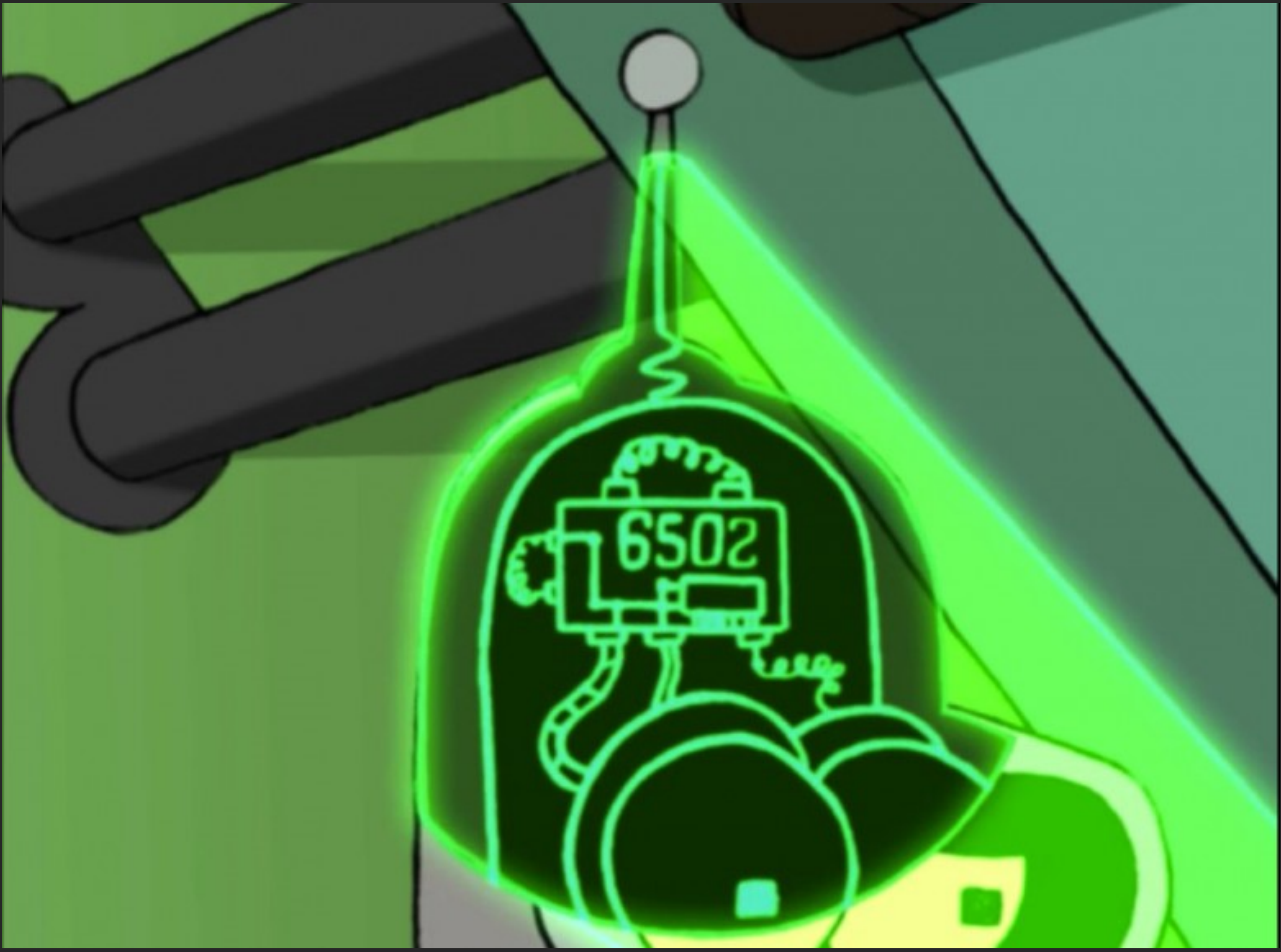
Breaking NES Book

6502 Core



A book on how the MOS 6502 processor works, but basically just a copy of the wiki from the GitHub.

Translated with www.DeepL.com/Translator (free version)

Rev. A2

Foreword

What to say...

Apparently it is time to summarize all the research on the 6502 processor in the form of a book.

