Atari 2600 Specifications

Programming Specifications

Writing this emulator simultaneously while US hi-tec machinery was attacking Bagdad, it must be said that the Atari 2600 was shipped with a war game called Combat, the Atari 2600's hardware sprites are notably called "Players", "Missiles", and "Ball". Being manufactured in the cold war era, the implied purpose of this gaming console is: Playing joyful war games. War is not a game, war is no fun. War is murder.

Technical Data

Atari 2600 Video Computer System (VCS)
Manufacured 1977-1991

Chipset
CPU 6507 - 8bit, 1.19MHz
TIA 1A - Television Interface Adaptor Model 1A (Video, Audio, Input Ports)
PIA 6532 - (128 bytes RAM, I/O Ports, Timer)

CPU and Memory
```
  CPU:   6507, 8bit, 1.19MHz (cutdown 6502 with only 8K address space)
  RAM:   128 Bytes (additional 128 or 256 bytes in some cartridges)
  VRAM:  None (Picture controlled by I/O Ports only)
  ROM:   External Game Cartridge (usually 2KB or 4KB, or banked 2x4KB)
```
Video
```
  Output:      Line-by-line (Registers must be updated each scanline)
  Resolution:  160x192 pixels (NTSC 60Hz), 160x228 pixels (PAL 50Hz)
  Playfield:   40 dots horizontal resolution (rows of 4 pixels per dot)
  Colors:      4 colors at once (one color per object)
  Palette:     128 colors (NTSC), 104 colors (PAL), 8 colors (SECAM)
  Sprites:     2 sprites of 8pix width, 3 sprites of 1pix width
```
Input/Output Ports, Audio, Timer
```
  I/O:    Two 8bit I/O ports, six 1bit Input ports
  Timer:  One 8bit Timer (with prescaler; 1,8,64,1024 machine cycles)
  Audio:  Two sound channels (with Frequency, Volume, Noise control)
```
Console Switches
```
  Switches: Color/Mono Switch (PAL/NTSC only), and two Difficulty Switches
  Buttons:  Select Button, Reset Button (or Switches in older consoles)
  Hardware: Power Switch, TV Channel Select Switch (not software controlled)
```
Connectors
```
  Slot:     One 24 pin Cartridge Slot (with 4K address bus)
  Controls: Two 9 pin Joystick ports (also used for Paddles, Keyboards)
  TV:       One Cinch Socket (Video/Audio TV Signal)
  Power:    9V DC, 500mA (internally converted to 5V DC)
```

Memory and I/O Map

Overview

0000-002C	TIA Write
0000-000D	TIA Read (sometimes mirrored at 0030-003D)
0080-00FF	PIA RAM (128 bytes)
0280-0297	PIA Ports and Timer
F000-FFFF	Cartridge Memory (4 Kbytes area)

Below lists address, name, used bits (ordered 76543210, "1"=used), and function of all I/O ports and memory. Ports marked as <strobe> are executing the the specified function when wrinting any data to it (the written data itself is ignored).

TIA - WRITE ADDRESS SUMMARY (Write only)

00	VSYNC	......1.	vertical sync set-clear
01	VBLANK	11....1.	vertical blank set-clear
02	WSYNC	<strobe>	wait for leading edge of horizontal blank
03	RSYNC	<strobe>	reset horizontal sync counter
04	NUSIZ0	..111111	number-size player-missile 0
05	NUSIZ1	..111111	number-size player-missile 1
06	COLUP0	1111111.	color-lum player 0 and missile 0
07	COLUP1	1111111.	color-lum player 1 and missile 1
08	COLUPF	1111111.	color-lum playfield and ball
09	COLUBK	1111111.	color-lum background
0A	CTRLPF	..11.111	control playfield ball size & collisions
0B	REFP0	....1...	reflect player 0
0C	REFP1	....1...	reflect player 1
0D	PF0	1111....	playfield register byte 0
0E	PF1	11111111	playfield register byte 1
0F	PF2	11111111	playfield register byte 2
10	RESP0	<strobe>	reset player 0
11	RESP1	<strobe>	reset player 1
12	RESM0	<strobe>	reset missile 0
13	RESM1	<strobe>	reset missile 1
14	RESBL	<strobe>	reset ball
15	AUDC0	1111	audio control 0
16	AUDC1	1111	audio control 1
17	AUDF0	...11111	audio frequency 0
18	AUDF1	...11111	audio frequency 1
19	AUDV0	1111	audio volume 0
1A	AUDV1	1111	audio volume 1
1B	GRP0	11111111	graphics player 0
1C	GRP1	11111111	graphics player 1
1D	ENAM0	......1.	graphics (enable) missile 0
1E	ENAM1	......1.	graphics (enable) missile 1
1F	ENABL	......1.	graphics (enable) ball
20	HMP0	1111....	horizontal motion player 0
21	HMP1	1111....	horizontal motion player 1
22	HMM0	1111....	horizontal motion missile 0
23	HMM1	1111....	horizontal motion missile 1
24	HMBL	1111....	horizontal motion ball
25	VDELP0	1	vertical delay player 0
26	VDELP1	1	vertical delay player 1
27	VDELBL	1	vertical delay ball
28	RESMP0	......1.	reset missile 0 to player 0
29	RESMP1	......1.	reset missile 1 to player 1
2A	HMOVE	<strobe>	apply horizontal motion
2B	HMCLR	<strobe>	clear horizontal motion registers
2C	CXCLR	<strobe>	clear collision latches

TIA - READ ADDRESS SUMMARY (Read only)

30	CXM0P	11......	read collision M0-P1, M0-P0 (Bit 7,6)
31	CXM1P	11......	read collision M1-P0, M1-P1
32	CXP0FB	11......	read collision P0-PF, P0-BL
33	CXP1FB	11......	read collision P1-PF, P1-BL
34	CXM0FB	11......	read collision M0-PF, M0-BL
35	CXM1FB	11......	read collision M1-PF, M1-BL
36	CXBLPF	1.......	read collision BL-PF, unused
37	CXPPMM	11......	read collision P0-P1, M0-M1
38	INPT0	1.......	read pot port
39	INPT1	1.......	read pot port
3A	INPT2	1.......	read pot port
3B	INPT3	1.......	read pot port
3C	INPT4	1.......	read input
3D	INPT5	1.......	read input

PIA 6532 - RAM, Switches, and Timer (Read/Write)

80..FF	RAM	11111111	128 bytes RAM (in PIA chip) for variables and stack
0280	SWCHA	11111111	Port A; input or output (read or write)
0281	SWACNT	11111111	Port A DDR, 0= input, 1=output
0282	SWCHB	11111111	Port B; console switches (read only)
0283	SWBCNT	11111111	Port B DDR (hardwired as input)
0284	INTIM	11111111	Timer output (read only)
0285	INSTAT	11......	Timer Status (read only, undocumented)
0294	TIM1T	11111111	set 1 clock interval (838 nsec/interval)
0295	TIM8T	11111111	set 8 clock interval (6.7 usec/interval)
0296	TIM64T	11111111	set 64 clock interval (53.6 usec/interval)
0297	T1024T	11111111	set 1024 clock interval (858.2 usec/interval)

Cartridge Memory (4 Kbytes area)

F000-FFFF	ROM	11111111	Cartridge ROM (4 Kbytes max)
F000-F07F	RAMW	11111111	Cartridge RAM Write (optional 128 bytes)
F000-F0FF	RAMW	11111111	Cartridge RAM Write (optional 256 bytes)
F080-F0FF	RAMR	11111111	Cartridge RAM Read (optional 128 bytes)
F100-F1FF	RAMR	11111111	Cartridge RAM Read (optional 256 bytes)
003F	BANK	11	Cart Bank Switching (for some 8K ROMs, 4x2K)
FFF4-FFFB	BANK	<strobe>	Cart Bank Switching (for ROMs greater 4K)
FFFC-FFFD	ENTRY	11111111	Cart Entrypoint (16bit pointer)
FFFE-FFFF	BREAK	11111111	Cart Breakpoint (16bit pointer)

See also:
Memory Mirrors
Cartridges

Memory Mirrors

CPU 8K Address Space Mirrors (Step 2000h)
The 6507 CPU is having only 13 address pins (8KBytes, 0000h-1FFFh). All memory is mirrored in steps of 2000h (ie. at 2000h-3FFFh, 4000h-5FFFh, and so on, up to E000h-FFFFh). The lower half of each area contains internal memory (RAM, I/O Ports), the upper half contains external memory (cartridge ROM or expansion RAM).

TIA Mirrors (Step 10h/40h,100h)
The TIA chip is addressed by A12=0, A7=0. Located in memory at 0000h-007Fh, 0100h-017Fh, 0200h-027Fh, etc., 0F00h-0F7Fh. Output ports are mirrored twice in each window (at XX00h and XX40h), input port are mirrored eight times in each window (at XX00h, XX10h, XX20h, XX30h, etc. XX70h).

PIA RAM Mirrors (Step 100h,400h)
PIA RAM is selected by A12=0, A9=0, A7=1. Located in memory at 0080h-00FFh, 0180h-01FFh, 0480h-04FFh, 0580h-05FFh, etc. The mirror at 0180h is particulary important because it allows the CPU to access RAM for stack operations.

PIA I/O Mirrors (Step 2h,8h,100h,400h)
PIA I/O is selected by A12=0, A9=1, A7=1. Located in memory at 0280h-02FFh, 0380h-03FFh, 0680h-06FFh, 0780h-07FFh, etc. Each 80h-area contains 16 mirrors of the PIA I/O ports at XX80h, XX88h, XX90h, XX98h, etc. And, each 8h-area contains two copies of INTIM (eg. 0284h,0286h) and INSTAT (eg. 0285h,0287h).

Cartridge Mirrors
Cartridge memory is selected by A12=1. Small 2K cartridges are usually mirrored twice into the 4K cartridge area. ROM-Bank ports (eg. 1FF8h) aren't necessarily connected to A12, and may overlap the upper PIA I/O mirror (eg. 0FF8h).

Note (Commonly used Mirrors)
In some cases, two different ports are located at the same memory address, one port being Read-only, and the other being Write-only, of course, there is no physical conflict with such 'overlapping' ports.
However, human users may prefer to use different memory addresses for different ports. The following mirrors are used by many games (and in this document): Timer outputs at 294h-297h (instead of 284h-287h), TIA inputs at 30h-3Dh (instead of 00h-0Dh).

Video

The display controller operates on a line by line basis, always outputting the same information every television line unless new data is written into it by software.

"The 5 moveable objects can be positioned anywhere, and consists of 2 players, 2 missiles, and a ball.
The playfield, players, missiles, and ball are created and manipulated by a series of registers in the TIA that the microprocessor can address and write into.
Each type of object has certain defined capabilities."

Video Synchronization and Blanking

00h - VSYNC - Vertical sync set-clear
```
  Bit  Expl.
  0    Not used
  1    Vertical Sync            (0=Stop sync, 1=Start sync)
  2-7  Not used
```
Sync should be output for a duration of 3 scanlines, between the lower and upper screen border periods.

