

Pac-Man Emulation Guide v0.1, Oct 2008

This document is a reference for the Pac-Man video game arcade hardware, mainly written to document my findings as I create a Pac-Man emulator for learning purposes.

The information was gathered from the Internet and my own investigations and was merged and edited into this document. There are various red **TODOs** left in the document where I hope to add or clarify material.

This was written by Chris Lomont in Oct 2008. If you have comments or fixes then email me at (my first name) at [lomont.org](mailto:lolomont@gmail.com). Also check out my website www.lomont.org.

To start, Figure 1 shows a Pac-Man “attract” screen and the start of a level.



Figure 1 – Two views of Pac-Man

Introduction

Throughout this document decimal numbers appear as usual, and hexadecimal numbers are suffixed with an 'h'. For example 1K is 1024 bytes = 400h bytes.

Pac-Man hardware is at the highest level:

- Main CPU : Z80A at 3.072 MHZ
- [ROM](#): 16K of ROM in four 4K chips
- [RAM](#): almost 2K of working RAM
- [Video](#):
 - 2K of RAM as follows
 - 1K holds tile information.
 - 1K holds per tile palette indices.
 - The visible screen is 28x36 tiles.
 - Each tile is an 8x8 pixel 4-color image from 16 possible colors.
 - Video resolution is thus 224x288 pixels.
 - Eight 16x16 pixel hardware sprites, each with a 4-color palette from 16 possible colors.
 - 60.61 frames per second (60.61fps).
- [Sound Chip](#) : Custom monophonic 3-voice 4-bit Wave Sound Generator

Z80

The heart of the Pac-Man arcade machine is the 8-bit Z80 Central Processing Unit (CPU). It was released in July 1976, borrowing ideas from the Intel 8080. In fact it was designed to be binary compatible with the 8080.

The Pac-Man Z80 runs at 3.072 MHZ.

One important feature is that it addresses 16 bits of memory, so all Pac-Man addresses must lie in the range 0-65535 (0000h-FFFFh). Pac-Man actually uses less than half of this.

The Z80 also supports three interrupt modes, although the game only uses one, the so-called "mode 2" set via the Z80 instruction "IM 2." An emulator must support port 0 in hardware, which is used to store a byte (used to form an address for an interrupt vector) which must be returned when triggering an interrupt. See the section on [interrupts](#).

Memory Map

All hardware is memory-mapped, that is, can be accessed by reading and writing memory addresses from the CPU.

The game ROM uses the lowest 16K of memory, followed by 2K of video memory for tile selection and per tile palettes (1K each), and then a little less than 2K of general purpose RAM. This is followed at address 5000h with 256 bytes of memory-mapped hardware devices such as counters, sound hardware, and joystick switches.

Addresses	Bytes	Description
0000h-3FFFh	16,384	Game ROM in four 1K chips: 0000h-0FFFh pacman.6e 1000h-1FFFh pacman.6f 2000h-2FFFh pacman.6h 3000h-3FFFh pacman.6j
4000h-43FFh	1024	Video RAM (tile information)
4400h-47FFh	1024	Video RAM (tile palettes)
4800h-4FEFh	2032	RAM
4FF0h-4FFFh	16	Sprite #, x and y flip bits (write only). 8 pairs of 2 bytes: First byte is sprite number 0-31, (bits 2-7), X flip (bit1), Y flip(bit 0) Second byte is sprite palette
Memory Mapped Registers*		
5000h-503Fh read	1	IN0 (joystick and coin slot) (each byte returns same value)
5000h write	1	Interrupt enable (bit 0: 0 = disabled, 1 = enabled)
5001h write	1	Sound enable (bit 0: 0 = disabled, 1 = enabled)
5002h write	1	??? Aux board enable?
5003h	1	Flip screen (bit 0: 0 = normal, 1 = flipped)
5004h	1	1 player start lamp (bit 0: 0 = on, 1 = off)
5005h	1	2 players start lamp (bit 0: 0 = on, 1 = off)
5006h	1	Coin lockout (bit 0: 0 = unlocked, 1 = locked)
5007h	1	Coin counter (trigger by changing bit 0 from 0 to 1)
5040h-507Fh read	64	IN1 (joystick and start buttons) (each byte returns same value)
5040h-5045h write	6	Sound voice 1 5040-5044 – accumulator (low nibbles, used by H/W only) 5045 – waveform (low nibble)
5046h-504Ah write	5	Sound voice 2, laid out like voice 1, missing low accumulator nibble
504Bh-504Fh write	5	Sound voice 3, laid out like voice 1, missing low accumulator nibble
5050h-5054h write	5	Voice 1 frequency (low nibbles)
5055h write	1	Voice 1 volume (low nibble)
5056h-505Ah write	5	Voice 2 frequency and volume, laid out like voice 1
505Bh-505Fh write	5	Voice 3 frequency and volume, laid out like voice 1
5060h-506Fh write	16	Sprite x, y coordinates
5080h-50BFh read	64	Dip Switch Settings (each byte returns same value)
50C0h-50FF write	64	Watchdog reset (each byte has the same function)

Table 1 - Pac-Man memory map

* Note some addresses have different read and write behavior.

ROMs

Pac-Man hardware needs various Read-Only-Memory chips to function. These hold program code, color and palette definitions, sprite and tile images, and sound waveforms. There are various sets from different manufacturers and countries.

Table 2 lists the ten ROMs needed to run Pac-Man. The filenames are from MAME¹ and come from the physical board description. These are for the Midway² 1980 version of Pac-Man.

Filename	ROM usage	Bytes	Comments	CRC ³
pacman.6e	Code ROM1	4096	4K of code	C1E6AB10
pacman.6f	Code ROM2	4096	4K of code	1A6FB2D4
pacman.6h	Code ROM3	4096	4K of code	BCDD1BEB
pacman.6j	Code ROM4	4096	4K of code	817D94E3
82s123.7f	Color ROM	32	32 one-byte colors	2FC650BD
82s126.4a	Palette ROM	256	64 four-byte palettes	3EB3A8E4
pacman.5e	Tile ROM	4096	256 8x8 pixel tile image	0C944964
pacman.5f	Sprite ROM	4096	64 16x16 sprite images	958FEDF9
82s126.1m	Sound ROM 1	256	8 waveforms	A9CC86BF
82s126.3m	Sound ROM 2	256	8 waveforms	77245B66

Table 2 – Pac-Man ROMs

Obtaining ROMs

These ROM sets can be found on the internet and in commercial emulators such as *Microsoft Arcade Pocket Pack*. Since they are copyrighted by the original owners, you should obtain a legal copy from an emulator or purchase a real Pac-Man machine from eBay! You also could purchase a legal emulator, such as the one mentioned, and strip out the ROMs.

Code ROMs

The four code ROMs are unencrypted binary Z80 code. Disassembled versions are on the web, although they are poorly commented. **TODO – finish commenting disassembly and post?**

Color ROM

Colors are defined in a 32-entry palette ROM (82s123.7f) where each 8-bit entry stores a 3-bit value for each of red and green and a 2-bit value for blue. The hardware maps these values through resistors of value 1000s ohm, 470 ohms, and 220 ohms, dropping the 1000 ohm resistor for blue⁴. Thus Pac-Man hardware can use 32 colors, however the ROM (and hence the game) only contains 16 colors in the lower 16 bytes; the upper 16 bytes are zeros for all black.

¹ MAME = Multi-Arcade Machine Emulator. <http://en.wikipedia.org/wiki/MAME>

² **TODO – more ROMsets?**

³ **TODO – CRC type?**

⁴ I think this is correct. **TODO – check this.**

Each color entry corresponds to color intensity on the output hardware. For a fixed voltage V, current (giving intensity) is proportional to 1/Resistance. The maximum current is then

$$\frac{V}{220} + \frac{V}{470} + \frac{V}{1000} = \frac{3967 V}{517000}$$

The highest intensity, 3967, is scaled to the highest color-channel value FFh. Noting

$$\frac{1}{220} = \frac{2350}{517000}$$

$$\frac{1}{470} = \frac{1100}{517000}$$

$$\frac{1}{1000} = \frac{517}{517000}$$

then the 220 ohm resistor (and the corresponding bit) gives a weight of 2350 * FFh/3967 =97h. Similarly 470 ohm gives 47h and 1000 gives 21h.

Using this to compute the output RGB values gives the weights 97h, 47h, and 21h. Since the blue channel only has 2 bits with 470 and 220 ohm resistors, similar calculations give weights 51h and AEh respectively. This is summarized in Table 3.

Bit	Color	Connected to resistor	Weight
0	Red (Least amount)	1000 ohm	21h
1	Red	470 ohm	47h
2	Red (Most amount)	220 ohm	97h
3	Green (Least amount)	1000 ohm	21h
4	Green	470 ohm	47h
5	Green (Most amount)	220 ohm	97h
6	Blue	470 ohm	51h
7	Blue	220 ohm	AEh

Table 3 – Color ROM bits

The first byte in the ROM is color 0, the second byte is color 2, etc. Although the ROM has space for 32 colors, only the first 16 are used, with the remaining 16 bytes being 0 which is black. The first 16 colors are shown in Figure 2, and the corresponding RGB values (0-255 range) are

{0,0,0}, {255,0,0}, {222,151,81}, {255,184,255}, {0,0,0}, {0,255,255}, {71,184,255}, {255,184,81}

{0,0,0}, {255,255,0}, {0,0,0}, {33,33,255}, {0,255,0}, {71,184,174}, {255,184,174}, {222,222,255}



Figure 2 – Pac-Man Colors

For some reason four of the entries are black⁵.