The necessity to write a book arose because of the need to fill up the wiki on the Breaking NES project to the end, but it needs additional motivation. The book is a good way :smiley:

Another need is that as of now (2021) there are still no 6502 emulators that replicate exactly all of its operations, especially the so-called undocumented instructions (which are really just Undefined Behavior of its operation).

Contents

Decoder

The decoder 6502 is an ordinary demultiplexer, but a very large one. The formula for the demultiplexer is 21-to-130. Sometimes the 6502 instruction decoder is also called a PLA.

Topologically, the decoder is divided by ground lines into several groups, so we'll stick to the same division, for convenience.

The input lines are:

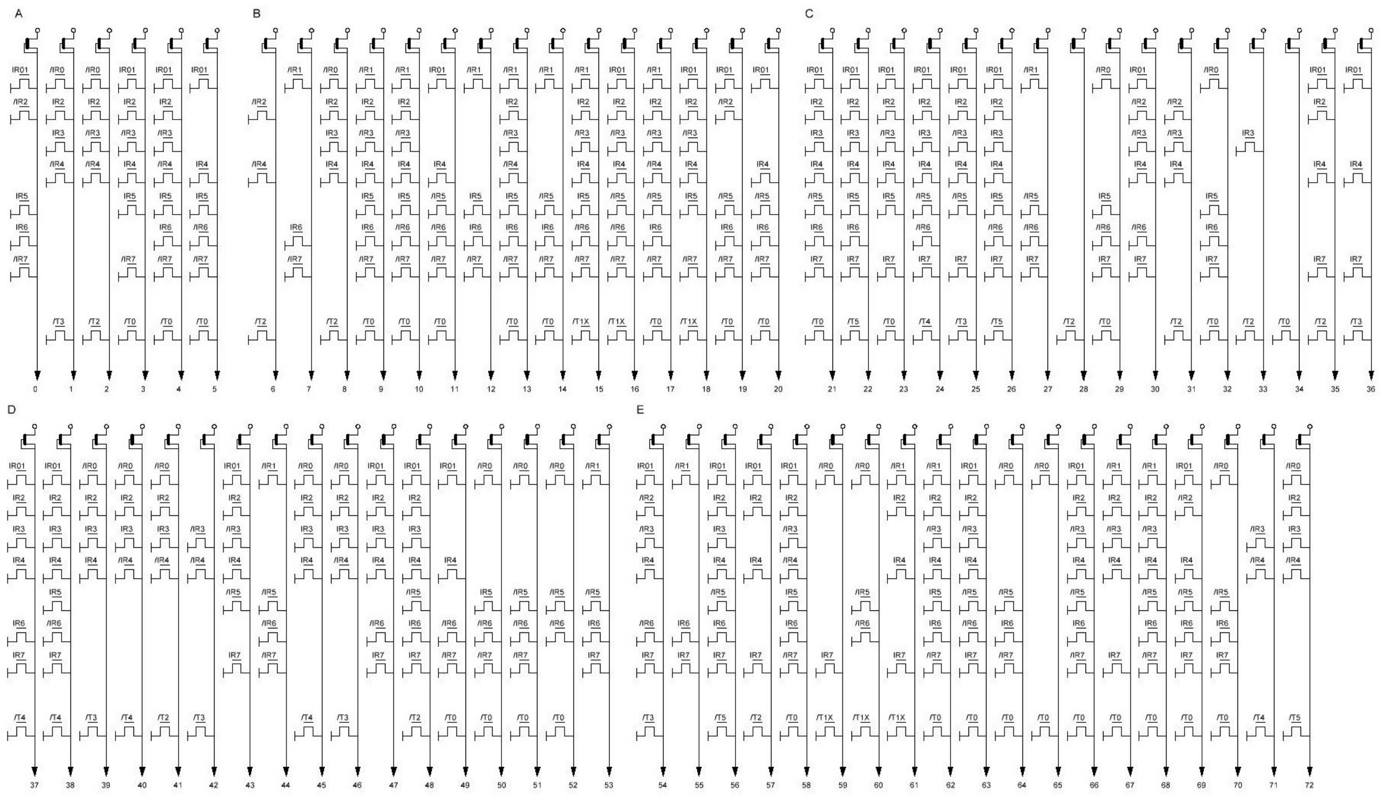
- /T0, /T1X: current cycle for short (2 clock) instructions. These signals are output from *dispatch logic*.
- /T2, /T3, /T4, /T5: current cycle for long instructions. Signals are output from *extended cycle counter*.