01h - VBLANK - Vertical blank set-clear
```
  Bit  Expl.
  0    Not used
  1    Vertical Blank           (0=Stop blanking, 1=Start blanking)
  2-5  Not used
  6    INPT4-INPT5 Control      (0=Normal Input, 1=Latched Input)
  7    INPT0-INPT3 Control      (0=Normal Input, 1=Dumped to ground)
```

Note: Disable latches (Bit6=0) also resets latches to logic true.
The blanking bit should be set for the duration of upper and lower screen border periods. During blanking, playfield and moveable objects are hidden and the screen area becomes black.

02h - WSYNC <strobe> - Wait for leading edge of horizontal blank
Writing any value to this address halts microporcessor (by RDY signal) until end of current scanline, ie. until begin of next H-Blank period.
Note: Works also in 'hidden' scanlines, ie. during VBlank/VSync.

03h - RSYNC <strobe> - Reset horizontal sync counter
Writing any value to this address resets the horizontal sync counter to begin of horizontal blank time (intended for chip testing purposes).

Video Dimensions and Timings

Vertical Timing and Resolution
Vertical sync and blanking must be controlled by software. VBLANK should be enabled during vsync and upper/lower border periods.

```
  Type_______NTSC__PAL/SECAM______
  V-Sync      3     3    scanlines
  V-Blank     37    45   scanlines (upper border)
  Picture     192   228  scanlines
  Overscan    30    36   scanlines (lower border)
  Frame Rate  60    50   Hz
  Frame Time  262   312  scanlines
```

Because there is not enough CPU time to change all registers per scanline, some objects must use a lower vertical resolution than 192 or 228 pixels.
Notes: V-Sync should be exactly 3 scanlines (if it is 2 or 4 scanlines, my TV tends to ignore colors and to produce a black & white picture). Many PAL games use only 192 picture scanlines (and according 45+18 upper, and 36+18 lower border lines).

Horizontal Timing and Resolution
Horizontal sync and blanking are done automatically by hardware. The software may re-synchronize itself to begin of H-Blank by writing to WSYNC.

```
  Blanking  22.6 machine clocks (68 color clocks)
  Picture   53.3 machine clocks (160 color clocks) (160 pixels)
  Total     76.0 machine clocks (228 color clocks)
```

The playfield (which takes up the main part of the screen) is restricted to a horizontal resolution of 40 dots. Moveable objects can be positioned and drawn at full 160 pixels resolution.

Clocks

```
  Video Color Clock: 3.579545 MHz (NTSC), 3.546894 MHz (PAL)
  CPU Machine Clock: 1.193182 MHz (NTSC), 1.182298 MHz (PAL)
```

The CPU Clock is derived from Video clock divided by three.
One color clock = 1 pixel. One machine clock = 3 pixels.

Note: For the vertical blanking/overscan periods, it'd be recommended to program the PIA Timer to blanking time (allowing to execute larger parts of code without timing synchronization), followed by one WSYNC when the timer has expired.

Note: Most PAL games increase upper/lower border heights rather than actually using the full 228 pixels picture height.

Video Playfield

Officially, a playfield consists of walls, clouds, barriers, and other seldom moved objects. In practice, it consists of a 20 bit value, used to display 20 low resolution dots (rows of 4 pixels per dot). The whole screen width is 40 dots, the same 20 bit value is used both for the left and right half (however, the right half may be horizontally mirrored, and, software may change the PF registers somewhere in the middle of the scanline).

0Dh - PF0 - Playfield Register 0 (LSB first) (only upper 4bit used)
0Eh - PF1 - Playfield Register 1 (MSB first) (8bit)
0Fh - PF2 - Playfield Register 2 (LSB first) (8bit)
Defines the colors of the separate playfield dots (0=COLUBK, 1=COLUPF). Depending on the REF bit (CTRLPF.0), the screen output is composed of the following bits/ordering:
```
  Normal (REF=0) : PF0.4-7 PF1.7-0 PF2.0-7  PF0.4-7 PF1.7-0 PF2.0-7
  Mirror (REF=1) : PF0.4-7 PF1.7-0 PF2.0-7  PF2.7-0 PF1.0-7 PF0.7-4
```
PF0, PF1, and PF2, should not be written to while the respective register is currently output to the screen.

0Ah - CTRLPF - Control Playfield and Ball size
```
  Bit  Expl.
  0    Playfield Reflection     (0=Normal, 1=Mirror right half)
  1    Playfield Color          (0=Normal, 1=Score Mode, only if Bit2=0)
  2    Playfield/Ball Priority  (0=Normal, 1=Above Players/Missiles)
  3    Not used
  4-5  Ball size                (0..3 = 1,2,4,8 pixels width)
  6-7  Not used
```
In "score" mode, the playfield is drawn by COLUP0/COLUP1 for left/right half of screen (instead COLUPF in both halves). Intended to display 2-player mode scores at different colors by low-resolution playfield graphics (used by older games like Combat).

Video MOB Shape/Enable

(Ball Size: See CTRLPF)

04h - NUSIZ0 - Number-size player-missile 0
05h - NUSIZ1 - Number-size player-missile 1
These addresses control the number and size of players and missiles.
```
  Bit  Expl.
  0-2  Player-Missile number & Player size (See table below)
  3    Not used ???
  4-5  Missile Size  (0..3 = 1,2,4,8 pixels width)
  6-7  Not used
```
Player-Missile number & player size (Picture shows shape, 8 pix per char)
```
  0  One copy              (X.........)
  1  Two copies - close    (X.X.......)
  2  Two copies - medium   (X...X.....)
  3  Three copies - close  (X.X.X.....)
  4  Two copies - wide     (X.......X.)
  5  Double sized player   (XX........)
  6  Three copies - medium (X...X...X.)
  7  Quad sized player     (XXXX......)
```
1/2 television line (80 clocks), 8 clocks per character

1Bh - GRP0 - Graphics (enable) player 0
1Ch - GRP1 - Graphics (enable) player 1

```
  Bit  Expl.
  0-7  Eight dots player shape   (0=Transparent, 1=Player color)
```

The dots are usually drawn MSB first (from left to right), but can be horizontally mirroed by REFP0 and REFP1 registers.
Use a value of 00h to hide/disable the object.

1Dh - ENAM0 - Graphics (enable) missile 0
1Eh - ENAM1 - Graphics (enable) missile 1
1Fh - ENABL - Graphics (enable) ball

```
  Bit  Expl.
  0    Not used
  1    Missile/Ball Enable      (0=Transparent, 1=Missile/Ball color)
  2-7  Not used
```

The "shape" for these objects consists of a single dot, which may be horizontally repeated or stretched by NUSIZ/CTRLPF registers.
Use a value of 00h to hide/disable the object. Missiles can be also hidden by RESMPx registers (see Positioning chapter).

0Bh - REFP0 - Reflect player 0
0Ch - REFP1 - Reflect player 1

```
  Bit  Expl.
  0-2  Not used
  3    Reflect Player Graphics   (0=Normal/MSB first, 1=Mirror/LSB first)
  4-7  Not used
```

Used to mirror the GRP0 and GRP1 registers.

25h - VDELP0 - Vertical delay player 0 (Delay GRP0 until writing to GRP1)
26h - VDELP1 - Vertical delay player 1 (Delay GRP1 until writing to GRP0)
27h - VDELBL - Vertical delay ball (Delay ENABL until writing to GRP1)
When VDELPx is set, writes to GRPx are delayed until GRPy is written to - which may be done in the same scanline, in one of the next scanlines, or in the next frame, or never - and may thus actually cause a horizontal, vertical, framerate, or even endless delay.

```
  Bit  Expl.
  0    Vertical Delay  (0=No delay, 1=Delay until writing to GRP0/GRP1)
  1-7  Not used
```

The misleading name "vertical" delay is used because these registers were originally intended to be used to update GRPx in one scanline and to have the changes delayed until GRPy is updated in the next scanline.

Notes on High Resolution Text and Vertical Delay
Many newer games (eg. Pacman) use high resolution text or graphics output, of 6 characters or 48 pixels width. This is done by configuring NUSIZ to "three copies close", and positioning GRP0 ("0_0_0") eight pixels to the left of GRP1 ("1_1_1"), resulting in the pattern "010101".
Updating GRP0 and GRP1 multiple times within each scanline can be done to change the pattern to 6 different characters ("012345"), the drawing procedure requires proper timing, and typically uses "vertical" delay (in this case behaving as horizontal delay), note that one usually needs 7 writes to GRP registers to draw 6 characters (with one final dummy write to apply the last (delayed) character).
The GRPx registers actually consist of two registers each, hereby called GRPxA (delayed data latch) and GRPxB (actual display data). When VDELx is disabled, writes to GRPx go directly to GRPxB. When VDELx is enabled, writes to GRPx go to GRPxA, and GRPxA is copied to GRPxB at the time when writing to GRPy.

Video MOB Horizontal Positioning

10h - RESP0 <strobe> - Reset player 0
11h - RESP1 <strobe> - Reset player 1

12h - RESM0 <strobe> - Reset missile 0
13h - RESM1 <strobe> - Reset missile 1
14h - RESBL <strobe> - Reset ball
Writing any value to these addresses sets the associated objects horizontal position equal to the current position of the cathode ray beam, if the write takes place anywhere within horizontal blanking then the position is set to the left edge of the screen (plus a few pixels towards right: 3 pixels for P0/P1, and only 2 pixels for M0/M1/BL).
Note: Because of opcode execution times, it is usually necessary to adjust the resulting position to the desired value by subsequently using the Horizontal Motion function.

28h - RESMP0 - Reset missile 0 to player 0
29h - RESMP1 - Reset missile 1 to player 1
```
  Bit  Expl.
  0    Not used
  1    Reset Missile to Player  (0=Normal, 1=Hide and Lock on player)
  2-7  Not used
```
As long as Bit 1 is set, the missile is hidden and its horizontal position is centered on the players position. The centering offset is +3 for normal, +6 for double, and +10 quad sized player (that is giving good centering results with missile widths of 2, 4, and 8 respectively).

Note on Vertical Positioning
None such supported by the video controller. Moveable objects (and playfield) must be enabled or disabled (or modified in shape) in the appropriate scanlines by software.

Video MOB Horizontal Motion

20h - HMP0 - Horizontal motion player 0
21h - HMP1 - Horizontal motion player 1
22h - HMM0 - Horizontal motion missile 0
23h - HMM1 - Horizontal motion missile 1
24h - HMBL - Horizontal motion ball
Specifies the motion direction and step (in pixels). Writing to these registers does not directly affect the objects position, unless when subsequently using the HMOVE command.
```
  Bit  Expl.
  0-3  Not used
  4-7  Signed Motion Value (-8..-1=Right, 0=No motion, +1..+7=Left)
```
Repeatedly writing to HMOVE will repeat the same movement (without having to rewrite the motion registers).

2Ah - HMOVE <strobe> - Apply horizontal motion
Writing any value to this address applies horizontal motion: The motion registers (HMP0, HMP1, etc) are added to the position of the moveable objects (P0, P1, etc):
```
  NewPos = ( OldPos +/- Motion ) MOD 160
```
Timing Restrictions: The HMOVE command should be used only immediately after a WSYNC command, this ensures that the operation starts at the beginning of, and finishes before end of, horizontal blanking (see notes below). Also, the software should not change any of the motion registers for at least 24 machine cycles after an HMOVE command.

