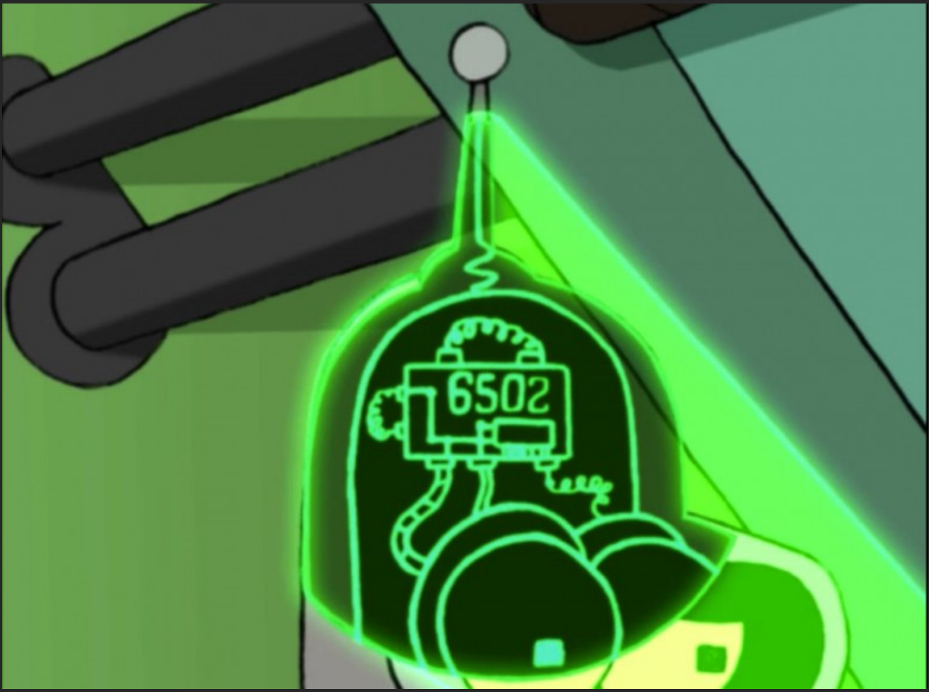
Breaking NES Book

6502 Core

andkorzh, HardWareMan, org



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. B3

Foreword

In this revision of the book all found errors are corrected and a detailed description of some 6502 instructions is added.

A C++ simulator has been written on the basis of the studied circuits, which has confirmed their correctness and has passed all functional tests of 6502 by Klaus Dormann.

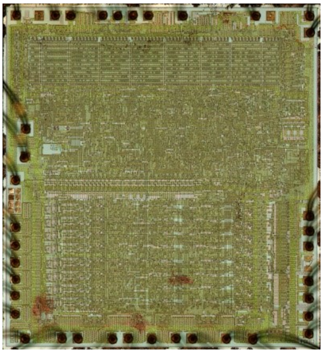
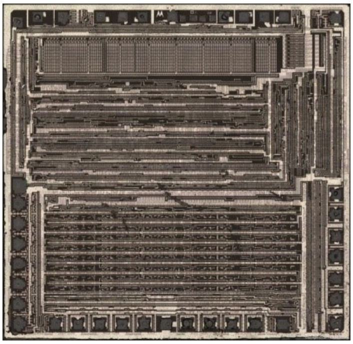
We are also thankful to ttlworks, for allowing the publication of the optimized 6502 circuits resulting from his 6509 (which is based on the 6502 core) research.

Contents

This page is needed so that the schematics on the spreads start on even pages.

6502 Overview

The 6502 processor was developed by MOS. It was based on the architecture of the Motorola 6800 processor:

6502	6800
	

In both cases the top part is occupied by the decoder and random logic, and the whole bottom part of the processor is occupied by the context and the ALU.

