROM     Castlevania
  RP2C04-0004 -         Clu Clu Land
  RP2C04-0003 MMC1      Dr. Mario
  RC2C03B     -         Duck Hunt (Lightgun) (one version)
  RC2C03C     -         Duck Hunt (Lightgun) (other/same version)
  RP2C04-0003 -         Excite Bike (instructions in demo-mode) (Nintendo) 1984
  RP2C04-0004 -         Excite Bike (status-bar in demo-mode) (Nintendo) 1984
  RP2C04-0001 MMC3      Freedom Force (Lightgun) (1988 Sunsoft)
  RP2C04-0002 -         Golf
  RC2C03B ?   -         Golf (Japan version?) XXX doesn't match ANY palette??
  RP2C04-0003 VRC1      Goonies
  RP2C04-0001 VRC1      Gradius
  RC2C05-03   40K+16K   Gumshoe (Lightgun)
  RP2C04-0001 -         Hogan's Alley (Lightgun)
  RP2C04-0004 -         Ice Climber (Unisystem)
  RP2C04-0004 DUAL      Ice Climber (Dualsystem) (JAPAN) "splitscreen-vertical"
  RP2C04-0002 -         Ladies Golf
  RP2C04-0002 -         Mach Rider (Endurance Course version)
  RP2C04-0001 -         Mach Rider (Fighting Course version) (Japan version)
  RC2C03B     DUAL      Mahjang (... Dualsystem only?)
  RC2C05-02   -         Mighty Bomb Jack (Japan)
  RC2C05-01   -         Ninja Jajamaru Kun (Japan)
  RP2C04-0001 -         Pinball
  RC2C03B     -         Pinball (Japan)
  RP2C04-0001 Sunsoft3  Platoon
  RP2C04-0002 DUMMY     Raid on Bungeling Bay (Japan)
  RP2C04-0002 -         Slalom
  RP2C04-0002 -         Soccer (japan version)
  RP2C04-0003 -         Soccer (other version)
 *RC2C03B     -         Star Luster
  RP2C04-0004 -         Super Mario Bros.
  RP2C04-0004 -         Super Mario Bros. (alternate version)
  RP2C04-0004 -         Super Mario Bros. (Super Skater/Skate Kids CHR-ROM hack)
 *RP2C04-0001 MMC3      Super Sky Kid
 *RP2C04-0001 MMC3+?    Super Xevious
  RC2C03B     DUAL      Tennis (... Dualsystem only?)
 *RP2C04-0001 -         Tetris (by Tengen)
 *RP2C04-0003 MMC3+?    TKO Boxing
  RC2C05-04   UNROM     Top Gun
  RP2C04-0002 DUAL      Wrecking Crew (... Dualsystem only?)
  RC2C05-05   ?         (this PPU is used by which game ???) (does it exist?)
  RP2C03B     ?         (this PPU is used by Playchoice 10; not by VS System)
  RP2C03G     ?         (this PPU is used by which game ???) (does it exist?)
```

Note: The "*" marked entries are third-party games that do support different PPUs (typically selected via DIP switch 6-8, the PPUs listed in the above table are used when all DIP switches are OFF).

VS System Mappers

Below should be the correct mapper numbers used by VS games. Observe that ROM-images do often contain incorrect mapper numbers (probably because there is no automatic tool for dumping the VS ROM-Sets) (in worst case, the .NES header is even missing the VS-flag).

```
  -         Mapper 99 (standard VS System mapper; games without daughterboard)
  40K+16K   Mapper 99 (with some extension to access 40K PRG-ROM)
  DUAL      Mapper 99 (Dualsystem, requires two ROM-sets, two CPUs/PPUs etc.)
  DUMMY     Mapper 99 (single-player, but requires Dummy PRG-ROM on second CPU)
  MMC1      Mapper 1  (MMC1, or actually some pre-MMC1 74xxx-logic)
  MMC3      Mapper 4  (MMC3, or actually some third-party pre-MMC3 chip)
  MMC3+?    Mapper 4  (plus protection chip at 5Exxh or 5xxxh)
  Sunsoft3  Mapper 67 (Sunsoft-3 chip)
  UNROM     Mapper 2  (UNROM)
  VRC1      Mapper 75 (Konami VRC1 chip)
```

The "Mapper 99" games consist of several 8Kbyte EPROMs (to be mounted directly on the mainboard). The other games use special daughterboards (to be mounted between CPU/PPU and mainboard).

The "MMC3" chips are actually some sort of pre-MMC3 28pin Shrink-DIP chips from Namco, with part number "108 JAPAN", so correct mapper number would be "Mapper 206" or so?

VS System PPUs

```
  RC2C03B (standard palette)                    ;\standard palette
  RC2C03C (standard palette)                    ;/
  RC2C05-01 (with ID ([2002h] AND xxh)=?)       ;\
  RC2C05-02 (with ID ([2002h] AND 3Fh)=3Dh)     ; standard palette, but with
  RC2C05-03 (with ID ([2002h] AND 1Fh)=1Ch)     ; swapped port 2000h/2001h,
  RC2C05-04 (with ID ([2002h] AND 1Fh)=1Bh)     ; and Chip ID in port 2002h
  RC2C05-05 (with ID ([2002h] AND xxh)=?)       ;/
  RP2C04-0001 (special palette 1)               ;\
  RP2C04-0002 (special palette 2)               ; special palettes
  RP2C04-0003 (special palette 3)               ;
  RP2C04-0004 (special palette 4)               ;/
```

For reference, below are some more RGB PPUs (although not used in VS System):

```
  RP2C03B (standard palette) ;<-- Playchoice 10 ;\standard palette, but not
  RP2C03G (standard palette) ;<-- used where?   ;/actually VS System related
```

And, RP2C03C and RC2C05-99 do exist (the above obscure RP2C03G and RC2C05-05 chip names might be actually mistyped names, and might be

actually being meant to mean RP2C03C and RC2C05-99).

Reportedly further VS games:

```
Babel no Tou               (by Namco, 1986)
Family Boxing              (by Namco/Wood Place, 1987; japanese "TKO Boxing")
Family Stadium '87         (by Namco, 1987; sequel to RBI Baseball)
Family Stadium '88         (by Namco, 1988; sequel to RBI Baseball)
Family Tennis              (by Namco, 1987)
Head to Head Baseball      (ever finished/released?, by Nintendo, 1986)
Lionex                     (prototype by Sunsoft, 1987)
Madura no Tsubasa          (prototype by Sunsoft, 1987)
Predators                  (prototype by Williams, 1984)
Pro Yakyuu Family Stadium  (by Namco, 1986; Japan version of RBI Baseball)
Quest of Ki                (by Namco/Game Studio, 1988)
Super Chinese              (by Namco/Culture Brain, 1988)
Toukaidou 53tsugi          (prototype by Sunsoft, 1985)
Trojan                     (by Capcom, 1987)
Urban Champion             (1984)
Volleyball                 (1986)
Walkure no Bouken          (by Namco, 1986)
Wild Gunman                (1984, light gun game)
```

and maybe Kung-Fu, Football, etc.

VS System PPUs and Palettes

PPU Name Tables

The VS System seems to be fitted with 4K VRAM. If that is true, and if there's no way to disable that, then VS games are probably always running in Four-Screen mode. Ie. ignore any different mirroring settings in .NES file header, and ignore any Name Table mode selections via VRC1/MMC1/MMC3/Sunsoft3 mappers.

PPU ID Codes

There are some PPUs with swapped control port addresses, and IDs in lower 5bit of the PPU status port.

```
RC2C05-01 (with ID ([2002h] AND xxh)=?)       ;\
RC2C05-02 (with ID ([2002h] AND 3Fh)=3Dh)     ; standard palette, but with
RC2C05-03 (with ID ([2002h] AND 1Fh)=1Ch)     ; swapped port 2000h/2001h,
RC2C05-04 (with ID ([2002h] AND 1Fh)=1Bh)     ; and Chip ID in port 2002h
RC2C05-05 (with ID ([2002h] AND xxh)=?)       ;/
```

Unknown: RC2C05-05 does it really exist, which game is using it, what's its ID?
Unknown: RC2C05-01 what ID does it have (if any) (game doesn't check it)?
Unknown: RC2C05-03 game wants a 6bit ID, does that PPU support the "Sprite Lost" flag, if yes, why/how is it set in the ID check? Maybe always initially set on Reset, or always occuring shortly after reset (in case OBJ y-coordinates are initially all same)?

PPU Palettes

The "Standard" Palette does resemble the NES palette. The other four palettes do have these colors rearranged, and have some additional colors.
All five VS palettes are much more colorful than the "pastelized" NES palette (roughly same as on a NES when setting the TV Set to MIN Brightness & MAX Color).
The PPU's Color Emphasis Bits are also different as on NES PPUs: On a VS-PPU, they are forcing the selected color(s) to max brightness. For example, when setting all three Emphasis Bits, the whole screen will become white (on a NES, the normal picture would be kept displayed at reduced brightness).

Below palette tables are showing 3:3:3 bit RGB values (eg. 700=Red, 070=Green, 007=Blue).

RP2C04-0001, Special Palette 1

```
755,637,700,447,044,120,222,704,777,333,750,503,403,660,320,777
357,653,310,360,467,657,764,027,760,276,000,200,666,444,707,014
003,567,757,070,077,022,053,507,000,420,747,510,407,006,740,000
000,140,555,031,572,326,770,630,020,036,040,111,773,737,430,473
```

RP2C04-0002, Special Palette 2

```
000,750,430,572,473,737,044,567,700,407,773,747,777,637,467,040
020,357,510,666,053,360,200,447,222,707,003,276,657,320,000,326
403,764,740,757,036,310,555,006,507,760,333,120,027,000,660,777
653,111,070,630,022,014,704,140,000,077,420,770,755,503,031,444
```

RP2C04-0003, Special Palette 3

```
507,737,473,555,040,777,567,120,014,000,764,320,704,666,653,467
447,044,503,027,140,430,630,053,333,326,000,006,700,510,747,755
637,020,003,770,111,750,740,777,360,403,357,707,036,444,000,310
077,200,572,757,420,070,660,222,031,000,657,773,407,276,760,022
```

RP2C04-0004, Special Palette 4

```
430,326,044,660,000,755,014,630,555,310,070,003,764,770,040,572
737,200,027,747,000,222,510,740,653,053,447,140,403,000,473,357
503,031,420,006,407,507,333,704,022,666,036,020,111,773,444,707
757,777,320,700,760,276,777,467,000,750,637,567,360,657,077,120
```

All other VS System PPUs reportedly use this (or similar?) Standard Palette

```
333,014,006,326,403,503,510,420,320,120,031,040,022,000,000,000
555,036,027,407,507,704,700,630,430,140,040,053,044,000,000,000
777,357,447,637,707,737,740,750,660,360,070,276,077,000,000,000
777,567,657,757,747,755,764,772,773,572,473,276,467,000,000,000
```

In the "standard" RGB palettes, color 0Dh is all black, which means that TWO of the composite NES gray-shades are missing on RGB PPUs. Note/caution: There is an "improved" RGB palette with THREE extra gray-shades (111,222,444) in the internet: that palette was invented/intended to make an open source emulator look "better" than real hardware - that "improvement" was adopted by many other emulators (probably without even being aware of the unreal grays).

French "RGB" (home-use NES consoles, not arcade)

France is using SECAM as television standard, frame rate is 50Hz (like PAL), but color encoding is different. The other big video standard in france appears to be to have TV Sets fitted with RGB input.

Like many other manufacturers, Nintendo didn't produce SECAM PPUs. Instead they have used a PAL to RGB converter attached to a standard PAL PPU. That pseudo "RGB" output is thus having same artifacts as composite PAL pictures.

VS System Protections

Unprotected Games

VS Games are consisting of sets of EPROMs without any special hardware (apart from custom cabinet front plates), the are no CIC lockout chips. That (and the relative high price of the games) made it quite inviting to upgrade the cabinets with illegal copies of newer games, or with unlicensed third-party games.

Nintendo Protections

To prevent piracy & unlicensed games, Nintendo has made a bunch of different PPUs: PPUs with different palettes, and PPUs with different Port 2000h/2001h/2002h.

Third-Party Protections

Thirty-Party games can be often DIP-switched to work with different palettes (thus bypassing Nintendo's protection). Instead, some thirty-party games do require some special "ID chip" mapped at 5xxxh or 5Exxh, and refuse to boot if the thing doesn't respond with expected values (see below for details).

Daughterboards

These are mainly used to access more memory. But, to some level they do also act as protection, since one needs to buy the boards. Some boards can be DIP-switched to work with different games with ROM capacites though.

Raid on Bungeling Bay

This game consists of seven 8Kbyte EPROMs. Six PRG/CHR EPROMs for first CPU, and one PRG EPROM for second CPU (without any CHR ROM here). This extra EPROM isn't doing anything useful (no sound/video output, and coprocessor-like maths), it's just doing a short handshake with the other CPU, and then it hangs in an endless loop.

The purpose is unknown; it may be some crude protection (won't work if the extra EPROM doesn't say "I am here"). Or maybe the game supports DUAL mode (and in UNI mode, requires the second CPU to say "I am NOT here").

Basically, the game needs a response IRQ from other CPU (with don't care response at 67E0h-67FFh), and additionally DIP 5 must be ON.

TKO Boxing Protection (Namco)

Protection unit is contained in a 28pin chip with part number "126 JAPAN".

```
[5E00h].Read   ;-reset data stream (returns unknown/dummy value)
[5E01h].Read   ;-return data stream (returns FFh,BFh,B7h,etc.)
```

TKO Boxing contains pre-computed 32 values in ROM:

```
FF,BF,B7,97,97,17,57,4F,6F,6B,EB,A9,B1,90,94,14    ;1st..16th read
56,4E,6F,6B,EB,A9,B1,90,D4,5C,3E,26,87,83,13,51    ;17th..32th read
```

That is, the initial value (FFh on first read) is XORed by following values:

```
40,08,20,00,80   ;XORed after 1st..5th read
40,18,20,04,80   ;XORed after 6th..10th read
42,18,21,04,80   ;XORed after 11th..15th read
42,18,21,04,80   ;etc.
42,18,21,44,88
62,18,A1,04,90
42
```

The exact way how that pattern is generated (and how it continues after 32 reads) is unknown. Note: The way how TKO Boxing is programmed, it CAN only verify the first 31 values, and actually DOES only verify first 7 values (unless there are further checks hidden "deeper" in the game).

Atari RBI Baseball Protection (Namco)

Protection unit is contained in a 28pin chip with part number "127 JAPAN".

```
[5E00h].Read   ;-reset data stream (returns unknown/dummy value)
[5E01h].Read   ;-return data stream (returns whatever...)
```

RBI Baseball verifies only two values: B4h on 5th read, and 6Fh on 10th read. This is different as in TKO Boxing (which would return 97h and 6Bh).

Super Xevious Protection (Namco)

Uses whatever protection chip (unknown part number).
The game is doing the following check upon Reset:
```
Write:  [5098h]=38h, [5132h]=9Eh, [5263h]=22h, [5300h]=90h
Verify: [54FFh]=05h, [5678h]=01h, [578Fh]=89h, [5567h]=37h
Write:  [5056h]=44h, [51C8h]=72h, [526Ah]=9Ah, [5300h]=FEh
Verify: [54FFh]=05h, [5678h]=00h, [578Fh]=D1h, [5567h]=3Eh
```
That can be faked by returning following values upon [5400h..57FFh] reads:
```
05h, 01h, 89h, 37h, 05h, 00h, D1h, 3Eh
```
Unknown how the hardware works in reality; it looks like four 8bit latches, and scrambled data, possibly by XORing data & address lines.

Controllers
Some VS games do reportedly some controller buttons exchanged with each other, so they can be played only when buying a special controller front panels or so. Unknown which games and which buttons that applies to.

Nintendo Playchoice 10

The Nintendo Playchoice 10 (PC10) arcade cabinets can contain up 10 NES games (on special cartridges). Aside from the NES-compatible hardware, the thing contains a Z80 CPU and a custom video circuit for handling game selection, coin/credits, game title/instructions display and bookkeeping.

PC10 Memory Map and I/O Ports

Z80 Memory Map
```
0000h-7FFFh  32K BIOS ROM Area (usually 16K chip, mirrored within 32K area)
8000h-87FFh  2K  Work RAM
8800h-8BFFh  1K  Lower 1K of Battery RAM
8C00h-8FFFh  1K  Upper 1K of Battery RAM (when disabled: mirror of Lower 1K)
9000h-97FFh  2K  Video RAM (write only, and disabled during video output)
9800h-BFFFh  10K Unused (open bus)
C000h-DFFFh  8K  Cartridge BIOS (resides on each game cartridge)
E000h-FFFFh  8K  Memory-mapped PROM I/O Ports
```

Z80 I/O Map - READ (by IN opcodes)
```
00h  Button/Status
  bit 0: Channel select button (aka Button 1) (0=Released, 1=Pressed)
  bit 1: Enter button          (aka Button 2) (0=Released, 1=Pressed)
  bit 2: Dual-Monitor with Reset button       (0=Released, 1=Pressed)
         Single-Monitor with Reset button     (0=Pressed, 1=Released)
         Single-Monitor without Reset button  (Always 0=No Reset Button)
           (Presence of the single monitor reset button works as so:
           The button may not be held down during power-up (obviously).
           The freeplay mode DIP-switch setting and R99 shunt-lead removal
           are accidentally causing the button-detection to be skipped.
           Bugfix: change the two jumps to address 0378h to address 036Bh)
  bit 3: Vblank NMI Occurred on NES Side (0=Yes, 1=No)
  bit 4: <zero>  unknown... is USED !  "R99 0 ohm Shunt Lead to GND" ?
          (allow to insert more coins for PRIME TIME ...?)
  bit 5: Coin 2 button         (0=None, 1=Coin Insterted)
  bit 6: Service button        (0=Released, 1=Pressed)
  bit 7: Coin 1 button         (0=None, 1=Coin Insterted)
01h  DIP-switch 1, Bits 0-7 (A..H) (0=Off, 1=On)
  bit 0..5: (A..F) Coinage (credits per left/right coin slot)
  bit 6:    (G)    Sound enable for Attraction/Demo mode (0=Off, 1=On)
  bit 7:    (H)    Automatic Selftest upon Power-Up      (0=Off, 1=On)
02h  DIP-switch 2, Bits 0-7 (I..P) (0=Off, 1=On)
  bit 0..5: (I..N) Timer Speed (credit decrease rate)
  bit 6:    (O)    Divide all coinage settings by 2 (see A..F)
  bit 7:    (P)    Freeplay Mode (only when bit0..6 = zero)
03h  Reading from this address sets Bit3 of read Port 00h back to 1=False
```
Port 04h..FFFFh are mirrors of Port 00h..03h.

Z80 I/O Map - WRITE (by OUT opcodes) (all write ports are using bit 0 only)
```
00h  VRAM Access                (0=by Z80 CPU, 1=by Video circuit)
01h  Game Controls              (0=Disable, 1=Enable)
02h  PPU RP2C03B Display output (0=Disable, 1=Enable)
```

```
  03h  APU N2A03 Sound output     (0=Disable, 1=Enable)
  04h  CPU N2A03 Reset            (0=Reset, 1=Run)
  05h  CPU N2A03 Stop             (0=Stop, 1=Run)
  06h  Display Output Select      (0=Z80/Video circuit, 1=NES/RP2C03B PPU)
       (Only on single monitor version)
  07h  Unknown...? is USED ! (even in single-monitor bios!)
         or, 06h,07h = both N/C
  08h  Z80 NMI Control            (0=Disable, 1=Enable)
  09h  Watchdog Control           (0=Enable, 1=Disable)
  0Ah  PPU RP2C03B Reset          (0=Reset PPU, 1=Run PPU)
  0Bh  Game Channel Select Bit0      ;\Slot Select (0-9) (0Ah..0Fh=Open Bus)
  0Ch  Game Channel Select Bit1      ; affects NES CPU/PPU memory,
  0Dh  Game Channel Select Bit2      ; and Z80 INST-ROM and PROM
  0Eh  Game Channel Select Bit3      ;/
  0Fh  Upper 1K of Battery RAM    (0=Disable, 1=Enable)
```

Below 4bit Ports in Single-Monitor version only (remaining time display):

```
  10h  7-Segment LED 4th Digit (LSB) ;\Four 74HC4511 BCD-to-7-segment drivers,
  11h  7-Segment LED 3rd Digit       ; and four GL-8E040 7-segment LED displays
  12h  7-Segment LED 2nd Digit       ; (bit0-3:00h..09h="0..9", 0Ah..0Fh=Blank)
  13h  7-Segment LED 1st Digit (MSB) ;/
```

Mirrors should be as so:

```
  10h..1Fh   Mirrors of Port 00h..0Fh (Dual-Monitor version only)
  14h..1Fh   Mirrors of Port 10h..13h (Single-Monitor version only)
  20h..FFFFh Mirrors of Port 00h..1Fh (all versions)
```

Watchdog

The watchdog is disabled (and its frame counter is reset to zero) when Port 08h and/or Port 09h are set to nonzero values; ie. the watchdog is active ONLY during periods when NMIs are off (Port 08h=0) and only when the Watchdog is on (Port 09h=0). When active, the watchdog resets the Z80 CPU after 8 frames.

Z80 Memory Mapped I/O at E000h..FFFFh - Ricoh RP5H01 serial 72bit PROM

Data Write:

```
  7-5  Unused
  4    PROM Test Mode (0=Low=6bit Address, 1=High=7bit Address)
  3    PROM Clock     (0=Low, 1=High) ;increment address on 1-to-0 transition
  2-1  Unused
  0    PROM Address Reset (0=High=Reset to zero, 1=Low=No Change)
```

Data Read and Opcode Fetch:

```
  7-5  Always set (MSBs of RST Opcode)
  4    PROM Counter Out (0=High=One, 1=Low=Zero) ;PROM Address Bit5 ;\both
  3    PROM Data Out    (0=High=One, 1=Low=Zero) ;PROM Data         ;/inverted
  2-0  Always set (LSBs of RST Opcode)
```

The 72bit PROM contains 9 bytes (8 "normal" bytes, and 1 "test" byte which was originally intended for testing purposes, but can be also mis-used to contain "normal" data). After resetting the address to zero, the the PROM will be (repeatedly) outputting the following bytes (LSB first):

```
  DATA (when TEST=0):   a, b, c, d, e, f, g, h, a, b, c, d, e, f, g, h
  DATA (when TEST=1):   a, b, c, d, e, f, g, h, i, i, 00,00,i, i, 00,00
  COUNTER OUT (always): 00,00,00,00,FF,FF,FF,FF,00,00,00,00,FF,FF,FF,FF
  (the PC10 mainboard inverts that signals, so Z80 will see inverse of above)
```

In the PC10, the PROM contains a decryption key, used for deciphering the INST-ROM content. Most of the decryption is done by code in the INST-ROM (so one could completely omit that part in homebrew INST-ROMs). The BIOS accesses the PROM in only two places: In the INST-ROM checksum calculation (which thus requires whatever stable data coming from the PROM). And, the Z80 NMI handler (which checks that CounterOut is High after 32 reads).

PC10 Video Circuit

PC10 VRAM (32x28 character "BG Map")

VRAM is located at 9000h-97FFh. VRAM is write-only, and can be accessed only when video output is disabled via Port OUT[00h], to avoid flickering, this should be usually done only during VBlank (ie. in the Z80 NMI handler).

Each character line occupies 40h bytes (20h words). The words are:

```
  Bit0-10  Character number (000h..7FFh) (usually only 000h..3FFh installed)
  Bit11-15 Palette number   (00h..1Fh)
```

The upper 2 lines and lower 2 lines are masked for V-Blank period, so actually used VRAM consists of 28 lines at 9080h..977Fh.

PC10 Charset (for the Z80 Video Circuit)

```
  '0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ.'!-"",:    abcdefghijkl+ ' ;000-03F
  '0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ.'!-"",:mnopq?()/=+ ' ;040-07F
  'abcdefghijklABCmEnoHIpqLMNOPrRSTUstuYv!wxyzr' ;080-0BF
  'stuvwxyz0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ,-!'()       ' ;0C0-0FF
  '0 1 2 3 4 5 6 7 8 9 TIMEInsertCoin                ' ;100-13F
  '   Tim       NearUp   FillUpTimeAnd        ' ;140-17F
  '' ;180-1BF
  '' ;1C0-1FF
  'ChannelSelectEnter.ButtonsGameTart                        ' ;200-23F
  '                                                          ' ;240-27F
  '                                                          ' ;280-2BF
  '                                                        ' ;2C0-2FF
```

```
'                               ' ;300-33F
'cntrlTM' ;340-37F
'' ;380-3BF
'c1986nintendo ' ;3C0-3FF
```

The 400h characters are stored in three 2764 chips (three 8Kx8 EPROMs): IC "8P"=CHR0, IC "8M"=CHR1, IC "8K"=CHR2 (each chip containing one color-plane of the 8-color characters). Optionally, for 800h characters, the mainboard could be fitted with 27128 chips.
The "" graphics symbols are used by the BIOS (and by instruction pages in Excitebike, Mario Bros, and Super Mario Bros). There are no japanese characters.

PC10 Palette (for the Z80 Video Circuit)

Below palette table is showing 4:4:4 bit RGB values (all digits are inverted, 0=Brightest, F=Darkest, eg. 0FF=Red, F0F=Green, FF0=Blue).