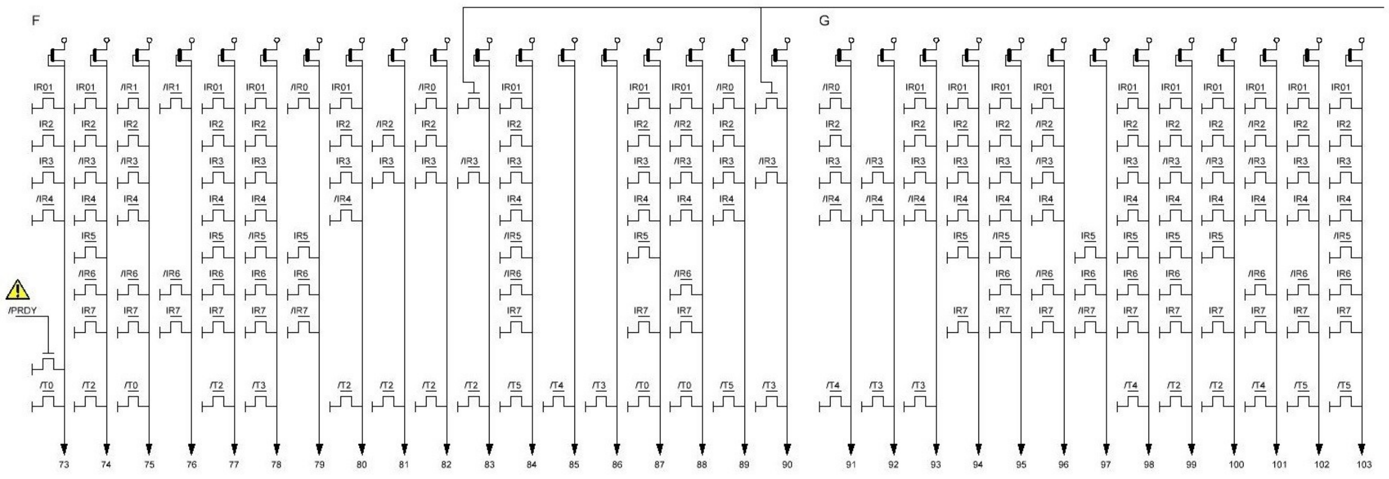
- /IR0, /IR1, IR01: the lower bits of the operation code from *instruction register*. To reduce the number of lines 0 and 1 bits are combined into one control line `IR01`.

- IR2-IR7, /IR2-/IR7: direct and inverse values of the remaining bits. The direct and inverse forms are needed to check the bit for 0 and 1.

The decoder logic is based on the exclusion principle. Schematically, each output is a multi-input NOR element, which means that if at least one of the inputs has a 1, the whole line will NOT work.

That is, the decoder outputs are not in inverse logic (as is usual), but in direct logic.