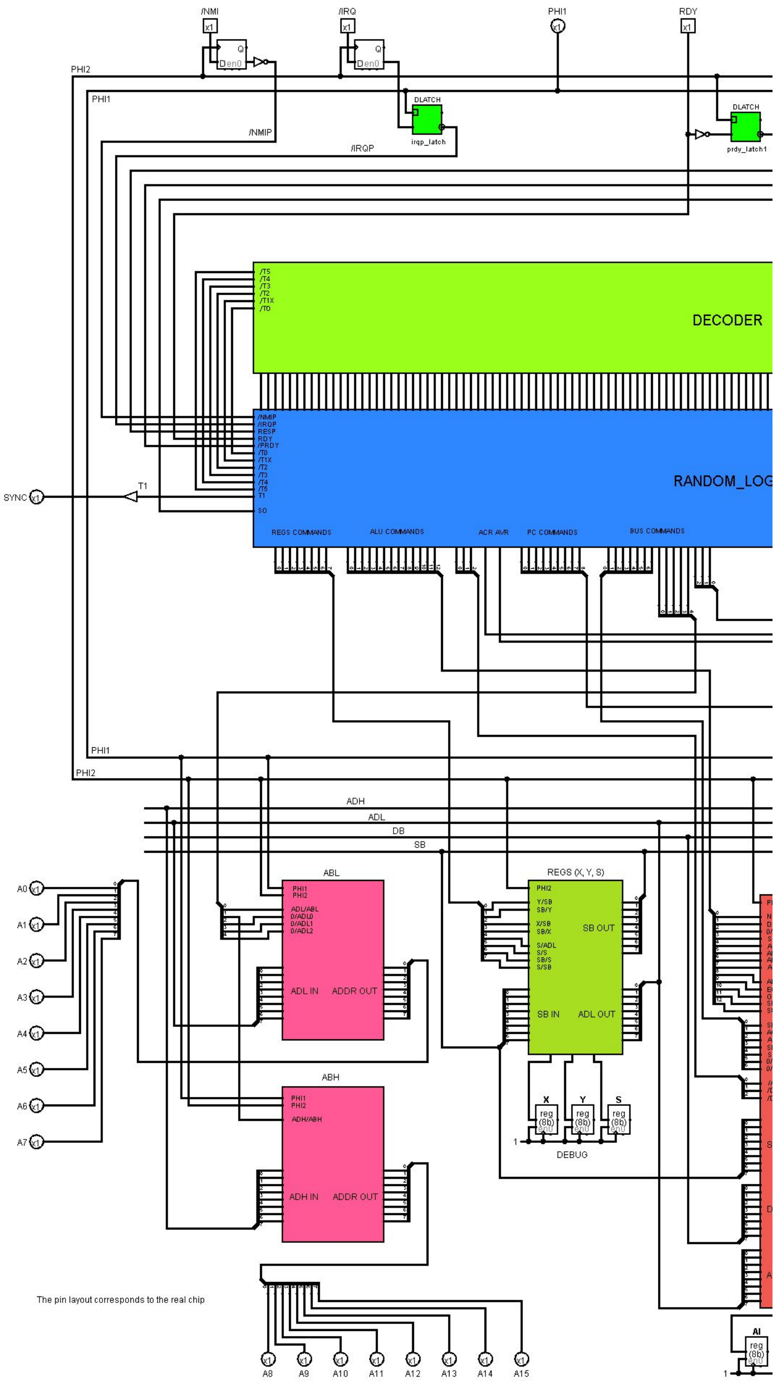
Architecture

The processor is divided into two parts: the upper part and the lower part.