```
Pal <-------- Color 0..7 --------->     Pal <-------- Color 0..7 --------->
00: FFF,000,000,000,000,000,000,000     10: FAF,0BF,000,027,00B,FAF,FA9,F08
01: 94D,FFF,000,00B,024,1FF,AFF,FD8     11: FAF,A5D,303,707,00B,FAF,307,FFF
02: F9F,03F,FAF,000,000,000,FFF,FFF     12: 000,000,000,000,000,000,000,000
03: FFF,000,FFF,9FF,06F,0FF,047,F40     13: 000,000,000,000,000,000,000,000
04: 048,9FF,000,000,000,F95,FFA,FFF     14: 000,048,000,000,000,F9F,72F,FFF
05: F9F,0FF,F0B,00A,000,000,FFF,FFF     15: 000,000,000,000,000,000,000,000
06: F9F,88F,0FF,000,000,000,FFF,FFF     16: 000,000,000,000,000,000,000,000
07: F9F,11F,5FF,000,000,000,FFF,FFF     17: 000,FAF,000,000,000,FCF,039,FFF
08: 000,000,000,000,000,000,000,000     18: F8F,000,00B,000,000,036,FFF,FFF
09: 000,000,000,000,000,000,000,000     19: FAB,2CF,F07,029,111,FFF,4F1,F93
0A: 000,000,000,000,000,000,000,000     1A: 000,000,000,000,000,000,000,000
0B: 000,000,000,000,000,000,000,000     1B: FAF,000,FF5,FF0,F40,5AF,024,0FF
0C: 000,000,000,000,000,000,000,000     1C: FAF,0FF,046,6D9,008,FF4,F30,300
0D: 000,000,000,000,000,000,000,000     1D: FAF,0F9,F7F,838,415,7BE,27D,048
0E: 000,000,000,000,000,000,000,000     1E: FAF,0FF,000,02F,F4F,5DF,888,766
0F: 000,000,000,000,000,000,000,000     1F: F9F,007,000,00F,0FF,027,FFF,FFF
```

The 100h palette entries are containing 42h different colors. The Palette data is stored in three N82S129N chips (three 256x4bit PROMs): IC "6D"=Blue, IC "6E"=Green, IC "6F"=Red.

PC10 Character Sets - Available Characters per Color

The PC10's character set ROM contains four character sets, all having identically looking characters, but with different color numbers, and with different amounts of defined characters:

```
Charset0:  0123456789.'!-",:+?()/=ABCDEFGHIJKLMNOPQRSTUVWXYZabc..xyz
Charset1:  0123456789 '!- ,    ()  ABCDEFGHIJKLMNOPQRSTUVWXYZabc..xyz
Charset2:  0123456789.'!-",:+      ABCDEFGHIJKLMNOPQRSTUVWXYZ
Charset3:              !           ABC E  HI  LMNOP RSTU   Y
```

Title strings must use Charset2 (and will be changed to Charset0 for the currently selected title).
Instruction text can have the Charset ROM colors combined with different VRAM palette attributes. To match up with the window border, the background color should be dark green (as so in palette 10h-11h and also in 1Bh-1Eh). Possible Charset vs Palette combinations for Instruction Texts are:

```
Palette:   10h       11h       1Bh       1Ch       1Dh       1Eh
Charset0:  White     Mint      D.Blue    Orange2   D.Green   White
Charset1:  Orange    L.Green   Blue      Magenta   P.Green   Orange3
Charset2:  Red       M.Green   White     Red       Pink      Red
Charset3:  Yellow    Yellow    -         -         -         -
```

Accordingly, there are several restrictions on coloring. For example, Red cannot be used with lowercase letters.

PC10 Z80 Video Circuit Timings - Dual Monitor Version

The Z80 Video Circuit is producing roughly NTSC-style timings (the output is passed straight to the built-in RGB monitor, so there has been no need to match exact television standards).

```
Resolution 32x28 tiles of 8x8 pix each (=256x224 pixels)
Dotclk = 5.040MHz (X2 oscillator, 20.160MHz, divided by 4)
Dots per line = 327 total dots (256 visible dots, plus 71 h-blank dots)
Lines per frame = 256 total lines (224 visible lines, plus 32 v-blank lines)
VBlank at Y=F0h..0Fh (when V5=1, V6=1, V7=1, latched at raising V4)
Vsync at Y=F8h..FBh (when Vblank=1, V4=1, V3=1, V2=0)
Hsync when HBlank=1, H1=1, H2=0
Screen Border Color during H/V-Blanking is Color 0 of Palette 0 (Black)
Frame Rate = 60.20642202Hz  = 5.040MHz / 256 Lines / 327 Dots
Z80 NMI (if enabled) is generated on begin of Z80 Circuit's Vblank (Y=F0h)
```

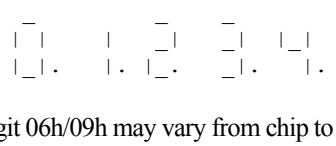
The Z80 Video Circuit's frame rate is NOT synchronized with the NES PPU.

PC10 Z80 Video Circuit Timings - Single Monitor Version

Not much known here. There's no schematic for the Single Monitor version. According to a component list, the X2 oscillator for the Z80 video circuit is 21.47727MHz (same as the X3 oscillator for the NES). Accordingly, one may assume that dots/line and lines/frame are also matched to NES timings (else the display would shake when switching between Z80 picture and NES picture).

LED Character Set (74HC4511 BCD-driver with yellow-green GL-8E040 display)

The Single-Monitor version is having a 4-Digit LED display for displaying the remaining time during Game (and also in Menu). The LED Digits 00h..09h are as so:



Digit 06h/09h may vary from chip to chip: existing Playchoice cabinets are drawing them as shown above (without upper/lower horizontal line, as specified

by Toshiba and Philips) (whilst Texas Instruments specifies them with extra lines). The "." dot (eighth LED segment) is always off in the Playchoice.

PC10 Title/Instructions (INST ROM)

Hardcoded INST-ROM Addresses

```
C000h 16bit ptr to "Data" for 40h-byte Area (passed on stack to C0FFh)
C001h 8bit value (initial chksum value)
C0FEh 16bit ptr to rst 38h handler (not used by BIOS) overlaps below C0FFh!
C0FFh code: rst 00h handler when I<>00h (get_40h_byte_area_function)
C3C3h code: Opcode E9h (JP HL)
C784h 16bit ptr to rst 30h handler (not used by BIOS)
C9BEh code: wait_1_frame function
        XXX the BIOS NMI handler does occassionally remap a different slot
            while the mainprogram is executing the C9BEh function,
            so, the C9BEh stuff must be identical (or very similar)
            in all INST-ROMs
C9C9h code: Opcode C9h (RET)
D000h homebrew title string (used by the gamehacker's homebrew BIOS)
D0FEh 16bit ptr to rst 28h handler (not used by BIOS) overlaps below D0FFh!
D0FFh code: rst 10h handler (not used by BIOS, used only by INST ROM)
D3D3h 16bit ptr to rst 20h handler (not used by BIOS)
D784h 16bit ptr to rst 18h handler (RET P, LD HL,[80E3], LD [HL],00, RET)
DFFFh 8bit value (can be used as final chksum adjustment byte)
```

40h-byte Area (aka HDR) (retrieved via C0FFh)

```
00h  1   GameID (must be unique number for Bookkeeping) (or 00h=Empty Slot)
01h  2   16bit ptr to 16bit ptr to Token code (demo_duration_function)
03h  2   16bit ptr to 16bit ptr to Z80 code   (draw_instruction_function)
05h  1   Decryption Offset (to be subtracted from bytes at [05h..1Ch])
06h  16h Unknown/Unused (eleven words in range C000h..FFFFh or so?)
1Ch  1   Opcode E9h (JP HL)
1Dh  3   Zerofilled (overwritten by leading Slot number for Title string)
20h  18h Title (18h bytes) (space, title, space, dotted-line, player symbols)
38h  8   Zerofilled (overwritten by variables)
```

Title (18h bytes) (space, title, space, dotted-line, player symbols)

Must consist of following characters:

```
00h..2Bh  0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ.'!-"",:
3Ch       +
3Eh       Dotted line
3Fh       Space
3Dh,F9h   Lightgun Symbol
FBh..FEh  Two Player Symbol ("@/@@")
```

Last 5 bytes of title should be:

```
3E,3F,3F,3F,3F   Normal single-player game (eg. 1942, Castlevania, Metroid)
3E,FB,FC,FD,FE   Two-player game (eg. Balloon, Contra)
3D,F9,3F,3F,3F   Lightgun game (eg. Hogan)
```

Instructions (aka "How-to-Play" screens)

Instructions are displayed in a 30x22 character window (originated at 90C2h in VRAM), existing games are containing 1-3 pages of text.
The text is displayed by repeatedly calling functions in INST ROM during Vblank (that functions should display only one text line at a time; to avoid flickering that would occur when exceeding vblank time).
The initial function call goes to [HDR[03h]], aside from drawing text, that function may manipulate the call address at [DE-1] (so further calls would go to different addresses), and some sort of general purpose index at [DE-2] (bit0-6 are initially 01h, and must be kept nonzero, and can be otherwise used for general purpose such like drawing yloc, commercial games use it as decryption offset) (bit7 should be set at the end of a page, causing the BIOS to wait for Enter button).

Token Interpreter (for defining the Demo Duration)

The demo duration is, for some strange reason, defined by tokens. The token program pointer consists of a 16bit base plus 8bit index. The token entrypoint is Base=[HDR[01h]] with Index=00h. The token interpreter is very simple: It can wait... and wait... and wait, and not much more.

```
Token     Param(s)                Function
00h..0Fh  -                       sleep_1_frame
10h..1Fh  -                       mark_finished (to be followed by token F0h)
20h..2Fh  Index,BaseLsb,BaseMsb   jump
30h..3Fh  Index,BaseLsb,BaseMsb   jump_if_user_is_not_interested
40h..4Fh  Index,BaseLsb,BaseMsb   jump_if_user_is_interested
50h..7Fh  NNh (max 0Fh else bugs) sleep_n_seconds (aka 64-frame units)
80h..8Fh  NNh                     sleep_n_frames
80h..8Fh  02h                     jump_if_single_monitor
90h..FFh  -                       terminate (to be preceeded by token 10h)
```

The "jump_if_single_monitor" acts as normal on Dual-Monitor BIOS (sleeps 2 frames), but has a special jump function on Single-Monitor BIOS (jumps to Base=Base-0008h, Index=00h).
User is "interested" means that the user has manually entered the instructions mode; that "interest" should be usually handled by executing additional sleep tokens to increase the demo duration.

Checksum
The INST-ROM checksum must be C9h. The checksum is calculated by adding/xoring 8192 bytes from INST-ROM with 8192 PROM reads (for details, see the BIOS code near the "compare A,C9h" opcode).

Tools
The a22i assembler (in no$nes debugger's utility menu) contains some .pc10 directives for creating INST ROMs. See the magicnes.zip package on no$nes webpage for sample source code. The INST ROMs will work without decryption PROM (they will only require a minimalistic circuit that drags the two decryption signals to HIGH upon /CHANNEL_SELECT).

PC10 NES-Side

The PC10 cartridges are usually containing PRG-ROM and CHR-ROM on EPROMs (though, despite of the EPROM-storage, they are usually byte-identical to normal NES ROM-cartridge versions). Nonetheless, there are a few things to recurse when making PC10 games (and eventually customize NES games accordingly):

NES PPU Palette
The PC10 uses a RGB palette, the colors are similar to NES Composite colors, but not 100% same: PC10 colors are more colorful than the pastelized NES colors. Pastel-cyan and two grayshades are completely missing. And, some colors are having wrong intensities, so fade-in/fade-out effects programmed for the NES can produce ugly flickering on the PC10. The color emphasis bits are also working differently as on NES. The PC10 palette is same as the "standard" VS System palette, for details see:
VS System PPUs and Palettes

NES NMI Disable
The PC10 BIOS is watching the NES NMI signal. If the NES disables NMIs for a longer period (around 255 frames), then it'll treat the game to have locked up, and will terminate the game. If that happens more than twice, then it will additionally remove the game from the menu's cartridge list.

NES Controller Disable
The PC10 can disable the NES controllers in demo mode. For some reason, it doesn't disable the serial shift-register outputs, but rather only the Select/Start parallel inputs (and lightgun trigger). Accordingly, to prevent the game being played in demo mode, starting the game should work ONLY via Start button (ie. not via Button A/B).

NES Controller Connection
The PC10 has two "joypads" and one "zapper", unlike as on NES, it isn't possible to disconnect them, so the game must work with that hardware connected (which should be usually no problem).
Whereas, the lightgun seems to be an optional add-on, not installed in all cabinets (especially not in tabletop cabinets).
The Start/Select buttons are mapped to the Joypad 1 input. The Joypad 2 input doesn't have any such buttons (similar as japanese Famicom joypads).

NES Pause / Timings
It seems to be possible to "pause" the NES CPU. Unknown if this is actually done during game-execution... if so, pausing/unpausing may get the CPU offsync with the PPU, which might confuse & crash a few NES games. Initial CPU Reset time vs PPU Reset time may be also different as on normal NES. And, reportedly, the PC10 RGB-PPU timing isn't exactly same as NES NTSC-PPU timing (reportedly something to do with missing dots on some NTSC scanlines or so).

NES IRQs
On older PC10 mainboards, the /IRQ pin is wired as output (rather than as input), making it incompatible with games that do use IRQs. The mainboards can be upgraded (with two wires) to support IRQs. The upgrade may be found on many mainboards as it's required for PCH1-01-ROM-G game cards.

Communicating between NES and Z80 CPUs
There is no intended CPU-to-CPU communication support. However, with some trickery, it should be possible. On the Z80 side, one could place custom code in the C9BEh function (which is called once every mainloop cycle). For NES-to-Z80 transfers one could enable/disable NES NMIs to transfer 1bit per frame. For Z80-to-NES transfers one could eventually issue specially timed CPU or PPU Reset pulses.

PC10 Games and Cartridge PCBs

PC10 Cartridge PCB Versions

```
PCH1-01-ROM      Mapper 0   (NROM)
PCH1-01-ROM-A    Mapper 3   (CNROM - VROM/8K)
PCH1-01-ROM-B    Mapper 2   (UNROM - PRG/16K)
PCH1-01-ROM-C    Mapper 87  (Jaleco/Konami 16K VROM - VROM/8K)
PCH1-01-ROM-D    Mapper 1   (MMC1 with VRAM)
PCH1-01-ROM-E    Mapper 9   (MMC2)           (...plus battery ??)
PCH1-01-ROM-F    Mapper 1   (MMC1 with VROM)
PCH1-01-ROM-G    Mapper 4   (MMC3 xxx)
PCH1-01-ROM-H    Mapper 119 (MMC3 TQROM)
PCH1-01-ROM-I    Mapper 7   (AOROM - PRG/32K, Name Table Select)
PCH1-01-ROM-J    N/A ?
PCH1-01-ROM-K    Mapper 1   (MMC1 with VRAM... plus battery ??)
```

Known/released PC10 Games

```
PCH1-02-ROM    1942 (1986/1985) Capcom
PCH1-01-ROM    Balloon Fight (1985/1986) Nintendo
PCH1-01-ROM    Baseball (1985/1984) Nintendo
PCH1-02-ROM-F  Baseball Stars (1989/1986) SNK
PCH1-01-ROM-I  Captain Skyhawk (1990/1989) Milton Bradley
PCH1-02-ROM-B  Castlevania (1987) Konami
PCH1-02-ROM-F  Chip 'n Dale Rescue Rangers (1990) Capcom
PCH1-02-ROM-B  Contra (1988) Konami
PCH1-01-ROM-F  Double Dragon (1988) Technos
PCH1-02-ROM-B  Double Dribble (1987) Konami
PCH1-02-ROM-F  Dr. Mario (1990) Nintendo
PCH1-01-ROM    Duck Hunt (1985) Nintendo
PCH1-01-ROM    Excitebike (1985) Nintendo
PCH1-01-ROM-F  Fester's Quest (1989) Sunsoft
PCH1-01-ROM-G  Gauntlet (1985) Atari/Tengen
PCH1-01-ROM    Golf (1985) Nintendo
PCH1-01-ROM-C  Goonies, The (1986) Konami
PCH1-01-ROM-A  Gradius (1986) Konami (said to exist in two PRG-ROM versions)
PCH1-01-ROM    Hogan's Alley (1985) Nintendo
PCH1-02-ROM    Kung Fu (1985) Irem
CUSTOM         Magic Floor (2012) nocash (homebrew)
PCH1-01-ROM    Mario Bros. (1984/1986) Nintendo
PCH1-01-ROM-K  Mario's Open Golf (1991) Nintendo
PCH1-04-ROM-G  Mega Man 3 (1990) Capcom
PCH1-01-ROM-D  Metroid (1986) Nintendo
PCH1-01-ROM-E  Mike Tyson's Punch-Out!! (1987) Nintendo
PCH1-01-ROM-F  Ninja Gaiden (1989) Tecmo
PCH1-03-ROM-G  Ninja Gaiden II: The Dark Sword of Chaos (1990) Tecmo
PCH1-03-ROM-G  Ninja Gaiden III: The Ancient Ship of Doom (1991) Tecmo
PCH1-02-ROM-G  Nintendo World Cup (Soccer) (1990) Technos
PCH1-01-ROM-H  Pinbot (1990/1988) Rare
PCH1-02-ROM-G  Power Blade (1991) Taito
PCH1-02-ROM-B  Pro Wrestling (1987/1986) Nintendo
PCH1-02-ROM-D  Rad Racer (1987) Square
PCH1-02-ROM-G  Rad Racer II (1990) Square
PCH1-01-ROM-F  R.C. Pro-Am (1988/1987) Rare
PCH1-02-ROM-G  Rockin' Kats (1991) Atlus
PCH1-01-ROM-B  Rush'n Attack (1987) Konami
PCH1-01-ROM-B  Rygar (1987) Tecmo
PCH1-01-ROM-I  Solar Jetman: Hunt for the Golden War(p)ship (1990) Rare
PCH1-02-ROM-G  Super C (1990) Konami
PCH1-02-ROM    Super Mario Bros. (1985) Nintendo
PCH1-01-ROM-G  Super Mario Bros. 2 (1988) Nintendo
PCH1-01-ROM-G  Super Mario Bros. 3 (1990) Nintendo
PCH1-01-ROM-F  Tecmo Bowl (1989) Tecmo Inc.
PCH1-01-ROM-F  Teenage Mutant Ninja Turtles (1989) Konami
PCH1-03-ROM-G  Teenage Mutant Ninja Turtles II: The Arcade Game (1990) Konami
PCH1-01-ROM    Tennis (1985) Nintendo
PCH1-01-ROM-A  Track & Field (1987) Konami
PCH1-01-ROM-B  Trojan (1987/1986) Capcom
PCH1-01-ROM    Volleyball (1987/1986) Nintendo
PCH1-01-ROM    Wild Gunman (1985) Nintendo
PCH1-02-ROM-F  Yo! Noid (1990) Capcom
```

As seen above, release dates are somewhat unclear; dates before 1986 are apparently referring to the original NES version, not the PC10 release date.

Mystical PC10 Games