Palette ROM

Each palette in the 256 byte palette ROM (82s126.4a) consists of 4 bytes, so this ROM stores 64 palettes. The first four bytes are palette 0, the next four bytes are palette 1, etc. The values in each entry reference a color from the color ROM. Since the color ROM only uses 16 colors, each byte only uses the lower 4 bits to store a number 0-15. Finally, only the first 32 palettes are not completely black, and of these several are all black.

Figure 3 shows the first 32 palettes, left to right, one palette per column. Palette color 0 is the bottom and palette color 3 is the top. By inspection the ghost palettes are 1, 3, 5, and 7.



Figure 3 – Pac-Man Palettes

Tile (Character) ROM

The background, scores, bonus items, and dots are drawn on a static background using images from the 4096 byte tile ROM (pacman.5e). Figure 4 shows the images decoded from the tile ROM using palette 1.

Each tile is an 8x8 pixel image, stored as 2 bits per pixel, thus using 16 bytes per tile. The 4096 byte ROM thus holds 256 tiles, including a character font and two sets of differently spaced digits.

Drawing a tile uses a byte in Tile RAM to select one of the 256 four-color tiles to display, and also uses a byte in the Palette RAM to select the palette used to color the tile.

Decoding images from the Tile ROM is slightly messy. Each byte stores 4 pixels, but they are stored in bitplanes. Thus the least significant bit of each pixel is stored in the lower four bits of the byte, and the most significant bit of each pixel is stored in the higher 4 bits of the byte (see Table 4).

Bit	Usage
0	Bit 0 of pixel #1
1	Bit 0 of pixel #2
2	Bit 0 of pixel #3
3	Bit 0 of pixel #4
4	Bit 1 of pixel #1
5	Bit 1 of pixel #2
6	Bit 1 of pixel #3
7	Bit 1 of pixel #4

Table 4 – Tile ROM byte layout

⁵ TODO – figure out why.

The resulting 4 pixels from each byte are vertical strips. The first 8 bytes draw the lower half of the tile, left to right, top to bottom. The last 8 bytes defining the tile draw the top half of the tile⁶. The images are stored in columns since the Pac-Man hardware has the screen rotated 90 degrees, and reading them in this order results in easier scanline drawing in hardware (TODO - conjecture!?).

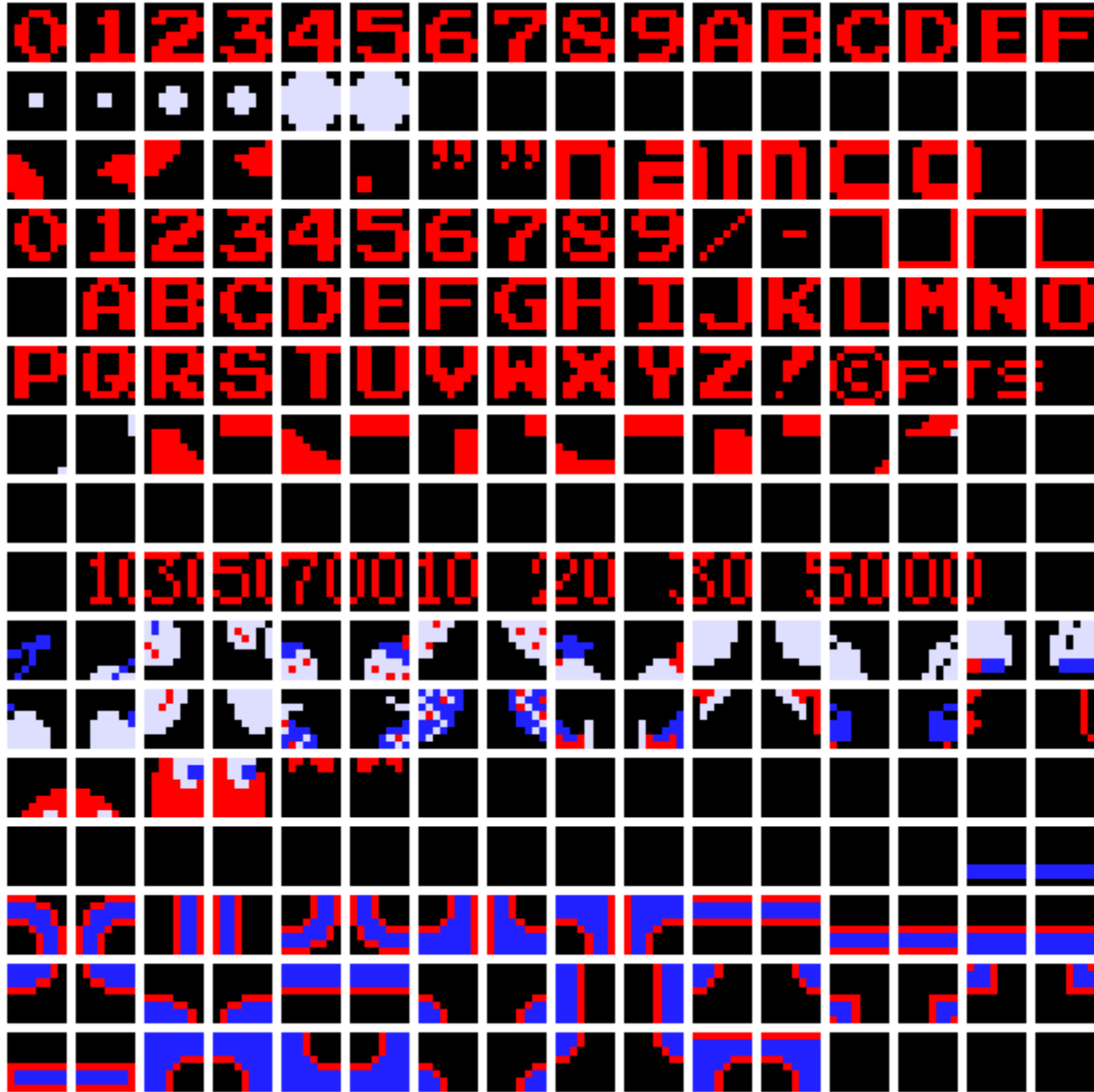


Figure 4 – Pac-Man tiles in the tile ROM

Sprite ROM

The 4096 byte Sprite ROM (pacman.5f) stores 16x16 pixel sprites. Each pixel uses 2 bits, resulting in each sprite using 64 bytes of ROM. $4096/64 = 64$ sprites stored in ROM. Figure 5 shows the decoded sprites

⁶ TODO – check this is correct.

with sprites 0-7 across the top, etc. The image is drawn using palette 1 which is black, light gray, blue, and red.