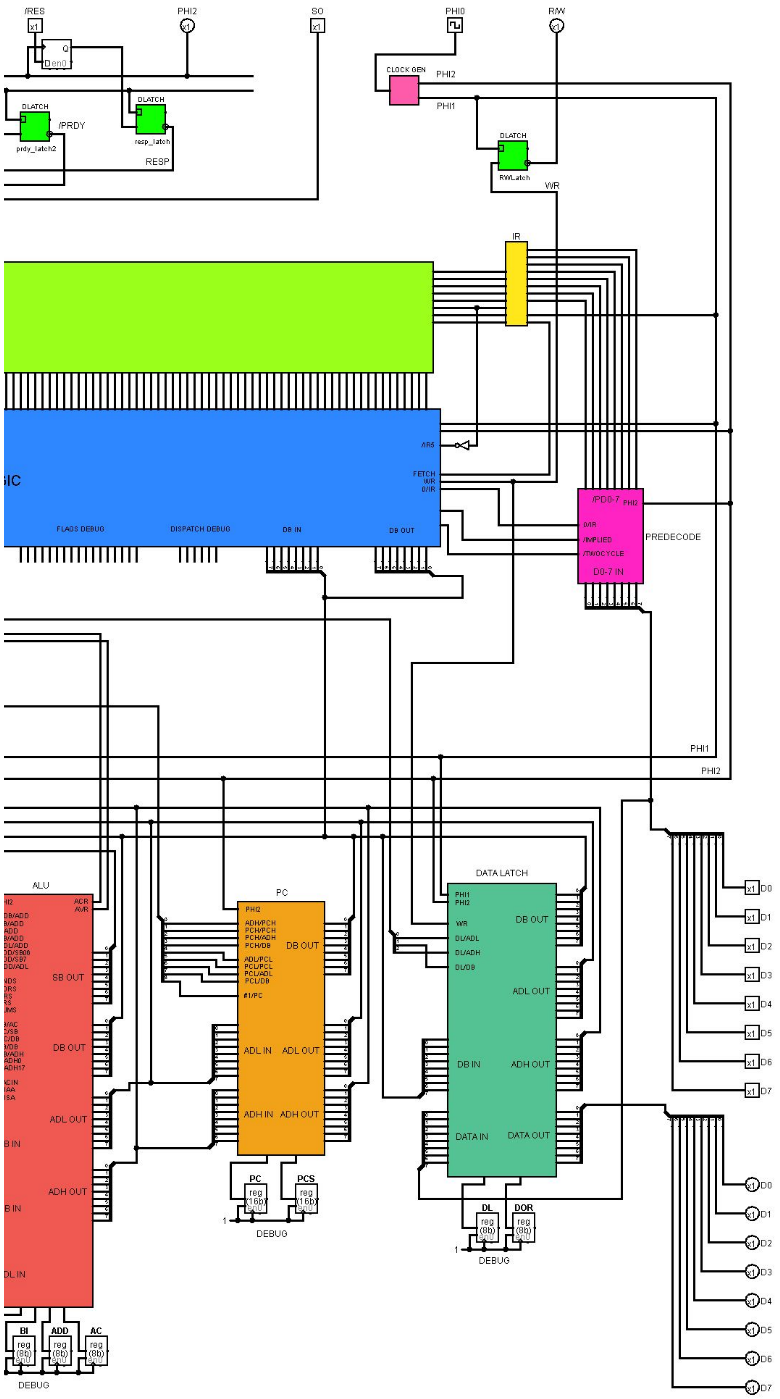
The upper part contains the control logic, which issues a number of control lines ("commands") to the lower part. The lower part contains the context of the processor: internal buses and registers, with one exception - the flags register (P) is in the upper part in a "spread out" form.

Also in the lower part is the *ALU*.

The processor is clocked by the PHI0 clock pulse, both half-cycles are used. During the first half-cycle (PHI1) the processor is in "Set Address and R/W Mode" mode. During the second half-cycle (PHI2) the processor is in "Read/Write Data" mode, during this half-cycle external devices can put data on the data bus and get data from the processor.



The pin layout corresponds to the real chip



/RES
PHI2
SO
PHI0
R/W
CLOCK GEN
PHI2
PHI1
DLATCH
prdy_latch2
/PRDY
DLATCH
resp_latch
RESP
DLATCH
RWLatch
WR
IR
/IR5
FETCH
WR
0/IR
FLAGS DEBUG
DISPATCH DEBUG
DB IN
DB OUT
/PD0-7
PHI2
0/IR
/IMPLIED
/TWOCYCLE
D0-7 IN
PREDECODE
PHI1
PHI2
ALU
ACR
AVR
SB OUT
DB OUT
ADL OUT
ADH OUT
PC
PHI2
ADH/PCH
PCH/PCH
PCH/ADH
PCH/DB
ADL/PCL
PCL/PCL
PCL/ADL
PCL/DB
#1/PC
DB OUT
ADL IN
ADL OUT
ADH IN
ADH OUT
DATA LATCH
PHI1
PHI2
WR
DL/ADL
DL/ADH
DL/DB
DB OUT
ADL OUT
DB IN
ADH OUT
DATA IN
DATA OUT
PC
PCS
DEBUG
DL
DOR
DEBUG
BI
ADD
AC
DEBUG
D0
D1
D2
D3
D4
D5
D6
D7
D0
D1
D2
D3
D4
D5
D6
D7

Registers

- PD: current operation code for precoding
- IR: instruction register (stores the current operation code)
- X, Y: index registers
- S: Stack pointer
- AI, BI: input values for ALU
- ADD: Intermediate result of an ALU operation
- AC: accumulator
- PCH/PCL: program counter in two halves
- PCHS/PCLS: program counter auxiliary registers (S stands for "Select")
- ABH/ABL: registers for output to the external address bus
- DL: data latch, stores the last read value of the external data bus
- DOR: data output register, holds the value which will be written to the data bus
- P: flag register, actually consists of a set of latches scattered around the circuit

The following registers are directly available to the programmer: A (accumulator), X, Y, S, P, PC.