2Bh - HMCLR <strobe> - Clear horizontal motion registers
Writing any value to this address resets all five motion registers (HMP0, HMP1, etc) to zero.

HMOVE Extra Blanking
When HMOVE is executed inside of the horizontal blanking period (which should be usually be so), then the leftmost 8 pixels of the following scanline are covered by the border color (black).
That "feature" is probably intended to hide minor dirt effects in moveable objects while processing motion. The resulting black lines could be treated as a major dirt effect though (eg. Espial).

Some games (eg. Hero) issue HMOVEs in all scanlines (so that all scanlines have masked left pixels, which looks better than only some lines being masked). Also, the blanking may be of some use with objects that are moved "half offscreen" (which would usually re-appear at the other side of the screen).
Note: Collision checks still take place "below" of the extra blanking area.

HMOVE Extra Motion Offsets
When HMOVE is executed outside of the horizontal blanking period (which should usually not be done), additional motion offets are added to the position registers (additonally to the normal HMxx motion values):
Player Offsets:
```
  6,4,4,2,0           ;-more before blanking, disp RIGHT  \
  -2,-2,-4,-6,-8,-8   ;-area before blanking, disp LEFT   /
  14 dup (0)          ;-blanking area, disp ZERO         |
  2,2,4,6,8,8,10,12   ;-area after blanking, disp RIGHT   \
  8,8,8,8,8...        ;-remaining area, disp FAST RIGHT    ""--__
```
Missile/Ball Offsets
```
  -1,-2,-2,-3,-4,-5,-5,-6,-7,-8,-8  ;disp LEFT           /
  14 dup (0)          ;-blanking area, disp ZERO         |
  1,1,2,3,4,4,5,6     ;-area after blanking, disp RIGHT   \
  0,0,0,0,0...        ;-remaining area, disp ZERO         |
```
In such cases, motion may occur even if HMxx registers are all zero.

Actually...
The above extra-offsets seem to apply ONLY when HMxx=0.
For other HMxx values the extra-offsets seem to change,
such like a LEFT offset becomes RIGHT offset or vice versa,
ie. there seem to be some kind of overflows occuring...?
And/or, the above extra-offsets apply only if the sprites
are at a specific position?
Anyways, HMOVE should be issued only immediately after WSYNC
(or N*76 cycles after WSYNC), otherwise the results are kinda
unpredictable.

Video Collision

30h - CXM0P (R) - Collision Latch M0-P1, M0-P0 (Bit 7,6) (Read only)
31h - CXM1P (R) - Collision Latch M1-P0, M1-P1 (Bit 7,6) (Read only)
32h - CXP0FB (R) - Collision Latch P0-PF, P0-BL (Bit 7,6) (Read only)
33h - CXP1FB (R) - Collision Latch P1-PF, P1-BL (Bit 7,6) (Read only)
34h - CXM0FB (R) - Collision Latch M0-PF, M0-BL (Bit 7,6) (Read only)
35h - CXM1FB (R) - Collision Latch M1-PF, M1-BL (Bit 7,6) (Read only)
36h - CXBLPF (R) - Collision Latch BL-PF (Bit 7) (Read only)
37h - CXPPMM (R) - Collision Latch P0-P1, M0-M1 (Bit 7,6) (Read only)
The TIA detects collisions between any of the 6 objects (the playfield and 5 moveable objects). There are 15 possible two-object collisions which are stored in 15 one bit latches.
```
  Bit  Expl.
  0-5  Not used
  6,7  Two Collsion Flags   (0=No collision, 1=Collision occured)
```
The collision registers could be read at any time but is usually done during vertical blank after all possible collisions have occurred.
Note: Use CXCLR to reset the collision latches after reading.

2Ch - CXCLR (W) <strobe> - Clear all collision latches
Writing any value to this address resets all collision latches (CXM0P-CXPPMM) to zero.

Video Colors

See also CTRLPF.1 (SCORE), and SWCHB.3 (B/W) - and CTRLPF priority

06h - COLUP0 - Color Luminosity Player 0, Missile 0
07h - COLUP1 - Color Luminosity Player 1, Missile 1
08h - COLUPF - Color Luminosity Playfield, Ball
09h - COLUBK - Color Luminosity Background
Selects color and luminance (brightness) for the specified object. The console can display 128 colors (NTSC), or 104 colors (PAL), or 8 colors (SECAM) on color TV sets, and 8 grayshades (NTSC/PAL/SECAM) on monochrome TV sets.

```
  Bit  Expl.
  0    Not used
  1-3  NTSC/PAL Luminance (0-7)  - SECAM: RGB (0-7) - Mono TV: Grayshade (0-7)
  4-7  NTSC/PAL Color     (0-15) - SECAM: Ignored   - Mono TV: Ignored
```

A smaller luminance value drags the NTSC/PAL colors towards black, a larger luminance value drags it towards white (pastelized). Software should process the Color/Mono switch to select between different color schemes for Color/Mono TV sets (if necessary). PAL Software may choose luminance values that look okay both on Mono TV sets and on SECAM color TV sets.

Color & Luminance Values

NTSC Colors	PAL Colors	NTSC/PAL Lum.	SECAM RGB
0 White	0 White	0 Black	0 Black
1 Gold	1 Same as color 0	1 Dark grey	1 Blue
2 Orange	2 Yellow	2 ...	2 Red
3 Bright orange	3 Green-yellow	3 Grey	3 Magenta
4 Pink	4 Orange	4 ...	4 Green
5 Purple	5 Green	5 ...	5 Cyan
6 Purple-blue	6 Pink-orange	6 Light grey	6 Yellow
7 Blue	7 Green-blue	7 White	7 White
8 Blue	8 Pink		
9 Light blue	9 Turquois		
A Torq.	A Pink-blue		
B Green-blue	B Light blue		
C Green	C Blue-red		
D Yellow-green	D Dark blue		
E Orange-green	E Same as color 0		
F Light orange	F Same as color 0		

SECAM is a little weird. It takes the PAL software, but the console color/black & white switch is hardwired as black & white. Therefore, it reads the PAL black & white tables in software and assigns a fixed color to each lum of black & white according to the table.

Screen Border Color
The screen border (HBLANK and VBLANK periods) is black.

Video Priority

See also CTRLPF.1 (SCORE), and CTRLPF.2 (PRIORITY)

Object Colors and Priorities
When pixels of two or more objects overlap each other, only the pixel of the object with topmost priority is drawn to the screen. The normal priority ordering is:

```
  Priority     Color    Objects
  1 (highest)  COLUP0   P0, M0  (and left side of PF in SCORE-mode)
  2            COLUP1   P1, M1  (and right side of PF in SCORE-mode)
  3            COLUPF   BL, PF  (only BL in SCORE-mode)
  4 (lowest)   COLUBK   BK
```

Optionally, the playfield and ball may be assigned to have higher priority (by setting CTRLPF.2). The priority ordering is then:

```
  Priority     Color    Objects
  1 (highest)  COLUPF   PF, BL  (always, the SCORE-bit is ignored)
  2            COLUP0   P0, M0
  3            COLUP1   P1, M1
  4 (lowest)   COLUBK   BK
```

Objects of the same color are having the same priority (it makes no difference which pixel is drawn topmost if both pixels are having the same color).

Interval Timer

284h - INTIM - PIA Current Timer Value (Read only)
Reading from this 8bit register returns the current timer value.
The initial value and interval can be set by TIM1T-T1024T registers.

285h - INSTAT - PIA Timer Status (Read only) (Undocumented)

```
  Bit  Expl.
  0-5  Not used           (Seems to be always zero)
  6    Underflow Flag 1   (1=Underflow, since last INSTAT read)
  7    Underflow Flag 2   (1=Underflow, since last TIMnnT write)
```

Both bit 6 and 7 are set on underflow. Bit 6 is reset when reading from INSTAT. Bit 7 is reset when writing to TIM1T..T1024T.

294h - TIM1T - PIA Set Timer & Select 1 clock interval (838ns/interval)
295h - TIM8T - PIA Set Timer & Select 8 clock interval (6.7us/interval)
296h - TIM64T - PIA Set Timer & Select 64 clock interval (53.6us/interval)
297h - T1024T - PIA Set Timer & Select 1024 clock interval (858.2us/interval)
Writing a 8bit value in range of 00h..FFh to any of these four addresses TIM1T-T1024T sets the timer to that value, and selects the prescaler interval.

Normal Timer Decrement
The timer is decremented once immediately after writing (ie. value 00h does immediately underflow). It is then decremented once every N clock cycle interval (depending on the port address used).

Timer Underflow / Highspeed Decrement
Once when the timer does underflow, it restarts at FFh, and is then decremented once per clock cycle, regardless of the selected interval (this feature allows to calculate the exact underflow time at one clock cycle resolution even if a larger interval was used).

However, the interval is automatically re-actived when reading from the INTIM register, ie. the timer does then continue to decrement at interval speed (originated at the current value).

Note
The timer is particulary useful to specify and determine the end of the vertical blanking periods (upper and lower screen border), allowing to execute program code during that periods without permanent synchronization with the cathode ray beam.

Audio

The TIA chip is having two sound channels which are output, as mono signal, to the TV set. According to the specs, it can make sounds like a "flute", a "rocket motor", and an "explosion".

19h - AUDV0 - Audio Volume, Channel 0
1Ah - AUDV1 - Audio Volume, Channel 1
```
  Bit 0-3, Volume 0-15 (0=Off, 15=Loudest)
```
Note: The recommended method to disable a sound channel is to set its volume to zero.
Does that REALLY FULLY disable sound, or would it be required to be combined with audc=0 to be fully muted ???

17h - AUDF0 - Audio Frequency Divider, Channel 0
18h - AUDF1 - Audio Frequency Divider, Channel 1
```
  Bit 0-4, Frequency Divider (0..31 = 30KHz/1..30KHz/32)
```
The so-called "30KHz" is clocked twice per scanline, ie. at CPUCLK/38 aka DOTCLK/114, so the exact rate is:
```
  PAL:  31113.1 Hz (3.546894MHz/114)
  NTSC: 31399.5 Hz (3.579545MHz/114)
```
After applying the Frequency Divider, the resulting frequency is subsequently divided by AUDC0/AUDC1, which provides additional division by 2, 6, 31, or 93 for pure tone; so the tone frequency range is circa 10Hz..15kHz (ie. 30KHz/32/93..30KHz/1/2).

15h - AUDC0 - Audio Noise/Division Control, Channel 0
16h - AUDC1 - Audio Noise/Division Control, Channel 1
```
    These registers control nine bit shift counters, and may select
    different shift counter feedback taps and count lengths to produce
    a variety of noise and tone qualities.
```
Bit 0-3, Noise/Division Control (0-15, see below)
```
  0  set to 1                     8  9 bit poly (white noise)
  1  4 bit poly                   9  5 bit poly
  2  div 15 -> 4 bit poly         A  div 31 : pure tone
  3  5 bit poly -> 4 bit poly     B  set last 4 bits to 1
  4  div 2 : pure tone            C  div 6 : pure tone
  5  div 2 : pure tone            D  div 6 : pure tone
  6  div 31 : pure tone           E  div 93 : pure tone
  7  5 bit poly -> div 2          F  5 bit poly div 6
```
A setting of zero will disable the sound (PAL/NTSC), or it will produce an "obnoxious background sound" (SECAM).