```
(unseen)       Goonies II, The (19xx) Konami
(unseen)       RBI Baseball (1987) Atari/Tengen
(unseen)       Shatterhand (1991) Jaleco
```

These titles have been somewhere announced, and might have been actually released, but so far no collectors ever seem to have ever found these cartridges.

PC10 Cabinet and BIOS Versions

Dual-Monitor Upright Cabinets

The upper monitor (for Z80 menu, instructions, and remaining time/credit display) is usually smaller than the lower one (NES game picture). There seems to be also a variant with two big monitors. Front panel is having 5 control buttons.

Single-Monitor Cabinets

The BIOS can switch the monitor to display either the Z80 or NES picture. The remaining credit/time is displayed via 4-digit 7-segment LED displays. Front panel can have 4 or 5 control buttons. Switching the BIOS to the correct front panel type seems to rely on inversion of the reset button signal, and seems to work properly only on certain dip-switch settings.

Single-Monitor Upright Cabinets

These are usually upgraded VS System cabinets with a PC10 mainboard (either original VS System cabinets, or even older cabinets that have been formerly upgraded to become a VS System). The cabinets are having VS System style front panels with only four buttons (named 1-4) instead of the normal five

PC10 buttons (Channel Select, Enter, Reset, Start, Game Select).

Single-Monitor Countertop/Sitdown Cabinets
The Playchoice "countertop" version (a white box to be placed on a bar counter or table) is a native PC10 cabinet (with 5 control buttons). The "red tent" version (a red tent-shaped console with two legs) is an upgraded VS Dual System sitdown version (with 4 control buttons).

Official BIOSes (IC "8T")
```
  PCH1-C.8T   Dual-Monitor Version   (CRC32=D52FA07Ah) (16Kbytes)
  PCK1-C.8T   Single-Monitor Version (CRC32=503EE8B1h) (16Kbytes)
```
There aren't any further known versions/revisions.
```
        0BE8CEB4
```

Homebrew BIOSes
```
  Oliver Achten's BIOS v0.1 (2002)   (CRC32=FB96DE76h) (8Kbytes)
  gamehacker's BIOS v1.0             (CRC32=18B4E0B5h) (16Kbytes)
  gamehacker's BIOS v1.1             (CRC32=80BD20EFh) (16Kbytes)
```
The homebrew BIOSes are allowing to play NES games without PROMs. They are suffering some restrictions:
Oliver's BIOS doesn't have any support for Titles/Instructions (neither for original nor homebrew games). The BIOS can be DIP-Switched to Single/Dual Monitor mode. Unknown if coin inputs are supported.
Gamehacker's BIOSes are supporting Titles (from original games and homebrew games; for the latter, the title can be stored in custom INST ROM format, or in SRAM; which probably only lasts for some days/weeks?). Instructions aren't supported (neither for original nor homebrew games). Unknown if this BIOS supports both Single and Dual Monitor versions. The BIOS doesn't support coin inputs (freeplay mode only)

PC10 Pin-Outs

PC10 Cartridge Slot (3x32 pins per slot; 10 slots)
```
  A1  -           B1  PROM.RESET    C1  Z80./INST.ROM.SEL
  A2  -           B2  PROM.CLOCK    C2  Z80.DD7
  A3  -           B3  PROM.TEST     C3  Z80.DD6
  A4  -           B4  Z80.AD7       C4  Z80.DD5
  A5  -           B5  Z80.AD6       C5  Z80.DD4
  A6  -           B6  Z80.AD5       C6  Z80.DD3
  A7  Z80.AD11    B7  Z80.AD4       C7  Z80.DD2
  A8  Z80.AD10    B8  Z80.AD3       C8  Z80.DD1
  A9  Z80.AD9     B9  Z80.AD2       C9  Z80.DD0
  A10 Z80.AD8     B10 Z80.AD1       C10 Z80.AD12
  A11 PROM.ADDR5  B11 Z80.AD0       C11 Z80./MREQ.RD
  A12 PROM.DATA   B12 PPU./PA13     C12 PPU./VRAMCS
  A13 PPU.PA8     B13 PPU.PD7       C13 PPU.VRAMA10
  A14 PPU.PA9     B14 PPU.PD6       C14 PPU.PA7
  A15 PPU.PA10    B15 PPU.PD5       C15 PPU.PA6
  A16 PPU.PA11    B16 PPU.PD4       C16 PPU.PA5
  A17 PPU.PA12    B17 PPU.PD3       C17 PPU.PA4
  A18 PPU.PA13    B18 PPU.PD2       C18 PPU.PA3
  A19 PPU./WE     B19 PPU.PD1       C19 PPU.PA2
  A20 PPU./RD     B20 PPU.PD0       C20 PPU.PA1
  A21 NES.PHI2    B21 NES./ROM.SEL  C21 PPU.PA0
  A22 NES.A8      B22 NES.R/W       C22 NES./IRQ (see note)
  A23 NES.A9      B23 NES.D7        C23 NES.A0
  A24 NES.A10     B24 NES.D6        C24 NES.A1
  A25 NES.A11     B25 NES.D5        C25 NES.A2
  A26 NES.A12     B26 NES.D4        C26 NES.A3
  A27 NES.A13     B27 NES.D3        C27 NES.A4
  A28 NES.A14     B28 NES.D2        C28 NES.A5
  A29 /CH1..10    B29 NES.D1        C29 NES.A6
  A30 SOUND       B30 NES.D0        C30 NES.A7
  A31 VCC         B31 VCC           C31 VCC
  A32 GND         B32 GND           C32 GND
```
Note: The NES./IRQ appears to be somehow bugged on older mainboards (PCH1-01-CPU and PCH1-02-CPU), Nintendo seems to have released a document that describes how to upgrade that mainboards (for PCH1-01-ROM-G game card compatibilty). The document says to disconnect pin 11 and 12 of IC "4L", and to rewire that pins to whatever locations on the mainboard (not quite clear what signals they shall be wired to exactly) (the bug as such seems to be that IC "4L" is passing /IRQ as output, rather than as input) (so, maybe, the bugfix just means to say to swap the two pins?).

Wiring of 74LS367 on PC10 pro wrestling cart
```
                  .----._.----.
         /chsel 1| /G1   Vcc |16 VCC
  (Conn) Chr A13 2| 1A1   /G2 |15 /chsel
      +5 pullup 3| 1Y1   2A1 |14 CHR /A13 (PC10 Connector)
  (Conn) PRG /CE 4| 1A2   2Y1 |13 CIRAM /CE (PC10 Connector)
      +5 pullup 5| 1Y2   2A2 |12 /CSel INST ROM (PC10 Conn)
  (Brd) CIRAM A10 6| 1A3   2Y2 |11 Pin 20 U4
  (Conn)CIRAM A10 7| 1Y3   2A3 |10 N/C
            GND 8| GND   2Y3 | 9 N/C
                  '-----------'
```

PC10 "P1" Connector (called "P2" in schematic)

```
 1  /MRED (Main Screen / NES Picture)
 2  /MGRN
 3  /MBLUE
 4  MVGND
 5  /SRED (Sub Screen / Z80 Picture)
 6  /SGRN
 7  /SBLU
 8  SVGND
 9  /SSYNC
10 /MSYNC
11 GND
12 GND
13 GND
14 GND
15 +5V
16 +5V
17 +5V
18 +5V
19 ?
20 ? as 19
21 ?
22 ? as 21
23 +24V
24 Service
25 ?                    ;schematic: Counter 2
26 Counter1
27 ?
28 ?
29 ?
30 ?
31 ? via jumper to 26   ;schematic: Coin 2
32 Coin1
33 2.B Button
34 2.A Button
35 ? to GND
36 2.Right
37 ? to GND
38 2.Left
39 ? to GND
40 2.Up
41 ? to GND
42 2.Down
43 ? to GND
44 GameSelect
45 ? to GND
46 ?
47 ? NC
48 Reset
49 ? NC
50 Start
51 SSound (shortcut with MSound)
52 MSound (shortcut with SSound)
53 ?
54 GND (Sound)
55 GND
56 GND
```

PC10 "P2-to-P1" Connector (called "P1" in schematic)

```
 1 1.Right
 2 1.Left
 3 1.Down
 4 1.Up
 5 GND
 6 ?    to 4016h.Bit3
 7 1.B Button
 8 1.A Button
 9 GND
10 ?   to 4016h.Bit4 (INP0.D4)
11 CH.Select
12 Enter
13 GND
14 GND
15 ?
16 GND
17 GunTrigger
18 GunHit
19 +5V
20 +5V
```

PC10 Single-Monitor version Connectors?

P2 (36 pin)

```
      Parts Side                |         Solder Side
         GND            | A/19 | 1 |          GND
         LED00-DD3      | B/20 | 2 |       2P  Right
         LED01-DD2      | C/21 | 3 |       2P  Left
         LED02-DD1      | D/22 | 4 |       2P  Up
         LED03-DD0      | E/23 | 5 |       2P  Down
         +5V            | F/24 | 6 |       2P  ButtonA
         +5V            | H/25 | 7 |       2P  ButtonB
         LED04-AD4      | J/26 | 8 |       Button1/Channel Select
         LED05-AD1      | K/27 | 9 |       Button3/Game Select
         LED06-AD0      | L/28 | 10|       Button2/Channel Enter
         LED07-/IOWR    | M/29 | 11|       Button4/Game Start
                        | N/30 | 12|       1P  Right
                        | P/31 | 13|       1P  Left
         +5V            | R/32 | 14|       1P  Up
                        | S/33 | 15|       1P  Down
                        | T/34 | 16|       1P  ButtonA
   ResetButton(if any?) | U/35 | 17|       1P  ButtonB
         GND            | V/36 | 18|          GND
```

P1 (44pin)

```
      Parts Side                |         Solder Side
   -----------------------------+-------------------------------
         GND            | A/23 | 1 |          GND
         GND            | B/24 | 2 |          GND
         +5V            | C/25 | 3 |          +5V
         +5V            | D/26 | 4 |          +5V
         +12V           | E/27 | 5 |          +12V
                        | F/28 | 6 |
                        | H/29 | 7 |       Coin Switch #1
                        | J/30 | 8 |       Coin Switch #2 (if any?)
                        | K/31 | 9 |       Video Red (inv)
                        | L/32 | 10|       Video Green (inv)
         GND            | M/33 | 11|          GND
                        | N/34 | 12|       Video Blue (inv)
                        | P/35 | 13|       Video Sync
                        | R/36 | 14|
                        | S/37 | 15|
         -5V (or +12V?) | T/38 | 16|          -5V (or +12V?)
           (?)          | U/39 | 17|            (?) (or COUNTER?)
                        | V/40 | 18|
                        | W/41 | 19|            (?) (or SERVICE Button?)
                        | X/42 | 20|       Audio (no amp)
         GND            | Y/43 | 21|          GND
         GND            | Z/44 | 22|          GND
   -----------------------------------------------------
Note:    Connect Video Ground and Audio Ground to GND
         Video color signals need to be inverted for regular Jamma
         Audio signal needs to be amplified for regular Jamma
```

Z80 CPU Specifications

Z80 Register Set

Register Summary

```
  16bit Hi   Lo   Name/Function
  -------------------------------------
  AF    A    -    Accumulator & Flags
  BC    B    C    BC
  DE    D    E    DE
```

```
HL    H    L    HL
AF'   -    -    Second AF
BC'   -    -    Second BC
DE'   -    -    Second DE
HL'   -    -    Second HL
IX    IXH  IXL  Index register 1
IY    IYH  IYL  Index register 2
SP    -    -    Stack Pointer
PC    -    -    Program Counter/Pointer
-     I    R    Interrupt & Refresh
```

Normal 8bit and 16bit Registers
The Accumulator (A) is the allround register for 8bit operations. Registers B, C, D, E, H, L are normal 8bit registers, which can be also accessed as 16bit register pairs BC, DE, HL.
The HL register pair is used as allround register for 16bit operations. B and BC are sometimes used as counters. DE is used as DEstination pointer in block transfer commands.

Second Register Set
The Z80 includes a second register set (AF',BC',DE',HL') these registers cannot be accessed directly, but can be exchanged with the normal registers by using the EX AF,AF and EXX instructions.

Refresh Register
The lower 7 bits of the Refresh Register (R) are incremented with every instruction. Instructions with at least one prefix-byte (CB,DD,ED,FD, or DDCB,FDCB) will increment the register twice. Bit 7 can be used by programmer to store data. Permanent writing to this register will suppress memory refresh signals, causing Dynamic RAM to lose data.

Interrupt Register
The Interrupt Register (I) is used in interrupt mode 2 only (see command "im 2"). In other modes it can be used as simple 8bit data register.

IX and IY Registers
IX and IY are able to manage almost all the things that HL is able to do. When used as memory pointers they are additionally including a signed index byte (IX+d). The disadvantage is that the opcodes occupy more memory bytes, and that they are less fast than HL-instructions.

Undocumented 8bit Registers
IXH, IXL, IYH, IYL are undocumented 8bit registers which can be used to access high and low bytes of the IX and IY registers (much like H and L for HL). Even though these registers do not officially exist, they seem to be available in all Z80 CPUs, and are quite commonly used by various software.

Z80 Flags

Flag Summary
The Flags are located in the lower eight bits of the AF register pair.

```
Bit Name  Set  Clr  Expl.
0   C     C    NC   Carry Flag
1   N     -    -    Add/Sub-Flag (BCD)
2   P/V   PE   PO   Parity/Overflow-Flag
3   -     -    -    Undocumented
4   H     -    -    Half-Carry Flag (BCD)
5   -     -    -    Undocumented
6   Z     Z    NZ   Zero-Flag
7   S     M    P    Sign-Flag
```

Carry Flag (C)
This flag signalizes if the result of an arithmetic operation exceeded the maximum range of 8 or 16 bits, ie. the flag is set if the result was less than Zero, or greater than 255 (8bit) or 65535 (16bit). After rotate/shift operations the bit that has been 'shifted out' is stored in the carry flag.

Zero Flag (Z)
Signalizes if the result of an operation has been zero (Z) or not zero (NZ). Note that the flag is set (1) if the result was zero (0).

Sign Flag (S)
Signalizes if the result of an operation is negative (M) or positive (P), the sign flag is just a copy of the most significant bit of the result.

Parity/Overflow Flag (P/V)
This flag is used as Parity Flag, or as Overflow Flag, or for other purposes, depending on the instruction.
Parity: Bit7 XOR Bit6 XOR Bit5 ... XOR Bit0 XOR 1.
8bit Overflow: Indicates if the result was greater/less than +127/-128.
HL Overflow: Indicates if the result was greater/less than +32767/-32768.
After LD A,I or LD A,R: Contains current state of IFF2.
After LDI,LDD,CPI,CPD,CPIR,CPDR: Set if BC<>0 at end of operation.

BCD Flags (H,N)

These bits are solely supposed to be used by the DAA instruction. The N flag signalizes if the previous operation has be an addition or substraction. The H flag indicates if the lower 4 bits exceeded the range from 0-0Fh. (For 16bit instructions: H indicates if the lower 12 bits exceeded the range from 0-0FFFh.) After adding/subtracting two 8bit BCD values (0-99h) the DAA instruction can be used to convert the hexadecimal result in the A register (0-FFh) back to BCD format (0-99h). Note that DAA also requires the carry flag to be set correctly, and thus should not be used after INC A or DEC A.

Undocumented Flags (Bit 3,5)
The content of these undocumented bits is filled by garbage by all instructions that affect one or more of the normal flags (for more info read the chapter Garbage in Flag Register), the only way to read out these flags would be to copy the flags register onto the stack by using the PUSH AF instruction. However, the existence of these bits makes the AF register a full 16bit register, so that for example the code sequence PUSH DE, POP AF, PUSH AF, POP HL would set HL=DE with all 16bits intact.

Z80 Instruction Format

Commands and Parameters
Each instruction consists of a command, and optionally one or two parameters. Usually the leftmost parameter is modified by the operation when two parameters are specified.

Parameter Placeholders
The following placeholders are used in the following chapters:

```
r      8bit  register A,B,C,D,E,H,L
rr     16bit register BC, DE, HL/IX/IY, AF/SP   (as described)
i      8bit  register A,B,C,D,E,IXH/IYH,IXL/IYL
ii     16bit register IX,IY
n      8bit  immediate 00-FFh                   (unless described else)
nn     16bit immediate 0000-FFFFh
d      8bit  signed offset -128..+127
f      flag  condition nz,z,nc,c AND/OR po,pe,p,m  (as described)
(..)   16bit pointer to byte/word in memory
```

Opcode Bytes
Each command (including parameters) consists of 1-4 bytes. The respective bytes are described in the following chapters. In some cases the register number or other parameters are encoded into some bits of the opcode, in that case the opcode is specified as "xx". Opcode prefix bytes "DD" (IX) and "FD" (IY) are abbreviated as "pD".

Clock Cycles
The clock cycle values in the following chapters specify the execution time of the instruction. For example, an 8-cycle instruction would take 2 microseconds on a CPU which is operated at 4MHz (8/4 ms). For conditional instructions two values are specified, for example, 17;10 means 17 cycles if condition true, and 10 cycles if false.
Note that in case that WAIT signals are sent to the CPU by the hardware then the execution may take longer.

Affected Flags
The instruction tables below are including a six character wide field for the six flags: Sign, Zero, Halfcarry, Parity/Overflow, N-Flag, and Carry (in that order). The meaning of the separate characters is:

```
s    Indicates Signed result
z    Indicates Zero
h    Indicates Halfcarry
o    Indicates Overflow
p    Indicates Parity
c    Indicates Carry
-    Flag is not affected
0    Flag is cleared
1    Flag is set
x    Flag is destroyed (unspecified)
i    State of IFF2
e    Indicates BC<>0 for LDX(R) and CPX(R), or B=0 for INX(R) and OUTX(R)
```

Z80 Load Commands

8bit Load Commands

```
Instruction    Opcode  Cycles Flags  Notes
ld   r,r       xx           4 ------ r=r
ld   i,i       pD xx        8 ------ i=i
ld   r,n       xx nn        7 ------ r=n
ld   i,n       pD xx nn    11 ------ i=n
ld   r,(HL)    xx           7 ------ r=(HL)
ld   r,(ii+d)  pD xx dd    19 ------ r=(ii+d)
ld   (HL),r    7x           7 ------ (HL)=r
ld   (ii+d),r  pD 7x dd    19 ------
ld   (HL),n    36 nn       10 ------
ld   (ii+d),n  pD 36 dd nn 19 ------
ld   A,(BC)    0A           7 ------
```

```
ld   A,(DE)     1A           7 ------
ld   A,(nn)     3A nn nn    13 ------
ld   (BC),A     02           7 ------
ld   (DE),A     12           7 ------
ld   (nn),A     32 nn nn    13 ------
ld   A,I        ED 57        9 sz0i0- A=I  ;Interrupt Register
ld   A,R        ED 5F        9 sz0i0- A=R  ;Refresh Register
ld   I,A        ED 47        9 ------
ld   R,A        ED 4F        9 ------
```

16bit Load Commands

```
Instruction    Opcode  Cycles Flags  Notes
ld   rr,nn     x1 nn nn    10 ------ rr=nn    ;rr may be BC,DE,HL or SP
ld   ii,nn     pD 21 nn nn 13 ------ ii=nn
ld   HL,(nn)   2A nn nn    16 ------ HL=(nn)
ld   ii,(nn)   pD 2A nn nn 20 ------ ii=(nn)
ld   rr,(nn)   ED xB nn nn 20 ------ rr=(nn)  ;rr may be BC,DE,HL or SP
ld   (nn),HL   22 nn nn    16 ------ (nn)=HL
ld   (nn),ii   pD 22 nn nn 20 ------ (nn)=ii
ld   (nn),rr   ED x3 nn nn 20 ------ (nn)=rr  ;rr may be BC,DE,HL or SP
ld   SP,HL     F9           6 ------ SP=HL
ld   SP,ii     pD F9       10 ------ SP=ii
push rr        x5          11 ------ SP=SP-2, (SP)=rr  ;rr may be BC,DE,HL,AF
push ii        pD E5       15 ------ SP=SP-2, (SP)=ii
pop  rr        x1          10 (-AF-) rr=(SP), SP=SP+2  ;rr may be BC,DE,HL,AF
pop  ii        pD E1       14 ------ ii=(SP), SP=SP+2
ex   DE,HL     EB           4 ------ exchange DE <--> HL
ex   AF,AF     08           4 xxxxxx exchange AF <--> AF'
exx            D9           4 ------ exchange BC,DE,HL <--> BC',DE',HL'
ex   (SP),HL   E3          19 ------ exchange (SP) <--> HL
ex   (SP),ii   pD E3       23 ------ exchange (SP) <--> ii
```

Blocktransfer

```
Instruction    Opcode  Cycles Flags  Notes
ldi            ED A0       16 --0e0- (DE)=(HL), HL=HL+1, DE=DE+1, BC=BC-1
ldd            ED A8       16 --0e0- (DE)=(HL), HL=HL-1, DE=DE-1, BC=BC-1
ldir           ED B0  bc*21-5 --0?0- ldi-repeat until BC=0
lddr           ED B8  bc*21-5 --0?0- ldd-repeat until BC=0
```

Z80 Arithmetic/Logical Commands

8bit Arithmetic/Logical Commands

```
Instruction    Opcode  Cycles Flags  Notes
daa            27           4 szxp-x decimal adjust akku
cpl            2F           4 --1-1- A = A xor FF
neg            ED 44        8 szho1c A = 00-A
<arit>  r      xx           4 szhonc see below
<arit>  i      pD xx        8 szhonc see below, UNDOCUMENTED
<arit>  n      xx nn        7 szhonc see below
<arit>  (HL)   xx           7 szhonc see below
<arit>  (ii+d) pD xx dd    19 szhonc see below
<cnt>   r      xx           4 szhon- see below
<cnt>   i      pD xx        8 szhon- see below, UNDOCUMENTED
<cnt>   (HL)   xx          11 szhon- see below
<cnt>   (ii+d) pD xx dd    23 szhon- see below
<logi>  r      xx           4 szhp00 see below
<logi>  i      pD xx        8 szhp00 see below, UNDOCUMENTED
<logi>  n      xx nn        7 szhp00 see below
<logi>  (HL)   xx           7 szhp00 see below
<logi>  (ii+d) pD xx dd    19 szhp00 see below
```

Arithmetic <arit> commands:

```
add   A,op     see above 4-19 szho0c A=A+op
adc   A,op     see above 4-19 szho0c A=A+op+cy
sub   op       see above 4-19 szho1c A=A-op
sbc   A,op     see above 4-19 szho1c A=A-op-cy
cp    op       see above 4-19 szho1c compare, ie. VOID=A-op
```

Increment/Decrement <cnt> commands:

```
inc   op       see above 4-23 szho0- op=op+1
dec   op       see above 4-23 szho1- op=op-1
```

Logical <logi> commands:

```
and   op       see above 4-19 sz1p00 A=A & op
xor   op       see above 4-19 sz0p00 A=A XOR op
or    op       see above 4-19 sz0p00 A=A | op
```

16bit Arithmetic Commands