External Buses

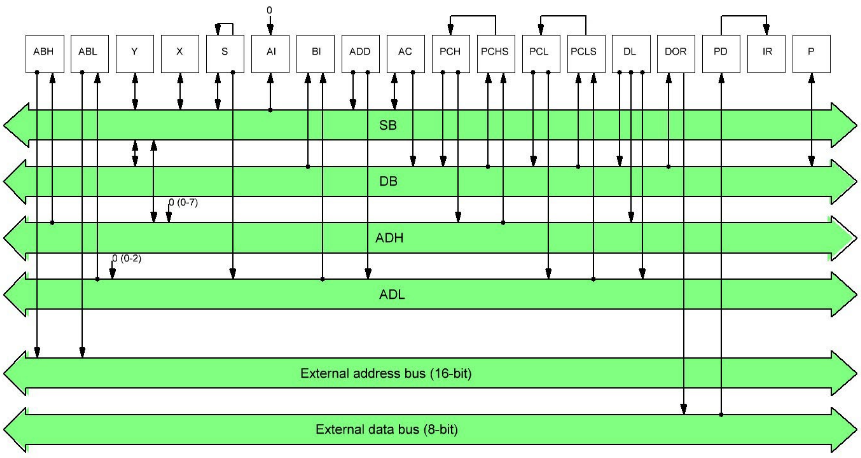
There are only two external buses: a 16-bit address bus (ADDR) and an 8-bit data bus (DATA). The address bus is one-way - only the processor can write to it. The data bus is bidirectional.

Internal Buses

- ADH/ADL: address bus
- SB: Special bus, register exchange bus
- DB: Internal data bus

During the second half-step (PHI2) all internal buses are precharged and have a value of 0xFF. This is done because it is faster to "discharge" the transistor at the right moment than to "charge" it (the change of 1=>0 is faster than the change of 0=>1).

Register-Bus Connections



By connecting buses and registers in series, the processor executes a variety of instructions. The variety of connections provides a variety of processor instructions, and the division of instructions into clock cycles allows complex actions to be performed. In addition, the ALU is controlled (addition, logical operations, etc.).

Software Model

Addressing Modes

Addressing modes are described here because they should be kept in mind when analyzing circuits.

Addressing is a way to get the operand to (or load it from) the desired memory location. The developers of the 6502 were very generous and added as many as two X and Y index registers to the context.

"Indexed" means that an offset is added to the memory address in a certain way to get a new address. This is usually needed to access arrays. In this case the beginning of the array will be a fixed address and the value in the index register will be the array index (offset).

List of addressing modes:

- Immediate (immediate operand). In this case the operand is stored in the instruction itself (usually the second byte, after the operation code). Example `LDA #$1C`: A = 0x1C
- Absolute (absolute addressing). The instruction specifies the full 16-bit address from which to get the operand. For example `LDA $1234`: A = [$1234]
- Zero Page Absolute: Developers have made an optimized version of absolute addressing by adding the ability to address only page zero (pages are 256 bytes in size). Example `LDA $56`: In this case the processor itself makes the highest 8 bits of the address equal to 0x00, while the lowest 8 bits are taken from the instruction. The final address is 0x0056. A = [0x0056]. This is done to save instruction size (one byte is saved).
- Indexed: In this addressing mode an offset from the X or Y register is added to the constant address value. For example `LDA $1234, X`: A = [$1234 + X]
- Zero Page Indexed: Similar to Indexed but only the X register can be used. Example `LDA $33, X`: A = [$0033 + X]

And then the special magic begins:

- Pre-indexed Indirect: The value of the operand which is the address in page zero is added to the value of register X and the indirect address is obtained. The address the indirect address refers to is then used to get the value of the operand. Example `LDA ($34, X)`: A = [[$0034 + X]]. Important: When you add an address and a value in the X register, it "wraps" around 256 bytes. That is, it does not wrap to the higher half of the address. (0xFF + 0x02 will be 0x0001, not 0x0101). **Indirect** means "take address by address".
- Post-indexed Indirect: Different from the previous one in that the indirect address from page zero is selected first, and then the index register Y value is added to it. Example `LDA ($2A), Y`: A = [[$002A] + Y].

Instruction Set

The 6502 has all the necessary instructions and also includes such rather handy instructions as bit rotation (ROL/ROR) and bit testing (BIT). Not all processors of the time contained such instructions.