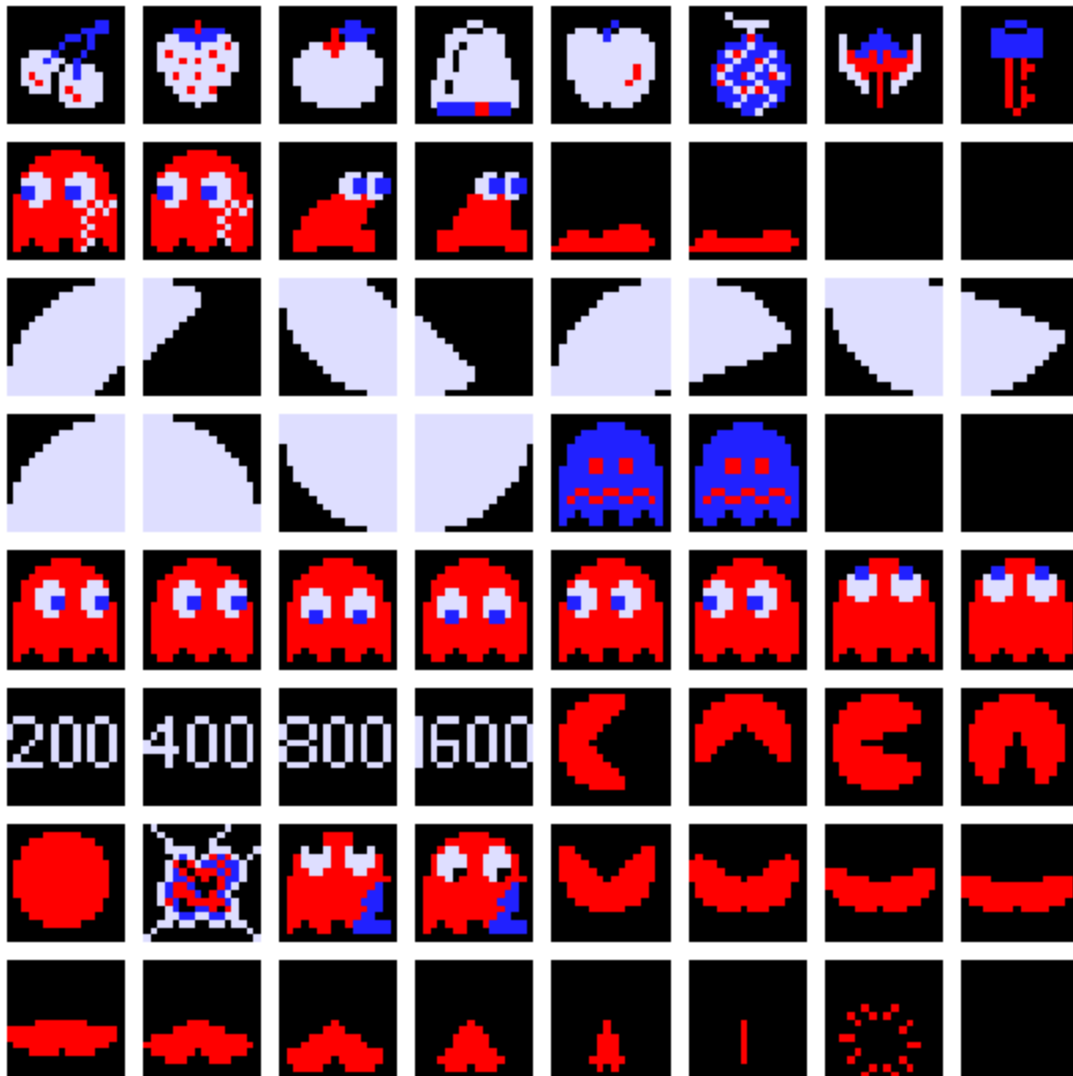


Figure 5 – Pac-Man sprite ROM

Decoding the sprites works similarly to decoding the tiles. Each byte represents 4 pixels in a column, stored low bits then high bits just like the tiles. Each 8 bytes draw a strip of 8x4 pixels, just like tiles. These 8 strips are then arranged

5	1
6	2
7	3
4	0

The complete pixel ordering is then⁷

⁷ TODO – check these orderings.

188	184	180	176	172	168	164	160	60	56	52	48	44	40	36	32
189	185	181	177	173	169	165	161	61	57	53	49	45	41	37	33
190	186	182	178	174	170	166	162	62	58	54	50	46	42	38	34
191	187	183	179	175	171	167	163	63	59	55	51	47	43	39	35
220	216	212	208	204	200	196	192	92	88	84	80	76	72	68	64
221	217	213	209	205	201	197	193	93	89	85	81	77	73	69	65
222	218	214	210	206	202	198	194	94	90	86	82	78	74	70	66
223	219	215	211	207	203	199	195	95	91	87	83	79	75	71	67
252	248	244	240	236	232	228	224	124	120	116	112	108	104	100	96
253	249	245	241	237	233	229	225	125	121	117	113	109	105	101	97
254	250	246	242	238	234	230	226	126	122	118	114	110	106	102	98
255	251	247	243	239	235	231	227	127	123	119	115	111	107	103	99
156	152	148	144	140	136	132	128	28	24	20	16	12	8	4	0
157	153	149	145	141	137	133	129	29	25	21	17	13	9	5	1
158	154	150	146	142	138	134	130	30	26	22	18	14	10	6	2
159	155	151	147	143	139	135	131	31	27	23	19	15	11	7	3

Sound ROMs

Sound is generated by playing waveforms stored in the two 256 byte Sound ROMs (82s126.1m and 82s126.3m). The sound hardware only takes 4-bit samples, so each byte entry has the top nibble set to 0. This gives in total 512 four-bit sound samples. These are organized into 16 waveforms, each 32 samples long, and each sample value from 0-15.

Ordering them from Sound ROM1 (82s126.1m) to Sound ROM2 (82s126.3m), the waveforms are shown in Figure 6.

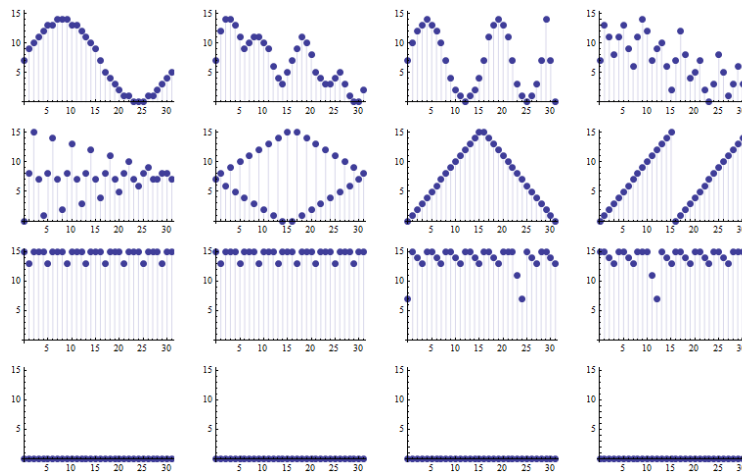


Figure 6 – Pac-Man Sound ROM waveforms

These are ordered 0-3 in the top row, left to right. The next row left to right is 4,5,6,7, etc. It seems the last four are not used, and the previous four may also be unused⁸.

Video

⁸ TODO – check this from the disassembly.

The video is drawn from two components: tiles and sprites. The hardware supports

- 8 sprites that are 16x16 pixel images,
- a 28 wide by 36 tall grid of tiles, each capable holding an 8x8 pixel image.

Thus the effective pixel resolution is 224x288 (28*8 = 224, 36*8=288). The tiles are used to draw a static background, and the sprites are used to draw moving items.

Layout

To understand how video is drawn, the mapping from memory address to screen locations is needed.

The following description works for both the 1K tile RAM 4000h-43FFh (each byte selects a tile to draw) and the 1K palette RAM 4400h-47FFh.

The first 64 addresses draw two rows, right to left, top to bottom, along the bottom of the screen, each row 32 tiles wide, with 4 of the tiles off-screen on both ends. Then the next 380h (896 decimal) addresses fill in the main screen, top to bottom, right to left, using offsets from 40h to 3BFh. This gives a 28x32 tile center region. Finally the last 64 addresses in each bank draw two more 32 tile wide rows along the top, right to left, top to bottom, again with 4 tiles off-screen. The off-screen locations could be used to hide unused sprites, which are always on⁹. The result is a 28x36 tile visible screen. With each tile 8x8 pixels, this gives the 224x288 pixel resolution.