```
Instruction    Opcode  Cycles Flags  Notes
add  HL,rr     x9          11 --h-0c HL = HL+rr    ;rr may be BC,DE,HL,SP
add  ii,rr     pD x9       15 --h-0c ii = ii+rr    ;rr may be BC,DE,ii,SP (!)
adc  HL,rr     ED xA       15 szho0c HL = HL+rr+cy ;rr may be BC,DE,HL,SP
```

```
sbc  HL,rr     ED x2       15 szho1c HL = HL-rr-cy ;rr may be BC,DE,HL,SP
inc  rr        x3           6 ------ rr = rr+1     ;rr may be BC,DE,HL,SP
inc  ii        pD 23       10 ------ ii = ii+1
dec  rr        xB           6 ------ rr = rr-1     ;rr may be BC,DE,HL,SP
dec  ii        pD 2B       10 ------ ii = ii-1
```

Searchcommands

```
Instruction    Opcode  Cycles Flags  Notes
cpi            ED A1       16 szhe1- compare A-(HL), HL=HL+1, DE=DE+1, BC=BC-1
cpd            ED A9       16 szhe1- compare A-(HL), HL=HL-1, DE=DE-1, BC=BC-1
cpir           ED B1   x*21-5 szhe1- cpi-repeat until BC=0 or compare fits
cpdr           ED B9   x*21-5 szhe1- cpd-repeat until BC=0 or compare fits
```

Z80 Rotate/Shift and Singlebit Operations

Rotate and Shift Commands

```
Instruction    Opcode  Cycles Flags  Notes
rlca           07           4 --0-0c rotate akku left
rla            17           4 --0-0c rotate akku left through carry
rrca           0F           4 --0-0c rotate akku right
rra            1F           4 --0-0c rotate akku right through carry
rld            ED 6F       18 sz0p0- rotate left low digit of A through (HL)
rrd            ED 67       18 sz0p0- rotate right low digit of A through (HL)
<cmd> r        CB xx        8 sz0p0c see below
<cmd> (HL)     CB xx       15 sz0p0c see below
<cmd> (ii+d)   pD CB dd xx 23 sz0p0c see below
<cmd> r,(ii+d) pD CB dd xx 23 sz0p0c see below, UNDOCUMENTED modify and load
```

Whereas <cmd> may be:

```
rlc    rotate left
rl     rotate left through carry
rrc    rotate right
rr     rotate right through carry
sla    shift left arithmetic (b0=0)
sll    UNDOCUMENTED shift left (b0=1)
sra    shift right arithmetic (b7=b7)
srl    shift right logical (b7=0)
```

Singlebit Operations

```
Instruction    Opcode  Cycles Flags  Notes
bit  n,r       CB xx        8 xz1x0- test bit n  ;n=0..7
bit  n,(HL)    CB xx       12 xz1x0-
bit  n,(ii+d)  pD CB dd xx 20 xz1x0-
set  n,r       CB xx        8 ------ set bit n   ;n=0..7
set  n,(HL)    CB xx       15 ------
set  n,(ii+d)  pD CB dd xx 23 ------
set r,n,(ii+d) pD CB dd xx 23 ------ UNDOCUMENTED set n,(ii+d) and ld r,(ii+d)
res  n,r       CB xx        8 ------ reset bit n ;n=0..7
res  n,(HL)    CB xx       15 ------
res  n,(ii+d)  pD CB dd xx 23 ------
res r,n,(ii+d) pD CB dd xx 23 ------ UNDOCUMENTED res n,(ii+d) and ld r,(ii+d)
ccf            3F           4 --h-0c h=cy, cy=cy xor 1
scf            37           4 --0-01 cy=1
```

Z80 Jumpcommands & Interrupts

General Jump Commands

```
Instruction    Opcode  Cycles Flags  Notes
jp   nn        C3 nn nn    10 ------ jump to nn, ie. PC=nn
jp   HL        E9           4 ------ jump to HL, ie. PC=HL
jp   ii        pD E9        8 ------ jump to ii, ie. PC=ii
jp   f,nn      xx nn nn 10;10 ------ jump to nn if nz,z,nc,c,po,pe,p,m
jr   nn        18 dd       12 ------ relative jump to nn, ie. PC=PC+d
jr   f,nn      xx dd     12;7 ------ relative jump to nn if nz,z,nc,c
djnz nn        10 dd     13;8 ------ B=B-1 and relative jump to nn if B<>0
call nn        CD nn nn    17 ------ call nn ie. SP=SP-2, (SP)=PC, PC=nn
call f,nn      xx nn nn 17;10 ------ call nn if nz,z,nc,c,po,pe,p,m
ret            C9          10 ------ pop PC ie. PC=(SP), SP=SP+2
ret  f         xx        11;5 ------ pop PC if nz,z,nc,c,po,pe,p,m
rst  n         xx          11 ------ call n  ;n=00,08,10,18,20,28,30,38
nop            00           4 ------ no operation
```

Interrupt Related Commands

```
Instruction    Opcode  Cycles Flags  Notes
di             F3           4 ------ IFF1=0, IFF2=0  ;disable interrupts
ei             FB           4 ------ IFF1=1, IFF2=1  ;enable interrupts
im   0         ED 46        8 ------ read opcode from databus on interrupt
im   1         ED 56        8 ------ execute call 0038h on interrupt
```

```
im   2          ED 5E        8 ------ execute call (i*100h+databus) on int.
halt            76         N*4 ------ repeat until interrupt occurs
reti            ED 4D       14 ------ pop PC, IFF1=IFF2, ACK (ret from INT)
retn            ED 45       14 ------ pop PC, IFF1=IFF2      (ret from NMI)
</INT=LOW,IM=0,IFF1=1>  1+var ------ IFF1=0,IFF2=0, exec opcode from databus
</INT=LOW,IM=1,IFF1=1>     12 ------ IFF1=0,IFF2=0, CALL 0038h
</INT=LOW,IM=2,IFF1=1>     18 ------ IFF1=0,IFF2=0, CALL [I*100h+databus]
</NMI=falling_edge>         ? ------ IFF1=0,        CALL 0066h
```

Z80 I/O Commands

```
Instruction    Opcode  Cycles Flags  Notes
in   A,(n)     DB nn       11 ------ A=PORT(A*100h+n)
in   r,(C)     ED xx       12 sz0p0- r=PORT(BC)
in   (C)       ED 70       12 sz0p0- **undoc/illegal** VOID=PORT(BC)
out  (n),A     D3 nn       11 ------ PORT(A*100h+n)=A
out  (C),r     ED xx       12 ------ PORT(BC)=r
out  (C),0     ED 71       12 ------ **undoc/illegal** PORT(BC)=00
ini            ED A2       16 xexxxx MEM(HL)=PORT(BC), HL=HL+1, B=B-1
ind            ED AA       16 xexxxx MEM(HL)=PORT(BC), HL=HL-1, B=B-1
outi           ED A3       16 xexxxx B=B-1, PORT(BC)=MEM(HL), HL=HL+1
outd           ED AB       16 xexxxx B=B-1, PORT(BC)=MEM(HL), HL=HL-1
inir           ED B2   b*21-5 x1xxxx same than ini, repeat until b=0
indr           ED BA   b*21-5 x1xxxx same than ind, repeat until b=0
otir           ED B3   b*21-5 x1xxxx same than outi, repeat until b=0
otdr           ED BB   b*21-5 x1xxxx same than outd, repeat until b=0
```

Z80 Interrupts

Interrupt Flip-Flop (IFF1,IFF2)
The IFF1 flag is used to enable/disable INTs (maskable interrupts).
In a raw INT-based system, IFF2 is always having the same state than IFF1. However, in a NMI-based system the IFF2 flag is used to backup the recent IFF1 state prior to NMI execution, and may be used to restore IFF1 upon NMI completion by RETN opcode.
Beside for the above 'backup' function, IFF2 itself is having no effect. Neither IFF1 nor IFF2 affect NMIs which are always enabled.
The following opcodes/events are modifying IFF1 and/or IFF2:

```
EI      IFF1=1, IFF2=1
DI      IFF1=0, IFF2=0
<INT>   IFF1=0, IFF2=0
<NMI>   IFF1=0
RETI    IFF1=IFF2
RETN    IFF1=IFF2
```

When using the EI instruction, the new IFF state isn't applied until the next instruction has completed (this ensures that an interrupt handler which is using the sequence "EI, RET" may return to the main program before the next interrupt is executed).
Interrupts can be disabled by the DI instruction (IFF=0), and are additionally automatically each time when an interrupt is executed.

Interrupt Execution
An interrupt is executed when an interrupt is requested by the hardware, and IFF is set. Whenever both conditions are true, the interrupt is executed after the completion of the current opcode.
Note that repeated block commands (such like LDIR) can be interrupted also, the interrupt return address on the stack then points to the interrupted opcode, so that the instruction may continue as normal once the interrupt handler returns.

Interrupt Modes (IM 0,1,2)
The Z80 supports three interrupt modes which can be selected by IM 0, IM 1, and IM 2 instructions. The table below describes the respective operation and execution time in each mode.

```
Mode  Cycles  Refresh  Operation
0     1+var   0+var    IFF1=0,IFF2=0, read and execute opcode from databus
1     12      1        IFF1=0,IFF2=0, CALL 0038h
2     18      1        IFF1=0,IFF2=0, CALL [I*100h+databus]
```

Mode 0 requires an opcode to be output to the databus by external hardware, in case that no byte is output, and provided that the 'empty' databus is free of garbage, then the CPU might tend to read a value of FFh (opcode RST 38h, 11 cycles, 1 refresh) - the clock cycles (11+1), refresh cycles (1), and executed operation are then fully identical as in Mode 1.
Mode 1 interrupts always perform a CALL 0038h operation. The downside is that many systems may have ROM located at this address, making it impossible to hook the interrupt handler directly.

Mode 2 calls to a 16bit address which is read from a table in memory, the table pointer is calculated from the "I" register (initialized by LD I,A instruction) multiplied by 100h, plus an index byte which is read from the databus. The following trick may be used to gain stable results in Mode 2 even if no index byte is supplied on the databus: For example, set I=40h the origin of the table will be then at 4000h in memory. Now fill the entire area from 4000h to 4100h (101h bytes, including 4100h) by the value 41h. The CPU will then perform a CALL 4141h upon interrupt execution - regardless of whether the randomized index byte is an even or odd number.

Non-Maskable Interrupts (NMIs)

Unlike INTs, NMIs cannot be disabled by the CPU, ie. DI and EI instructions and the state of IFF1 and IFF2 do not have effect on NMIs. The NMI handler address is fixed at 0066h, regardless of the interrupt mode (IM). Upon NMI execution, IFF1 is cleared (disabeling maskable INTs - NMIs remain enabled, which may result in nested execution if the handler does not return before next NMI is requested). IFF2 remains unchanged, thus containing the most recent state of IFF1, which may be used to restore IFF1 if the NMI handler returns by RETN instruction.
Execution time for NMIs is unknown (?).

RETN (return from NMI and restore IFF1)
Intended to return from NMI and to restore the old IFF1 state (assuming the old state was IFF1/IFF2 both set or both cleared).

RETI (return from INT with external acknowledge)
Intended to return from INT and to notify peripherals about completion of the INT handler, the Z80 itself doesn't send any such acknowledge signal (instead, peripherals like Z80-PIO or Z80-SIO must decode the databus during /M1 cycles, and identify the opcode sequence EDh,4Fh as RETI). Aside from such external handling, internally, RETI is exactly same as RETN, and, like RETN it does set IFF1=IFF2 (though in case of RETI this is a dirt effect without practical use; within INT handlers IFF1 and IFF2 are always both zero, or when EI was used both set). Recommended methods to return from INT are: EI+RETI (when needing the external acknowledge), or EI+RET (faster).

Z80 Meaningless and Duplicated Opcodes

Mirrored Instructions
NEG (ED44) is mirrored to ED4C,54,5C,64,6C,74,7C.
RETN (ED45) is mirrored to ED55,65,75.
RETI (ED4D) is mirrored to ED5D,6D,7D.

Mirrored IM Instructions
IM 0,X,1,2 (ED46,4E,56,5E) are mirrored to ED66,6E,76,7E.
Whereas IM X is an undocumented mirrored instruction itself which appears to be identical to either IM 0 or IM 1 instruction (?).

Duplicated LD HL Instructions
LD (nn),HL (opcode 22NNNN) is mirrored to ED63NNNN.
LD HL,(nn) (opcode 2ANNNN) is mirrored to ED6BNNNN.
Unlike the other instructions in this chapter, these two opcodes are officially documented. The clock/refresh cycles for the mirrored instructions are then 20/2 instead of 16/1 as for the native 8080 instructions.

Mirrored BIT N,(ii+d) Instructions
Unlike as for RES and SET, the BIT instruction does not support a third operand, ie. DD or FD prefixes cannot be used on a BIT N,r instruction in order to produce a BIT r,N,(ii+d) instruction. When attempting this, the 'r' operand is ignored, and the resulting instruction is identical to BIT N,(ii+d).
Except that, not tested yet, maybe undocumented flags are then read from 'r' instead of from ii+d(?).

Non-Functional Opcodes
The following opcodes behave much like the NOP instruction.
ED00-3F, ED77, ED7F, ED80-9F, EDA4-A7, EDAC-AF, EDB4-B7, EDBC-BF, EDC0-FF.
The execution time for these opcodes is 8 clock cycles, 2 refresh cycles.
Note that some of these opcodes appear to be used for additional instructions by the R800 CPU in newer turbo R (MSX) models.

Ignored DD and FD Prefixes
In some cases, DD-prefixes (IX) and FD-prefixes (IY) may be ignored by the CPU. This happens when using one (or more) of the above prefixes prior to instructions that already contain an ED, DD, or FD prefix, or prior to any instructions that do not support IX, IY, IXL, IXH, IYL, IYH operands. In such cases, 4 clock cycles and 1 refresh cycle are counted for each ignored prefix byte.

Z80 Garbage in Flag Register

Nocash Z80-flags description
This chapter describes the undocumented Z80 flags (bit 3 and 5 of the Flags Register), these flags are affected by ALL instructions that modify one or more of the normal flags - all OTHER instructions do NOT affect the undocumented flags.

For some instructions, the content of some flags has been officially documented as 'destroyed', indicating that the flags contain garbage, the exact garbage calculation for these instructions will be described here also.

All information below just for curiosity. Keep in mind that Z80 compatible CPUs (or emulators) may not supply identical results, so that it wouldn't be a good idea to use these flags in any programs (not that they could be very useful anyways).

Normal Behaviour for Undocumented Flags
In most cases, undocumented flags are copied from the Bit 3 and Bit 5 of the result byte. That is "A AND 28h" for:

```
RLD; CPL; RLCA; RLA; LD A,I; ADD OP; ADC OP; XOR OP; AND OP;
RRD; NEG; RRCA; RRA; LD A,R; SUB OP; SBC OP; OR OP ; DAA.
```

When other operands than A may be modified, "OP AND 28h" for:

```
RLC OP; RL OP; SLA OP; SLL OP; INC OP; IN OP,(C);
RRC OP; RR OP; SRA OP; SRL OP; DEC OP
```

For 16bit instructions flags are calculated as "RR AND 2800h":

```
ADD RR,XX; ADC RR,XX; SBC RR,XX.
```

Slightly Special Undocumented Flags

For 'CP OP' flags are calculated as "OP AND 28h", that is the unmodified operand, and NOT the internally calculated result of the comparision.
For 'SCF' and 'CCF' flags are calculated as "(A OR F) AND 28h", ie. the flags remain set if they have been previously set.
For 'BIT N,R' flags are calculated as "OP AND 28h", additionally the P-Flag is set to the same value than the Z-Flag (ie. the Parity of "OP AND MASK"), and the S-flag is set to "OP AND MASK AND 80h".

Fatal MEMPTR Undocumented Flags

For 'BIT N,(HL)' the P- and S-flags are set as for BIT N,R, but the undocumented flags are calculated as "MEMPTR AND 2800h", for more info about MEMPTR read on below.
The same applies to 'BIT N,(ii+d)', but the result is less unpredictable because the instruction sets MEMPTR=ii+d, so that undocumented flags are "<ii+d> AND 2800h".

Memory Block Command Undocumented Flags

For LDI, LDD, LDIR, LDDR, undocumented flags are "((A+DATA) AND 08h) + ((A+DATA) AND 02h)*10h".
For CPI, CPD, CPIR, CPDR, undocumented flags are "((A-DATA-FLG_H) AND 08h) + ((A-DATA-FLG_H) AND 02h)*10h", whereas the CPU first calculates A-DATA, and then internally subtracts the resulting H-flag from the result.

Chaotic I/O Block Command Flags

The INI, IND, INIR, INDR, OUTI, OUTD, OTIR, OTDR instructions are doing a lot of obscure things, to simplify the description a placeholder called DUMMY is used in the formulas.

```
DUMMY = "REG_C+DATA+1"    ;for INI/INIR
DUMMY = "REG_C+DATA-1"    ;for IND/INDR
DUMMY = "REG_L+DATA"      ;for OUTI,OUTD,OTIR,OTDR
FLG_C = Carry  of above "DUMMY" calculation
FLG_H = Carry  of above "DUMMY" calculation (same as FLG_C)
FLG_N = Sign   of "DATA"
FLG_P = Parity of "REG_B XOR (DUMMY AND 07h)"
FLG_S = Sign   of "REG_B"
UNDOC = Bit3,5 of "REG_B AND 28h"
```

The above registers L and B are meant to contain the new values which are already incremented/decremented by the instruction.
Note that the official docs mis-described the N-Flag as set, and the C-Flag as not affected.

DAA Flags

Addition (if N was 0):

```
FLG_H = (OLD_A AND 0Fh) > 09h
FLG_C = Carry of result
```

Subtraction (if N was 1):

```
FLG_H = (NEW_A AND 0Fh) > 09h
FLG_C = OLD_CARRY OR (OLD_A>99h)
```

For both addition and subtraction, N remains unmodified, and S, Z, P contain "Sign", Zero, and Parity of result (A). Undocumented flags are set to (A AND 28h) as normal.

Mis-documented Flags

For all XOR/OR: H=N=C=0, and for all AND: H=1, N=C=0, unlike described else in Z80 docs. Also note C,N flag description bug for I/O block commands (see above).

Internal MEMPTR Register

This is an internal Z80 register, modified by some instructions, and usually completely hidden to the user, except that Bit 11 and Bit 13 can be read out at a later time by BIT N,(HL) instructions.
The following list specifies the resulting content of the MEMPTR register caused by the respective instructions.

```
Content Instruction
A*100h  LD (xx),A              ;xx=BC,DE,nn
xx+1    LD A,(xx)              ;xx=BC,DE,nn
nn+1    LD (nn),rr; LD rr,(nn) ;rr=BC,DE,HL,IX,IY
rr      EX (SP),rr             ;rr=HL,IX,IY (MEMPTR=new value of rr)
rr+1    ADD/ADC/SBC rr,xx      ;rr=HL,IX,IY (MEMPTR=old value of rr+1)
HL+1    RLD and RRD
dest    JP nn; CALL nn; JR nn  ;dest=nn
dest    JP f,nn; CALL f,nn     ;regardless of condition true/false
dest    RET; RETI; RETN        ;dest=value read from (sp)
dest    RET f; JR f,nn; DJNZ nn ;only if condition=true
00XX    RST n
adr+1   IN A,(n)               ;adr=A*100h+n, memptr=A*100h+n+1
bc+1    IN r,(BC); OUT (BC),r  ;adr=bc
ii+d    All instructions with operand (ii+d)
```

Also the following might or might not affect MEMPTR, not tested yet:

```
OUT (N),A and block commands LDXX, CPXX, INXX, OUTXX
and probably interrupts in IM 0, 1, 2
```

All other commands do not affect the MEMPTR register - this includes all instructions with operand (HL), all PUSH and POP instructions, not executed conditionals JR f,d, DJNZ d, RET f (ie. with condition=false), and the JP HL/IX/IY jump instructions.

Z80 Compatibility

The Z80 CPU is (almost) fully backwards compatible to older 8080 and 8085 CPUs.

Instruction Format
The Z80 syntax simplifies the chaotic 8080/8085 syntax. For example, Z80 uses the command "LD" for all load instructions, 8080/8085 used various different commands depending on whether the operands are 8bit registers, 16bit registers, memory pointers, and/or an immediates. However, these changes apply to the source code only - the generated binary code is identical for both CPUs.

Parity/Overflow Flag
The Z80 CPU uses Bit 2 of the flag register as Overflow flag for arithmetic instructions, and as Parity flag for other instructions. 8080/8085 CPUs are always using this bit as Parity flag for both arithmetic and non-arithmetic instructions.

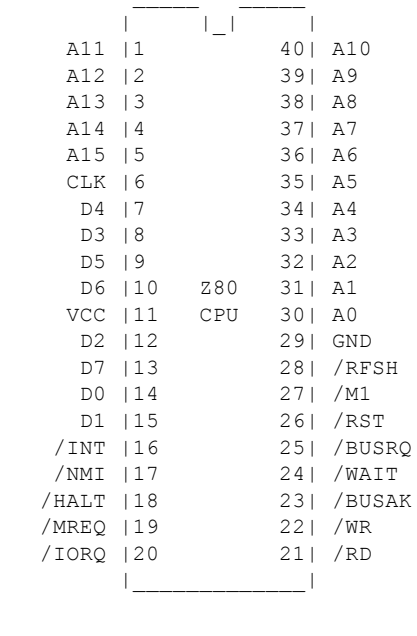
Z80 Specific Instructions
The following instructions are available for Z80 CPUs only, but not for older 8080/8085 CPUs:
All CB-prefixed opcodes (most Shift/Rotate, all BIT/SET/RES commands).
All ED-prefixed opcodes (various instructions, and all block commands).
All DD/FD-prefixed opcodes (registers IX and IY).
As well as DJNZ nn; JR nn; JR f,nn; EX AF,AF; and EXX.

8085 Specific Instructions
The 8085 instruction set includes two specific opcodes in addition to the 8080 instruction set, used to control 8085-specifc interrupts and SID and SOD input/output signals. These opcodes, RIM (20h) and SIM (30h), are not supported by Z80/8080 CPUs.

Z80 vs Z80A
Both Z80 and Z80A are including the same instruction set, the only difference is the supported clock frequency (Z80 = max 2.5MHz, Z80A = max 4MHz).

NEC-780 vs Zilog-Z80
These CPUs are apparently fully compatible to each other, including for undocumented flags and undocumented opcodes.

Z80 Pin-Outs



Z80 Local Usage

Playchoice 10 (Z80)
Clocked at 4.000MHz.
NMIs are triggered on Vblank start (of the Z80 video circuit).
Normal interrupts are not used. There is no DRAM (no Refresh used).
There is memory mapped I/O at E000h..FFFFh. Writes to that area are done by LD [RR],R and LD [RR],NN and PUSH RR opcodes. Reads from that area are done by BIT N,[HL], XOR A,[HL], and JP HL opcodes (JP HL is executing a RST NN opcode in the memory mapped I/O area).

FamicomBox

Nintendo FamicomBox (SSS-CDS) aka Sharp FamicomStation.

Info from Kevin Horton.

FamicomBox Memory and I/O Maps

Overall Memory Map