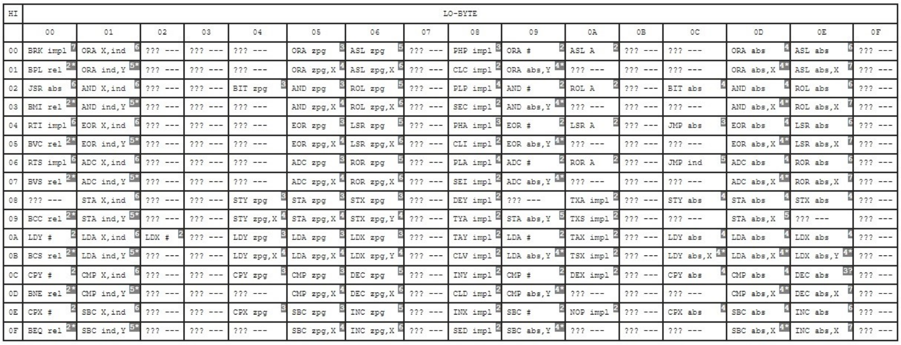
The instruction type and address mode are fully contained in the operation code, to simplify decoding, but the bus width (8 bits) does not allow all instructions to be executed in a single clock cycle. Also, the decoder is somewhat unoptimized, so the minimum instruction execution time is 2 clock cycles, with the first clock cycle always taken by sampling the operation code (the first byte of the instruction).

Summary of instructions:

Instruction	Description
ADC	Add Memory to Accumulator with Carry
AND	"AND" Memory with Accumulator
ASL	Shift Left One Bit (Memory or Accumulator)
BCC	Branch on Carry Clear
BCS	Branch on Carry Set
BEQ	Branch on Result Zero
BIT	Test Bits in Memory with Accumulator
BMI	Branch on Result Minus
BNE	Branch on Result not Zero
BPL	Branch on Result Plus
BRK	Force Break
BVC	Branch on Overflow Clear
BVS	Branch on Overflow Set
CLC	Clear Carry Flag
CLD	Clear Decimal Mode
CLI	Clear interrupt Disable Bit
CLV	Clear Overflow Flag
CMP	Compare Memory and Accumulator
CPX	Compare Memory and Index X
CPY	Compare Memory and Index Y
DEC	Decrement Memory by One
DEX	Decrement Index X by One
DEY	Decrement Index Y by One
EOR	"Exclusive-Or" Memory with Accumulator
INC	Increment Memory by One
INX	Increment Index X by One
INY	Increment Index Y by One
JMP	Jump to New Location
JSR	Jump to New Location Saving Return Address
LDA	Load Accumulator with Memory
LDX	Load Index X with Memory
LDY	Load Index Y with Memory
LSR	Shift Right One Bit (Memory or Accumulator)
NOP	No Operation
ORA	"OR" Memory with Accumulator
PHA	Push Accumulator on Stack
PHP	Push Processor Status on Stack
PLA	Pull Accumulator from Stack
PLP	Pull Processor Status from Stack
ROL	Rotate One Bit Left (Memory or Accumulator)
ROR	Rotate One Bit Right (Memory or Accumulator)
RTI	Return from Interrupt
RTS	Return from Subroutine
SBC	Subtract Memory from Accumulator with
SEC	Set Carry Flag
SED	Set Decimal Mode
SEI	Set Interrupt Disable Status
STA	Store Accumulator in Memory
STX	Store Index X in Memory
STY	Store Index Y in Memory
TAX	Transfer Accumulator to Index X
TAY	Transfer Accumulator to Index Y
TSX	Transfer Stack Pointer to Index X
TXA	Transfer Index X to Accumulator
TXS	Transfer Index X to Stack Pointer
TYA	Transfer Index Y to Accumulator

The developers chose the encoding so that it would be easier to process by *decoder* and *random 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	??? ---

You can find a description of the instructions in any Reference Manual for 6502.

Interrupts

6502 interrupts:

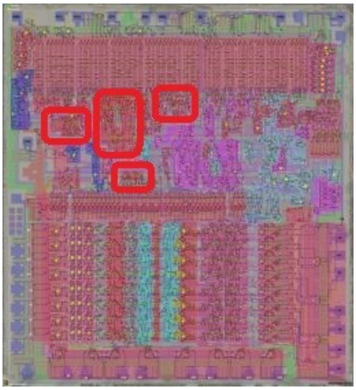
- IRQ: hardware interrupt. Can be disabled with flag I (interrupt disable), if flag I=1 the interrupt is "disabled" and does not go to the CPU.
- NMI: non-maskable interrupt. It has higher priority than IRQ, triggered on falling edge.
- RES: hardware reset. After powering up the 6502 it is necessary to set the /RES pin to 0 for a few cycles so that the processor "comes to its senses".
- BRK: software interrupt. It is initiated by the `BRK` instruction.