For clarity examine Figure 7 and Figure 8 for offsets. Figure 9 shows an overlaid Pac-Man screen.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	3DF	3DE	3DD	3DC	3DB	3DA	3D9	3D8	3D7	3D6	3D5	3D4	3D3	3D2	3D1	3D0	3CF	3CE	3CD	3CC	3CB	3CA	3C9	3C8	3C7	3C6	3C5	3C4	3C3	3C2	3C1	3C0
1	3FF	3FE	3FD	3FC	3FB	3FA	3F9	3F8	3F7	3F6	3F5	3F4	3F3	3F2	3F1	3F0	3EF	3EE	3ED	3EC	3EB	3EA	3E9	3E8	3E7	3E6	3E5	3E4	3E3	3E2	3E1	3E0
2	3A0	380	360	340	320	300	2E0	2C0	2A0	280	260	240	220	200	1F0	1C0	1A0	180	160	140	120	100	0E0	0C0	0A0	080	060	040				
3	3A1	381	361	341	321	301	2E1	2C1	2A1	281	261	241	221	201	1E1	1C1	1A1	181	161	141	121	101	0E1	0C1	0A1	081	061	041				
4	3A2	382	362	342	322	302	2E2	2C2	2A2	282	262	242	222	202	1E2	1C2	1A2	182	162	142	122	102	0E2	0C2	0A2	082	062	042				
5	3A3	383	363	343	323	303	2E3	2C3	2A3	283	263	243	223	203	1E3	1C3	1A3	183	163	143	123	103	0E3	0C3	0A3	083	063	043				
6	3A4	384	364	344	324	304	2E4	2C4	2A4	284	264	244	224	204	1E4	1C4	1A4	184	164	144	124	104	0E4	0C4	0A4	084	064	044				
7	3A5	385	365	345	325	305	2E5	2C5	2A5	285	265	245	225	205	1E5	1C5	1A5	185	165	145	125	105	0E5	0C5	0A5	085	065	045				
8	3A6	386	366	346	326	306	2E6	2C6	2A6	286	266	246	226	206	1E6	1C6	1A6	186	166	146	126	106	0E6	0C6	0A6	086	066	046				
9	3A7	387	367	347	327	307	2E7	2C7	2A7	287	267	247	227	207	1E7	1C7	1A7	187	167	147	127	107	0E7	0C7	0A7	087	067	047				
10	3A8	388	368	348	328	308	2E8	2C8	2A8	288	268	248	228	208	1E8	1C8	1A8	188	168	148	128	108	0E8	0C8	0A8	088	068	048				
11	3A9	389	369	349	329	309	2E9	2C9	2A9	289	269	249	229	209	1E9	1C9	1A9	189	169	149	129	109	0E9	0C9	0A9	089	069	049				
12	3AA	38A	36A	34A	32A	30A	2EA	2CA	2AA	28A	26A	24A	22A	20A	1EA	1CA	1AA	18A	16A	14A	12A	10A	0EA	0CA	0AA	08A	06A	04A				
13	3AB	38B	36B	34B	32B	30B	2EB	2CB	2AB	28B	26B	24B	22B	20B	1EB	1CB	1AB	18B	16B	14B	12B	10B	0EB	0CB	0AB	08B	06B	04B				
14	3AC	38C	36C	34C	32C	30C	2EC	2CC	2AC	28C	26C	24C	22C	20C	1EC	1CC	1AC	18C	16C	14C	12C	10C	0EC	0CC	0AC	08C	06C	04C				
15	3AD	38D	36D	34D	32D	30D	2ED	2CD	2AD	28D	26D	24D	22D	20D	1ED	1CD	1AD	18D	16D	14D	12D	10D	0ED	0CD	0AD	08D	06D	04D				
16	3AE	38E	36E	34E	32E	30E	2EE	2CE	2AE	28E	26E	24E	22E	20E	1EE	1CE	1AE	18E	16E	14E	12E	10E	0EE	0CE	0AE	08E	06E	04E				
17	3AF	38F	36F	34F	32F	30F	2EF	2CF	2AF	28F	26F	24F	22F	20F	1EF	1CF	1AF	18F	16F	14F	12F	10F	0EF	0CF	0AF	08F	06F	04F				
18	3B0	390	370	350	330	310	2F0	2D0	2B0	290	270	250	230	210	1F0	1D0	1B0	190	170	150	130	110	0F0	0D0	0B0	090	070	050				
19	3B1	391	371	351	331	311	2F1	2D1	2B1	291	271	251	231	211	1F1	1D1	1B1	191	171	151	131	111	0F1	0D1	0B1	091	071	051				
20	3B2	392	372	352	332	312	2F2	2D2	2B2	292	272	252	232	212	1F2	1D2	1B2	192	172	152	132	112	0F2	0D2	0B2	092	072	052				
21	3B3	393	373	353	333	313	2F3	2D3	2B3	293	273	253	233	213	1F3	1D3	1B3	193	173	153	133	113	0F3	0D3	0B3	093	073	053				
22	3B4	394	374	354	334	314	2F4	2D4	2B4	294	274	254	234	214	1F4	1D4	1B4	194	174	154	134	114	0F4	0D4	0B4	094	074	054				
23	3B5	395	375	355	335	315	2F5	2D5	2B5	295	275	255	235	215	1F5	1D5	1B5	195	175	155	135	115	0F5	0D5	0B5	095	075	055				
24	3B6	396	376	356	336	316	2F6	2D6	2B6	296	276	256	236	216	1F6	1D6	1B6	196	176	156	136	116	0F6	0D6	0B6	096	076	056				
25	3B7	397	377	357	337	317	2F7	2D7	2B7	297	277	257	237	217	1F7	1D7	1B7	197	177	157	137	117	0F7	0D7	0B7	097	077	057				
26	3B8	398	378	358	338	318	2F8	2D8	2B8	298	278	258	238	218	1F8	1D8	1B8	198	178	158	138	118	0F8	0D8	0B8	098	078	058				
27	3B9	399	379	359	339	319	2F9	2D9	2B9	299	279	259	239	219	1F9	1D9	1B9	199	179	159	139	119	0F9	0D9	0B9	099	079	059				
28	3BA	39A	37A	35A	33A	31A	2FA	2DA	2BA	29A	27A	25A	23A	21A	1FA	1DA	1BA	19A	17A	15A	13A	11A	0FA	0DA	0BA	09A	07A	05A				
29	3BB	39B	37B	35B	33B	31B	2FB	2DB	2BB	29B	27B	25B	23B	21B	1FB	1DB	1BB	19B	17B	15B	13B	11B	0FB	0DB	0BB	09B	07B	05B				
30	3BC	39C	37C	35C	33C	31C	2FC	2DC	2BC	29C	27C	25C	23C	21C	1FC	1DC	1BC	19C	17C	15C	13C	11C	0FC	0DC	0BC	09C	07C	05C				
31	3BD	39D	37D	35D	33D	31D	2FD	2DD	2BD	29D	27D	25D	23D	21D	1FD	1DD	1BD	19D	17D	15D	13D	11D	0FD	0DD	0BD	09D	07D	05D				
32	3BE	39E	37E	35E	33E	31E	2FE	2DE	2BE	29E	27E	25E	23E	21E	1FE	1DE	1BE	19E	17E	15E	13E	11E	0FE	0DE	0BE	09E	07E	05E				
33	3BF	39F	37F	35F	33F	31F	2FF	2DF	2BF	29F	27F	25F	23F	21F	1FF	1DF	1BF	19F	17F	15F	13F	11F	0FF	0DF	0BF	09F	07F	05F				
34	01F	01E	01D	01C	01B	01A	019	018	017	016	015	014	013	012	011	010	00F	00E	00D	00C	00B	00A	009	008	007	006	005	004	003	002	001	000
35	03F	03E	03D	03C	03B	03A	039	038	037	036	035	034	033	032	031	030	02F	02E	02D	02C	02B	02A	029	028	027	026	025	024	023	022	021	020

Figure 7 – Pac-Man video address offsets

⁹ TODO – check this. Perhaps a blank sprite is drawn?

3DF	3DE	3DD	3DC	3DB	3DA	3D9	3D8	3D7	3D6	3D5	3D4	3D3	3D2	3D1	3D0	3CF	3CE	3CD	3CC	3CB	3CA	3C9	3C8	3C7	3C6	3C5	3C4	3C3	3C2	3C1	3C0
3FF	3FE	3FD	3FC	3FB	3FA	3F9	3F8	3F7	3F6	3F5	3F4	3F3	3F2	3F1	3F0	3EF	3EE	3ED	3EC	3EB	3EA	3E9	3E8	3E7	3E6	3E5	3E4	3E3	3E2	3E1	3E0
X	X	3A0	380	360	340	320	300	2E0	2C0	2A0	280	260	240	220	200	1E0	1C0	1A0	180	160	140	120	100	0E0	0C0	0A0	080	060	040	X	X
X	X	3A1	381	361	341	321	301	2E1	2C1	2A1	281	261	241	221	201	1E1	1C1	1A1	181	161	141	121	101	0E1	0C1	0A1	081	061	041	X	X
X	X	3A2	382	362	342	322	302	2E2	2C2	2A2	282	262	242	222	202	1E2	1C2	1A2	182	162	142	122	102	0E2	0C2	0A2	082	062	042	X	X
X	X	3A3	383	363	343	323	303	2E3	2C3	2A3	283	263	243	223	203	1E3	1C3	1A3	183	163	143	123	103	0E3	0C3	0A3	083	063	043	X	X
X	X	3A4	384	364	344	324	304	2E4	2C4	2A4	284	264	244	224	204	1E4	1C4	1A4	184	164	144	124	104	0E4	0C4	0A4	084	064	044	X	X
X	X	3A5	385	365	345	325	305	2E5	2C5	2A5	285	265	245	225	205	1E5	1C5	1A5	185	165	145	125	105	0E5	0C5	0A5	085	065	045	X	X
X	X	3A6	386	366	346	326	306	2E6	2C6	2A6	286	266	246	226	206	1E6	1C6	1A6	186	166	146	126	106	0E6	0C6	0A6	086	066	046	X	X
X	X	3A7	387	367	347	327	307	2E7	2C7	2A7	287	267	247	227	207	1E7	1C7	1A7	187	167	147	127	107	0E7	0C7	0A7	087	067	047	X	X
X	X	3A8	388	368	348	328	308	2E8	2C8	2A8	288	268	248	228	208	1E8	1C8	1A8	188	168	148	128	108	0E8	0C8	0A8	088	068	048	X	X
X	X	3A9	389	369	349	329	309	2E9	2C9	2A9	289	269	249	229	209	1E9	1C9	1A9	189	169	149	129	109	0E9	0C9	0A9	089	069	049	X	X
X	X	3AA	38A	36A	34A	32A	30A	2EA	2CA	2AA	28A	26A	24A	22A	20A	1EA	1CA	1AA	18A	16A	14A	12A	10A	0EA	0CA	0AA	08A	06A	04A	X	X
X	X	3AB	38B	36B	34B	32B	30B	2EB	2CB	2AB	28B	26B	24B	22B	20B	1EB	1CB	1AB	18B	16B	14B	12B	10B	0EB	0CB	0AB	08B	06B	04B	X	X
X	X	3AC	38C	36C	34C	32C	30C	2EC	2CC	2AC	28C	26C	24C	22C	20C	1EC	1CC	1AC	18C	16C	14C	12C	10C	0EC	0CC	0AC	08C	06C	04C	X	X
X	X	3AD	38D	36D	34D	32D	30D	2ED	2CD	2AD	28D	26D	24D	22D	20D	1ED	1CD	1AD	18D	16D	14D	12D	10D	0ED	0CD	0AD	08D	06D	04D	X	X
X	X	3AE	38E	36E	34E	32E	30E	2EE	2CE	2AE	28E	26E	24E	22E	20E	1EE	1CE	1AE	18E	16E	14E	12E	10E	0EE	0CE	0AE	08E	06E	04E	X	X
X	X	3AF	38F	36F	34F	32F	30F	2EF	2CF	2AF	28F	26F	24F	22F	20F	1EF	1CF	1AF	18F	16F	14F	12F	10F	0EF	0CF	0AF	08F	06F	04F	X	X
X	X	3B0	390	370	350	330	310	2F0	2D0	2B0	290	270	250	230	210	1F0	1D0	1B0	190	170	150	130	110	0F0	0D0	0B0	090	070	050	X	X
X	X	3B1	391	371	351	331	311	2F1	2D1	2B1	291	271	251	231	211	1F1	1D1	1B1	191	171	151	131	111	0F1	0D1	0B1	091	071	051	X	X
X	X	3B2	392	372	352	332	312	2F2	2D2	2B2	292	272	252	232	212	1F2	1D2	1B2	192	172	152	132	112	0F2	0D2	0B2	092	072	052	X	X
X	X	3B3	393	373	353	333	313	2F3	2D3	2B3	293	273	253	233	213	1F3	1D3	1B3	193	173	153	133	113	0F3	0D3	0B3	093	073	053	X	X
X	X	3B4	394	374	354	334	314	2F4	2D4	2B4	294	274	254	234	214	1F4	1D4	1B4	194	174	154	134	114	0F4	0D4	0B4	094	074	054	X	X
X	X	3B5	395	375	355	335	315	2F5	2D5	2B5	295	275	255	235	215	1F5	1D5	1B5	195	175	155	135	115	0F5	0D5	0B5	095	075	055	X	X
X	X	3B6	396	376	356	336	316	2F6	2D6	2B6	296	276	256	236	216	1F6	1D6	1B6	196	176	156	136	116	0F6	0D6	0B6	096	076	056	X	X
X	X	3B7	397	377	357	337	317	2F7	2D7	2B7	297	277	257	237	217	1F7	1D7	1B7	197	177	157	137	117	0F7	0D7	0B7	097	077	057	X	X
X	X	3B8	398	378	358	338	318	2F8	2D8	2B8	298	278	258	238	218	1F8	1D8	1B8	198	178	158	138	118	0F8	0D8	0B8	098	078	058	X	X
X	X	3B9	399	379	359	339	319	2F9	2D9	2B9	299	279	259	239	219	1F9	1D9	1B9	199	179	159	139	119	0F9	0D9	0B9	099	079	059	X	X
X	X	3BA	39A	37A	35A	33A	31A	2FA	2DA	2BA	29A	27A	25A	23A	21A	1FA	1DA	1BA	19A	17A	15A	13A	11A	0FA	0DA	0BA	09A	07A	05A	X	X
X	X	3BB	39B	37B	35B	33B	31B	2FB	2DB	2BB	29B	27B	25B	23B	21B	1FB	1DB	1BB	19B	17B	15B	13B	11B	0FB	0DB	0BB	09B	07B	05B	X	X
X	X	3BC	39C	37C	35C	33C	31C	2FC	2DC	2BC	29C	27C	25C	23C	21C	1FC	1DC	1BC	19C	17C	15C	13C	11C	0FC	0DC	0BC	09C	07C	05C	X	X
X	X	3BD	39D	37D	35D	33D	31D	2FD	2DD	2BD	29D	27D	25D	23D	21D	1FD	1DD	1BD	19D	17D	15D	13D	11D	0FD	0DD	0BD	09D	07D	05D	X	X
X	X	3BE	39E	37E	35E	33E	31E	2FE	2DE	2BE	29E	27E	25E	23E	21E	1FE	1DE	1BE	19E	17E	15E	13E	11E	0FE	0DE	0BE	09E	07E	05E	X	X
X	X	3BF	39F	37F	35F	33F	31F	2FF	2DF	2BF	29F	27F	25F	23F	21F	1FF	1DF	1BF	19F	17F	15F	13F	11F	0FF	0DF	0BF	09F	07F	05F	X	X
01F	01E	01D	01C	01B	01A	019	018	017	016	015	014	013	012	011	010	00F	00E	00D	00C	00B	00A	009	008	007	006	005	004	003	002	001	000
03F	03E	03D	03C	03B	03A	039	038	037	036	035	034	033	032	031	030	02F	02E	02D	02C	02B	02A	029	028	027	026	025	024	023	022	021	020