AUDC Pattern Shapes
```
  x Rep Pattern Shape
  0   1 ----------------------------------------------------------------------
  1  15 ----___-__--_-_----___-__--_-_----___-__--_-_----___-__--_-_----___-__
  2 465 --------------------------------------------------------------________
  3 465 ------______-___---__-----___-------___----___--__--_____---------___-
  4   2 -_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_
  5   2 -_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_
  6  31 ------------------_____________------------------_____________--------
  7  31 -----___--_---_-_-____-__-_--__-----___--_---_-_-____-__-_--__-----___
  8 511 ---------_____----_-----___-_---__--__-_____-__-_-__---_--_-___----__-
  9  31 -----___--_---_-_-____-__-_--__-----___--_---_-_-____-__-_--__-----___
  A  31 ------------------_____________------------------_____________--------
  B   1 ----------------------------------------------------------------------
  C   6 ---___---___---___---___---___---___---___---___---___---___---___---_
  D   6 ---___---___---___---___---___---___---___---___---___---___---___---_
  E  93 -------------------------------------------------_____________________
  F  93 ----------_____---_______----__________------___------____---------___
```

AUDC Pattern Type Notes

Types 1,8,9 produce 4bit,9bit,5bit poly sound (see Formulas below).
Type 7 produces exactly the same sound as Type 9 (the additional "div2" is eventually meant to use div2 transitions as clock, which is identical as normal 1 cycle clock).
Type 2 fetches next 4bit poly (type 1) on each div31 (type 6) transition, the duration of each Poly4 bit is thus multiplied by 18 or by 13.
Type 3 uses the most complicated mechanism (for reverse engineering). The Poly5 operation takes place each cycle. If Poly5-Carry is 1, then the 4bit poly operation is executed and the Poly4-Carry is output to the sound channel, otherwise Poly4 (and sound output) are kept unchanged.
Types 4,5 both produce div2 tone, consisting of 1 HIGH and 1 LOW cycle.
Types 6,A both produce div31 tone, consisting of 18 HIGH and 13 LOW cycles (that odd numbers are eventually based on 5bit poly being applied only when lower 4bits all same, maybe).
Types 0,B both produce a permanent HIGH signal and the sound is effectively disabled (at least with PAL/NTSC, however SECAM reportedly produces "obnoxious background sound" in this case).
Types C,D,E,F are using same mechanism as types 4,5,6,7 but only each 3rd transition is applied.

Internal Poly Formulas

Poly/Noise is generated by shift/xor operations. The rightmost bit of the shift register is output to the sound channel (0=low, 1=high). The register is shifted right and the new leftmost bit is calculated by an XOR operation:

```
  poly  old bits   output  new bits   calculation
  4bit  abcd       d       zabc       z = c xor d
  5bit  abcde      e       zabcd      z = c xor e
  9bit  abcdefghi  i       zabcdefgh  z = e xor i
```

Internal Poly Bitstreams

Below are example bitstreams, the rightmost bit is output to the sound channel, the two x-marked bits are XORed by each other and used as new leftmost bit in next cycle. The dots "..." indicate frozen poly4 settings.

```
  4bit    5bit     9bit poly    5bit->4bit
 ---xx-  ---x-x-  -----x---x-  ---x-x---xx-
  1111    11111    111111111    10011 1111
  0111    01111    011111111    11001 0111
  0011    00111    001111111    11100 ...1
  0001    00011    000111111    11110 ...1
  1000    10001    000011111    11111 0011
  0100    11000    000001111    01111 0001
  0010    01100    100000111    00111 1000
  1001    10110    110000011    00011 0100
  1100    11011    111000001    10001 0010
  0110    11101    111100000    11000 ...0
  1011    01110    011110000    01100 ...0
  0101    10111    101111000    10110 ...0
  1010    01011    110111100    11011 1001
  1101    10101    111011110    11101 1100
  1110    01010    111101111    01110 ...0
  1111    00101    111110111    10111 0110
  0111    00010    011111011    01011 1011
  0011    00001    001111101    10101 0101
  0001    10000    000111110    01010 ...1
  1000    01000    100011111    00101 1010
  0100    00100    010001111    00010 ...0
  0010    10010    101000111    00001 1101
  1001    01001    110100011    10000 ...1
  1100    10100    111010001    01000 ...1
  0110    11010    011101000    00100 ...1
  1011    01101    001110100    10010 ...1
  0101    00110    100111010    01001 1110
  1010    10011    110011101    10100 ...0
  1101    11001    011001110    11010 ...0
  1110    11100    001100111    01101 1111
  1111    11110    100110011    00110 ...1
```

Controllers

Specific Controller Types

Basic Tech Info

Controllers: Joysticks

Joysticks are the most common controllers. Two joysticks can be connected.

Normal 4-Direction / 1-Button Joysticks
Configure Port A as input (SWACNT=00h). Joystick data can be then read from SWCHA and INPT4-5 as follows:

```
  Pin  PlayerP0  PlayerP1  Expl.
  1    SWCHA.4   SWCHA.0   Up     (0=Moved, 1=Not moved)
  2    SWCHA.5   SWCHA.1   Down   ("")
  3    SWCHA.6   SWCHA.2   Left   ("")
  4    SWCHA.7   SWCHA.3   Right  ("")
  6    INPT4.7   INPT5.7   Button (0=Pressed, 1=Not pressed)
```

Note: Connect all direction switches and push buttons to GND (Pin 8).

3-Button Joystick Adapter
The device fits on the shaft of standard Atari joysticks, it includes two additional buttons, and is connected by separate cable between the normal joystick plug and consoles joystick port.
The device comes up without any external pullup/pulldown resistors, it acts somewhat similar than leftmost/rightmost paddle positions and must deal with the capacitors/resistors inside of the console (see Paddles chapter for details).
In general, software must drag INPT0-3 to ground for a moment (eg. during vertical blanking period):

```
  Pin  Bit       Expl.
  5,9  VBLANK.7  INPT0-INPT3 Control   (0=Normal Input, 1=Dumped to ground)
```

Software must then wait a moment (eg. until end of picture drawing period) before it can read the adapters button states:

```
  Pin  PlayerP0  PlayerP1  Expl.
  5    INPT0.7   INPT2.7   2nd Button  (1=Pressed/Charged)
  9    INPT1.7   INPT3.7   3rd Button  (1=Pressed/Charged)
```

Note: Connect the extra buttons to VCC (Pin 7).
The 3-button adapter is used by Omega Race and Thrust (both games aren't actually using all 3 buttons for 3 unique purposes though).

Controllers: Paddles

A paddle consists of single knob (similar like the volume control of HiFi amplifiers), and a single push button. Two paddles can be connected to each port (four paddles can be used when using both ports). Paddles are usually shipped as pair of two controllers which are attached (by Y-cable) to a single plug.

To read paddle buttons, configure SWCHA as input (SWACNT=00h), then read from SWCHA:
```
  Pin  Bit      Expl.
  3    SWCHA.2  Paddle #3  button (right paddle, P1) (0=Pressed)
  4    SWCHA.3  Paddle #2  button (left paddle, P1)
  3    SWCHA.6  Paddle #1  button (right paddle, P0)
  4    SWCHA.7  Paddle #0  button (left paddle, P0)
```
To read paddle pots, switch INPT0-3 to LOW by setting VBLANK.7, this discharges the 68nF capacitors (inside of the console), then reset VBLANK.7 again, the capacitors are now charged through linear 1M Ohm potentiometers in the paddle (in series with 1.8K Ohm resistors in the console).
```
  Pin  Bit      Expl.
  5,9  VBLANK.7 INPT0-INPT3 Control  (0=Normal Input, 1=Dumped to ground)
```
Then measure the time it takes until INPTx pins are getting high:
```
  Pin  Bit      Expl.
  9    INPT3.7  Paddle #3  pot    (right paddle, P1) (1=Charged/Ready)
  5    INPT2.7  Paddle #2  pot    (left paddle, P1)
  9    INPT1.7  Paddle #1  pot    (right paddle, P0)
  5    INPT0.7  Paddle #0  pot    (left paddle, P0)
```
Note: Connect buttons to GND (Pin 8), pots to VCC (Pin 7), leave 3rd pot pin not connected.

POT Timing Table (counted in scanline units)
Analogue timings are, of course, unstable. Expect the scanline count to vary by at least +/-1 even if the POT is not moved. Counter range may vary with different hardware, depending on accuracy/tolerance of the components in the paddle and in the console, and on the room temperature. Also, larger counter values may be measured when the circuit has warmed up (after some minutes of operation).
```
  Position   Left <--------- Center ---------> Right
  Degrees    -135   -90   -45   0   +45   +90   +135
  Ohms       1M     ...   ...  500K  ...   ...     0
  Scanlines  380    ...   ...  190   ...   ...     0
```
Note that it may take two or more frames until the maximum count is reached, ideally, software should check INPTs throughout drawing and blanking periods, and carry on in the next frame. Implement a timeout after 1000 scanlines (to prevent the software getting hung, and/or to detect the presence of paddles).

According to AtariAge FAQ, paddles are used by 31 games: Astroblast (joystick ok too), Bachelor Party, Backgammon, Beat Em' & Eat Em, Blackjack, Breakout (Breakaway IV), Bugs, Bumper Bash, Canyon Bomber, Casino (Poker Plus), Circus Atari (Circus), Demons to Diamonds, Eggomania, Encounter at L-5, Fireball, Guardian, Kaboom!, Music Machine, Night Driver, Party Mix, Picnic, Piece O Cake, Solar Storm, Star Wars: Jedi Arena, Steeplechase, Street Racer (Speedway II), Super Breakout, Tac-Scan, Video Olympics (Pong Sports), Warlords, Warplock.
Also by "actionmn.bin" ???

Controllers: Keypad

The keypad or touchpad controller has 12 buttons arranged into 4 rows and 3 columns. The names "0..9,#,*" are printed directly on the controller. Aside from this predefined names, software is usually shipped with overlays: small cards (with holes for the keys) with custom names or symbols printed on the card. The Basic Programming cartridge uses two separate keypads (giving it 24 keys in total).

To access the keypad, configure SWCHA as output (SWACNT=FFh), then select a row by writing to SWCHA, then wait 400us, then read column bits from INPTx registers.

Pin	PlayerP0	PlayerP1	Dir	Expl.		Keys
1	SWCHA.4	SWCHA.0	Out	Upper row	(0=Select)	1,2,3
2	SWCHA.5	SWCHA.1	Out	Second row	(0=Select)	4,5,6
3	SWCHA.6	SWCHA.2	Out	Third row	(0=Select)	7,8,9
4	SWCHA.7	SWCHA.3	Out	Bottom row	(0=Select)	*,0,#
5	INPT0.7	INPT2.7	In	Left column	(0=Pressed)	1,4,7,*
9	INPT1.7	INPT3.7	In	Middle column	(0=Pressed)	2,5,8,0
6	INPT4.7	INPT5.7	In	Right column	(0=Pressed)	3,6,9,#

Note: Pin 9 and 5 require external 4.7K Ohm pullups to pin 7.