```
0000h-1FFFh  8K RAM (with write-protect feature, see port 5002h.W)
2000h-3FFFh  PPU Registers
4000h-4FFFh  CPU/APU RP2A03E Registers
5000h-5FFFh  FamicomBox Registers
6000h-7FFFh  8K RAM (temporary storage, used for TEST mode only)
8000h-FFFFh  Cartridge space
```

RAM Usage

```
0000h..07FFh Standard NES Work RAM (variables for MENU and GAME)
0800h..08FFh FamicomBox Variables
0900h..09FFh FamicomBox Slot Counters (Total & per-game counters)
0A00h..0A18h FamicomBox Variables
0A19h..0DFFh FamicomBox Unused
0E00h..0FFFh FamicomBox Slot Directory (Game Titles with Chksum and Status)
1000h..1FFFh FamicomBox Program Code (relocated from Menu ROM)
6000h..7FFFh Unknown purpose (temporary storage, used for TEST mode only)
```

Memory at 0800h..1FFFh is write-protected during Game execution (as far as known, it isn't battery-backed, but the Famibox doesn't have a power-switch, so the RAM is permanently powered; unless one unplugs the power supply).
Unknown if 6000h..7FFFh is also mapped during Game execution - if so, it may clash with (or replace and act as?) SRAM used by Game cartridges?

I/O Map (Output)

```
4016h.W - Joypad Strobe
5000h.W - Exception Trap Enable Bits        (reset to 00h on power-up only)
5001h.W - Coin Chip Params & CATV Outputs   (reset to 00h on power-up only)
5002h.W - Slot LED and RAM protect register (reset to 00h when CPU is reset)
5003h.W - Exception Trap Attraction Timer
5004h.W - Slot ROM Cart control register    (reset to 00h when CPU is reset)
5005h.W - Misc Control                      (reset to 00h on power-up only)
5006h.W - Test Connector DB-25 Outputs (not reset, uninitialzed at power-up)
5007h.W - Expansion 50-pin Edge Connector, 8bit Output
```

I/O Map (Input)

```
4016h.R - Watchdog Reload, and Joypad 1
4017h.R - Watchdog Reload, and Joypad 2, and Zapper
5000h.R - Exception Trap Flags
5001h.R - Not used
5002h.R - Dip Switch Register
5003h.R - Keyswitch Position & Coin Chip Status
5004h.R - Test Connector DB-25 Inputs
5005h.R - Expansion 50-pin Edge Connector, 8bit Input 1
5006h.R - Expansion 50-pin Edge Connector, 8bit Input 2
5007h.R - Misc Status
```

FamicomBox I/O Ports

5000h.W - Exception Trap Enable Bits (reset to 00h on power-up only)

```
0    6.82Hz interrupt source          (0=Enable, 1=Disable)
1    Attraction Timer                 (0=Disable, 1=Enable)
2    Controller reads                 (0=Disable, 1=Enable)
3    Keyswitch rotation               (0=Disable, 1=Enable)
4    Coin insertion (and/or EXPIRED?) (0=Disable, 1=Enable)
5    Reset Button                     (0=Disable, 1=Enable)
6    Not used                         (Watchdog is always Enabled)
7    CATV connector Pin 1 detection   (0=Disable, 1=Enable)
```

5000h.R - Exception Trap Flags

```
  0    6.82Hz interrupt source hit              (0=Trapped, 1=Normal)
  1    Attraction Timer Expired                 (0=Trapped, 1=Normal)
  2    Controller(s) were read (and PRESSED?)   (0=Trapped, 1=Normal)
  3    Keyswitch was rotated                    (0=Trapped, 1=Normal)
  4    Coin was inserted (and/or EXPIRED?)      (0=Trapped, 1=Normal)
  5    Reset Button was pressed                 (0=Trapped, 1=Normal)
  6    Watchdog Timer Expired (after 17.5s)     (0=Trapped, 1=Normal)
  7    CATV connector Pin 1 went high/open      (0=Trapped, 1=Normal)
```

When 5000h.R is read, this register is reset, and the exception trap controller is reset.
There seems to be a 9th exception flag in 5007h.R.Bit0. If the NINE flags are all ones, then the reset can be considered to be caused by a "PROTECT IC" (aka CIC, presumably) (see menu error 0Fh at 9067h).
When one of the above exceptions occurs, the CPU is reset. The menu code then reads to see which of the sources caused the reset, and acts accordingly.

5003h.W - Exception Trap Attraction Timer

```
  0-7  Counter value (decremented at 6.8274Hz) (255=max=37 seconds)
```

When the timer wraps from 00h to FFh it triggers an exception, if it is enabled. (See 5000h.W and 5000h.R)
This timer is used for the "attract" mode. It lets the game run for several seconds before it brings the menu back.

5001h.W - Coin Chip Params & CATV Outputs (reset to 00h on power-up only)

```
  0    Coin Chip pin 1  (1=Ten Minutes per Coin)
  1    Coin Chip pin 2  (1=Twenty Minutes per Coin)
  2    Coin Chip pin 3  (should be always 0, except in test mode)
  3    Coin Chip pin 4  (should be always 0, except in test mode)
  4    Coin Chip pin 12 (should be always 0, except in test mode)
  5    Coin Chip pin 14 (1=Apply Minutes?)
  6    CATV Connector pin 7 (0=Free-Menu/Demo, 1=Play-and-Pay; unless TV Mode?)
  7    CATV Connector pin 8 (0=High=Okay, 1=Low=Joypad/Zapper-Disconnect-Alert)
```

5002h.W - Slot LED and RAM protect register (reset to 00h when CPU is reset)

```
  0-3  LED Select (00h=None, 01h..0Fh=Cartridge Slot LED 1..15)
  4-6  RAM Write Enable (0=None, 1=0-07FFh, 2=0-0FFFh, 3=0-17FFh, 4-7=0-1FFFh)
  7    LED Flash (3.414Hz) (high=flash, low=steady)
```

BIT7: err... 1=high, 0=low or what? err... flash=blinking? and steady... means always on? err... what LED(s)... the currently selected cart-led? power-led?

5004h.W - Slot ROM Cart control register (reset to 00h when CPU is reset)

```
  0-3  Cart Number select (00h=Menu, 01h..0Fh=Game Cart1-15)
  4-5  Cart Row select    (00h=Menu, 01h=Cart1-5, 02h=Cart6-10, 03h=Cart11-15)
  6    Lock (0=No change, 1=Disable Port 500xh or so; until Reset)
  7    NC
```

Proper cart selection requires setting up BOTH the desired cart number (00h-0Fh), AND the proper row.

5005h.W - Misc Control (reset to 00h on power-up only)

```
  0    Latching Relay (1=Flip to position A) (coil on pins 1 & 10)
  1    Coin Chip pin ?    ;err... which pin?  (0=Operate, 1=Config/Reset?)
  2    Zapper GND (0=Disable/Low-Z; for 5007h.R connect-check, 1=Enable/GND)
  3    enable 40% input of modulator (0=Disable, 1=Enable)
  4    NC
  5    maps to 5007h.R.Bit7     (warmboot flag or so?)  (and to 50-pin ??)
  6    Joypad Enable  (0=Enable, 1=Disable)
  7    Joypad Swap    (0=Swap, 1=Normal)       swapping only swaps D0 and CLK
```

err... what is this register having to do with "CATV"...? maybe the relay?

5007h.R - Misc Status

```
  0    TV type selection (0=TV, 1=Game)   ;or... trap flag bit8... CIC, 1=trap?
  1    Keyswitch turned  (0=No, 1=in middle of two positions)
  2    Zapper GND (0=Enabled or Disabled+Disconnected, 1=Disabled+Connected)
  3    Expansion 50-pin Edge Connector, Pin 21 (inverted)
  4    CATV Connector pin 8 (0=Low, 1=High) (see also: 5001h.W.Bit7)
  5    Relay Position (0=Position A, 1=Position B) (For CATV 0=Force Demo Only)
  6    Expansion 50-pin Edge Connector, Pin 22 (inverted)
  7    5005h.W.Bit5 (inverted)                 ;coldboot/warmboot flag?
  7    err, and/or Expansion 50-pin Edge Connector, Pin 23
```

5002h.R - Dip Switch Register

```
  0   DIP SW 1 Self Test            (0=Off=Normal, 1=On=Do continuous selftest)
  1   DIP SW 2 Coin timeout period  (0=Off=10 minutes, 1=On=20 minutes)
  2   DIP SW 3 Not used             (0=Off, 1=On)
  3   DIP SW 4 Famicombox menu time (0=Off=7 seconds, 1=On=12 seconds)
  4   DIP SW 5 Attract time, bit0 ;\(0..3 = 12,17,23,7 seconds)
  5   DIP SW 6 Attract time, bit1 ;/
  6   DIP SW 7 Mode, bit0 ;\(0=KEY MODE, 1=CATV MODE, 2=COIN MODE, 3=FREEPLAY)
  7   DIP SW 8 Mode, bit1 ;/
 N/A  DIP SW 9 Feep Disable         (On=Mute the Coin feep)
 N/A  DIP SW 10 Controller 2 D3,D4  (Off=Disable, On=Enable) (Zapper pins)
```

Attract time: How long each game plays while in attract mode before switching to the next.

5003h.R - Keyswitch Position & Coin Chip Status

```
0    Key Position 0 (0=No, 1=Yes) (??? from left) ;-(demo mode?)
1    Key Position 1 (0=No, 1=Yes) (??? from left) ;\SELECT GAME (play modes)
2    Key Position 2 (0=No, 1=Yes) (??? from left) ;/
3    Key Position 3 (0=No, 1=Yes) (??? from left) ;-SELF CHECK and GAME CHECK
4    Key Position 4 (0=No, 1=Yes) (??? from left) ;-Lockup ???
5    Key Position 6 (0=No, 1=Yes) (1st from left) ;-GAME TITLE & COUNT
6    Coin Chip Pin 9  (0=Empty/Expired, 1="Money system enabled")
7    Coin Chip Pin 10 (not used by existing software) (MAYBE expired-trap?)
```

Note: There are Black visitor keys, and red master keys. The visitor keys can be probably turned to only 2 positions.
The naming for the key positions is very confusing. Obvious names would be "Nth from Left" or "Bit0-5" (which may not be numbered straightly). Other naming schemes include "ON-OFF" (on front panel, without markings on the other four positions), "1 2 3 4 5 6" (used in the Menu cartridge), "0 1 2 3 4 6" (used by kevtris, mabe pin-numbers, with 5=GND), and "1-OFF-ON-2-3" (seen on a japanese webpage).

5001h.R - Not used

```
0-7  Not used (open bus)
```

5004h.R - Test Connector DB-25 Inputs

```
0-7  Input R0..R7 (DB-25 pins 2,15,3,16,4,17,5,18) (inverted; 0=High, 1=Low)
```

5006h.W - Test Connector DB-25 Outputs (not reset, uninitialzed at power-up)

```
0-7  Output W0..W7 (DB-25 pins 6,15,7,16,8,17,9,18)
```

5005h.R - Expansion 50-pin Edge Connector, 8bit Input 1

```
0-7  Input from CPU Databus; with 5005h.Read signal on Expansion Port pin 28
```

5006h.R - Expansion 50-pin Edge Connector, 8bit Input 2

```
0-7  Input from CPU Databus; with 5006h.Read signal on Expansion Port pin 27
```

5007h.W - Expansion 50-pin Edge Connector, 8bit Output

```
0-7  Output to CPU Databus; with 5007h.Write signal on Expansion Port pin 26
```

4016h.W - Joypad Strobe

```
0    DB-15 pin 12 and Joypad 1/2 Strobe pin 3
1    DB-15 pin 11
2    DB-15 pin 10
3-7  Not used
```

4016h.R - Watchdog Reload, and Joypad 1

```
0    Joypad 1 D0 pin 4                  ;Joypad 1 (or joypad 2 when swapped)
1    DB-15 pin 14
2    Expansion 50-pin Edge Connector, Pin 44 (would be Microphone in Famicom)
3    Joypad 1 D3 pin 6
4    Joypad 1 D4 pin 7
5    Expansion 50-pin Edge Connector, Pin 19
6    Expansion 50-pin Edge Connector, Pin 45
7    Expansion 50-pin Edge Connector, Pin 20
```

Reading 4016h or 4017h resets the 4bit Watchdog timer (this must be done at least every 17.5 seconds; ie. every fifteen 0.853Hz steps).

4017h.R - Watchdog Reload, and Joypad 2 and Zapper

```
0    DB-15 pin 8 and Joypad 2 D0 pin 4  ;Joypad 2 (or joypad 1 when swapped)
1    DB-15 pin 7
2    DB-15 pin 6
3    DB-15 pin 5 and Joypad 2 D3 pin 6  ;Zapper Light   ;\when DIP10=Off:
4    DB-15 pin 4 and Joypad 2 D4 pin 7  ;Zapper Trigger ;/only DB-15 pins
5    Expansion 50-pin Edge Connector, Pin 17
6    Expansion 50-pin Edge Connector, Pin 43
7    Expansion 50-pin Edge Connector, Pin 18
```

Reading 4016h or 4017h resets the 4bit Watchdog timer (this must be done at least every 17.5 seconds; ie. every fifteen 0.853Hz steps).

FamicomBox Misc

Counter chain

```
21.47727MHz/3/2000h =  873.91Hz (feep tone, when coin inserted)
21.47727MHz/3/100000h = 6.827Hz (clock for 8bit attraction timer)
21.47727MHz/3/200000h = 3.414Hz (LED flash)
21.47727MHz/3/800000h = 0.853Hz (clock for 4bit watchdog timer)
```

Watchdog timer

A 4 bit binary up counter is used. It is clocked at 0.85Hz. When a controller is read (either one), this timer is reset. When the count wraps from 0Fh to 00h, an exception is generated. (see 5000R and 5000W)

Audio chain

The audio chain is kinda interesting. The audio passes from the two pins on the CPU, through two hex inverter audio amplifiers, and then the outputs of these go to the 50-pin expansion connector, and then through two more hex inverter amplifiers. The outputs of those two amps is combined with external audio (again from the 50-pin expansion connector) and the coin beep, which is finally fed to the DB-15, DB-25, and the modulator and audio jack.
The beep is made whenever coin is inserted.

Lockout
The 3198 lockout chip in the famicombox carts is kinda odd. The resistor network connects 4 lines from the expansion pins (on the cart) to 4 pins on the lockout chip.
Unlike the regular lockout chips, the 3198 is designed to work in a network of up to 16 lockout chips. Each of the 15 cart slots on the front + the menu board has a unique mapping of the 4 lines; pulled high or low depending on which slot it is in.
The master lockout chip on the main board has its 4 address lines connected to a latch which is written to by the CPU to select 1 of the 16 slaves on the various carts. The slave lockout chip matching the desired chip sent out by the master responds, while the other up to 15 chips do not.
This is how the system addresses each cart's lockout chip to see if a cart exists at that address.

FamicomBox Cartridges

Software Requirements
For whatever reason, most or all game cartridges are containing PRG-ROM and CHR-ROM stored on EPROMs rather than ROMs. The FamicomBox EPROMs are usually (maybe always) containing exactly the same data as normal Famicom ROMs, without any FamicomBox specific modifications.
However, as far as known, the Watchdog feature is active even in Game-mode, so games MUST read the controller ports (either 4016h or 4017h) at least every 17.5 seconds; that might be a problem with a few games (with excessive intros without controller input).
And, the games MUST contain a "header" with valid Checksums and Title (see below). The only exception are older games (that were produced before the FamicomBox was released; ie. before 1986/1987), for such games, the FamicomBox menu cartridge contains a database with about 150 known checksums & titles.
FamicomBox ROM Header (at FFE0h)

Hardware Restrictions
According to Kevin Horton, FamicomBox games cannot use IRQs, and can contain memory and I/O ports in the 8000h-FFFFh area only (though unknown what exactly is causing that restrictions) (in case of the SRAM area at 6000h-7FFFh: unknown if the games are allowed to use the internal SRAM on the FamicomBox mainboard; instead of battery-backed SRAM used in retail game cartridges).

FamicomBox ROM Header (at FFE0h)

Some cartridges include header information in the last 20h bytes of PRG-ROM (or at the end of EVERY 16K PRG-ROM bank, eg. in "Derby Stallion - Zenkoku Han (J)"). The header was inventend around 1987 for use by the FamicomBox, but it's also present in around 30% of the existing normal retail cartridges.

Summary
```
FFE0h 16 Title in ASCII, max 16 chars (right-justified, at FFExh..FFEFh)
FFF0h 2  PRG-ROM Checksum (big-endian) (ALL bytes, or only last 16Kbytes)
FFF2h 2  CHR-ROM Checksum (big-endian) (ALL bytes, 0000h if no CHR-ROM)
FFF4h 1  Unknown/Unused Cartridge Size (MSB=PRG-ROM, LSB=CHR-ROM/RAM) (or so)
FFF5h 1  Mapper Type (implies Checksum Type), and bit7=Name Table Mirroring
FFF6h 1  Unknown/Unused (00h=NoTitle?, 01h=Normal, 02h=Mapper4?)
FFF7h 1  Length of Title minus 1 (typically 02h..0Fh) (or often 10h=Corrupt)
FFF8h 1  Maker Code (same as for Gameboy and SNES) (01h=Nintendo, etc.)
FFF9h 1  Header Checksum (00h minus all bytes at [FFF2h..FFF8h])
FFFAh 2  CPU NMI Vector
FFFCh 2  CPU RESET Vector
FFFEh 2  CPU IRQ/BRK Vector
```

PRG-ROM Checksum
This is the sum of either ALL or SOME bytes of PRG-ROM added together, minus the checksum bytes at [FFF0h,FFF1h). The FamicomBox DOES verify this value, and works only if the checksum is correctly matched to the mapper type in [FFF5h].Bit0-6:
```
00h NROM    PRG=8K,16K,32K
01h CNROM   PRG=8K,16K,32K
02h UNROM   PRG=128K (fixed size) (8 banks of 16K) chksum per whole 128K
03h GNROM?  PRG=128K (fixed size) (4 banks of 32K) chksum per 32K bank
04h MMC's   PRG=16K  chksum per (any) mappable 16K bank(s) at C000h-FFFFh
05h..7Fh    Invalid
```
For NROM/CNROM, the FamicomBox is autodetecting PRG-ROM sizes of 8K, 16K, and 32K and computes the checksum across that region (if the PRG-ROM is smaller, then the checksum is including mirror(s) in the 8K region).
For UNROM, the FamicomBox assumes a fixed size of 128K (aka 8 banks of 16K), and computes ONE checksum across the whole 128K area (if the actual ROM should be bigger/smaller, then it must be clipped/mirrored to 128K size).
For GNROM, the FamicomBox assumes a fixed size of 128K (aka 4 banks of 32K), and computes FOUR checksums across the separate 32K areas; so the ROM must have FOUR headers with FOUR different checksums.
For MMC's, the FamicomBox simply computes a checksum on the 16K area at C000h-FFFFh without trying to perform any bankswitching. If that 16K area is mappable, then all 16K blocks must contain separate headers with separate checksums (unless maybe if the cartridge contains reset logic that does

ensure a specific initial bank number to be mapped during checksumming).

CHR-ROM Checksum
This is simply the sum of ALL bytes in CHR-ROM added together (or 0000h if there's is no CHR-ROM). The FamicomBox doesn't verify this value, and thus doesn't require any mapper specific checksumming mechanisms for CHR-ROM.

Bad Headers
The "FamicomBox" headers found in retail cartridges are often incomplete or corrupted. For example, some carts have the title centered or left-justified (rather than right-justified within the 16-byte region). Many carts have 16-byte title length defined as 10h (rather than 0Fh). Some have checksums in little-endian (instead of big-endian), or completely missing or incorrect checksums.
Unknown if the headers are also used by other consoles (such like maybe the M82 demonstration unit).

FamicomBox Pinouts

72-Pin Cartridge Slots (16 slots: for menu cart and 15 game carts)
Mostly same as 72-pin NES cartridges, see:
Cartridge Pin-Outs
Pin 16-19 are the 4bit SlotID; for use by the FamicomBox CIC chip (on NES carts, these pins are unused "expansion" pins).

26pin Front Panel connector (from Mainboard to Front Panel)
```
 1 - +5V
 2 - +5V
 3 - Front LED anode (cathode is grounded) (green LED "indicating TV/game")
 4 - TV/Game Button (5V on other end, connected to pin 6)
 5 - RESET Button (Low=Pressed) (used to return to Menu)
 6 - +5V
 7 - Coin Connector Pin 3 and connects to LED (err... what LED??)
 8 - Coin Connector Pin 1 and TEST Button (Low=Add credit)
 9 - Controller Pin 1 on Port 3:   (5007h.R.Bit2, 5005h.W.Bit2, Zapper GND)
10 - Controller Pin 4 on Port 1:   (4016h.R.Bit0, Joypad 1 Data)  ;\can be
11 - Controller Pin 4 on Port 2:   (4017h.R.Bit0, Joypad 2 Data)  ;/swapped
12 - Controller Pin 2 on Port 1:   (4016h.Read, Joypad 1 Clock)   ;\can be
13 - Controller Pin 2 on Port 2-3: (4017h.Read, Joypad 2 Clock)   ;/swapped
14 - Controller Pin 3 on Port 1-2: (4016h.W.Bit0, Joypad 1/2 Strobe, OUT0)
15 - Controller Pin 7 on Port 2-3: (4017h.R.Bit4, Zapper Trigger) ;\only when
16 - Controller Pin 6 on Port 2-3: (4017h.R.Bit3, Zapper Light)   ;/DIP10=On
17 - Controller Pin 6 on Port 1:   (4016h.R.Bit3)
18 - Controller Pin 7 on Port 1:   (4016h.R.Bit4)
19 - Keyswitch position 0 (Low=Selected) (5003h.R.Bit0)
20 - Keyswitch position 1 (Low=Selected) (5003h.R.Bit1)
21 - Keyswitch position 2 (Low=Selected) (5003h.R.Bit2)
22 - Keyswitch position 3 (Low=Selected) (5003h.R.Bit3)
23 - Keyswitch position 4 (Low=Selected) (5003h.R.Bit4)
24 - Keyswitch position 6 (Low=Selected) (5003h.R.Bit5)
25 - GND
26 - GND
```

3pin Coin Mechanics connector (from Front Panel to External Coin Unit)
```
1 - (pin8 of front panel PCB) (Low=Add credit) (also wired to TEST button)
2 - GND
3 - (pin7 of front panel PCB) "connects to LED"   err.. what LED, what for?
```

7pin Controller Ports (3 pieces; for 2 Joypads and 1 Zapper)
```
Pin Purpose        Port 1 (Joy1) Port 2 (Joy2)       Port 3 (Zapper)
1 - Ground         GND           GND                 5005h.W.Bit2/5007h.R.Bit2
2 - Joypad Clock   4016h.R       4017h.R             4017h.R       .---------.
3 - Joypad Strobe  4016h.W.Bit0  4016h.W.Bit0        NC            | 4 3 2 1 |
4 - Joypad Data    4016h.R.Bit0  4017h.R.Bit0        NC            | 7 6 5  /
5 - Supply         +5V           +5V                 +5V           '-------'
6 - Zapper Light   4016h.R.Bit3  4017h.R.Bit3/DIP10 4017h.R.Bit3/DIP10
7 - Zapper Button  4016h.R.Bit4  4017h.R.Bit4/DIP10 4017h.R.Bit4/DIP10
```

DB-15 on the back of unit (marked "15P expand"):
```
 1 - GND                9 - 4017h.R enable
 2 - audio output      10 - 4016h.W.2
 3 - /IRQ              11 - 4016h.W.1
 4 - 4017h.R.4         12 - 4016h.W.0
 5 - 4017h.R.3         13 - 4016h.R.1
 6 - 4017h.R.2         14 - 4016h.R enable
 7 - 4017h.R.1         15 - +5V
 8 - 4017h.R.0
```
Note: The pinout is the same as the Famicom's DB-15.

Test Connector DB-25 on the back of the unit (marked "25P expand"):
```
 1 - +5V               14 - /IRQ
```