Figure 8 –Pac-Man Video addresses



Figure 9 – Pac-Man screen overlaid on video addresses

Tiles

Each byte in Video RAM entry from 4000h-43FFh (Tile RAM) causes a tile from the Tile ROM to be drawn. The corresponding byte in 4400h-47FFh (Palette RAM) selects a palette from the palette ROM to apply to the tile, and this palette in turn references colors defined in the color ROM.

Each tile is an 8x8 pixel image; the resulting grid is a 28x36 grid of tiles. The first two rows, 32 bytes each, are used to hold the fruit/credit/prize display at the screen bottom. The next 28 rows are rotated as explained above, and contain the maze and game play area. The top two rows contain the score and related information.

Sprites

The Pac-Man hardware supports 8 sprites; each is a 16x16 pixel image that can be drawn at any location. Each can be flipped in the x and/or y directions, reducing the amount of sprite images stored in ROM. For example, the left and right facing Pac-Man images are the same sprite with a different x-flip bit set. The cocktail mode Pac-Man games flip the screen for two-player modes by flipping sprites and reordering tiles.

The eight hardware sprites are programmed through addresses 4FF0h-4FFh and 5060h-506Fh, listed in Table 5. For each of the 8 sprites, the sprite image number 0-63, x-flip, y-flip, and palette are set through the addresses shown in Table 6.

Sprite	X-location	Y-location	Sprite#, flip-x, flip-y	Palette
0	5060h	5061h	4FF0h	4FF1h
1	5062h	5063h	4FF2h	4FF3h
2	5064h	5065h	4FF4h	4FF5h
3	5066h	5067h	4FF6h	4FF7h
4	5068h	5069h	4FF8h	4FF9h
5	506Ah	506Bh	4FFAh	4FFBh
6	506Ch	506Dh	4FFCh	4FFDh
7	506Eh	506Fh	4FFEh	4FFFh

Table 5 – Pac-Man sprite addresses

Byte 0								Byte 1							
7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Sprite # 0-63						X flip	Y flip	Palette entry 0-15							

Table 6 – Sprite register definition

Each sprite can thus select a different palette. Color 0 in sprite images are drawn as transparent, allowing the background tiles to show through. Setting the X and Y locations allows the sprites to be positioned anywhere on the screen with pixel accuracy. They're positioned offscreen when not needed. Sprites are displayed in reverse order; higher numbered overlapped sprites are overwritten by lower numbered sprites.

The X and Y locations denote the upper left corner of the sprite image. The screen coordinates start on the lower right corner of the screen. Thus, including hidden coordinates, the lowest coordinate pair where a 16x16 sprite is completely visible is location (31,16). If drawn lower, it wraps to the top of the screen. The sprite does not show up on the top or bottom two rows where the video is laid out differently, as explained the section on [Video Layout](#). Visible pixel coordinates are in Figure 10.

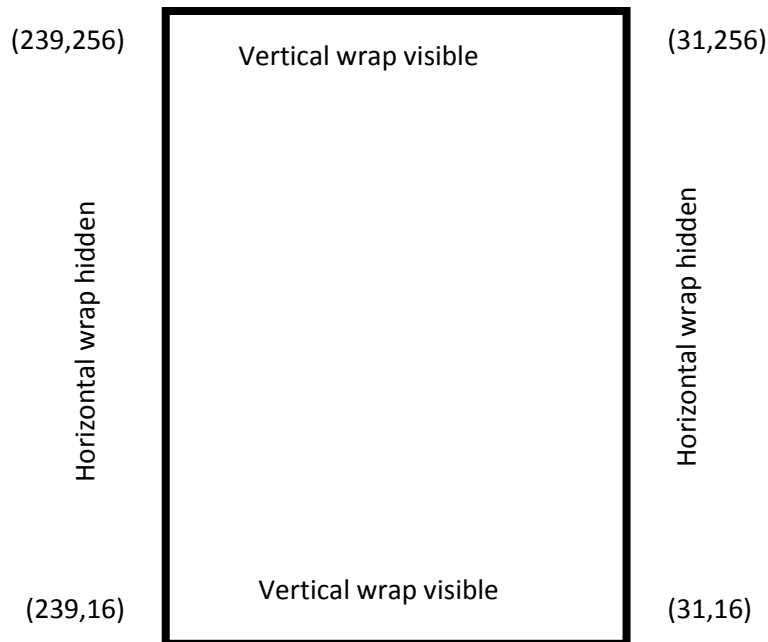


Figure 10 – Screen coordinates

Sound

The sound chip is a custom-made 3-channel Waveform Sound Generator (WSG). It can play three simultaneous voices each with independent volume, frequency, and waveform. It is accessed using 4-bit memory-mapped registers, whose addresses are in Table 7. The WSG runs at main CPU clock/32, i.e., $3.072 \text{ MHz}/32 = 96 \text{ kHz}$.

Register	Address	Notes
Voice 1 Waveform	5045h	low 3 bits used – selects waveform 0-7 from ROM
Voice 1 Frequency	5050h-5054h	20 bits in low nibbles
Voice 1 Volume	5055h	low nibble – 0 off to 15 loudest
Voice 1 Accumulator	5040h-5044h	low nibbles, used by H/W only
Voice 2 Waveform	504Ah	low 3 bits used – selects waveform 0-7 from ROM
Voice 2 Frequency	5056h-5059h	16 bits in low nibbles
Voice 2 Volume	505Ah	low nibble – 0 off to 15 loudest
Voice 2 Accumulator	5046h-5049h	low nibbles, used by H/W only
Voice 3 Waveform	504Fh	low 3 bits used – selects waveform 0-7 from ROM
Voice 3 Frequency	505Bh-505Eh	16 bits in low nibbles
Voice 3 Volume	505Fh	low nibble – 0 off to 15 loudest
Voice 3 Accumulator	504Bh-504Eh	low nibbles, used by H/W only

Table 7 – Pac-Man sound registers.