According to AtariAge FAQ, kid's controllers/keypads are used by 11 games: A Game of Concentration (Hunt & Score Memory Match), Alpha Beam with Ernie, BASIC Programming, Big Bird's Egg Catch, Brain Games, Codebreaker, Cookie Monster Munch, Grover's Music Maker (prototype), MagiCard, Oscar's Trash Race, Star Raiders.

Controllers: Steering

Which 2600 games use the driving controllers?

```
  Indy 500 and
  Stell-A-Sketch (homebrew program)
```

The device looks about exactly like a paddle, using a knob as steering 'wheel', and a push button. Unlike paddles, only one device can be connected to each game port (ie. no Y-cable pairs). Also unlike paddles, the knob is attached to a 16 position rotary binary switch, which allows endless rotation (ie. by more than 360 degrees). Even though the 16 position switch provides 4 bit combinations, only 2 bits are connected, the output starts over at zero once every four positions.
Configure Port A as input (SWACNT=00h). Steering data can be then read from SWCHA and INPT4-5 as follows:

Pin	PlayerP0	PlayerP1	Expl.	
1	SWCHA.4	SWCHA.0	Pos	(0=Closed, 1=Not closed)
2	SWCHA.5	SWCHA.1	Pos	("")
6	INPT4.7	INPT5.7	Button	(0=Pressed, 1=Not pressed)

Note: Connect switch and push buttons to GND (Pin 8).

16 Position Real Code Binary Rotary DIP-Switch:

```
 ________
|   BIT1 |____________________ Pin 1 (Up)
|   BIT2 |__      |__ ____
|   BIT3 |__         |XOR \___ Pin 2 (Down)
|   BIT0 |___________|____/
| Rotary |        |__      ____ Pin 7 (Vcc)
|  DIP   |           [10K]
| Switch |____________________ Pin 8 (Gnd)
|    GND |        |  Button
|________|        |___o--o____ Pin 6 (Button)
```

Controllers: Lightgun

Lightgun
The XG-1 lightgun has been released in 1987 (originally for use with Atari XE computers). There are only a few Atari 2600 games supporting it:

```
  Sentinel (1990) (Atari)
  Shooting Arcade (prototype only)
  Bobby needs Food (homebrew)
  Guntest (homebrew test program) (Eckhard Stolberg)
```

"You can deduce the atari lightgun pinout combinig the sega light phaser pinout with the schematic for the SMS gun to 7800 adapter. It should be like this:"

```
  1   trigger         (active high)
  2   -
  3   -
  4   -
  5   -
  6   light sensor    (active low)
  7   +5V
  8   GND
  9   -
```

So if the lightgun is connected in the left controller port you have:

```
  bit 4 of SWCHA = 1 if the trigger is pressed, 0 otherwise.
  bit 7 of INPT4 = 0 if the sensor sees light, 1 otherwise.
```

If in the right controller port:

```
  bit 0 of SWCHA = 1 if the trigger is pressed, 0 otherwise.
  bit 7 of INPT5 = 0 if the sensor sees light, 1 otherwise.
```

Controllers: Trackball

The Atari "Trak-Ball" is some unreal thing.

Existing Trackball Hardware
Atari does have produced trackballs. In fact, they have even produced a whole mess of different/incompatible variants, often without even changing the CX-22 and CX-80 product numbers:
Trackballs with DIFFERENT product numbers aren't ALWAYS compatible with each other. And worse, trackballs with SAME product numbers aren't ALWAYS compatible with themselves.

Existing Trackball Software
Basically, there is NONE. The CX-22 manual states that the thing can be used with Atari 2600 joystick games, and, in trackball mode, with special games - but none such special games have been ever released for the Atari 2600. The only exception is a homebrew hack of the Missile Command game (the game is originally joystick controlled; different versions of the hack allow to use it in trackball mode & in Atari/Amiga mouse modes).

Atari Trak-Ball Schematic

```
   GND--||-------o----------------------------------- XCLK/Pin2   ;\Trackball
       470pF     |        .------------. 100nF       (YCLK/Pin4)  ;/Mode
   VCC---[10K]---o   VCC--|RST 4538   C|--||--.
                 |   VCC--|-TR multi RC|------o--[91K]--VCC
 X1--o--[430K]---o     .--|+TR vibr.  Q|--.
(Y1) |  .-----.  |     |  '------------'  |  .----.
     '--|+     |  |     |  .--------.      o--|4011|
        |LM339|--o-----o--|D       |      |  |NAND|-- Right/Pin4  ;\
  Vref--|-     |           |       Q|---------|    |   (Down/Pin2) ;
        '-----'           | 4113 /Q|---.  |  '----'               ; Joystick
   VCC---[10K]---.        | flip    |   |  |  .----.               ; Mode
                 |        | flop    |   |  '--|4011|               ;
 X2--o--[430K]---o        |         |   |     |NAND|-- Left/Pin2    ;
(Y2) |  .-----.  |   GND--|SET      |   o-----|    |    (Up/Pin1)  ;/
     '--|+     |  |   GND--|RES      |   |     '----'
        |LM339|--o--------|CK       |   '------------- XDIR/Pin1   ;\Trackball
  Vref--|-     |           '--------'                 (YDIR/Pin3)  ;/Mode
        '-----'                       ____
 Vref--------.                   GND--o  o--.   VCC--[L1]--Pin7   ;\Supply
             |                        ____  |   GND--------Pin8   ; and Fire
 GND--[8K1]--o--[100K]---VCC     GND--o  o--o-------- Fire/Pin6   ;/
```

The schematic is showing only X-Axis (and equivalent signals for Y-Axis in brackets). Mode selection via 4019/4030 isn't shown. Component list is:

```
  LM339 quad comperator/amplifier ;-mouse and all trak-ball versions
  4113  dual flip flop            ;\
  4538  dual multivibrator        ; all trak-ball versions only
  4011  quad NAND gates           ;/
  4019  eight-to-four multiplexer ;-trak-ball with trackball-mode only
  4030  quad XOR gates            ;-trak-ball with mouse-mode only
  SW    two position switch       ;-trak-ball with trackball or mouse mode only
```

Joystick Mode
The low-level X1/X2 signals are amplified via LM339. X2 raising edge is loading X1 as direction bit into the flip-flop, which is then output to joystick Left/Right signals accordingly. If the trackball isn't moved, then the multivibrator and NAND gates will mask off the signals.
Joystick mode is compatible with normal joystick games, but aside from that, it's just crap: the "analog" speed/direction informations are lost, and the result is likely to be less precise than real joysticks.

Trackball Mode
This version contains a J/T mode switch and multiplexer which bypasses the multivibrator and NAND stuff: The amplified X1 is directly output as XCLK signal (toggles when ball is moved), and the /Q flipflop output is output as direction signal.

Mouse Mode
The Atari mouse does contain only the LM339 chip, and amplified X1/X2 signals are then output directly to the joystick port. The Amiga mouse works the same, but has swapped pins on the connector.
The Trak-Ball with mouse-mode is containing flipflop, multivibrator, and NAND gates for joystick mode, and extra XOR gates for mouse mode (details are unknown, not quite clear if/how the XOR gates can switch between mouse and joystick modes).

Trackball & Mouse - Models & Modes

Type/Modes	Joystick	Trackball	Atari-Mouse	Amiga-Mouse
Old Atari CX-22 Trak-Ball	Yes	-	-	-
New Atari CX-22 Trak-Ball	Yes	Yes	-	-
Old Atari CX-80 Trak-Ball	Yes	Yes	-	-
New Atari CX-80 Trak-Ball	Yes?	-	Yes	-
Atari Mouse	-	-	Yes	-
Amiga Mouse	-	-	-	Yes

Controllers: Other

Foot Craz
Video Jogger (Exus)
Video Reflex (Exus)

Joyboard
Mogul Maniac (Amiga)
Off Your Rocker (Amiga)
Surf's Up (Amiga)

Controllers: Console Switches

Aside from external controllers, the console includes five built-in switches or buttons.

Accessing Console Switches

Configure Port B as input (SWBCNT=00h). Switch/Button data can be then read from SWCHB as follows:

```
  Bit        Expl.
  SWCHB.0    Reset Button         (0=Pressed)
  SWCHB.1    Select Button        (0=Pressed)
  SWCHB.2    Not used
  SWCHB.3    Color Switch         (0=B/W, 1=Color) (Always 0 for SECAM)
  SWCHB.4-5  Not used
  SWCHB.6    P0 Difficulty Switch  (0=Beginner (B), 1=Advanced (A))
  SWCHB.7    P1 Difficulty Switch  (0=Beginner (B), 1=Advanced (A))
```

Note: Older consoles used switches instead of buttons for reset/select.
SECAM consoles do not include a color switch (Bit 3 is grounded).

Software Switches/Buttons

Above buttons/switches may (or may not) be used by software for whatever purpose. Namely, the Reset Button does not reset any hardware, and the Color Switch does not have any effect on the video output (unless the software processes it as such).

Hardware Switches

On the contrary, the Power Switch and TV Channel Select Switch cannot be processed by software.

Data Direction Register (SWBCNT)

Port B is hardwired to console switches (inputs), normally SWBCNT should be always 00h. However, the three unused bits could be configured as output, the respective SWCHB bits can be then used to store 3bits of custom read/writeable data (as done by Combat).

Controllers: I/O Registers

38h - INPT0 (R) - 1....... TIA Dumped Input Port 1 (Pot 1)
39h - INPT1 (R) - 1....... TIA Dumped Input Port 2 (Pot 2)
3Ah - INPT2 (R) - 1....... TIA Dumped Input Port 3 (Pot 3)
3Bh - INPT3 (R) - 1....... TIA Dumped Input Port 4 (Pot 4)

As long as bit 7 of register VBLANK is zero, these four ports are general purpose high impedance input ports. When this bit is a 1 these ports are grounded.
INPT0-3 can be used to read up to four paddle controllers, and, INPT0-1 can be used to read Keyboard columns 0 and 1.

3Ch - INPT4 (R) - 1....... TIA Latched Input Port 4
3Dh - INPT5 (R) - 1....... TIA Latched Input Port 5

These two ports have latches that are both enabled by writing a "1" or disabled by writing a "0" to D6 of VBLANK.
When disabled, the microprocessor reads the logic level of the port directly.
When enabled, the latch is set for logic one and remains that way until its port goes LOW (not sure what happens if it was already LOW ???). When the port goes LOW the latch goes LOW and remains that way (until re-disabled by VBLANK Bit 6) regardless of what the port does.

280h - SWCHA - PIA Port A; input or output (R/W)

```
  Bit  Expl.
  0-7  Data Inputs or Outputs (Direction depends on SWACNT)
```

Used to access various types of external controllers (joysticks, paddles, or keyboard, etc.).

281h - SWACNT - PIA Port A DDR Data Direction Register (R/W)

```
  Bit  Expl.
  0-7  Data Direction for SWCHA Bit 0-7   (0=Input, 1=Output)
```

Use value 00h for joysticks and paddles. Use FFh for keypad.