```
 2 - 5004h.R.0          15 - 5004h.R.1
 3 - 5004h.R.2          16 - 5004h.R.3
 4 - 5004h.R.4          17 - 5004h.R.5
 5 - 5004h.R.6          18 - 5004h.R.7
 6 - 5006h.W.0          19 - 5006h.W.1
 7 - 5006h.W.2          20 - 5006h.W.3
 8 - 5006h.W.4          21 - 5006h.W.5
 9 - 5006h.W.6          22 - 5006h.W.7
10 - GND                23 - GND
11 - GND                24 - M2
12 - GND                25 - GND
13 - GND
```

Could be used as general-purpose I/O port. Some of the Menu's selftest functions expect the DB-25 to be strapped to the DB-15 via an external test-adaptor (and the /IRQ pin strapped to a CATV pin?).

8pin Screw Terminal Block (marked "CATV INTERFACE")

```
1 - 5000h.RW.Bit7 Exception Trap (usually strapped to GND, ie. CATV pin4)
2 - Unknown
3 - Unknown
4 - GND
5 - +5V
6 - Unknown
7 - 5001h.W.Bit6                  ;Play-and-Pay (unless TV mode?)
8 - 5001h.W.Bit7 and 5007h.R.Bit4 ;Joypad/Zapper-Disconnect-Alert
```

Expansion 50-pin Edge Connector (behind a small door on the back left side)

```
 1 - +5V                      26 - 5007h.W enable
 2 - +5V                      27 - 5006h.R enable
 3 - +5V                      28 - 5005h.R enable
 4 - M2                       29 - +5V
 5 - Audio Input              30 - +5V
 6 - +5V                      31 - +5V
 7 - PRG A6                   32 - PRG A7
 8 - PRG A4                   33 - PRG A5
 9 - PRG A2                   34 - PRG A3
10 - PRG A0                   35 - PRG A1
11 - PRG D1                   36 - PRG D0
12 - PRG D3                   37 - PRG D2
13 - PRG D5                   38 - PRG D4
14 - PRG D7                   39 - PRG D6
15 - PRG R/W                  40 - /(5000h-5FFFh)
16 - pin #1 audio             41 - /IRQ
17 - 4017h.R.5                42 - pin #2 audio
18 - 4017h.R.7                43 - 4017h.R.6
19 - 4016h.R.5                44 - 4016h.R.2 (microphone)
20 - 4016h.R.7                45 - 4016h.R.6
21 - 5007h.R.3                46 - GND
22 - 5007h.R.6                47 - GND
23 - 5007h.R.7 (+5005h.W.5 ?) 48 - GND
24 - GND                      49 - GND
25 - GND                      50 - GND
```

3198 - 16pin Lockout Chip (CIC) (17 pieces; 1 on mainboard, 16 in carts)

Lockout chip with additional 4bit SlotID (unlike normal NES lockout chips).

Cartridge Cicurity Chip (CIC) (Lockout Chip)

3199 - 16pin Coin Timer/Coin Chip (1 piece; on mainboard)

This is probably the same type of 4bit CPU which is used in CICs.

```
1   P00  5001h.W.Bit0 (unknown purpose)
2   P01  5001h.W.Bit1 (unknown purpose)
3   P02  5001h.W.Bit2 (unknown purpose)
4   P03  5001h.W.Bit3 (unknown purpose)
5   CL2  whatever MHz clock (or maybe NC)
6   CL1  whatever MHz clock (this one needed)
7   RES  whatever type of reset (maybe I/O controlled?)
8   GND  Supply GND
9   P10  5003h.R.Bit6 (reportedly "Money system enabled")
10  P11  5003h.R.Bit7 (unknown purpose)
11  P12              (unknown purpose)
12  P13  5001h.W.Bit4 (unknown purpose)
13  P20              (unknown purpose)
14  P21  5001h.W.Bit5 (unknown purpose)
15  P22              (unknown purpose)
16  VCC  Supply VCC
```

Note: Reportedly, 5005h.W.Bit1 also connects to whatever 3199 pin. Moreover, the 3199 chip should connect to "a LED", sound feep, 40% video dimming, and coin-insert signal.

Modulator 7pin connector (connection from Mainboard to Modulator)

```
Audio - Mono audio from amplifier+external+coin feep
Video - Comes from the usual transistor circuit on the PPU
```

```
B+    - Supply 5V
B+SW  - Switch (Low=Switches the game in, High=Lets TV signal pass through)
GND   - Supply Ground
40%   - Dimming (undermodulates video by 40%) (Low=Dim, High=Normal)
?     - Unknown (there are seven wires, not only six)
```

The 40% input is weird. When the time is about to expire on the coin mechanism, it feeps and flashes the screen by using this 40% input. The LED on the coin mech also flashes in time with the feeping.

Modems

Japanese Famicom Modems

There are at least 4 different Famicom modems:

```
Famicom Network System HVC-050
Famicom Network System FCNS-A
Famicom Network System Dataship 1200 (Nintendo)
TV-NET MC-1200B
```

All of them do connect to cartridge slot and telephone line (that seems to include the TV-NET thing - it appears to be a regular modem; not a video text decoder).

All four modems seem to include some kind of "cartridge" slot, whereas the "cartridges" seem to be very flat plastic cards; they are almost looking a bit too flat to contain any ROM, SRAM or FLASH memory; so, possibly they are only containing some sort of ID codes for selecting specific online services; encoded in whatever form... possibly magnets, notches, reflectors (?)

Specifically for use with the modems, there have been some numeric keypads, in form of extended joypads, and some in form of (non-wireless) TV-style remote controls:

Controllers - Keypads

Other than that, there isn't anything known about these modems. Unknown if and how far they are compatible with each other. There don't seem to be any dumps of the modem BIOSes.

There seems to be no external storage, so any downloaded content is probably lost when switching off the modem - unless it contains some kind of internal storage. As far as known, the modems have been used for reading news online, and for JRA-PAT horse-betting. So far, for that purposes, it may not have used/required any storage, except possibly for storing details like user names.

US Baton Teleplay Modem

An unreleased NES modem that did exist only in prototype form. Unlike the japanese modems, it has been intended to support multiplayer games.

US Minnesota State Lottery Modem (Control Data Corporation) (1991)

Another unreleased NES modem, intended for online gambling.

Unpredictable Things

Reading Garbage from unused PPU Bits

Semi-stable garbage is returned for 8bit write-only PPU registers (2000h, 2001h, 2003h, 2005h, 2006h), for lower 5bits of the PPU status register (Port 2002h), and for upper 2bit of palette values (Port 2007h at PPU address 3F00h-3FFFh). That garbage value is:

1) The 8bit-value most recently written to any PPU port (2000h-2007h).
2) The 8bit-value most recently read from 2004h or 2007h,
3) The 3bit-value most recently read from 2002h (lower 5bit unchanged).
4) Zero if none of the above has "updated" the garbage for longer period.

Reading from Palette Memory

Palette entries are 6bit values, when reading from palette memory, the upper two bits are garbage (see above). In monochrome mode (Port 2001h/Bit0=1) the returned lower 4bit are zero. Also, palette reads appear a bit unstable, and occasionally return incorrect values (at least on my PAL NES console).

Reading from empty Expansion / SRAM area at 4100h-7FFFh

Returns the most recently fetched data byte. That is: The third opcode byte (direct 16bit addressing), or the second zero-page byte (indirect addressing). In either case, the returned value is the MSB of 16bit BASE address, regardless of any index value which may wrap the MSB to the next page. For example: [4510h]=45h and [44FFh+11h]=44h are receiving different values, even though both are reading from the same memory address.

Reading from empty Expansion / APU area at 4000h-40FFh

Empty bits and bytes are: Write-only APU Ports 4000h-4014h, unused expansion addresses 4018h-40FFh, and unused bits in APU/Joypad Ports: 4015h/Bit5, 4016h/Bit7-5 (NES) or Bit7-3 (Famicom), and 4017h/Bit7-5. Normally returned garbage is 40h (base MSB, as described for 4100h-7FFFh), unless when indexing causes a page-wrap (from 3Fxxh+yyh to 40zzh), in that case several "unpredictable" things are happening:

The CPU adds the index to the address LSB, and reads from that address (3Fzzh) by mistake, in a second cycle the CPU adds the carry-out to the address MSB, and tries to read from the correct address (40zzh). The hardware doesn't output data for 40zzh, so that the CPU receives the most recently fetched data byte, which has been [3Fzzh], which is a mirror of PPU register [200zh]. To the worst, most PPU registers are either write-only, or cannot be read during rendering, see PPU Garbage above.

Cleverly used, this allows to detect the number of unused bits in 4016h, and to separate between NES or Famicom hardware.

CPU 65XX Microprocessor

Overview

Instruction Set

Other Info

CPU Registers and Flags

The 65XX CPUs are equipped with not more than three 8bit general purpose registers (A, X, Y). However, the limited number of registers (and complete lack of 16bit registers other than PC) is parts of covered by comfortable memory operations, especially page 0 of memory (address 0000h-00FFh) may be used for relative fast and complicated operations, in so far one might say that the CPU has about 256 8bit 'registers' (or 128 16bit 'registers') in memory. For details see Memory Addressing chapter.

Registers

Bits	Name	Expl.
8	A	Accumulator
8	X	Index Register X
8	Y	Index Register Y
16	PC	Program Counter
8	S	Stack Pointer (see below)
8	P	Processor Status Register (see below)

Stack Pointer
The stack pointer is addressing 256 bytes in page 1 of memory, ie. values 00h-FFh will address memory at 0100h-01FFh. As for most other CPUs, the stack pointer is decrementing when storing data. However, in the 65XX world, it points to the first FREE byte on stack, so, when initializing stack to top set S=(1)FFh (rather than S=(2)00h).

Processor Status Register (Flags)

Bit	Name	Expl.	
0	C	Carry	(0=No Carry, 1=Carry)
1	Z	Zero	(0=Nonzero, 1=Zero)
2	I	IRQ Disable	(0=IRQ Enable, 1=IRQ Disable)
3	D	Decimal Mode	(0=Normal, 1=BCD Mode for ADC/SBC opcodes)
4	B	Break Flag	(0=IRQ/NMI, 1=RESET or BRK/PHP opcode)
5	-	Not used	(Always 1)
6	V	Overflow	(0=No Overflow, 1=Overflow)
7	N	Negative/Sign	(0=Positive, 1=Negative)

Carry Flag (C)
Caution: When used for subtractions (SBC and CMP), the carry flag is having opposite meaning as for normal 80x86 and Z80 CPUs, ie. it is SET when above-or-equal. For all other instructions (ADC, ASL, LSR, ROL, ROR) it works as normal, whereas ROL/ROR are rotating <through> carry (ie. much like 80x86 RCL/RCR and not like ROL/ROR).

Zero Flag (Z), Negative/Sign Flag (N), Overflow Flag (V)
Works just as everywhere, Z it is set when result (or destination register, in case of some 'move' instructions) is zero, N is set when signed (ie. same as Bit 7 of result/destination). V is set when an addition/subtraction exceeded the maximum range for signed numbers (-128..+127).

IRQ Disable Flag (I)
Disables IRQs when set. NMIs (non maskable interrupts) and BRK instructions cannot be disabled.

Decimal Mode Flag (D)
Packed BCD mode (range 00h..99h) for ADC and SBC opcodes.

Break Flag (B)
The Break flag is intended to separate between IRQ and BRK which are both using the same vector, [FFFEh]. The flag cannot be accessed directly, but there are 4 situations which are writing the P register to stack, which are then allowing the examine the B-bit in the pushed value: The BRK and PHP

opcodes always write "1" into the bit, IRQ/NMI execution always write "0".

CPU Memory Addressing

Opcode Addressing Modes

Name	Native	Nocash
Implied	-	A,X,Y,S,P
Immediate	#nn	nn
Zero Page	nn	[nn]
Zero Page,X	nn,X	[nn+X]
Zero Page,Y	nn,Y	[nn+Y]
Absolute	nnnn	[nnnn]
Absolute,X	nnnn,X	[nnnn+X]
Absolute,Y	nnnn,Y	[nnnn+Y]
(Indirect,X)	(nn,X)	[[nn+X]]
(Indirect),Y	(nn),Y	[[nn]+Y]

Zero Page - [nn] [nn+X] [nn+Y]
Uses an 8bit parameter (one byte) to address the first 256 of memory at 0000h..00FFh. This limited range is used even for "nn+X" and "nn+Y", ie. "C0h+60h" will access 0020h (not 0120h).

Absolute - [nnnn] [nnnn+X] [nnnn+Y]
Uses a 16bit parameter (two bytes) to address the whole 64K of memory at 0000h..FFFFh. Because of the additional parameter bytes, this is a bit slower than Zero Page accesses.

Indirect - [[nn+X]] [[nn]+Y]
Uses an 8bit parameter that points to a 16bit parameter in page zero.
Even though the CPU doesn't support 16bit registers (except for the program counter), this (double-)indirect addressing mode allows to use variable 16bit pointers.

On-Chip Bi-directional I/O port
Addresses (00)00h and (00)01h are occupied by an I/O port which is built-in into 6510, 8500, 7501, 8501 CPUs (eg. used in C64 and C16), be sure not to use the addresses as normal memory. For description read chapter about I/O ports.

Caution
Because of the identical format, assemblers will be more or less unable to separate between [XXh+r] and [00XXh+r], the assembler will most likely produce [XXh+r] when address is already known to be located in page 0, and [00XXh+r] in case of forward references.
Beside for different opcode size/time, [XXh+r] will always access page 0 memory (even when XXh+r>FFh), while [00XXh+r] may direct to memory in page 0 or 1, to avoid unpredictable results be sure not to use (00)XXh+r>FFh if possible.

CPU Memory and Register Transfers

Register/Immediate to Register Transfer

Opcode	Flags	Clk	Native	Nocash	Expl.
A8	nz----	2	TAY	MOV Y,A	;Y=A
AA	nz----	2	TAX	MOV X,A	;X=A
BA	nz----	2	TSX	MOV X,S	;X=S
98	nz----	2	TYA	MOV A,Y	;A=Y
8A	nz----	2	TXA	MOV A,X	;A=X
9A	------	2	TXS	MOV S,X	;S=X
A9 nn	nz----	2	LDA #nn	MOV A,nn	;A=nn
A2 nn	nz----	2	LDX #nn	MOV X,nn	;X=nn
A0 nn	nz----	2	LDY #nn	MOV Y,nn	;Y=nn

Load Register from Memory

A5 nn	nz----	3	LDA nn	MOV A,[nn]	;A=[nn]
B5 nn	nz----	4	LDA nn,X	MOV A,[nn+X]	;A=[nn+X]
AD nn nn	nz----	4	LDA nnnn	MOV A,[nnnn]	;A=[nnnn]
BD nn nn	nz----	4*	LDA nnnn,X	MOV A,[nnnn+X]	;A=[nnnn+X]
B9 nn nn	nz----	4*	LDA nnnn,Y	MOV A,[nnnn+Y]	;A=[nnnn+Y]
A1 nn	nz----	6	LDA (nn,X)	MOV A,[[nn+X]]	;A=[WORD[nn+X]]
B1 nn	nz----	5*	LDA (nn),Y	MOV A,[[nn]+Y]	;A=[WORD[nn]+Y]
A6 nn	nz----	3	LDX nn	MOV X,[nn]	;X=[nn]
B6 nn	nz----	4	LDX nn,Y	MOV X,[nn+Y]	;X=[nn+Y]
AE nn nn	nz----	4	LDX nnnn	MOV X,[nnnn]	;X=[nnnn]
BE nn nn	nz----	4*	LDX nnnn,Y	MOV X,[nnnn+Y]	;X=[nnnn+Y]
A4 nn	nz----	3	LDY nn	MOV Y,[nn]	;Y=[nn]
B4 nn	nz----	4	LDY nn,X	MOV Y,[nn+X]	;Y=[nn+X]
AC nn nn	nz----	4	LDY nnnn	MOV Y,[nnnn]	;Y=[nnnn]
BC nn nn	nz----	4*	LDY nnnn,X	MOV Y,[nnnn+X]	;Y=[nnnn+X]

* Add one cycle if indexing crosses a page boundary.

Store Register in Memory

```
  85 nn      ------  3   STA nn      MOV [nn],A          ;[nn]=A
  95 nn      ------  4   STA nn,X    MOV [nn+X],A        ;[nn+X]=A
  8D nn nn   ------  4   STA nnnn    MOV [nnnn],A        ;[nnnn]=A
  9D nn nn   ------  5   STA nnnn,X  MOV [nnnn+X],A      ;[nnnn+X]=A
  99 nn nn   ------  5   STA nnnn,Y  MOV [nnnn+Y],A      ;[nnnn+Y]=A
  81 nn      ------  6   STA (nn,X)  MOV [[nn+x]],A      ;[WORD[nn+x]]=A
  91 nn      ------  6   STA (nn),Y  MOV [[nn]+y],A      ;[WORD[nn]+y]=A
  86 nn      ------  3   STX nn      MOV [nn],X          ;[nn]=X
  96 nn      ------  4   STX nn,Y    MOV [nn+Y],X        ;[nn+Y]=X
  8E nn nn   ------  4   STX nnnn    MOV [nnnn],X        ;[nnnn]=X
  84 nn      ------  3   STY nn      MOV [nn],Y          ;[nn]=Y
  94 nn      ------  4   STY nn,X    MOV [nn+X],Y        ;[nn+X]=Y
  8C nn nn   ------  4   STY nnnn    MOV [nnnn],Y        ;[nnnn]=Y
```

Push/Pull

```
  48         ------  3   PHA         PUSH A              ;[S]=A, S=S-1
  08         ------  3   PHP         PUSH P              ;[S]=P, S=S-1 (flags)
  68         nz----  4   PLA         POP  A              ;S=S+1, A=[S]
  28         nzcidv  4   PLP         POP  P              ;S=S+1, P=[S] (flags)
```

Notes: PLA sets Z and N according to content of A. The B-flag and unused flags cannot be changed by PLP, these flags are always written as "1" by PHP.

CPU Arithmetic/Logical Operations

Add memory to accumulator with carry

```
  69 nn      nzc--v  2   ADC #nn     ADC A,nn            ;A=A+C+nn
  65 nn      nzc--v  3   ADC nn      ADC A,[nn]          ;A=A+C+[nn]
  75 nn      nzc--v  4   ADC nn,X    ADC A,[nn+X]        ;A=A+C+[nn+X]
  6D nn nn   nzc--v  4   ADC nnnn    ADC A,[nnnn]        ;A=A+C+[nnnn]
  7D nn nn   nzc--v  4*  ADC nnnn,X  ADC A,[nnnn+X]      ;A=A+C+[nnnn+X]
  79 nn nn   nzc--v  4*  ADC nnnn,Y  ADC A,[nnnn+Y]      ;A=A+C+[nnnn+Y]
  61 nn      nzc--v  6   ADC (nn,X)  ADC A,[[nn+X]]      ;A=A+C+[word[nn+X]]
  71 nn      nzc--v  5*  ADC (nn),Y  ADC A,[[nn]+Y]      ;A=A+C+[word[nn]+Y]
```

* Add one cycle if indexing crosses a page boundary.

Subtract memory from accumulator with borrow

```
  E9 nn      nzc--v  2   SBC #nn     SBC A,nn            ;A=A+C-1-nn
  E5 nn      nzc--v  3   SBC nn      SBC A,[nn]          ;A=A+C-1-[nn]
  F5 nn      nzc--v  4   SBC nn,X    SBC A,[nn+X]        ;A=A+C-1-[nn+X]
  ED nn nn   nzc--v  4   SBC nnnn    SBC A,[nnnn]        ;A=A+C-1-[nnnn]
  FD nn nn   nzc--v  4*  SBC nnnn,X  SBC A,[nnnn+X]      ;A=A+C-1-[nnnn+X]
  F9 nn nn   nzc--v  4*  SBC nnnn,Y  SBC A,[nnnn+Y]      ;A=A+C-1-[nnnn+Y]
  E1 nn      nzc--v  6   SBC (nn,X)  SBC A,[[nn+X]]      ;A=A+C-1-[word[nn+X]]
  F1 nn      nzc--v  5*  SBC (nn),Y  SBC A,[[nn]+Y]      ;A=A+C-1-[word[nn]+Y]
```

* Add one cycle if indexing crosses a page boundary.

Note: Compared with normal 80x86 and Z80 CPUs, incoming and resulting Carry Flag are reversed.

Logical AND memory with accumulator