Each register disregards the top nibble in each byte. The hardware supports 3 monophonic voices, each with 16 volume levels and an independent choice of 8 pre-defined waveforms. One voice has 20 bits of frequency choice, while the other two have 16 bits of frequency.

At each clock and for each voice the hardware performs the following steps:

1. Add the 20-bit (or 16-bit) voice frequency to the 20-bit (or 16-bit) accumulator.
2. Use the waveform 0-7 to lookup a 32-byte sample in the Sound ROM.
3. Take the top 5 bits of the accumulator to look up a nibble 0-31 in that sample.
4. Multiply that nibble by the volume nibble 0-15.
5. Send the result to the amplifier for output.

As a result, a voice is silent if the volume is zero.

Frequencies and counters are 20-bit or 16-bit values, depending on voice, and are stored in the least significant nibble of 5 (or 4) bytes, with the least significant nibble in the lowest address. Voices #2 and #3 do not implement the lowest nibble, which is always 0. Thus they have slightly lower dynamic range than voice #1, which is used in the code for **TODO**.

Waveforms

As explained in the Sound ROM section, samples are stored as 32 entries of 4-bits each. They can be selected by setting 0-7 in the appropriate voice.

Frequency

For each voice there are 4 registers controlling the tone. Voice #1 adds a 5th register allowing very low frequencies.

How does this convert to notes for a musical scale? Using scientific pitch notation¹⁰ to denote middle C as C₄, then the first A higher in pitch is known as A₄₄₀ (or as A₄) since it is 440 Hz. Here's a computation of the values to put in the frequency register to get a 440 Hz tone for Voice #1.

To play the A₄ tone (440 Hz) use sample 0, which is a pure sine tone and has one cycle per sample. If a sample has two cycles per sample this example would need changed accordingly.

Starting with the 96 kHz accumulator rate, sample 0 produces a complete sound cycle every 32 samples. To get 440 Hz, $440 \times 32 = 14080$ samples are needed per second. 96000 clocks per second / 14080 samples per second = $75/11$ clocks per sample. Since the top 5 bits of 20 select the sample, there are $20 - 5 = 15$ bits representing "fractional" values. $2^{15} / (75/11) = 4806 = 12C6h$. This needs stored in the frequency of a voice to use sample 0 for a note with A₄ = 440 Hz frequency.

Thus to get the register value V for frequency f using sample 0 use the formula

$$V = \frac{f \times 32}{96000} \times 2^{15} = \frac{4096}{375} f$$

¹⁰ http://en.wikipedia.org/wiki/Scientific_pitch_notation

Each octave is half/double the frequency of the one above/below it, so this gives counts for all octaves of A. Since there are 12 half-steps per octave, to go one half-step higher simple multiply the value by $\sqrt[12]{2}$. C₃ is 3 half-steps up, so has frequency $440 \times (\sqrt[12]{2})^3 = 523.251$ Hz. This then corresponds to $V = 5715 = 1653h$.

Filling a note table gives Table 8.

	Octave									
Note	0	1	2	3	4	5	6	7	8	9
C	B3h	165h	2CAh	595h	B2Ah	1653h	2CA7h	594Dh	B29Ah	16535h
C#/Db	BDh	17Ah	2F5h	5EAh	BD4h	17A7h	2F4Eh	5E9Dh	BD39h	17A72h
D	C8h	191h	322h	644h	C88h	190Fh	321Eh	643Dh	C87Ah	190F3h
D#/Eb	D4h	1A9h	352h	6A3h	D46h	1A8Dh	3519h	6A33h	D465h	1A8CBh
E	E1h	1C2h	384h	708h	E10h	1C21h	3842h	7083h	E107h	1C20Dh
F	EEh	1DDh	3BAh	773h	EE7h	1DCDh	3B9Ah	7734h	EE68h	1DCD0h
F#/Gb	FDh	1F9h	3F2h	7E5h	FC9h	1F93h	3F25h	7E4Bh	FC95h	1F92Ah
G	10Ch	217h	42Eh	85Dh	10BAh	2173h	42E7h	85CDh	10B9Ah	21734h
G#/Ab	11Ch	237h	46Eh	8DCh	11B8h	2370h	46E1h	8DC2h	11B84h	23708h
A	12Ch	259h	4B1h	963h	12C6h	258Ch	4B18h	9630h	12C60h	258BFh
A#/Bb	13Eh	27Ch	4F9h	9F2h	13E4h	27C8h	4F8Fh	9F1Eh	13E3Ch	27C78h
B	151h	2A2h	545h	A89h	1513h	2A25h	544Ah	A894h	15128h	2A251h

Table 8 – Note frequency chart

Note that this is for the 5 register voices. For the 4 register voice round the low nibble up; this will result in tones being slightly off.

Volume

Per-voice volume is a straightforward value 0-15, with 0 being muted to 15 being loudest.

Accumulator

The per-voice accumulator is used internally to select Sound ROM samples at some program defined frequency. It is not clear if these can be read (or set?) from the code (**TODO – find out**).

Sound Enable Register

Register 50C0h is the sound enable register, enabling or disabling all voices simultaneously. Write 0 to bit 0 of this register to disable sound and write 1 to bit 0 to enable sound.

Code

Sound register usage can be studied from the Pac-Man ROM sound routines through disassembly (if you know how, can read Z80, etc.). The main sound code is at addresses 2CC1h-2F54h, the interrupt portion is in two sections at 009Dh-00D2h (VBLANK1?) and 01B9h-01BCh (VBLANK2). Sound tables are at 3B30h-3CDDh. Details are at http://www.vecoven.com/elec/pacman/code/pac_z80.html.

Some routine highlights:

The software updates the sound registers 60 times a second during the VBLANK. Each voice plays an effect or a song.

An effect is encoded with eight bytes, and can be played on any voice.

0. Upper 3 bits are frequency shift, lower 3 bits are sample select.
1. Initial base frequency
2. Frequency increment
3. Upper bit causes reversal, lower 7 bits is duration.
4. Frequency increment added to base when repeat > 1
5. Repeat
6. Upper 4 bits denote volume adjust type, lower 4 bits is volume
7. Volume increment.

A song is a list of special bytes and tone information, and is played on voices 1 and 2 simultaneously.

A regular byte contains

- Upper 3 bits – log base 2 of duration
- Lower 4 bits – base frequency using lookup table
- Sometimes the lower 5 bits are assigned to W_DIR where the 5th bit has significance?!

Special song bytes include

- F0h followed by 2 bytes : address where song continues (jump in ROM)
- F1h followed by 1 byte : wave select
- F2h followed by one byte: frequency increment
- F3 followed by one byte: volume
- F4h followed by 1 byte : type
- FFh : end of song.

Interrupts

Interrupt Enable

Address 5000h is the interrupt enable register. Setting bit 0 to 0 disabled vertical blank (VBLANK) interrupts, and setting bit 0 to 1 enables VBLANK interrupts.

VBLANK

This is the vertical blank signal, which is triggered each time the electron gun scans to the bottom of the screen while drawing an image. This happens 60.61 times a second, which is every 16.67 milliseconds. If interrupts are enabled, the Z80 CPU is sent an interrupt signal. Depending on the interrupt mode set in the code different actions are taken.

Triggered by the VBLANK, the Z80 refreshes the sound frequencies, volumes, and waveforms.

In order for interrupts to work Port 0 has to behave properly.

Port 0

Port 0 from the CPU sets the interrupt vector. The code occasionally writes a byte to port 0, which has to be returned when a VBLANK interrupt is triggered.

The Z80 in Pac-Man code runs in Interrupt mode 2 (using the Z80 opcode IM 2). The Z80 instructions LD A,#CD; OUT (0),A for example set the interrupt address to I*256+CDh where I is the interrupt register.

Registers

Here are various memory-mapped registers not explained elsewhere, their uses and addresses.

Most of the control lines are normally closed, that is, the bit will be 1 when the switch is not pressed. Thus, for example, the joystick up button bit will read 1 when the joystick is not pressed up and will read 0 when pressed up. This is true for the coin slots also.

Also there are a few sets of registers where each byte in a range has the same function. They are marked in the [memory map](#).

Watchdog Timer

Lots of hardware/software systems have a “watchdog timer,” and Pac-Man is no different. A watchdog timer is a timer that the running software has to “kick” within a certain time period, or the watchdog resets the system. This is so if the software hangs, the system is restarted by the watchdog. Vernacular is to “kick the dog” when sending notification to the watchdog timer from software.

The watchdog timer at addresses 50C0h-50FFh (all bytes are the same function) constantly counts upwards, resetting the machine when it reaches its limit. It counts each VLBANK on a four bit counter (the 74LS161 at 9c on the Pac-Man board), so when it reaches 16 the carry from the counter will reset the machine. This means the code has to kick the dog faster than about 4 times per second. Precisely it needs reset before 16 frames of game elapse, at 60 frames per second. The Pac-Man code has relevant instructions are scattered throughout the ROMs to kick the dog, about 3 times per frame, mostly with zeroes, but occasionally (TODO) with a decreasing counter (TODO – why?).

Flip Screen

Flipped screen (write only byte at 5003h) is used in cocktail mode to flip the screen for the second player, which *flips the tiles only*. The sprites are flipped by the game code. Bit 0 set to 0 creates a normal screen; bit 0 set to 1 flips the screen.

IN0/IN1

The **IN0** port (read-only byte at address 5000h) is connected to the joystick and coin slots. The credit button works like a coin slots but adds a credit without incrementing the coin counter. The **IN1** port

(read-only byte at address 5040h) is connected to the second player joystick (only on a cocktail table version) and the start buttons.