282h - SWCHB - PIA Port B, Console Switches (R/W) (normally Read Only)

Bit	Expl.
0-7	Data Inputs or Outputs (Direction depends on SWBCNT)

Used to read console switches/buttons.

283h - SWBCNT - PIA Port B DDR Data Direction Register (R/W)

Bit	Expl.
0-7	Data Direction for SWCHB Bit 0-7 (0=Input, 1=Output)

Should be always 00h for reading buttons or switches.

Controllers: External Port Pin-Outs

Joystick Ports (two ports, male 9pin SUBD each)

Pin	Port 1	Port 2	Joystick	Paddles	Keyboard
1	SWCHA.4	SWACH.0	Up	-	Row 0
2	SWCHA.5	SWACH.1	Down	-	Row 1
3	SWCHA.6	SWCHA.2	Left	Button 1	Row 2
4	SWCHA.7	SWCHA.3	Right	Button 0	Row 3
5	INPT0	INPT2	-	Pot 0	Column 0
6	INPT4	INPT5	Button	-	Column 2
7	+5V	+5V	-	VCC/Pot	Pull-ups
8	GND	GND	GND	GND/Butt	-
9	INPT1	INPT3	-	Pot 1	Column 1

```
 .---------.
| 1 2 3 4 5 |
 \ 6 7 8 9 /
  '-------'
```

Cartridges

Cart Raw ROM (unbanked)

4K and 2K ROMs
Cartridges of 4KBytes fit directly into the 4K cartridge area.
Cartridges of 2KBytes are mirrored twice in the 4K cartridge area.

Smaller ROMs
Cartridges of 1KByte size haven't been manufactured, however, some homebrewn games may or may not use that filesize. A strange example is "Cave 1K", which is actually "2K", because it inserts 1024 bytes padding between (!) the program code and the reset vector (eventually intended for compatibility with software that accepts only 2K rom-images as minimum filesize).

Cart 4K-Banking (by Port FFxxh)

The bank number is selected by reading from (or by writing any value to) specific addresses, the addresses and corresponding bank numbers are:

```
  Size   Banks  FFF4 FFF5 FFF6 FFF7 FFF8 FFF9 FFFA FFFB
  2K,4K  1      -    -    -    -    -    -    -    -
  8K     2      -    -    -    -    0    1    -    -
  12K    3      -    -    -    -    0    1    2    -
  16K    4      -    -    0    1    2    3    -    -
  32K    8      0    1    2    3    4    5    6    7
```

Banks are usually selected by "BIT FFFx" opcodes, after execution of such opcodes the program counter is PC=PC+3, the new bank must contain valid code at that address (that is usually implemented by placing code at fixed addresses at the beginning or end of each ROM bank).
The initial bank number is undefined upon power-on, so each bank should contain a valid entry point at [FFFC], and a small power-on procedure.

Cart 2K-Banking (by Port FFxxh)

burgtime,he_man: eight 2K banks (16K)
also used by bnj (but first 8K empty, only upper 8K used)

```
  FFE0-FFE7 bank 0-7 for F000-F7FF
  FFE8 used by burgtime ONLY, don't know, initialize something ???
  expansion RAM at F800-F9FF
  bank 7 fixed at FA00-FFFF
```

The game BNJ consists of four 2K banks. Bank 3 is permanently mapped to F800-FFFF, and either bank 0,1,2 can be mapped to F000-F7FF, selected by accessing FFE4,FFE5,FFE6. Note: For whatever reason, the Bnj.bin rom-image at AtariAge is 16K, the first 8K are empty, the other 8K contain bank 0-3. (NB. this obscure file format allows to detect 2K-banking.)

Cart 1K-Banking (by Port FFxxh)

Memory Addresses for 8x1K (8K) Banking

```
  FFE0-FFE7 bank 0-7 for F000-F3FF
  FFE8-FFEF bank 0-7 for F400-F7FF
  FFF0-FFF7 bank 0-7 for F800-FBFF
  bank 7 fixed       at  FC00-FFFF
```

Used by: Popeye, Gyruss, Sprcobra, Tutank, Dethstar, Qbrtcube, Ewokadvn, Montzrev, Frogger2, Lordofth, James_bo, Swarcade, Toothpro.

Cart 2K-Banking (by Port 3Fh)

Memory Addresses for Port 3Fh 4x2K (8K) Banking

```
  MOV [3F],n     bank 0-2 for F000-F7FF
  bank 3 fixed            at  F800-FFFF
```

Used by: Springer, Espial, Minrvol2, Mnr2049r, Riverp, Polaris.

this port is NOT to be mirrored to [7F] - mnr2049r zerofills 40..FF memory

on power-on must be initially BANK 3 (polaris entrypt=7000, mirror of 7800)
access normal TIA ports only by mirrors at 40h-7Fh!

Cart 4K-Banking (by JSR/RTS opcodes)

JSR/RTS for banking (2x4K) (8K)
```
  bank1 select by CALL DXXX opcode,
  bank0 select by RET FXXX opcode
```
Used by Decathln only.

does not use any PUSH/POP, banking done by STACK accesses, probably only if SP.Bit0 is set (or cleared)
detect by clearmem-loop at F000, followed by call D000 opcode
entrypoint in bank0 only!

Cart Pitfall 2

The Pitfall 2 cartridge includes eight counters. (Five sequential video/data channels with automatically decrementing pointers, and one 3-channel audio/data generator). The program can read from the channels and can then write the data directly to the TIA ports (without having to adjust pointers/counters by software).

Cartridge I/O Map
```
  Channel   Data  Mask   Max/y1 Min/y2 Lsb   Msb
  ch0 video 1008  1010   1040   1048   1050  1058
  ch1 video 1009  -      -      -      1051  1059
  ch2 video 100A  -      -      -      1052  105A
  ch3 video 100B  1013   1043   104B   1053  105B
  ch4 video 100C  -      -      -      1054  105C
  ch5 audio /--\  -      1045   104D   1055  105D
  ch6 audio 1006  -      1046   104E   1056  105E
  ch7 audio \--/  -      1047   104F   1057  105F
  poly8     1000  -      -      -      -     -
  bank 0    1FF8  -      -      -      -     -
  bank 1    1FF9  -      -      -      -     -
```
Addresses 1000h-103Fh read only, 1040h and up write only.
Area 1000h-107Fh (and 1800h-187Fh) in ROM should be not used (FFh filled).

Display Data Channels
The first 5 channels are used to retrieve video data. The LSB and MSB registers contain a 11bit counter (the upper 5bit are unused?), used to address data in the 2K ROM area.
Normal data access: Reading from the DATA register returns the addressed byte and does automatically decrement the counter. Intended for general purpose display data (GRP0,PF0,COLUP0,etc.).
Masked data access: Reading from the MASK register works in the same way as above, but the returned data is forced to 00h if the channels output is disabled. The output is initially disabled (when writing to MIN/MAX/LSB/MSB?), output gets enabled when LSB=MAX, and gets disabled when LSB=MIN. Intended for vertical positioning of moveable objects (eg. hiding GRP0 in unused lines by forcing GRP0=00h).
(Pitfall2 program code uses masked access only for channel 0 and 3, not sure if other channels can use that feature as well?)

Audio Data Channels
The other 3 channels are used to generate 3 channel square? wave audio data. The LSB registers are 8bit counters automatically decremented at ca. 30kHz rate (or other speed?) (by external oscillator?), upon underflow, LSB is automatically reset to MAX?, the wave output is switched high when LSB=MAX, and low when LSB=MIN. For audio, the MSB serves as control register rather than as counter? MSB Bit5 seems to enable the

channel. MSB Bit4 does something/what: volume, length, noise, else? Other MSB Bits unused?
The output levels of the 3 channels are automatically added? together, probably high=5, low=0?, and can be read from Port 1006h, intended to be forwarded to AUDV0, with AUDC0=0 (always high), preferably at least once per scanline. The AUDx1 registers can be used as normal, as fourth sound channel.

ROM and Poly8
The first 8K (2000h) of the Pitfall ROM image contain normal program code/data, using normal 4K Port 1FF8h/1FF9h bank switching. The next 2K (800h) contain sequential data, used by channel 0-4, apparently in reversed order (in rom-image and/or real rom?), ie. addressed by "NOT counter". Finally, 255 bytes (FFh) of 8bit poly data are somewhat attached to the Pitfall2 ROM-images being found in internet (resulting in the obscure filesize of 28FFh bytes).
In reality, the poly data is likely to be generated by shift register, and probably not stored in a ROM? The poly data is used for what purpose? Audio? Not at all? Or Port 1000h? Pitfall reads from Port 1000h once every 4 frames, Bit 7 of the returned data seems to be used only? as color code for the blinking electric eels? If so, does one really need external hardware to produce blinking colors? The next poly value is generated when: at fixed rate or after reading from any or specific register?
The poly stream can be produced by taking the parity of Bits 7,5,4,3 of the current value, the new value is then shifted left, with the old parity (1=even) shifted in to Bit 0. Example: 00, 01, 03, 07, 0F, 1E, 3D, 7A, F4, E8, D0, A1, 43, 87, 0E, 1C, etc. The stream restarts each 255 cycles.

All info (and all question marks) gained from examining the pitfall2 rom-image, but without having viewed/listened/debugged the original cartridge.

NB.
When I first heard of the cartridge, having been wondering how it'd work, my first bet was to write a value of FFh to [TIA+mirror], allowing the cartridge to drag the databus to (FFh AND data). That method would take up only 3 cycles per write (instead 4+3 per read+write as actually used in pitfall2), and using different mirror address could be used for different channels. Not sure if it'd work stable, and if it'd be of any use to create games with higher video resolution, or for any other features.

Cart Expansion RAM

Expansion RAM
Some cartridges include additional 128 bytes (or 256 bytes) of external RAM, that memory is NOT (???) battery buffered. Because the cartridge slot does not include a read/write signal, different memory addresses must be used for reading and writing, mapped as such:

```
  Cart Type      4K-Banking 4K-Banking 2K-Banking
  RAM Size       128 bytes  256 bytes  256 bytes
  RAM Write Area F000-F07F  F000-F0FF  F800-F8FF
  RAM Read Area  F080-F0FF  F100-F1FF  F900-F9FF
  ROM Size       3.75K      3.5K       1.5K
  ROM Area       F100-FFFF  F200-FFFF  FA00-FFFF
```

The first 256 bytes (or 512 bytes) of each ROM bank are unused, these areas should be 00h or FFh filled in ROM-images. (NB. That allows to determine if a game uses external RAM. Omegarace and Digdug use garbage filled area though.)
RAM can be accessed from inside of all ROM-banks (but it is always the same 128 or 256 bytes of RAM, ie. RAM is NOT banked).
Note: Attempts to read from Write Addresses would overwrite the addressed byte by garbage. Thus, external RAM cannot be used by read-and-write opcodes (ie. INC, DEC, ASL, LSR, ROL, and ROR).