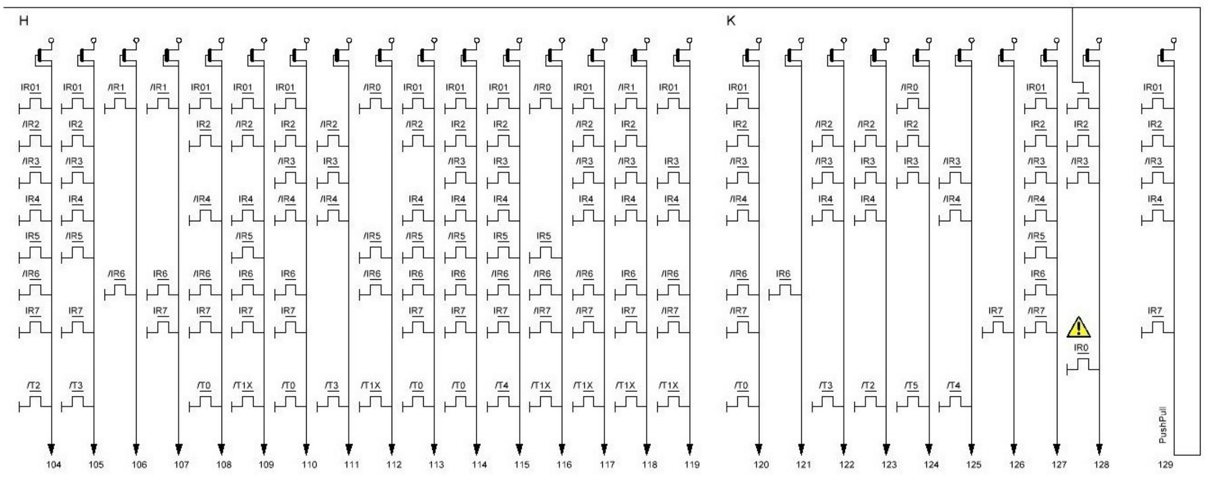
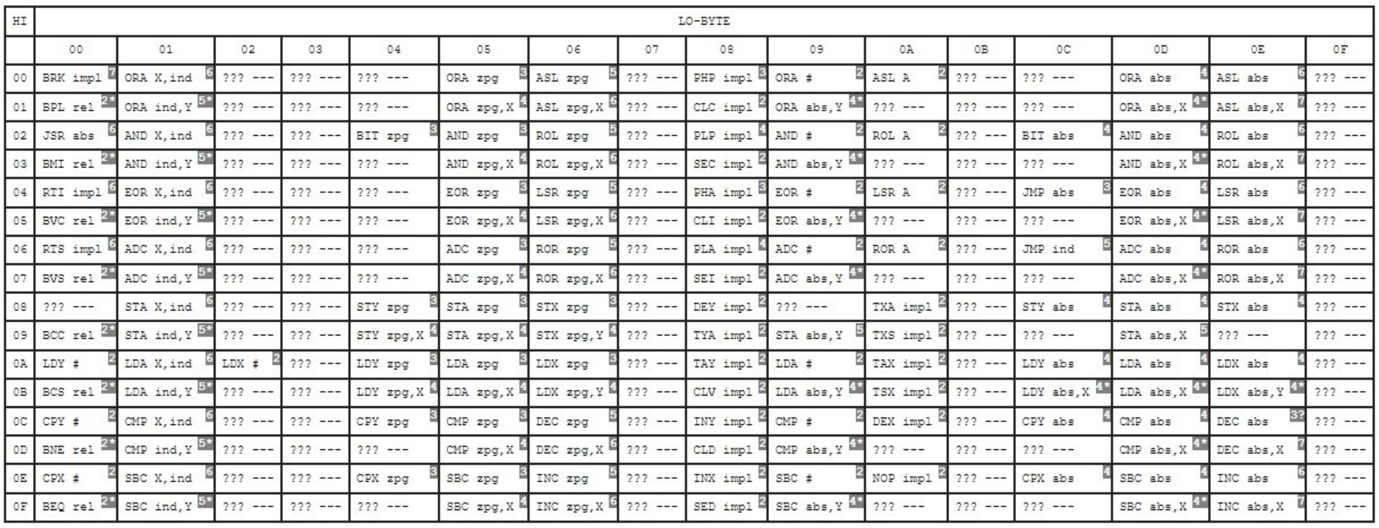


Table of 6502 opcodes (for reference):