Bit	Description of port IN0	Description of port IN1
0	Joystick up (0=preserved, 1 = released)	2 nd player joystick up (0=preserved, 1 = released)*
1	Joystick left (0=preserved, 1 = released)	2 nd player joystick left (0=preserved, 1 = released)*
2	Joystick right (0=preserved, 1 = released)	2 nd player joystick right (0=preserved, 1 = released)*
3	Joystick down (0=preserved, 1 = released)	2 nd player joystick down (0=preserved, 1 = released)*
4	Rack advance (switch, automatically advance to the next level, 0=on, 1 = off)	Board test (switch: 0 = on, 1 = off)
5	Coin slot 1 (trigger by changing from a 0 to 1)	One player start button (0=preserved, 1=released)
6	Coin slot 2 (trigger by changing from a 0 to 1)	Two player start button(0=preserved, 1=released)
7	Credit button (0 = preserved, 1 = released)	Cabinet mode (0=table, 1 = upright)

Table 9 – Input ports

* cocktail table only.

Dip Switches

Memory addresses 5080h-50BFh are memory mapped to a set of 8 DIP switches on the main board. Each byte is the same value, and is read only to read hardware settings on the board. On the hardware these are 8 small switches, with “OFF” setting the bit to 0, and “On” setting the bit to 1.

Bit	DIP Switch Description	Values
0,1	Coins per game:	0=free play 1=1 coin per game 2=1 coin per 2 games 3=2 coins per game
2,3	# lives per game:	0=1 life 1=2 lives 2=3 lives 3=5 lives
4,5	Bonus score for extra life:	0=10000 points 1=15000 points 2=20000 points 3=none
6	Difficulty (jumper pad):	0 = hard 1 = normal
7	Ghost names(jumper pad):	0=alternate 1 = normal

Table 10 – Dip Switch Settings

The difficulty switch changes the algorithm the ghosts use, with the effect of making less places Pac-Man can hide. There may be other changes also (**TODO – find out?**). Setting it breaks many patterns used in gameplay. Default and alternate ghost names are in Table 11.





Color	Image	Default name (nickname)	Alternate name (nickname)
Red		Shadow (Blinky)	AAAAAAAAA(BBBBBBBBB)
Pink		Speedy (Pinky)	CCCCCCCC (DDDDDDDD)
Blue		Bashful (Inky)	EEEEEEEE (FFFFFFF)
Orange		Pokey (Clyde)	GGGGGGG (HHHHHHH)

Table 11 – Ghost names

Start Lamps

Pac-Man does not use start lamps accessed from registers 5004h (1 player) and 5005h (2 player).

Coin Lockout

Coin lockout mechanisms are hardware designed to keep more coins from being added during game play, so that software does not have to check for new coins all the time. Pac-Man does not have this hardware, so this is unused?! The Pac-Man code must watch for coin drops.

Coin Counters

Register 5007h triggers coin counting by triggering bit 0 from value 0 to 1.

Miscellaneous

Restart

The CPU starts executing at ROM address 0000h on restart.

ROM Checksum

ROM modifications are detected using a simple checksum routine at address 3000h, shown in Figure 11. It sums all the bytes in each of the 4K banks, and the sum of the even bytes must be 0 and the sum of the odd bytes must be 0. The last two bytes (**TODO – right?**) in each of the four ROMs are set to values accomplishing this.

```

ROM:3000 CheckROM:
ROM:3000      ld      hl, 0
ROM:3003      loc_3003:
ROM:3003      ; CODE XREF: CheckROM+22↓j
ROM:3003      ; CheckROM+2B↓j
ROM:3003      ld      bc, 1000h ; we will walk one of the 4K ROMS
ROM:3006      loc_3006:
ROM:3006      ; CODE XREF: CheckROM+16↓j
ROM:3006      ld      (WatchdogReset), a
ROM:3009      loc_3009:
ROM:3009      ; CODE XREF: CheckROM+12↓j
ROM:3009      ld      a, c
ROM:300A      add     a, (hl) ; add data byte to checksum
ROM:300B      ld      c, a ; save it
ROM:300C      ld      a, 1 ; add 2 to data pointer
ROM:300D      add     a, 2
ROM:300E      ld      l, a
ROM:3010      cp      2 ; did it roll over?
ROM:3012      jp      nc, loc_3009 ; if not then loop
ROM:3015      inc     h ; next page of 256 bytes
ROM:3016      djnz   loc_3006
ROM:3018      ld      a, c
ROM:3019      and     a ; checksum must come to 0 else fail
ROM:301A      jr      nz, ChecksumFail ; Jump if checksum failed
ROM:301C      ld      (CoinCounter), a ; reset coin counter
ROM:301F      ld      a, h
ROM:3020      cp      30h ; '0' ; checksum other ROMs
ROM:3022      jp      nz, loc_3003 ; we will walk one of the 4K ROMS
ROM:3025      ld      h, 0 ; Prepare to check odd bytes
ROM:3027      inc     l ; now looking at odd bytes
ROM:3028      ld      a, 1
ROM:3029      cp      2 ; if l is 2, we're done with even and odd byte checksums
ROM:302B      jp      c, loc_3003 ; we will walk one of the 4K ROMS
ROM:302E      jp      ROMsPassed ; ROMs have passed checksums
ROM:3031 ; -----

```

Figure 11 – ROM Checksums

Copy Protection

Some claim that the port 0 byte stored as explained in the Interrupt section is used for copy protection, making it harder for bootleggers. However this is trivial to implement, and it seems more likely (without further documentation) that this feature merely makes programming interrupts easier and more flexible.

There is a false IM 1 handler at location 0038h which disables external interrupts (5000h), clears the coin counter at 5007h, then infinite loops. The watchdog timer restarts the machine soon thereafter.

Incorrect implementations of the port 0 hardware will cause trouble. However port 0 behavior might be designed to ease testing and troubleshooting of the hardware.

Tests

Turning on the game board test switch does diagnostics, shown in Figure 12. The first image shows bad video ram, the center shows a passed test and DIP Switch settings, and the final one shows a grid for testing the screen image. Different sounds are played on joystick movements, start buttons, and coin slot toggles.

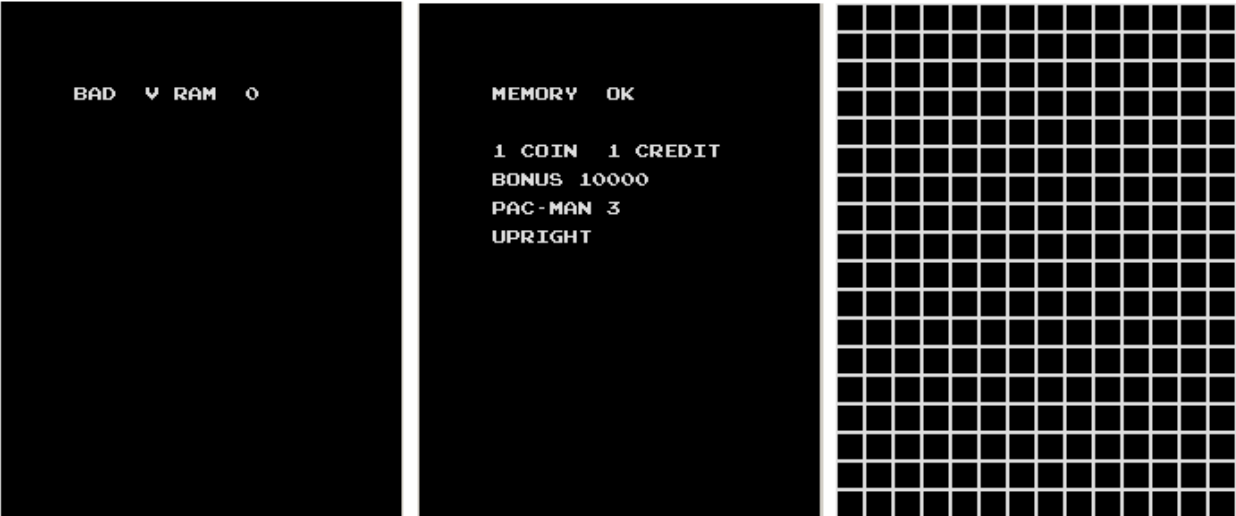


Figure 12 – Testing screens.

Easter Egg

There is an Easter Egg found in both the Midway and Namco ROMs. Hold down both player start buttons and toggle the test switch to pull up the test grid. Then push the joystick 4 times up, then 4 times left, then 4 times right then 4 times down. The result screen is in Figure 13.

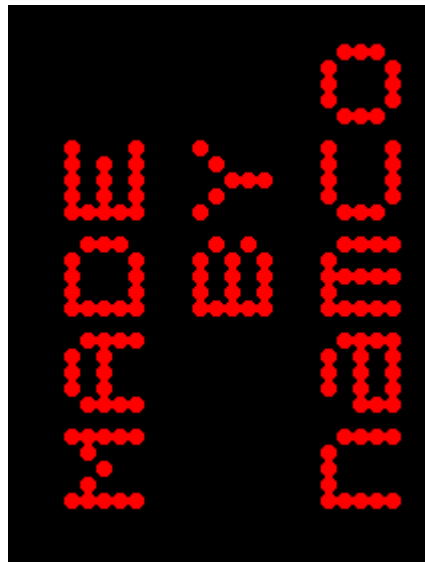


Figure 13 – Easter Egg

Gameplay

TODO?!