Cart Pin-Outs

Cartridge Slot (24 pins)
The A12 pin is used as CS (chip select, active HIGH), this works okay with the type of ROMs used in original cartridges. Standard EPROMs are expecting /CS though (active LOW), so that EPROM cartridges must include an inverter (transistor circuit, or 7404 chip, or similar).

```
+---------------------------------------------------+
|                                                   |
| /EX +5V  A8  A9  A11 A10 A12 D7  D6  D5  D4  D3   |
|  A7  A6  A5  A4  A3  A2  A1  A0  D0  D1  D2  GND  |
+---------------------------------------------------+
```

The /EXROM pin should be connected to GND inside of the cartridge, it isn't actually used, but it would allow the console to detect if a cartridge is inserted.

Cart Summary and General Info

Available ROM and Expansion RAM Sizes

```
ROM Type:Port RAM  Used by how many games
1K   -        -    1   (Cave1k - when removing 2K-padding)
2K   -        -    51
4K   -        -    296
8K   2x4K     -    111
8K   2x4K     128  3   (Stargate,Defendr2,Elevator)
8K   2x4K:JSR -    1   (Decathln)
8K   4x2K:3Fh -    6   (Springer,Espial,Minrvol2,Mnr2049r,Riverp,Polaris)
8K   4x2K     256  1   (Bnj - when removing 16K padding)
8K   8x1K     -    13
10K  2x4K+2K  -    1   (Pitfall2, with sequential data channels)
12K  3x4K     256  3   (Mtnking,Omegarac,Tunlrunr)
16K  4x4K     -    31
16K  4x4K     128  13
16K  8x2K     -    2   (Burgtime,He_man)
32K  8x4K     128  1   (Fatalrun)
```

AtariAge collection, 533 games in total.

ROM Origin & Entrypoint
ROM is usually originated at F000h, there are no headers or checksums required, only the Entrypoint (and Breakpoint, if using the BRK opcode) vectors are required to be located at following addresses:

```
FFFC-FFFD   CPU Entrypoint (16bit pointer)
FFFE-FFFF   CPU Breakpoint (16bit pointer)
```

That is, these addresses must contain a 16bit pointer (data), which points to the actual reset/break procedures (code). For small 2K ROMs, addresses FFFXh are mirrors of F7FXh.

Bank Switching
ROMs bigger than 4KBytes are split into several banks of 4KBytes each, and must include a small circuit that latches the current bank number.

Cartridge Mirrors
The 4K cartridge area is addressed by A12 signal, causing it to be mirrored to all addresses with Bit 12 set, 1000h, 3000h, 5000h, etc. (2K ROMs also to 1800h, 3800h, 5800h, etc.)
Most 2K and 4K programs are originated at F000h. Larger ROMS are often using different addresses for each bank - for example, bank 0 at D000h, and bank 1 at F000h (mainly to make .MAP files more obvious).

File Extension
The most common extension for Atart 2600 ROM-Images is ".bin".
Reportedly, some PAL games use ".pal". Also, some games use ".a26".

CPU 65XX Microprocessor

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

Atari 2600 - CPU 6507
The 6507 is a 6502-compatible CPU squeezed into a DIL-28 package, it's having only 8K address space, and doesn't have any IRQ or NMI inputs.
Clocked at 1.193182 MHz (NTSC), 1.182298 MHz (PAL).

Hardware / Soldering

Hardware Tuning
Nocash SRAM Circuit
Composite Video and Audio Out
Using a PC Power Supply

Cartridge Slot and Controller Ports
Controllers: External Port Pin-Outs
Cart Pin-Outs

Mainboard - CPU, PIA, TIA
Chipset Pin-Outs

Other Connectors
- TV: One Cinch Socket (Video/Audio TV Signal) ;doesn't work with all RF cables
- Power: 9V DC, 500mA (internally converted to 5V DC)

Nocash SRAM Circuit

Nocash SRAM Circuit

This circuit has been developed while testing no$2k6, it wired directly to the Atari mainboard, and allows to upload ROM-images from PC to Atari via function in no$2k6 utility menu. Features:

```
 - one-directional 4bit highspeed upload (when joystick is not moved)
 - bi-directional 1bit transmission for upload/download/debug terminal
 - works with external carts (if parallel cable disconnected or all high)
 - upload with automatic reset (autostart)
 - works with any older/newer one/two-directional PC parallel port
 - supports almost all existing cartridge types (except eight games)
```

Only a few exotic cartridges (eight games) are not supported: JSR banking (Decathln), 256 bytes RAM expansion (Mtnking, Omegarac, Tunlrunr), Port FFEX Nx2K banking (Bnj, Burgtime, He_man), and the sequential memory controller (Pitfall2).

Step 1 - Basic Connection (raw 1K,2K,4K unbanked ROM) (for 348 games)

```
                _____________                _____________
        D0-7 --|D0-7         |       D0-7 --|D0-7         |
   A0-6,8-10 --|A06,8-10     |  A0-6,8-10 --|A0-6,8-10    |
      XA7,11 --|A7,11        |  XA7,11-14 --|A7,11-14     |
         VCC --|A12-15 EPROM |       /WRS --|/WE   SRAM   |
       /MREQ --|/CS    64Kx8 |      /MREQ --|/CS   32Kx8  |
  /PROM.DTA3 --|/OE   27C512 | /SRAM.DTA2 --|/OE   62C256 |
               |_____________|              |_____________|

  XA7         --------- A7                /RES.DTA4   ---|<|--- /RES,TP2,C27(+)
  XA11        --------- A11               /PROM.DTA3  --[10K]-- VCC
  R/W         --------- /WRS              /SRAM.DTA2  --[10K]-- VCC
  LPT.DTA0    --------- SWCHB.5,PIA.18   CART.DTA5   --[10K]-- VCC
  LPT.DTA1    --------- SWCHB.4,PIA.19   XA12        --[10K]-- VCC
  LPT.BUSY    --------- SWCHB.2,PIA.22   XA13        --[10K]-- VCC
  LPT./STB    --------- SWCHA.0,PIA.8    XA14        --[10K]-- VCC
  LPT./AUTOLF --------- SWCHA.2,PIA.10                  ____
  LPT./INIT   --------- SWCHA.4,PIA.12          A12  --|AND \__ CS,SLOT.18
  LPT./SELECT --------- SWCHA.6,PIA.14    CART.DTA5  --|____/
  LPT.GND     --------- GND                     A12  --|NAND\__ /MREQ
  CS,SLOT.18  --/cut/-- A12                    PHI2  --|____/
```

Step 2 - Port 3Fh (Nx2K ROM) (for 6 games & control bits for next step)

```
                               ____               _______
  A11 --/cut/--- XA11   A6  --|    |-------------|LE      |
  GND ---------- 1K     A7  --|NOR |       D0 --|D0   Q0|-|<|- XA11
  LPT.DTA6 ----- 3F     A12 --|    |       D1 --|D1   Q1|-|<|- XA12
            ____        R/W --|    |       D2 --|D2   Q2|-|<|- XA13
  3F    ___|INV \_____________|    |       D3 --|D3   Q3|-|<|- XA14
           |____/    /3F      |____|       D4 --|D4   Q4|-- 1K
  A11   ---|NOR |   ____                   D5 --|D5   Q5|-- 4K
  3F    ---|    |__|INV \--|<|--XA11       D6 --|D6   Q6|-- /16K
  1K    ---|    |  |____/            ____  D7 --|D7   Q7|-- /32K
  GND   ---|    |              3F --|AND \_______|/OE     |
  GND   ---|____|             A11 --|____/       |_______| 74373
  XA11 --[10K]-- VCC         XA10 --[10K]-- VCC      1K --[10K]-- VCC
```

Step 3 - Port FFFXh (Nx4K ROM) (for ca. 160 games)

```
         ____             ____                              _________
  PHI2 -|    |   A2 -----|NAND\_ ____           ____   4K -|/CE1 /OE1|-- XRAM
  A5 ---|    |   /32K ---|____/ |XOR \_________|    |  3F -|/CE2 /OE2|-- GND
  A6 ---|NAND|   A1 -----|NAND\_|____/ /GOODA2 |    |______|CLK   RES|-- GND
  A7 ---|    |   16K ----|____/         _______|NOR |      |  74173  |
  A8 ---|    |   A2 -----|NAND\_ ____   GND    |    |  A0 -|D0     Q0|-|<|-XA12
  A9 ---|    |   16K32K -|____/ |XOR \_________|    |  A1 -|D1     Q1|-|<|-XA13
  A10 --|    |_  A3 ------------|____/ /GOODA3 |    |  A2 -|D2     Q2|-|<|-XA14
  A11 --|____| | GND --- XRAM  ________________|    |  NC -|D3     Q3|-- NC
         ____  |______________|  ____  /GOODFF |    |      |_________|
  /16K -|XOR \___            ___|INV \_________|    |  /16K --|XOR \__ 16K
  /32K -|____/   16K32K    A4   |____/ /GOODA4 |____|   VCC --|____/
```

Step 4 - XRAM (128 Bytes Expansion RAM) (for 17 games) (and Write Protect)

```
                ____                                              ____
  A8         ---|    |-- XRAM    GND --/cut/-- XRAM         XRAM -|OR  \__ XA7
  A9         ---|NOR |            A7 --/cut/-- XA7          A7  --|____/
  A10        ---|    |           R/W --/cut/-- /WRS    ____  R/W --|OR  \__ /WRS
  A11        ---|    |   /XRAM.DTA7 -----------------|AND \_______|____/
  /XRAM.DTA7 ---|____|   /3F        -----------------|____/ WPROT
```

Step 5 - Port FFXXh (Nx1K ROM) (for 13 games)

```
            __________                                           ____
  A0..2 ---|D0-2  Q0-2|--- XA10..12   A10 --/cut/-- XA10   1K --|OR  \-|<|-XA10
  A12 -----|D3      Q3|--- NC         GND --/cut/-- 1K    A10 --|____/
  A3 ------|WA0    RA0|--- A10                ____         3F __|INV \-|<|- 1K
  A4 ------|WA1    RA1|--- A11  ____ _______/AND |--- A10       |____/
  /GOODFF -|/WR    /RD|________/OR  |       \____|--- A11  1K __|INV \_____ /1K
           |__________| 74170  \____|__/1K                      |____/
```

Notes
The Eprom socket can be soldered directly on top of the SRAM (only Pins 1, 2, 22, 26, and 27 need different connection).
Implement the separate steps in incrementing order, steps 2 and up are optional, the device should be fully functional after completion of each step, do not attempt to implement all steps at once, establish 1-2 days per step.