```
  29 nn      nz----  2   AND #nn     AND A,nn            ;A=A AND nn
  25 nn      nz----  3   AND nn      AND A,[nn]          ;A=A AND [nn]
  35 nn      nz----  4   AND nn,X    AND A,[nn+X]        ;A=A AND [nn+X]
  2D nn nn   nz----  4   AND nnnn    AND A,[nnnn]        ;A=A AND [nnnn]
  3D nn nn   nz----  4*  AND nnnn,X  AND A,[nnnn+X]      ;A=A AND [nnnn+X]
  39 nn nn   nz----  4*  AND nnnn,Y  AND A,[nnnn+Y]      ;A=A AND [nnnn+Y]
  21 nn      nz----  6   AND (nn,X)  AND A,[[nn+X]]      ;A=A AND [word[nn+X]]
  31 nn      nz----  5*  AND (nn),Y  AND A,[[nn]+Y]      ;A=A AND [word[nn]+Y]
```

* Add one cycle if indexing crosses a page boundary.

Exclusive-OR memory with accumulator

```
  49 nn      nz----  2   EOR #nn     XOR A,nn            ;A=A XOR nn
  45 nn      nz----  3   EOR nn      XOR A,[nn]          ;A=A XOR [nn]
  55 nn      nz----  4   EOR nn,X    XOR A,[nn+X]        ;A=A XOR [nn+X]
  4D nn nn   nz----  4   EOR nnnn    XOR A,[nnnn]        ;A=A XOR [nnnn]
  5D nn nn   nz----  4*  EOR nnnn,X  XOR A,[nnnn+X]      ;A=A XOR [nnnn+X]
  59 nn nn   nz----  4*  EOR nnnn,Y  XOR A,[nnnn+Y]      ;A=A XOR [nnnn+Y]
  41 nn      nz----  6   EOR (nn,X)  XOR A,[[nn+X]]      ;A=A XOR [word[nn+X]]
  51 nn      nz----  5*  EOR (nn),Y  XOR A,[[nn]+Y]      ;A=A XOR [word[nn]+Y]
```

* Add one cycle if indexing crosses a page boundary.

Logical OR memory with accumulator

```
  09 nn      nz----  2   ORA #nn     OR  A,nn            ;A=A OR nn
  05 nn      nz----  3   ORA nn      OR  A,[nn]          ;A=A OR [nn]
  15 nn      nz----  4   ORA nn,X    OR  A,[nn+X]        ;A=A OR [nn+X]
  0D nn nn   nz----  4   ORA nnnn    OR  A,[nnnn]        ;A=A OR [nnnn]
  1D nn nn   nz----  4*  ORA nnnn,X  OR  A,[nnnn+X]      ;A=A OR [nnnn+X]
  19 nn nn   nz----  4*  ORA nnnn,Y  OR  A,[nnnn+Y]      ;A=A OR [nnnn+Y]
  01 nn      nz----  6   ORA (nn,X)  OR  A,[[nn+X]]      ;A=A OR [word[nn+X]]
```

```
  11 nn      nz----  5*  ORA (nn),Y  OR  A,[[nn]+Y]      ;A=A OR [word[nn]+Y]
```
* Add one cycle if indexing crosses a page boundary.

Compare
```
  C9 nn      nzc---  2   CMP #nn     CMP A,nn           ;A-nn
  C5 nn      nzc---  3   CMP nn      CMP A,[nn]         ;A-[nn]
  D5 nn      nzc---  4   CMP nn,X    CMP A,[nn+X]       ;A-[nn+X]
  CD nn nn   nzc---  4   CMP nnnn    CMP A,[nnnn]       ;A-[nnnn]
  DD nn nn   nzc---  4*  CMP nnnn,X  CMP A,[nnnn+X]     ;A-[nnnn+X]
  D9 nn nn   nzc---  4*  CMP nnnn,Y  CMP A,[nnnn+Y]     ;A-[nnnn+Y]
  C1 nn      nzc---  6   CMP (nn,X)  CMP A,[[nn+X]]     ;A-[word[nn+X]]
  D1 nn      nzc---  5*  CMP (nn),Y  CMP A,[[nn]+Y]     ;A-[word[nn]+Y]
  E0 nn      nzc---  2   CPX #nn     CMP X,nn           ;X-nn
  E4 nn      nzc---  3   CPX nn      CMP X,[nn]         ;X-[nn]
  EC nn nn   nzc---  4   CPX nnnn    CMP X,[nnnn]       ;X-[nnnn]
  C0 nn      nzc---  2   CPY #nn     CMP Y,nn           ;Y-nn
  C4 nn      nzc---  3   CPY nn      CMP Y,[nn]         ;Y-[nn]
  CC nn nn   nzc---  4   CPY nnnn    CMP Y,[nnnn]       ;Y-[nnnn]
```
* Add one cycle if indexing crosses a page boundary.
Note: Compared with normal 80x86 and Z80 CPUs, resulting Carry Flag is reversed.

Bit Test
```
  24 nn      xz---x  3   BIT nn      TEST A,[nn]        ;test and set flags
  2C nn nn   xz---x  4   BIT nnnn    TEST A,[nnnn]      ;test and set flags
```
Flags are set as so: Z=((A AND [addr])=00h), N=[addr].Bit7, V=[addr].Bit6. Note that N and V are affected only by [addr] (not by A).

Increment by one
```
  E6 nn      nz----  5   INC nn      INC [nn]           ;[nn]=[nn]+1
  F6 nn      nz----  6   INC nn,X    INC [nn+X]         ;[nn+X]=[nn+X]+1
  EE nn nn   nz----  6   INC nnnn    INC [nnnn]         ;[nnnn]=[nnnn]+1
  FE nn nn   nz----  7   INC nnnn,X  INC [nnnn+X]       ;[nnnn+X]=[nnnn+X]+1
  E8         nz----  2   INX         INC X              ;X=X+1
  C8         nz----  2   INY         INC Y              ;Y=Y+1
```

Decrement by one
```
  C6 nn      nz----  5   DEC nn      DEC [nn]           ;[nn]=[nn]-1
  D6 nn      nz----  6   DEC nn,X    DEC [nn+X]         ;[nn+X]=[nn+X]-1
  CE nn nn   nz----  6   DEC nnnn    DEC [nnnn]         ;[nnnn]=[nnnn]-1
  DE nn nn   nz----  7   DEC nnnn,X  DEC [nnnn+X]       ;[nnnn+X]=[nnnn+X]-1
  CA         nz----  2   DEX         DEC X              ;X=X-1
  88         nz----  2   DEY         DEC Y              ;Y=Y-1
```

CPU Rotate and Shift Instructions

Shift Left Logical/Arithmetic
```
  0A         nzc---  2   ASL A       SHL A              ;SHL A
  06 nn      nzc---  5   ASL nn      SHL [nn]           ;SHL [nn]
  16 nn      nzc---  6   ASL nn,X    SHL [nn+X]         ;SHL [nn+X]
  0E nn nn   nzc---  6   ASL nnnn    SHL [nnnn]         ;SHL [nnnn]
  1E nn nn   nzc---  7   ASL nnnn,X  SHL [nnnn+X]       ;SHL [nnnn+X]
```

Shift Right Logical
```
  4A         0zc---  2   LSR A       SHR A              ;SHR A
  46 nn      0zc---  5   LSR nn      SHR [nn]           ;SHR [nn]
  56 nn      0zc---  6   LSR nn,X    SHR [nn+X]         ;SHR [nn+X]
  4E nn nn   0zc---  6   LSR nnnn    SHR [nnnn]         ;SHR [nnnn]
  5E nn nn   0zc---  7   LSR nnnn,X  SHR [nnnn+X]       ;SHR [nnnn+X]
```

Rotate Left through Carry
```
  2A         nzc---  2   ROL A        RCL A             ;RCL A
  26 nn      nzc---  5   ROL nn       RCL [nn]          ;RCL [nn]
  36 nn      nzc---  6   ROL nn,X     RCL [nn+X]        ;RCL [nn+X]
  2E nn nn   nzc---  6   ROL nnnn     RCL [nnnn]        ;RCL [nnnn]
  3E nn nn   nzc---  7   ROL nnnn,X   RCL [nnnn+X]      ;RCL [nnnn+X]
```

Rotate Right through Carry
```
  6A         nzc---  2   ROR A        RCR A             ;RCR A
  66 nn      nzc---  5   ROR nn       RCR [nn]          ;RCR [nn]
  76 nn      nzc---  6   ROR nn,X     RCR [nn+X]        ;RCR [nn+X]
  6E nn nn   nzc---  6   ROR nnnn     RCR [nnnn]        ;RCR [nnnn]
  7E nn nn   nzc---  7   ROR nnnn,X   RCR [nnnn+X]      ;RCR [nnnn+X]
```

Notes:
ROR instruction is available on MCS650X microprocessors after June, 1976.
ROL and ROR rotate an 8bit value through carry (rotates 9bits in total).

CPU Jump and Control Instructions

Normal Jumps & Subroutine Calls/Returns

```
  4C nn nn  ------  3   JMP nnnn     JMP nnnn                ;PC=nnnn
  6C nn nn  ------  5   JMP (nnnn)   JMP [nnnn]              ;PC=WORD[nnnn]
  20 nn nn  ------  6   JSR nnnn     CALL nnnn               ;[S]=PC+2,PC=nnnn
  40        nzcidv  6   RTI          RETI ;(from BRK/IRQ/NMI) ;P=[S], PC=[S]
  60        ------  6   RTS          RET  ;(from CALL)        ;PC=[S]+1
```

Note: RTI cannot modify the B-Flag or the unused flag.
Glitch: For JMP [nnnn] the operand word cannot cross page boundaries, ie. JMP [03FFh] would fetch the MSB from [0300h] instead of [0400h]. Very simple workaround would be to place a ALIGN 2 before the data word.

Conditional Branches (conditional jump to PC=PC+/-dd)

```
  10 dd     ------  2** BPL nnn      JNS nnn     ;N=0 plus/positive
  30 dd     ------  2** BMI nnn      JS  nnn     ;N=1 minus/negative/signed
  50 dd     ------  2** BVC nnn      JNO nnn     ;V=0 no overflow
  70 dd     ------  2** BVS nnn      JO  nnn     ;V=1 overflow
  90 dd     ------  2** BCC/BLT nnn  JNC/JB  nnn ;C=0 less/below/no carry
  B0 dd     ------  2** BCS/BGE nnn  JC/JAE  nnn ;C=1 above/greater/equal/carry
  D0 dd     ------  2** BNE/BZC nnn  JNZ/JNE nnn ;Z=0 not zero/not equal
  F0 dd     ------  2** BEQ/BZS nnn  JZ/JE   nnn ;Z=1 zero/equal
```

** The execution time is 2 cycles if the condition is false (no branch executed). Otherwise, 3 cycles if the destination is in the same memory page, or 4 cycles if it crosses a page boundary (see below for exact info).
Note: After subtractions (SBC or CMP) carry=set indicates above-or-equal, unlike as for 80x86 and Z80 CPUs.

Interrupts, Exceptions, Breakpoints

```
  00        ---1--  7   BRK   Force Break B=1,[S]=PC+1,[S]=P,I=1,PC=[FFFE]
  --        ---1--  7   /IRQ  Interrupt   B=0,[S]=PC,  [S]=P,I=1,PC=[FFFE]
  --        ---1--  7   /NMI  NMI         B=0,[S]=PC,  [S]=P,I=1,PC=[FFFA]
  --        ---1-- T+6? /RESET Reset      B=1,S=S-3,         I=1,PC=[FFFC]
```

Notes: IRQs can be disabled by setting the I-flag. BRK command, /NMI signal, and /RESET signal cannot be masked by setting I.
BRK/IRQ/NMI first change the B-flag, then write P to stack, and then set the I-flag, the D-flag is NOT changed and should be cleared by software.
The same vector is shared for BRK and IRQ, software can separate between BRK and IRQ by examining the pushed B-flag only.
The RTI opcode can be used to return from BRK/IRQ/NMI, note that using the return address from BRK skips one dummy/parameter byte following after the BRK opcode.
Software or hardware must take care to acknowledge or reset /IRQ or /NMI signals after processing it.

```
  IRQs are executed whenever "/IRQ=LOW AND I=0".
  NMIs are executed whenever "/NMI changes from HIGH to LOW".
```

If /IRQ is kept LOW then same (old) interrupt is executed again as soon as setting I=0. If /NMI is kept LOW then no further NMIs can be executed.

CPU Control

```
  18        --0---  2   CLC       CLC    ;Clear carry flag            C=0
  58        ---0--  2   CLI       EI     ;Clear interrupt disable bit I=0
  D8        ----0-  2   CLD       CLD    ;Clear decimal mode          D=0
  B8        -----0  2   CLV       CLV    ;Clear overflow flag         V=0
  38        --1---  2   SEC       STC    ;Set carry flag              C=1
  78        ---1--  2   SEI       DI     ;Set interrupt disable bit   I=1
  F8        ----1-  2   SED       STD    ;Set decimal mode            D=1
```

No Operation

```
  EA        ------  2   NOP       NOP    ;No operation
```

Conditional Branch Page Crossing

The branch opcode with parameter takes up two bytes, causing the PC to get incremented twice (PC=PC+2), without any extra boundary cycle. The signed parameter is then added to the PC (PC+disp), the extra clock cycle occurs if the addition crosses a page boundary (next or previous 100h-page).

CPU Illegal Opcodes

SAX and LAX

```
  87 nn     ------  3   SAX nn      STA+STX  [nn]=A AND X
  97 nn     ------  4   SAX nn,Y    STA+STX  [nn+Y]=A AND X
  8F nn nn  ------  4   SAX nnnn    STA+STX  [nnnn]=A AND X
  83 nn     ------  6   SAX (nn,X)  STA+STX  [WORD[nn+X]]=A AND X
  A7 nn     nz----  3   LAX nn      LDA+LDX  A,X=[nn]
  B7 nn     nz----  4   LAX nn,Y    LDA+LDX  A,X=[nn+Y]
  AF nn nn  nz----  4   LAX nnnn    LDA+LDX  A,X=[nnnn]
  A3 nn     nz----  6   LAX (nn,X)  LDA+LDX  A,X=[WORD[nn+X]]
  B3 nn     nz----  5*  LAX (nn),Y  LDA+LDX  A,X=[WORD[nn]+Y]
```

For SAX, both A and X are output to databus, LOW-bits are stronger than HIGH-bits, resulting in a "forceful" AND operation.
For LAX, the same value is written to both A and X.

Combined ALU-Opcodes

Opcode high-bits, flags, commands:

```
  00+yy        nzc---  SLO op   ASL+ORA   op=op SHL 1 // A=A OR op
  20+yy        nzc---  RLA op   ROL+AND   op=op RCL 1 // A=A AND op
  40+yy        nzc---  SRE op   LSR+EOR   op=op SHR 1 // A=A XOR op
  60+yy        nzc--v  RRA op   ROR+ADC   op=op RCR 1 // A=A+op+cy
  C0+yy        nzc---  DCP op   DEC+CMP   op=op-1     // A-op
  E0+yy        nzc--v  ISC op   INC+SBC   op=op+1     // A=A-op-(1-cy)
```

Opcode low-bits, clock cycles, operands:

```
  07+xx nn        5    nn       [nn]
  17+xx nn        6    nn,X     [nn+X]
  03+xx nn        8    (nn,X)   [WORD[nn+X]]
  13+xx nn        8    (nn),Y   [WORD[nn]+Y]
  0F+xx nn nn     6    nnnn     [nnnn]
  1F+xx nn nn     7    nnnn,X   [nnnn+X]
  1B+xx nn nn     7    nnnn,Y   [nnnn+Y]
```

Other Illegal Opcodes

```
  0B nn     nzc---  2  ANC #nn          AND+ASL  A=A AND nn, C=N ;bit7 to carry
  2B nn     nzc---  2  ANC #nn          AND+ROL  A=A AND nn, C=N ;same as above
  4B nn     nzc---  2  ALR #nn          AND+LSR  A=(A AND nn) SHR 1
  6B nn     nzc--v  2  ARR #nn          AND+ROR  A=(A AND nn), V=Overflow(A+A),
                                                 A=A/2+C*80h, C=A.Bit6
  CB nn     nzc---  2  AXS #nn          CMP+DEX  X=(X AND A)-nn
  EB nn     nzc--v  2  SBC #nn          SBC+NOP  A=A-nn         cy?
  BB nn nn  nz----  4* LAS nnnn,Y       LDA+TSX  A,X,S = [nnnn+Y] AND S
```

NUL/NOP and KIL/JAM/HLT

```
  xx        ------  2   NOP         (xx=1A,3A,5A,7A,DA,FA)
  xx nn     ------  2   NOP #nn     (xx=80,82,89,C2,E2)
  xx nn     ------  3   NOP nn      (xx=04,44,64)
  xx nn     ------  4   NOP nn,X    (xx=14,34,54,74,D4,F4)
  xx nn nn  ------  4   NOP nnnn    (xx=0C)
  xx nn nn  ------  4*  NOP nnnn,X  (xx=1C,3C,5C,7C,DC,FC)
  xx        ------  -   KIL         (xx=02,12,22,32,42,52,62,72,92,B2,D2,F2)
```

NOP doesn't change any registers or flags, the operand (if any) is fetched, which may be useful for delays, patches, or for read-sensitive I/O ports. KIL halts the CPU, the data bus will be set to #$FF, KIL can be suspended by /RESET signal (not sure if also by /IRQ or /NMI ???).

Unstable Illegal Opcodes

```
  8B nn     nz----  2  XAA #nn    ((2)) TXA+AND  A=X AND nn
  AB nn     nz----  2  LAX #nn    ((2)) LDA+TAX  A,X=nn
  BF nn nn  nz----  4* LAX nnnn,X       LDA+LDX  A,X=[nnnn+X]
  93 nn     ------  6  AHX (nn),Y ((1))          [WORD[nn]+Y] = A AND X AND H
  9F nn nn  ------  5  AHX nnnn,Y ((1))          [nnnn+Y] = A AND X AND H
  9C nn nn  ------  5  SHY nnnn,X ((1))          [nnnn+X] = Y AND H
  9E nn nn  ------  5  SHX nnnn,Y ((1))          [nnnn+Y] = X AND H
  9B nn nn  ------  5  TAS nnnn,Y ((1)) STA+TXS  S=A AND X  // [nnnn+Y]=S AND H
```

note to XAA: DO NOT USE!!! Highly unstable!!!
note to LAX: DO NOT USE!!! On my C128, this opcode is stable, but on my C64-II
it loses bits so that the operation looks like this: ORA #? AND #{imm} TAX.
note to AXS: performs CMP and DEX at the same time, so that the MINUS sets
the flag like CMP, not SBC.
Combinations of STA/STX/STY:

```
 AHX {adr} = stores A&X&H into {adr}
 SHX {adr} = stores X&H into {adr}
 SHY {adr} = stores Y&H into {adr}
```

note: sometimes the &H drops off. Also page boundary crossing will not work as
expected (the bank where the value is stored may be equal to the value stored).
["H" probably meant to be the MSB aka Highbyte of the 16bit memory address?]

CPU Assembler Directives/Syntax

Below are some common 65XX assembler directives, and the corresponding expressions in 80XX-style language.