Patterns

Since all ghost actions are determined by Pac-Man location and actions, patterns can be devised to pass boards. They are often named based on the bonus item, such as “the Cherry Pattern” and “the 9th Key Pattern.” Find them on the internet.

Bugs

Split-Screen

There is a bug in the fruit drawing routine, which causes board 256 to be a garbled mess (Figure 14) that so far has been impassible despite enormous effort expended to conquer it. As demonstrated by Don Hodges in his article “Splitting Apart the Split Screen,” it is caused by an error in the code. A counter overflows on board 256, and the fruit-drawing routine (address 2B8Fh) trashes video memory by trying to draw 256 fruits. This results in not enough dots on the screen for Pac-Man to eat and pass the level. Don used a ROM disassembly to create a 9-byte in-place fix, which allows passing the level. At first glance it seems his fix does not alter the ghost behavior, but will require more checking.



Figure 14 – Split screen bug on level 256

Ghost pass through

A bug occasionally allows Pac-Man to pass through a non-blue ghost unharmed. Several patterns exploit this bug, and some can be seen on YouTube. An excellent one for 9th key patterns is shown in Figure 15. See it on YouTube at <http://www.youtube.com/watch?v=s82nzkZBaek&feature=related>.

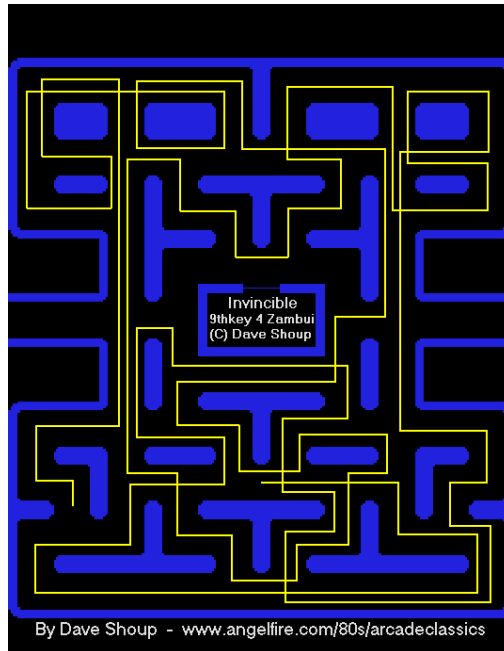


Figure 15 – Pac-Man passes through ghosts.

This bug is caused by the collision detection being done on an 8x8 grid, and when the Pac-Man enters a ghost grid at the exact time slice the ghost enters the Pac-Man cell they will pass without contact.

Hiding places

Not quite a bug, there are patterns resulting in places you can leave Pac-Man and ghosts will never find him. This is useful for breaks during marathon gaming sessions. See this YouTube pattern for an example <http://www.youtube.com/watch?v=S9vICsjF3zU&feature=related>.

When the ghosts are not closely pursuing Pac-Man, he can go to the spot immediately to the right and up from his starting location, facing upwards. The ghosts will never find him, and will continue to wander the board.

Up Direction

The MAME sourcecode states there is a minor bug in the up direction of Pac-Man animation: the circle animation is 1 pixel too low.

Fixes

TODO – later ROM versions have fixed some of these items?

ROM Sets

There are various ROM sets in the wild, and different difficulty levels. A good way to determine which set you have is the location Pac-Man dies in the attract mode animation.

There were three chip sets in the US. Each has two modes, easy and hard (or slow and fast as some call them). The speed can be as high as 8x factory “slow” speeds.

The three chipsets are the “original” one, the Atlantic one, and the rare Manhattan chipset. Patterns are different on each one, and most patterns on the internet are for the original ROMs.

ROM set	Slow death location	Fast death location	Comment
Original	Blue on square above lower left energizer.	Orange at top of lower right turn on board.	
Atlantic	Pinky kills Pac-Man right above ghost pen	Lower right quadrant, just left of the energizer. No energizers get eaten.	Allows fast (42 second) 9 th key patterns.
Manhattan	???? TODO	???? TODO	

Table 12 – ROM Versions

The Atlantic City chipset was developed to remove the hiding spot, but failed. Supposedly it allows a player to “kill” a ghost by eating the ghost and an energizer at the same, resulting in that ghost not coming back for the rest of that level.

Algorithms

Random numbers

TODO – explain the pseudo random number generator (PRNG) in AI.

Ghost Movement

Probably the most interesting algorithm is the ghost movement algorithm. This has been discussed online, and there is disassembly of the algorithms available.

The first point is there are “scatter” and “gather” periods. The ghosts start a level in “gather” mode, trying to reach Pac-Man. After a long time, they go to scatter mode, trying to avoid Pac-Man, and each going to a different quadrant of the board. Gather/scatter repeats a few times, is time based, and eventually ghosts return only to gather mode only.

The rough gather algorithms are:

Red: The only input is Pac-Man's location. Red attempts to reduce the distance between himself and Pac-Man, by turning in the direction that will immediately reduce the greater of the horizontal or vertical distance between himself and Pac-Man.

Pink: The input is four tiles in front of Pac-Man, which is then Pinky's "target." Pinky uses the same logic as Red to reach this location. Thus reversing on Pinky will chase him away.

Blue: the inputs are the square two tiles in front of Pac-Man and the location of the Red ghost. Draw a line from the Red ghost to the spot 2 tiles in front of Pac-Man, and extend that line twice as far in the same direction. Blue's target is that spot. Figure 16 shows Blue AI code disassembled.

Orange: Orange behaves differently is he is "near" (within 8 tiles of Pac-Man) or "far" (farther away than 8 tiles). “Far” mode duplicates Red's logic. “Near” mode causes Orange to head to his corner of the

maze, lower left, where he circles. So if Pac-Man is in the lower left area quadrant, Orange will hassle Pac-Man.

Edible ghost: The edible ghost uses a pseudo random number generator (PRNG) that creates a random address from 0000h to 1FFFh, which is used to look up a byte, reading the last bits. These bits determine the direction choices. This PRNG is reset at each level or life.

A small change to the code, such as never resetting this PRNG, would make Pac-Man level patterns impossible, since every game would start with an unknown state.

```
; Blue AI determination - find line
; from red to 2 spaces in front of
; Pac-Man and extend this to 2x that
; length. This is Blue's target spot

BlueAI:
ld    bc, (RedPosition)
ld    de, (PacPosition) ; get Pac-Man position
ld    hl, (PacOffset) ; get Pac-Man offset
add   hl, hl
add   hl, de           ; hl = 2hl+de = 2 spaces in front of Pac-Man
ld    a, l
add   a, a
sub   c
ld    l, a
ld    a, h
add   a, a
sub   b
ld    h, a           ; hl = hl + hl - red ghost
ex   de, hl
ld    hl, (word_4D0E)
ld    a, (BlueDirection) ; current direction
call  sub_2966       ; get best direction given de as destination
ld    (BlueOffset), hl ; save new choices
ld    (BlueDirection), a
ret
; End of function BlueAI
```

Figure 16 – Blue AI code

Notes

- Each dot resides in its own tile so 1 tile equals the distance from one dot to the next.
- Pac-Man can turn corners faster than the ghosts can, so turning corners helps escape ghosts.
- Eating dots slows Pac-Man a little bit, so eating dots allows ghosts to catch up.
- The ghosts never go upwards into the two passages immediately above the pen.

Sources

Pac-Man

1. Scott Lawrence, “Programming For Pac-Man And Pengo Hardware”
<http://www.cis.rit.edu/~jerry/Software/pengo>
2. Alessandro Scotti, Pac-Man Instructional Emulator (PIE),
<http://www.ascotti.org/programming/pie/pie.htm>
3. Scott Lawrence, “Programming for the Pac-Man or Pengo arcade platforms”,
<http://umlautllama.com/projects/pacdocs/#basics>

4. Frederic Vecoven, Pac-Man sound investigation, <http://www.vecoven.com/elec/pacman/pacman.html>
5. Simon Owen, "Pac-Man Emulator," <http://simonowen.com/sam/articles/pacemu/>
6. Don Hodges, "Splitting Apart the Split Screen," http://donhodes.com/how_high_can_you_get2.htm
7. Pac-Man from Wikipedia, <http://en.wikipedia.org/wiki/Pac-Man>
8. **TODO** <http://www.widel.com/>
9. **TODO** <http://www.mamedb.com/game/pacman>
10. **TODO**
<http://www.twingalaxies.com/forums/viewtopic.php?t=7224&highlight=&sid=5a5d90127cbd9233d6f4f110d7333d77> .
11. **TODO** Algorithm: <http://www.twingalaxies.com/forums/viewtopic.php?t=12231> , decoded by TwinGalaxies <http://www.atariage.com/forums/index.php?showtopic=68707&st=0>

Z80

1. Sean Young and Jan, "The Undocumented Z80 Documented," <http://www.myquest.nl/z80undocumented/>
2. "Decoding Z80 Opcodes," <http://www.z80.info/decoding.htm>
3. "Z80 Technical Information," http://www.tjornov.dk/spectrum/fag/tech_z80.html
4. Thomas Scherer, "Home of the Z80 CPU," <http://www.geocities.com/SiliconValley/Peaks/3938/>

THE END