Parts List
```
  1  27C512 EPROM (or EEPROM, or FLASH, or other size, min 1KByte)
  1  62C256 Static RAM (32KBytes or bigger)
  1  74LS04 Hex Inverter
  1  74LS08 Quad 2-input AND gate
  1  74LS30 Single 8-input NAND gate
  1  74LS32 Quad 2-input OR gate
  1  74LS86 Quad 2-input XOR gate
  1  74LS170 4x4 Register File open collector
  1  74LS173 4-bit 3-state flip-flop with clock enable
  2  74LS260 Dual 5-input NOR gate
  1  74LS374 8-bit 3-state flip-flop
  9  10K Ohm Resistors
 11  1N4148 diodes (for /Reset signal, and ANDed "XAnn" and "1K" outputs)
  1  Centronics socket 36pin female (and standard printer cable)
```
Plus socket(s) for EPROM (and any other chips), 100nF capacitors for all chips, wire, board, solder, tools, eprom burner.
Caution: 74LS260 outputs are at Pin 5 & 6 (unlike some internet-specs say)

BIOS EPROM
```
  0000 20 4A FC 9D FC 34 20 4C FC FF FC 26 20 25 FD 85
  0010 85 20 25 FD 85 86 A9 00 85 80 A9 F0 85 81 20 87
  0020 00 20 3F FC 4C 30 FC 20 87 00 20 3F FC 4C 30 FC
  0030 20 4A FC D1 FC 2E 20 4C FC 25 FD 27 4C 87 00 A5
  0040 82 20 FF FC A5 83 20 FF FC 60 A0 87 20 6C FC 85
  0050 80 20 6C FC 85 81 20 6C FC 85 84 98 AA A0 00 B1
  0060 80 95 00 C8 E8 C4 84 D0 F6 8A A8 60 BA F6 03 D0
  0070 02 F6 04 A1 03 60 A0 00 84 80 20 AE 00 85 3F 20
  0080 AE 00 85 81 0A F0 15 20 AE 00 91 80 18 65 82 85
  0090 82 A9 00 65 83 85 83 C8 D0 ED F0 DE 60 20 BB 00
  00A0 A6 85 DD 00 FF A0 00 B1 80 48 20 BB 00 68 18 65
  00B0 82 85 82 A9 00 65 83 85 83 E6 80 D0 E3 E6 81 D0
  00C0 DF A9 F0 85 81 A5 85 E6 85 C5 86 D0 D3 20 BB 00
  00D0 60 20 B5 00 85 3F 20 B5 00 A2 58 86 49 8D 97 00
  00E0 CD 00 FF A9 0F 85 3F A9 FF 8D 97 02 8D 80 02 8D
  00F0 82 02 A9 00 85 81 8D 81 02 8D 83 02 6C FC FF 85
  0100 84 A0 04 A9 10 06 84 69 03 8D 82 02 A9 10 2C 82
  0110 02 D0 FB 06 84 69 03 8D 82 02 A9 10 2C 82 02 F0
  0120 FB 88 D0 E1 60 A9 01 85 84 A9 10 2C 82 02 D0 FB
  0130 0A 2D 82 02 C9 20 26 84 A9 10 2C 82 02 F0 FB 0A
  0140 2D 82 02 C9 20 26 84 90 E0 A5 84 60 A9 10 2C 82
  0150 02 D0 FB 0E 80 02 2C 82 02 F0 FB AD 80 02 60 A2
  0160 00 BD 99 FD 20 FF FC E8 BD 99 FD D0 F4 A2 00 20
  0170 25 FD DD AA FD D0 FE E8 E0 08 D0 F3 A9 2B 20 FF
  0180 FC A2 00 A0 2B 20 4C FD DD AA FD F0 02 A0 2D E8
  0190 E0 08 D0 F1 98 20 FF FC 60 4E 4F 24 32 4B 36 20
  01A0 42 49 4F 53 20 56 31 2E 31 00 00 FF 55 AA 0F F0
  01B0 3C C3 20 25 FD 85 3F 20 25 FD CD F7 FF 20 25 FD
  01C0 85 3F 60 78 D8 A9 00 AA 95 00 9A E8 D0 FA A9 0E
  01D0 85 3F A9 04 8D 83 02 A9 AA 8D 81 02 A2 28 86 49
  01E0 20 5F FD 20 B2 FD 20 25 FD C9 31 F0 0B C9 34 F0
  01F0 1A C9 44 F0 25 4C F5 FD A2 00 86 49 20 4A FC 76
  0200 FC 27 20 4C FC 25 FD 27 4C 27 FC 20 4A FC 76 FC
  0210 27 20 4C FC 4C FD 13 4C 27 FC 4C 00 FC FF FF FF
  0220 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
  .... FF FF FF ............................ FF FF FF
  03F0 FF FF FF FF FF FF FF FF FF FF FF FF C3 FD 00 00
```

To be placed to highest memory location, eg. FC00h-FFFFh for 64K chips.

Composite Video and Audio Out

Composite Video Mod

This circuit from http://www.console-corner.de/videomod.html uses 5 resistors and 1 capacitor. The picture quality is very good, and it's less complicated than many other approaches (some of them require potentiometers, non-standard resistor values, transistors, chips, or need to disconnect the original RF modulator).

```
  LUM0'----[3K3]----o----------------o--------VIDEO OUT
                    |                |
  LUM1'----[2K2]----o               100p
                    |                |
  LUM2'----[1K]-----o    TIA.COLOR---o-----[1K]-----GND
                    |
  SYNC'----[330]----o    TIA.AUDIO------------AUDIO OUT
```

The LUMx' and SYNC' signals are found on the following pins of the 4050-chip:

```
  Type          LUM0'      LUM1'       LUM2'       SYNC'
  Atari 2600    4050.Pin2  4050.Pin12  4050.Pin15  4050.Pin4
  Atari 2600A   N/A        N/A         N/A         N/A
  Atari Junior  4050.Pin2  4050.Pin10  4050.Pin15  4050.Pin12
```

The COLOR, AUDIO, LUMx and SYNC signals are on these TIA (6526) pins:

```
  Type  COLOR     AUDIO         LUM0      LUM1      LUM2      SYNC
  PAL   TIA.Pin9  TIA.Pin13     TIA.Pin7  TIA.Pin5  TIA.Pin6  TIA.Pin2
  NTSC  TIA.Pin9  TIA.Pin13+12  TIA.Pin8  TIA.Pin5  TIA.Pin7  TIA.Pin2
```

Best use the LUMx' and SYNC' signals on the 4050-chip. Some mainboards do not include that chip, if it's absent: pick the LUMx and SYNC signals on the TIA-chip (you may need to change the resistor values in that case).

RF Notes

Installing the 4 resistors on the 4050 outputs has little (or no) effect on RF output, installing the Color-to-GND resistor makes dark RF colors slightly darker, installing the capacitor & connecting the video-out cable does reduce the quality of the RF output - but it's still working, and you'll no longer need it anyways.

Using a PC Power Supply

When using a PC (or other computer) to develop Atari games it might be recommended to connect the console directly to the PC's power supply, without having to use the external 9V power supply.

Using +5V DC Power Supply
Connect +5V (red floppy cable) directly to the 7805 voltage regulator's 5V output (not to the 9V input). Connect GND (black floppy cable) to middle 7805 pin (not required if centronics cable is connected). Most of the console works fine at 5V, but a few things still need to get modified to get it working at raw 5V only:
Move the power switch into 5V line (so that 9V are always on, and that 5V can be switched on and off). Replace the 750 Ohm LED resistor (R57) by 390 Ohm, and connect it to 5V instead of 9V. And, most important, connect color adjust potientometer (R9) to 5V instead of 6.2V, and re-adjust it so that it outputs approximately 3.38V (PAL) at the middle pin.
That's it. It's now everything working at 5V internally - and would still work alternately with 9V external supply.

Using +12V DC Power Supply
Alternately, +12V (yellow floppy cable) could be connected directly to the 9V input without any console modifications.
Drawbacks would be that the 7805 voltage converter would be producing more heat, and that some consoles are using a '3.5mm headphone socket' as power input - which would produce a shortcut (and shutdown the PC power supply) when inserting/removing the plug.
Note: An external 7809 would solve that problems, it'd split the heat to the 7809 and 7805, and it'd provide a more or less reliable protection against shortcuts (at least for short periodes, the 7809 would get very hot if the plug is stuck in 'half inserted' position).

Chipset Pin-Outs

CPU 6507 - Central Processing Unit (cut-down 6502 with only 28 pins)
```
  25..28   D0..D7
  4,2      VCC,GND
  5..17    A0..A12
  1,3,26   /RES,RDY,R/W
  27,28    PHI0,PHI2
```

PIA 6532 - (128 bytes RAM, I/O Ports, Timer)
```
  33..26   D0..D7
  7..2,40  A0..A6
  35       R/W
  25       IRQ (NC)
  36       /RS (A9)
  37       /CS2 (A12)
  38       CS1 (A7)
  39       PHI2
  34       /RES
  8..15    PA0..PA7 (UDLR2, UDLR1)
  1,20     GND,VCC
  16..24   PB7..PB0 (DIF1,DIF0,NC,NC,COLOR,SELECT,RESET)
```

TIA 6526 - Television Interface Adapter (Video, Audio, Pot/Button Inputs)
```
  20,23,22,1      VCC,VCC,GND,GND
  14..19,33..34   D0..D7
  32..27          A0..A5
  21,24           /CS1 (A7), /CS2 (A12)
  3,25,4,26,11    RDY,R/W,PHI0,PHI2,OSC
  40..37,36,35    P0..P3 (POT0..3), T0..T1 (BUTTON1,2)
  2,9,10          SYNC, COLOR, CADJ (+3.4V)
```
For PAL (6526P):

```
  7,5,6          LUM0,LUM1,LUM3
  13,12,8        AUD,PALS,PALI    ;AUD is audio AUD0+AUD1 merged on one pin
```
For NTSC:
```
  8,5,7          LUM0,LUM1,LUM2
  13,12,6        AUD0,AUD1,/BLK   ;Pin6 exist in 2600 only (not 2600A?)
```

4050 (non-inverting buffer; used as amplifier or edge-sharpener or so)
```
  3,5,7,9,11,14  INPUT A,B,C,D,E,F
  2,4,6,10,12,15 OUTPUT A,B,C,D,E,F
  1,8,13,16      VCC,GND,NC,NC
```
The chip exists in (most) PAL/NTSC consoles, connected like so:
```
  Atari 2600   A=LUM0, B=SYNC, C=JOY2, D=JOY1, E=LUM1, F=LUM2
  Atari 2600A  N/A     N/A     N/A     N/A     N/A     N/A
  Atari Junior A=LUM0, B=/RES, C=PALI, D=LUM1, E=SYNC, F=LUM2
```
Not sure about the purpose of C=PALI in PAL consoles, and no idea if/how that pin is used in NTSC consoles (?)

Links

AtariAge - Atari 2600 ROMs
About 500 games (ROM images) for the Atari 2600, about 2MB zipped.
```
  http://www.atariage.com/system_items.html?SystemID=2600&ItemTypeID=ROM
```
The webpage also includes a nice FAQ, very useful schematics, and various other info. The separate pages are badly generated bloated html, loading VERY slowly, and likely to crash your browser.

Stella Programmer's Guide
Official specs for Atari 2600, TIA and PIA chips (but lacks information about CPU and ROM), by Steve Wright 12/03/79, reconstructed by Charles Sinnett 6/11/93.
```
  http://alienbill.com/vgames/guide/docs/stella.html
```
This document is a matter of slightly increasing information, each chapter repeats the exact information from the previous chapter, and eventually reveals some additional details.

65xx processor series opcodes
Very nice summary of documented and undocumented 65XX opcodes.
```
  http://oxyron.net/graham/opcodes.html
```

nocash 2K6 specs
This document (or newer updates), in TXT and HTM format.
```
  http://nocash.emubase.de/2k6specs.txt
  http://nocash.emubase.de/2k6specs.htm
```
Atari 2600 Programming Specs. Describes TIA and PIA I/O ports, CPU, memory, cartridges, joysticks, paddles, etc.