```
  65XX-style    80XX-style         Expl.
  .native       .nocash            select native or nocash syntax
  *=$c100       org 0c100h         sets the assumed origin in memory
  *=*+8         org $+8            increments origin, does NOT produce data
  label         label:             sets a label equal to the current address
  label=$dc00   label equ 0dc00h   assigns a value or address to label
  .by $00       db 00h             defines a (list of) byte(s) in memory
  .byt $00      defb 00h           same as .by and db
  .wd $0000     dw 0000h           defines a (list of) word(s) in memory
  .end          end                indicates end of source code file
  |nn           [|nn]              force 16bit "00NN" instead 8bit "NN"
  #<nnnn        nnnn AND 0FFh      isolate lower 8bits of 16bit value
  #>nnnn        nnnn DIV 100h      isolate upper 8bits of 16bit value
```

```
  N/A (?)       fast label         ensure relative jump without page crossing
  N/A (?)       slow label         ensure relative jump with page crossing
```

Special Directives

```
  .65xx       Select 6502 Instruction Set
  .nes        Create NES ROM-Image with .NES extension
  .c64_prg    Create C64 file with .PRG extension/stub/fixed entry
  .c64_p00    Create C64 file with .P00 extension/stub/fixed entry/header
  .vic20_prg  Create VIC20/C64 file with .PRG extension/stub/relocated entry
  end entry   End of Source, the parameter specifies the entrypoint
```

The C64 files contain Basic Stub "10 SYS<entry>" with default ORG 80Eh.

VIC20 Stub

The VIC20 Stub is "10 SYSPEEK(44)*256+<entry>" with default ORG 1218h, this relocates the entryoint relative to the LOAD address (for C64: 818h, for VIC20: 1018h (Unexpanded), 0418h (3K Expanded), 1218h (8K and more Expansion). It does NOT relocate absolute addresses in the program, if the program wishes to run at a specific memory location, then it must de-relocate itself from the LOAD address to the desired address.

CPU Glitches

Dummy Read Cycles at Page-Wraps

Dummy reads occur when reads from [nnnn+X] or [nnnn+Y] or [WORD[nn]+Y] are crossing page boundaries, this applies only to raw read-opcodes, not for write or read-and-modify opcodes (ie. only for opcodes that include an extra clock cycle on page boundaries, such as LDA, CMP, etc.)
For above reads, the CPU adds the index register to the lower 8bits of the 16bit memory address, and does then read from the resulting memory address, if the addition caused a carry-out, then an extra clock cycle is used to increment the upper 8bits of the address, and to read from the correct memory location.
For example, a read from [1280h+X] with X=C0h produces a dummy read from [1240h], followed by the actual read from [1340h].
Dummy reads cause no problems with normal ROM or RAM, but may cause problems with read-sensitive I/O ports (eg. IRQ flags that are automatically cleared after reading, or data-registers that are automatically incrementing associated memory pointers, etc.)

Dummy Write Cycles in Read-Modify-Opcodes

Dummy writes occur in all read-modify opcodes, ie. all INC, DEC, Shift, Rotate opcodes with memory operands. The opcodes consist of three memory accesses: read original value, write dummy value, write result value.
Dummy writes cause no problems with normal RAM, but may cause problems (or may be useful) with write-sensitive I/O ports (eg. IRQ flags that are cleared by writing certain values, or data-registers that are automatically incrementing associated memory pointers, etc.)
On the C64 and C16, the written dummy value appears to be equal to the original value, a couple of programs are using this to acknowledge IRQs.
On the NES, the dummy value appears to be equal to the result value, though more or less unstable ANDed with a random number. Presumably 00h is output during the first half of the write cycle, and the result only during the second half, not leaving enough time to raise all bits from LOW to high. Also, dummy writes to [2007h] aren't always recognized (ie. the VRAM address register isn't always incremented twice), presumably because the PPU isn't fast enough to realize two write-signals immediately after each other, that maybe because it is attempting to synchronize CPU bus writes with the PPU bus.

CPU The 65XX Family

Different versions of the 6502:

All of these processors are the same concerning the software-side:

```
 6501   Some sort of 6502 prototype
 6502   Used in the CBM floppies and some other 8 bit computers.
 6507   Used in Atari 2600, 28pins (only 13 address lines, no /IRQ, no /NMI).
 6510   Used in C64, with built-in 6bit I/O port.
 7501   Used in C16,C116,Plus/4, with built-in 7bit I/O Port, without /NMI pin.
 8500   Used in C64-II, with different pin-outs.
 8501   Same as 7501
 8502   Used in C128s.
```

Some processors of the family which are not 100% compatible:

```
 65C02  Extension of the 6502
 65SC02 Small version of the 65C02 which lost a few opcodes again.
 65CE02 Extension of the 65C02, used in the C65.
 65816  Extended 6502 with new opcodes and 16 bit operation modes.
 2A03   Nintendo NES/Famicom, modified 6502 with built-in sound controller.
```

CPU Local Usage

CPU

The NES uses a customized NMOS 6502 CPU, engineered and produced by Ricoh. It's primary customization adds audio. Audio registers are mapped internal to the CPU; all waveform generation is done internal to the CPU as well.
The NES's 6502 does not contain support for decimal mode. Both the CLD and SED opcodes function normally, but the 'd' bit of P is unused in both ADC

and SBC. It is common practice for games to CLD prior to code execution, as the status of 'd' is unknown on power-on and on reset.
NMIs may be generated by PPU each VBlank.
IRQs may be generated by APU and by external hardware.
The CPU does include undocumented opcodes, just like normal 6502 CPUs.
The NTSC NES runs at 1.7897725MHz, and 1.662607MHz for PAL. Which is pretty fast for a 6502 compatible CPU, for example C64 used only 1MHz, and Atari 2600 only 1.2MHz.

Hardware Pin-Outs

Pin-Outs
Cartridge Pin-Outs
Cartridge Shell Dimensions
Controllers - Pin-Outs
Chipset Pin-Outs
NES Expansion Port

Upgrading
Nocash SRAM Circuit

Chipset Pin-Outs

2A03 Pin-Outs & Signal Description (CPU and APU)

Pin	Name	Dir	Expl.
1	ROUT	Out	Sound channel 1+2 output
2	COUT	Out	Sound channel 3+4+5 output
3	/RES	In	Resets several internal 2A03 registers, and the 6502.
4-19	A0-15	Out	Address Bus
20	GND	-	Supply Ground
21-28	D7-0	I/O	Data Bus
29	CLK	In	Master clock input (236,250/11 MHz), clocks an internal divide-by-12 counter.
30	?	In	Normally grounded in NES/FC consoles, this pin has unknown functionality. Might be an input controlling something, since the pin does draw a little current. Or might be simply some kind of shielding for the CLK signal?
31	PHI2	Out	Divide-by-12 result of the CLK signal (1.79 MHz). The internal 6502 along with function generating hardware, is clocked off this frequency, and is available externally here so that it can be used as a data bus enable signal (when at logic level 1) for external 6502 address decoder logic. The signal has a 62.5% duty cycle.
32	/IRQ	In	Interrupt Request (Low)
33	/NMI	In	Non-Maskable Interrupt (on High-to-Low Transition)
34	R/W	Out	Direction of 6502's data bus (0=Write/Out, 1=Read/In)
35	/JOY2	Out	Low if A0-A15=4017h, R/W=0, PHI2=1
36	/JOY1	Out	Low if A0-A15=4016h, R/W=0, PHI2=1
37-39	J2-0	Out	Bit2-0 of internal register 4016h (Bit0 = Joystick strobe)
40	VCC	-	Supply +5VDC

On Playchoice 10, Pin 30 is called "/SPECIAL" (seems to be related to pausing the CPU). On VS System, Pin 30 is called "NC" and is wired to GND.

2C02 Pin-Outs & Signal Descriptions (PPU)

Pin	Dir	Name	Expl
1	In	CPU R/W	Direction when /CS=LOW
2-9	I/O	CPU D0-D7	Data when /CS=LOW
10-12	In	CPU A2-A0	Register Select when /CS=LOW
13	In	CPU /CS	CPU read/write to/from PPU Registers
14-17	I/O	EXT0-EXT3	External Master/Slave Video signal (not used)
18	In	CLK	21.47727MHz NTSC, 26.601712MHz PAL
19	Out	/VBL NMI	VBlank, LOW max 20 scanlines or until acknowledged
20	In	VEE GND	Supply Ground
21	Out	VOUT	Composite Video output
22	In	/SYNC EXT	External Master /VBL for use by slave (not used) (*)
23,24	Out	PPU /W,/R	Video memory Write/Read requests
25-30	Out	PPU A13-A8	Video memory MSB-address lines
31-38	I/O	PPU AD7-AD0	Video memory LSB-address and data lines
39	Out	PPU ALE	Address Latch Enable, HIGH when A0-A7 output at AD0-AD7
40	In	VCC +5VDC	Supply

(*) On Famicom consoles, /SYNC EXT is always tied to logical one. On the NES however, this pin is tied in with the 2A03's reset input, and as a result, the picture is always disabled while the reset switch is held in on an NES.
On RGB PPUS (both Playchoice 10 and VS System ones), Pin 14-17,21,22 are R,G,B,GND,/SYNC,/RST.

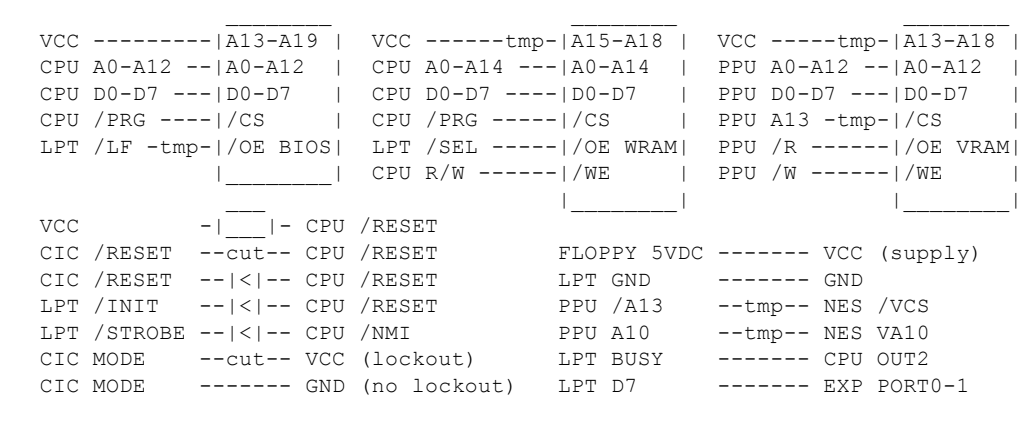
NES Expansion Port

NES Expansion Port, 48-pins, (at bottom of console, rarely used)

Pin	Dir	Expl.
1,48,2,47	Out	VCC,VCC,GND,GND (Supply +5VDC and Ground)
23	Out	VDD voltage from external power supply (usually +10VDC)
3	In	AIN (Audio Input)
21,22,23,24	Out	VOUT, AOUT (Video and Audio Outputs)
4,14,25-32	I/O	CPU /NMI,/IRQ,D7,D6,D5,D4,D3,D2,D1,D0
5,24	Out	CPU A15, CIC 4MHz
6-10,38-42	I/O	Cart Pin 51-55,20-16
43,44,45	Out	OUT0, OUT1, OUT2 (Port 4016h Bit0-2 Outputs)
34 and 37	Out	PORT0-CLK (both pins) (CPU Read from Port 4016h)
11 and 17	Out	PORT1-CLK (both pins) (CPU Read from Port 4017h)
35,12,33,13,36	In	PORT0-0,1,2,3,4 (Port 4016h Bit0-3 Inverted Inputs)
19,20,15,16,18	In	PORT1-0,1,2,3,4 (Port 4017h Bit0-3 Inverted Inputs)
46	-	Unused

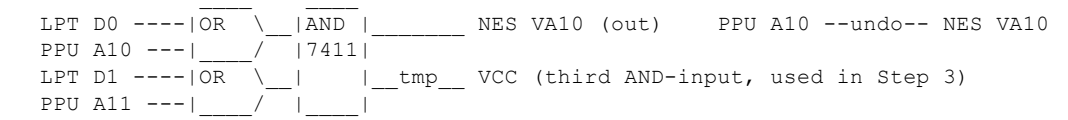
Nocash SRAM Circuit

Step 1 - Basic Connection, NROM support (32K+8K), Horizontal Mirroring

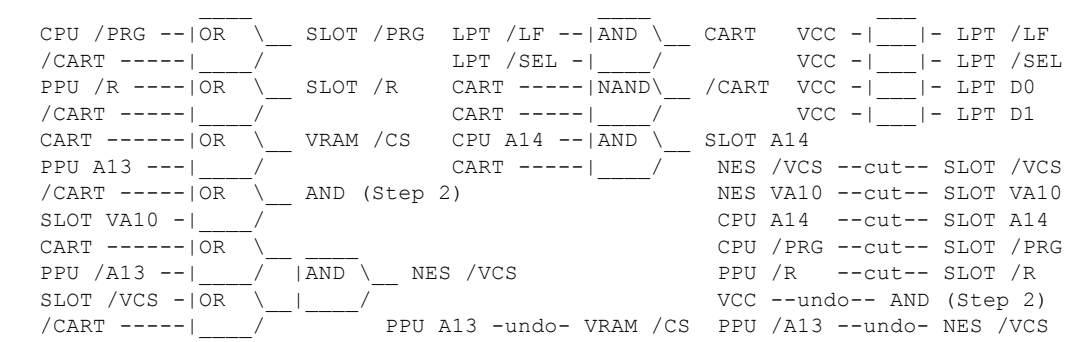


At this stage, the console won't work if an external cartridge is inserted.

Step 2 - Horizontal or Vertical Mirroring Control

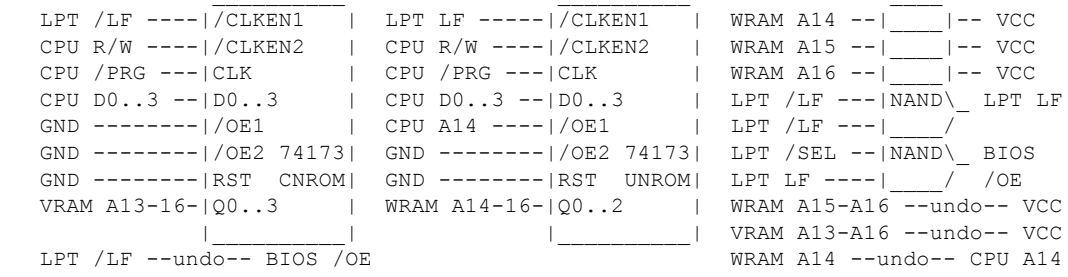


Step 3 - Internal Circuit Disable (Required for Internal Circuit only)

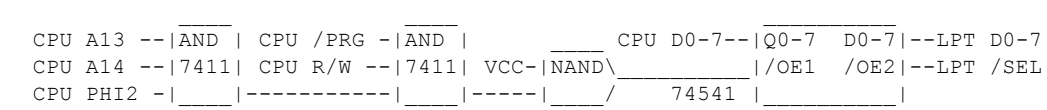


Also allows to disable the internal circuit so that external cartridges can be used when LPT signals are all HIGH).

Step 4 - UNROM (N*16K+8K) and CNROM (32K+N*8K) Bank Switching



Step 5 - Optional 8bit high-speed upload connection



The 1bit PORT0-1 connection is no longer used (may be disconnected if desired).

Compatibility Notes

WRAM and VRAM are not write-protected, and may get overwritten by accidental writes to ROM/VROM area, that applies also for writes to bank selection ports (no problem for cartridges that handle bus-conflicts, it will simply replace the value in RAM by the (same) written value). In NROM mode, bank selection ports are not protected against accidental ROM-area writes.

Soldering Notes

To reduce the amount of wires, the WRAM/VRAM chips can be stacked on top of the internal 2K SRAMs with 1:1 connection for most pins, also the BIOS EPROM socket can be stacked on the WRAM chip.
Optionally, the circuit could be connected externally to the cartridge slot (with /RESET and /NMI connected to unused cartridge/expansion port pins), the /CART and CART signals would be not required, Step 3 could be left out.

Parts List

```
2  SRAM     WRAM/VRAM, min 32K/8K, recommended 128K/32K, max 128K/128K
1  EPROM    BIOS, 27C64 or similar, min 8K
2  74LS32   quad 2-input OR gates
1  74LS08   quad 2-input AND gates
1  74LS00   quad 2-input NAND gates
1  74LS11   triple 3-input AND gates
2  74LS173  4-bit 3-state flip-flops
1  74LS541  8-bit 3-state buffer/line driver
8  10K      pull-up resitors
3  1N4148   diodes for /RESET and /NMI
```

Plus, eprom socket, optionally also sockets for all other chips, 100nF capacitors for power supply of all chips, centronics printer cable, centronics socket, wire, board, solder, eprom burner, etc.

BIOS ROM-Image

```
0000 85 04 48 8A 48 A9 19 8D FA FF A9 04 8D FB FF A2
0010 08 E0 00 D0 FC 68 AA 68 60 A9 00 06 04 69 03 8D
0020 16 40 CA 40 8A 48 A9 3B 8D FA FF A9 04 8D FB FF
0030 A2 08 E0 00 D0 FC 68 AA A5 04 60 24 0D 30 09 AD
0040 16 40 4A 4A 26 04 CA 40 AD 00 60 85 04 A2 00 40
0050 20 24 04 A2 7E A0 04 24 0D 30 04 A2 8D A0 04 8E
0060 FA FF 8C FB FF 8D FF FF A9 00 8D 01 20 8D 06 20
0070 8D 06 20 A2 00 A0 20 A9 01 85 04 4C 7B 04 AD 00
0080 60 8D 07 20 CA D0 FE 88 D0 FE 4C 1C 06 AD 16 40
0090 4A 4A 26 04 90 FE A5 04 8D 07 20 A9 01 85 04 CA
00A0 D0 FE 88 D0 FE 4C 1C 06 20 24 04 A2 DC A0 04 24
00B0 0D 30 04 A2 EE A0 04 8E FA FF 8C FB FF 8D FF FF
00C0 A0 BF A2 FF C9 FF D0 04 A0 FF A2 F9 8C FC 04 8C
00D0 E2 04 E8 A0 40 A9 01 85 04 4C D9 04 AD 00 60 CA
00E0 9D 00 FF D0 FE CE E2 04 88 D0 FE 4C 1C 06 AD 16
00F0 40 4A 4A 26 04 90 FE A5 04 CA 9D 00 FF A9 01 85
0100 04 E0 00 D0 FE CE FC 04 88 D0 FE 4C 1C 06 A2 00
0110 20 24 04 95 05 E8 E0 08 D0 F6 A2 00 B5 05 9D FA
0120 FF E8 E0 06 D0 F6 A1 05 81 05 4C 2A 05 A2 55 A0
0130 AA 8E FE FF 8C FF FF EC FE FF D0 F5 CC FF FF D0
0140 F0 8C FE FF 8E FF FF CC FE FF D0 E5 EC FF FF D0
0150 E0 60 A2 00 BD 61 05 20 00 04 E8 BD 61 05 D0 F4
0160 60 4E 4F 24 4E 45 53 20 42 49 4F 53 20 56 31 2E
0170 30 00 A9 00 85 0D 20 87 05 A9 80 85 0D 20 87 05
0180 F0 04 A9 00 85 0D 60 A2 00 A0 2B 20 24 04 DD A1
0190 05 F0 02 A0 2D E8 E0 08 D0 F1 98 20 00 04 C9 2B
01A0 60 00 FF 55 AA 0F F0 3C C3 A9 57 20 00 04 A2 FF
01B0 8D FF FF E8 8E 00 80 8E FF BF EC FF FF F0 06 E0
01C0 1F D0 F0 A2 01 E8 8A 20 00 04 60 A9 56 20 00 04
01D0 A2 40 A0 56 A9 00 8D 01 20 CA 8E FF FF 8D 06 20
01E0 8D 06 20 8C 07 20 8E 07 20 D0 EE A2 00 8E FF FF
01F0 8D 06 20 8D 06 20 CD 07 20 CC 07 20 D0 0A EC 07
0200 20 D0 05 E8 E0 40 D0 E5 8A 20 00 04 60 20 2D 05
0210 20 52 05 20 72 05 20 A9 05 20 CB 05 A9 52 20 00
0220 04 20 24 04 C9 57 D0 03 4C A8 04 C9 56 D0 03 4C
0230 50 04 C9 46 D0 03 4C 0E 05 4C 39 06 A2 00 BD 00
0240 E0 9D 00 04 BD 00 E1 9D 00 05 BD 00 E2 9D 00 06
0250 BD 00 E3 9D 00 07 E8 D0 E5 60 78 D8 A9 00 8D 00
0260 20 AD 02 20 A2 FF 9A 20 3C E2 4C 0D 06 FF FF FF
0270 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
.... FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
1FF0 FF FF FF FF FF FF FF FF FF FF 00 00 5A E2 00 00
```

To be copied to the highest memory location, ie. E000h-FFFFh for a 64K EPROM.

About Everynes

Everynes
Everything about NES and Famicom.
Nocash Technical Specifications written 2004 by Martin Korth.
Everynes Text and Html and Debugger versions and updates available at:
`http://problemkaputt.de/nes.htm`
I've originally written Everynes when collecting and sorting-out relevant info for making the no$nes emulator/debugger. I've included a copy of the resulting document in the debuggers help text, and also released raw txt/htm versions, which may be eventually of some use to NES/Famicom programmers.

Help welcome
Please let me know if you come across anything that is incomplete, incorrect, or unclear. A couple of details (marked by question marks) are definetly unclear to me - additional info would be very welcome!
My email address hides in no$nes.exe about box (for anti-spam reasons).

Thanks & Credits
Most of the Everynes document is based on information from many other documents found at nesdev.parodius.com - hoping that nobody gets angry about picking info from his/her docs - I'd like to send many thanks to the authors of that great documents, and to all people whom have contributed information to those docs, complete list as far as known to me - many thangs to:

(Feedback)
Thanks to Joe Lennox, and #nesdev on EFnet for Everynes corrections.

MAPPERS.NFO
Comprehensive NES Mapper Document v0.80 by \Firebug\
Thanks to FanWen, Y0SHi, D, Jim Geffre, Goroh, Paul Robson, Mark Knibbs.

2A03TECH.TXT
2A03 technical reference by Brad Taylor
Thanks to Matthew Conte, Kentaro Ishihara, Goroh, Memblers, FluBBa, Izumi,
Chibi-Tech, Quietust, SnowBro, Bananmos, Kevin Horton, and many others for
their time and help on and off the NESdev mailing list, and the Membled
Messageboards.

2C02TECH.TXT
NTSC 2C02 technical reference by Brad Taylor
Thanks to the NES community. http://nesdev.parodius.com.
Special thanks to Neal Tew for scrolling information.

FDSTECH.TXT
Famicom Disk System Technical Reference by Brad Taylor. Special thanks to Nori, Goroh, Ki, Sgt. Bowhack and "D".

NESTECH2.TXT
Nintendo Entertainment System Documentation, Version: 2.00
Alex Krasivsky
Andrew Davie
Avatar Z
Barubary
Bluefoot
CiXeL
Chi-Wen Yang
Chris Hickman
D
Dan Boris
David de Regt
Donald Moore
Fredrik Olsson
Icer Addis
Jon Merkel
Kevin Horton
Loopy
Marat Fayzullin
Mark Knibbs
Martin Nielsen
Matt Conte
Matthew Richey
Memblers

MiKael Iushin
Mike Perry
Morgan Johansson
Neill Corlett
Pat Mccomack
Patrik Alexandersson
Paul Robson
Ryan Auge
Stumble
Tennessee Carmel-Veilleux
Thomas Steen
Tony Young
Vince Indriolo
\FireBug\

FFPA.TXT
Famicom Four-Player Adapters Technical Document by Richard Hoelscher

NES4PLAY.TXT
NES 4player-adapter documentation by Fredrik Olsson
Special thanks to:
Juan Antonio Gomez Galvarez
Yoshi
Marat Fayzulin
Morgan Johansson

Pin-Outs
drk421
Siudym'2001

6205BUGS.TXT
Ivo van Poorten

NLOCKOUT.TXT
by Mark (Knipps?)

VSDOC.TXT
VS Unisystem information version 1.0, by Fx3

PC10DOC.TXT
Nintendo Playchoice 10 Hardware Description by Oliver Achten

NESGG.TXT
NES Game Genie Code Format DOC v0.71 by Benzene of Digital Emutations
Special thanks to Sardu, Opcode, Deuce, DrSplat, KingPin

NESFQ.HTM
NES Tech FAQ by Chris Covell

NES.HTM
Nintendo Entertainment System Architecture by Marat Fayzullin

```
Pascal Felber     Patrick Lesaard      Tink
Goroh             Pan of Anthrox       Bas Vijfwinkel
Kawasedo          Paul Robson
Marcel de Kogel   Serge Skorobogatov
Alex Krasivsky    John Stiles
```

KEYBOARD.TXT
Reverse Engineering the Keyboard of Family Computer
by goroh, english translation by Ki

LIGHTGUN.TXT
Family Computer Gun
by goroh, english translation by Ki

POWERPAD.TXT
Power Pad information Version: 1.2 (03/12/00) by Tennessee Carmel-Veilleux
Thanks to Jeremy D. Chadwick, Kevin Horton

CPU
Project64, Graham

The weird and wonderful CIC
Segher's rev-engineered instruction set for the CIC CPU.

Index

[extracted from no$nes v1.2 documentation]