HI	LO-BYTE															
	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
00	BRK impl 7	ORA X,ind 6	??? ---	??? ---	??? ---	ORA zpg 3	ASL zpg 5	??? ---	PHP impl 3	ORA # 2	ASL A 2	??? ---	??? ---	ORA abs 4	ASL abs 6	??? ---
01	BPL rel 2*	ORA ind,Y 5*	??? ---	??? ---	??? ---	ORA zpg,X 4	ASL zpg,X 6	??? ---	CLC impl 2	ORA abs,Y 4*	??? ---	??? ---	??? ---	ORA abs,X 4*	ASL abs,X 7	??? ---
02	JSR abs 6	AND X,ind 6	??? ---	??? ---	BIT zpg 3	AND zpg 3	ROL zpg 5	??? ---	PLP impl 4	AND # 2	ROL A 2	??? ---	BIT abs 4	AND abs 4	ROL abs 6	??? ---
03	BMI rel 2*	AND ind,Y 5*	??? ---	??? ---	??? ---	AND zpg,X 4	ROL zpg,X 6	??? ---	SEC impl 2	AND abs,Y 4*	??? ---	??? ---	??? ---	AND abs,X 4*	ROL abs,X 7	??? ---
04	RTI impl 6	EOR X,ind 6	??? ---	??? ---	??? ---	EOR zpg 3	LSR zpg 5	??? ---	PHA impl 3	EOR # 2	LSR A 2	??? ---	JMP abs 3	EOR abs 4	LSR abs 6	??? ---
05	BVC rel 2*	EOR ind,Y 5*	??? ---	??? ---	??? ---	EOR zpg,X 4	LSR zpg,X 6	??? ---	CLI impl 2	EOR abs,Y 4*	??? ---	??? ---	??? ---	EOR abs,X 4*	LSR abs,X 7	??? ---
06	RTS impl 6	ADC X,ind 6	??? ---	??? ---	??? ---	ADC zpg 3	ROR zpg 5	??? ---	PLA impl 4	ADC # 2	ROR A 2	??? ---	JMP ind 5	ADC abs 4	ROR abs 6	??? ---
07	BVS rel 2*	ADC ind,Y 5*	??? ---	??? ---	??? ---	ADC zpg,X 4	ROR zpg,X 6	??? ---	SEI impl 2	ADC abs,Y 4*	??? ---	??? ---	??? ---	ADC abs,X 4*	ROR abs,X 7	??? ---
08	??? ---	STA X,ind 6	??? ---	??? ---	STY zpg 3	STA zpg 3	STX zpg 3	??? ---	DEY impl 2	??? ---	TXA impl 2	??? ---	STY abs 4	STA abs 4	STX abs 4	??? ---
09	BCC rel 2*	STA ind,Y 5*	??? ---	??? ---	STY zpg,X 4	STA zpg,X 4	STX zpg,Y 4	??? ---	TYA impl 2	STA abs,Y 5	TXS impl 2	??? ---	??? ---	STA abs,X 5	??? ---	??? ---
0A	LDY # 2	LDA X,ind 6	LDX # 2	??? ---	LDY zpg 3	LDA zpg 3	LDX zpg 3	??? ---	TAY impl 2	LDA # 2	TAX impl 2	??? ---	LDY abs 4	LDA abs 4	LDX abs 4	??? ---
0B	BCS rel 2*	LDA ind,Y 5*	??? ---	??? ---	LDY zpg,X 4	LDA zpg,X 4	LDX zpg,Y 4	??? ---	CLV impl 2	LDA abs,Y 4*	TSX impl 2	??? ---	LDY abs,X 4*	LDA abs,X 4*	LDX abs,Y 4*	??? ---
0C	CPY # 2	CMP X,ind 6	??? ---	??? ---	CPY zpg 3	CMP zpg 3	DEC zpg 5	??? ---	INY impl 2	CMP # 2	DEX impl 2	??? ---	CPY abs 4	CMP abs 4	DEC abs 3?	??? ---
0D	BNE rel 2*	CMP ind,Y 5*	??? ---	??? ---	??? ---	CMP zpg,X 4	DEC zpg,X 6	??? ---	CLD impl 2	CMP abs,Y 4*	??? ---	??? ---	??? ---	CMP abs,X 4*	DEC abs,X 7	??? ---
0E	CPX # 2	SBC X,ind 6	??? ---	??? ---	CPX zpg 3	SBC zpg 3	INC zpg 5	??? ---	INX impl 2	SBC # 2	NOP impl 2	??? ---	CPX abs 4	SBC abs 4	INC abs 6	??? ---
0F	BEQ rel 2*	SBC ind,Y 5*	??? ---	??? ---	??? ---	SBC zpg,X 4	INC zpg,X 6	??? ---	SED impl 2	SBC abs,Y 4*	??? ---	??? ---	??? ---	SBC abs,X 4*	INC abs,X 7	??? ---

Special Lines

Additional logical operations are applied to some decoder outputs, which although territorially are in the decoder area, are actually part of *random logic*. Most likely this logic got into the decoder simply because it was more convenient to split the connections that way.

List:

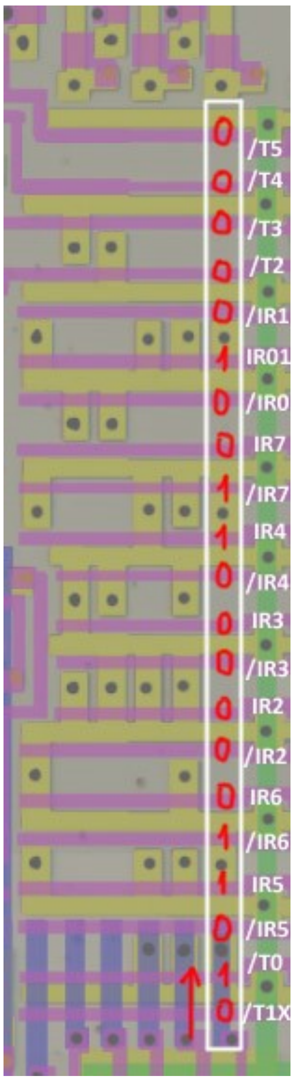
- Internal Push/Pull line: a special (129th) line that does not extend beyond the decoder. It is used to "cut off" Push/pull instructions when selecting instructions. It is used in three lines: 83, 90, and 128.
- /PRDY: this line goes to decoder line 73 (Branch T0)
- IR0: normally the common signal IR01 is used to check the two lowest bits of the operation code, but exclusively for the 128th line (IMPL), IR0 is used (IR0 is not included in the mask for the table below).

PLA Contents

Group	N	Mask value (Raw bits)	Decoded mask value	Cycle (T)	Comments	Where to use
A						
A01	0	000101100000100100000	100XX100	TX	STY	
A02	1	000000010110001000100	XXX100X1	T3	OP ind, Y	
A03	2	000000011010001001000	XXX110X1	T2	OP abs, Y	
A04	3	010100011001100100000	1X001000	T0	DEY INY	
A05	4	010101011010100100000	10011000	T0	TYA	
A06	5	010110000001100100000	1100XX00	T0	CPY INY	
B						
B01	6	000000100010000001000	XXX1X1XX	T2	OP zpg, X/Y & OP abs, X/Y	
B02	7	000001000000100010000	10XXXX1X	TX	LDX STX A<->X S<->X	
B03	8	000000010101001001000	XXX000X1	T2	OP ind, X	
B04	9	010101011001100010000	1000101X	T0	TXA	
B05	10	010110011001100010000	1100101X	T0	DEX	
B06	11	011010000001100100000	1110XX00	T0	CPX INX	
B07	12	000101000000100010000	100XXX1X	TX	STX TXA TXS	
B08	13	010101011010100010000	1001101X	T0	TXS	
B09	14	011001000000100010000	101XXX1X	T0	LDX TAX TSX	
B10	15	100110011001100010000	1100101X	T1	DEX	
B11	16	101010011001100100000	11101000	T1	INX	
B12	17	011001011010100010000	1011101X	T0	TSX	
B13	18	100100011001100100000	1X001000	T1	DEY INY	
B14	19	011001100000100100000	101XX100	T0	LDY	
B15	20	011001000001100100000	1010XX00	T0	LDY TAY	

TBD: The rest of the decoder groups are here.

What Raw bits mean



If you think of a decoder as a 21x130 ROM, where each bit represents a transistor, then the `Raw bits` value will represent one line of the decoder. This is why it is called the mask value.

For example, the picture shows the 5th line of the decoder. The bit counting starts from bottom to top. 0 means no transistor, 1 means present.

Online Decoder

You can use an online decoder to highlight opcodes: http://breaknes.com/files/6502/decoder.htm (You can also find it here: https://github.com/emu-russia/breaks/blob/master/Docs/6502/decoder.htm)

In the `Raw bits` field you can insert the mask value from the table above and when you press the `Make IR Mask` button you will get the decoded mask value (e.g. `11X00X00`). The decoded mask value can be inserted into the `IR` field and when the `Decode` button is pressed, the opcodes that correspond to the specified IR mask will be highlighted in the table.

Branch T0 Skip

From pin `RDY` a special line `/PRDY` comes through the delay line. If the processor was not ready when the *previous* instruction finished, then if the next instruction is a conditional branch, its cycle 0 (T0) is skipped.

The meaning of this operation is not known yet.

Why the decoder is so big and scary

Actually, there is nothing scary about it.

The decoder was compiled according to the requirements of random logic. Random logic is divided into several parts (domains) and each part corresponds to its own zone in the decoder, which was specially chosen so that the necessary opcodes were processed.

In other words - it is not random logic that adjusts to decoder, but vice versa. The impression that the decoder is "more important" is formed simply because it is above random logic.

prc. 6502 rub.

The book contains a description of all the MOS 6502 circuits, Python simulation code and other exclusive material.

This book is the second (in order of creation) in the Breaking NES series of books. The first, Breaking NES PPU, is already available.
The online version of the book is free, the print version can be ordered from various offices that print from pdf on paper.