Note on Transistor Circuits

The transistor circuits of each component are chopped into component parts so that they don't take up too much space.

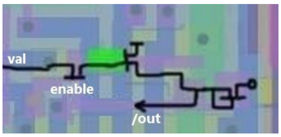
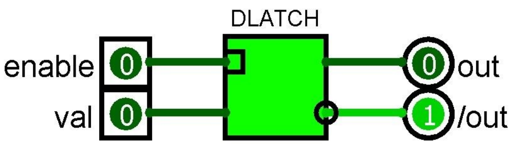
To keep you from getting lost, each section includes a special "locator" at the beginning that marks the approximate location of the component being studied on the large 6502 "family portrait" (https://github.com/emu-russia/breaks/blob/master/Docs/6502/6502.jpg)

Example locator:

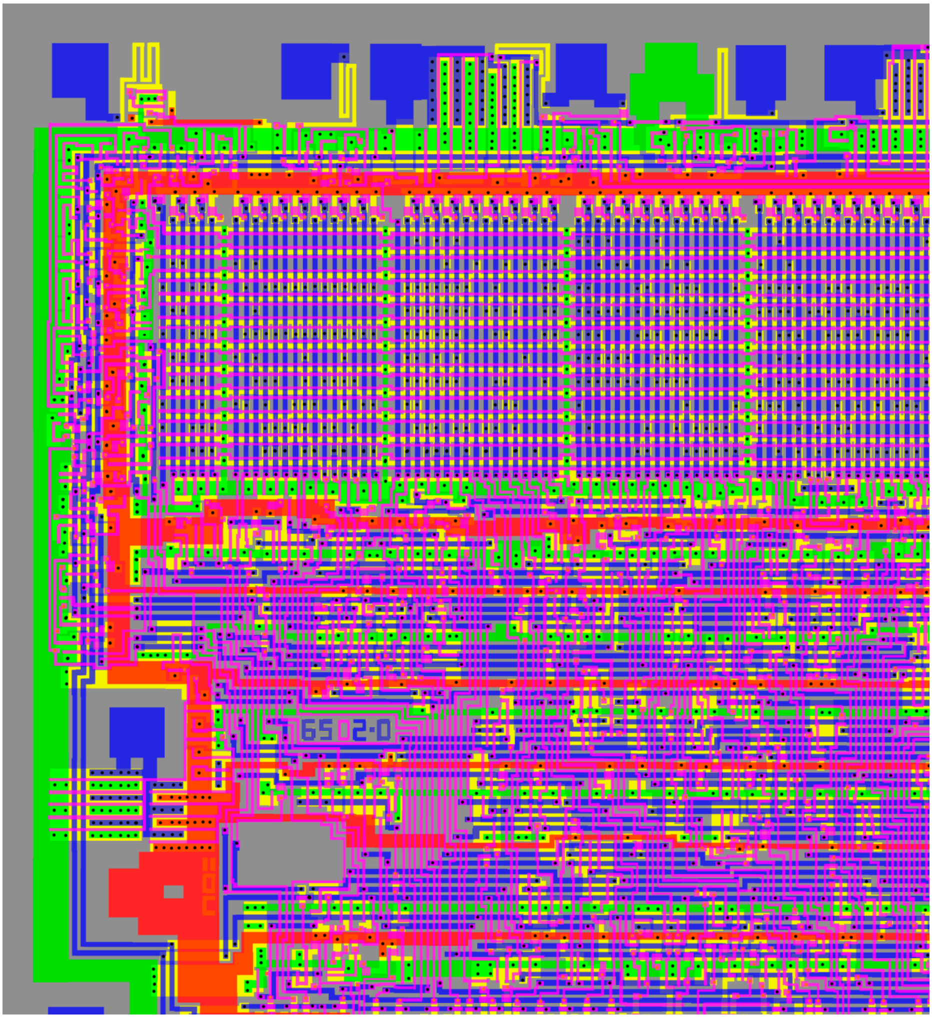


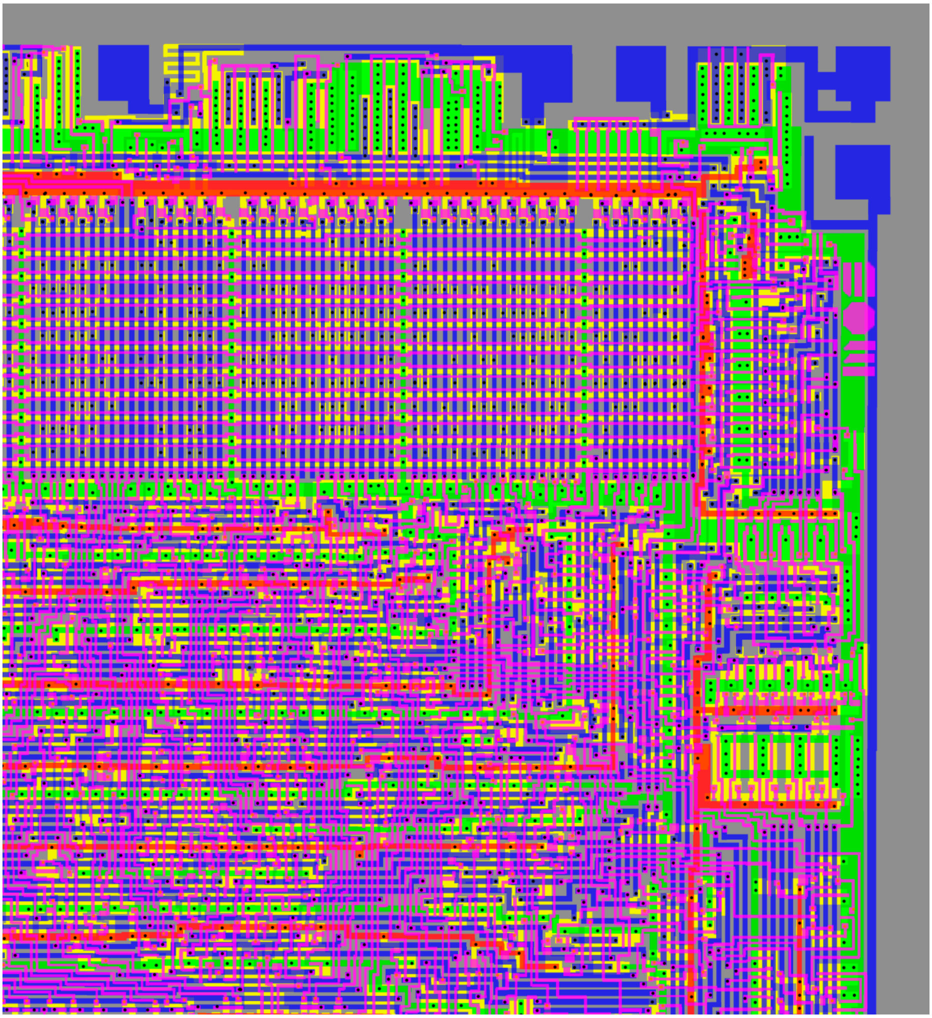
Note on Logic Circuits

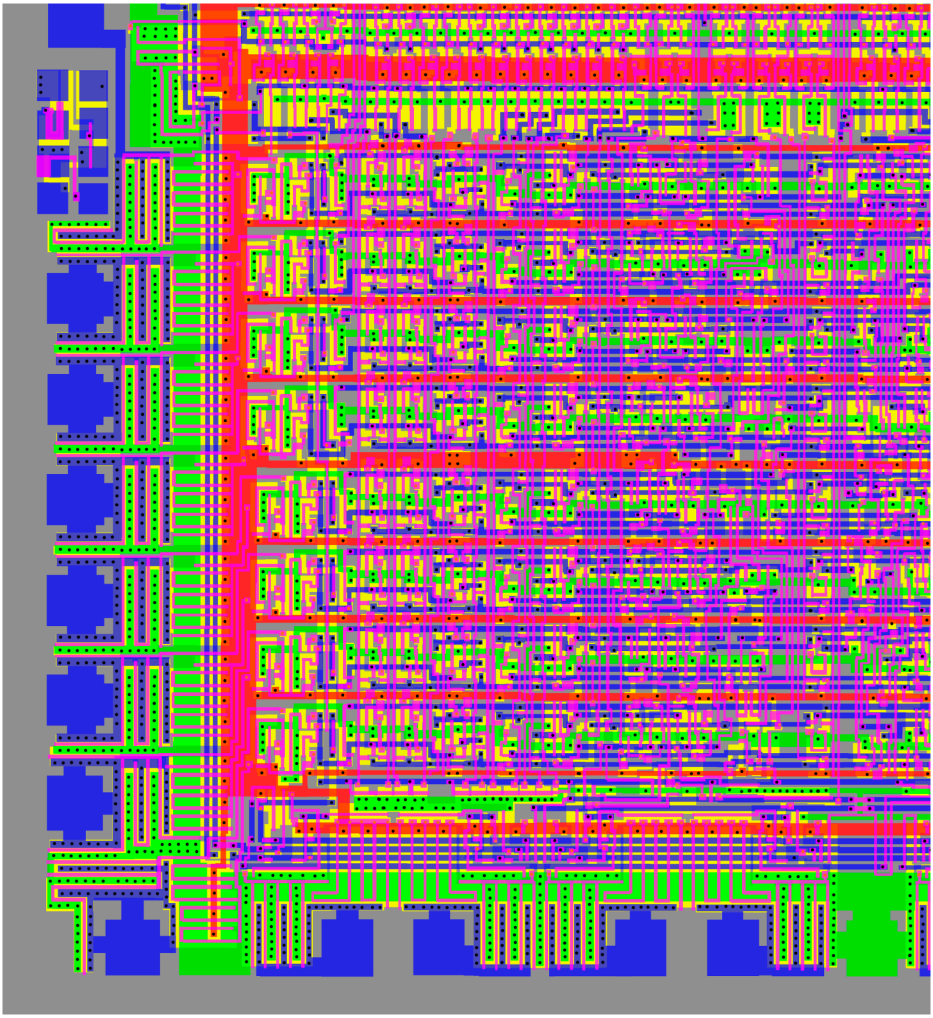
The logic circuits are mostly made in the Logisim program. The following element is used to denote DLatch:

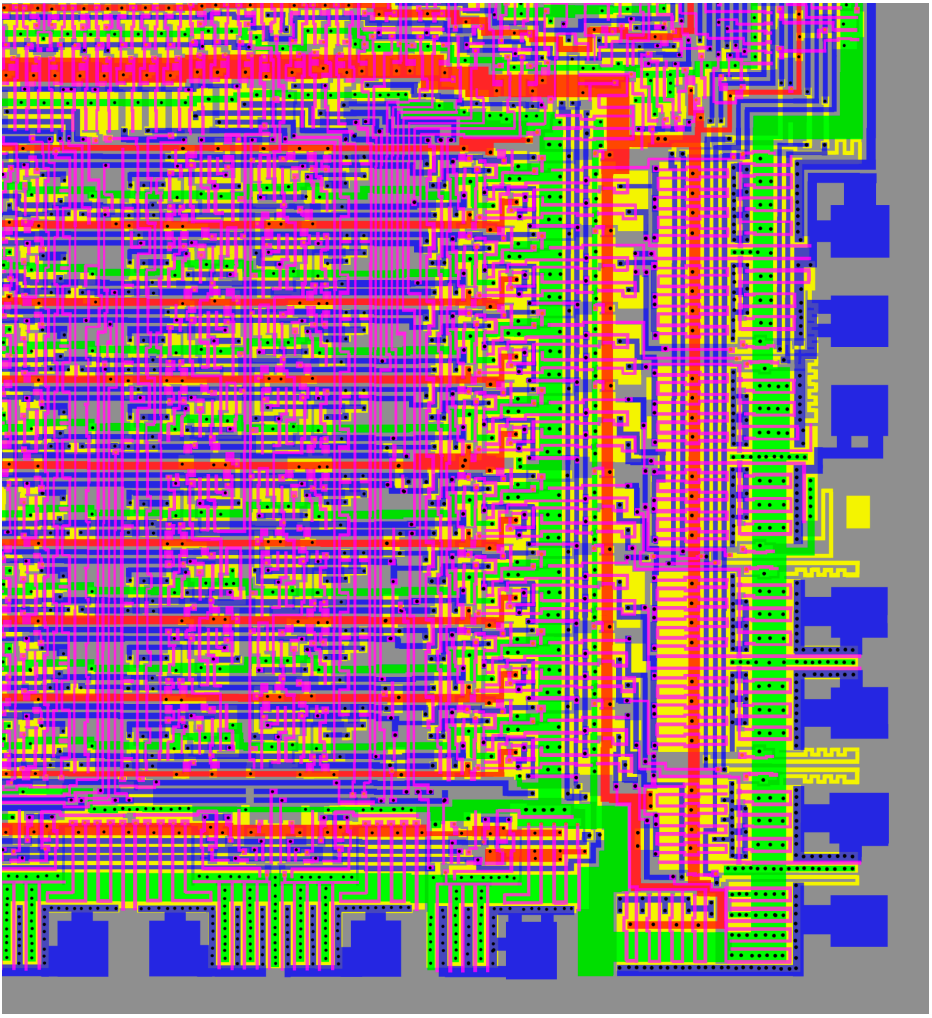
DLatch (transistor circuit)	DLatch (logic equivalent)
	

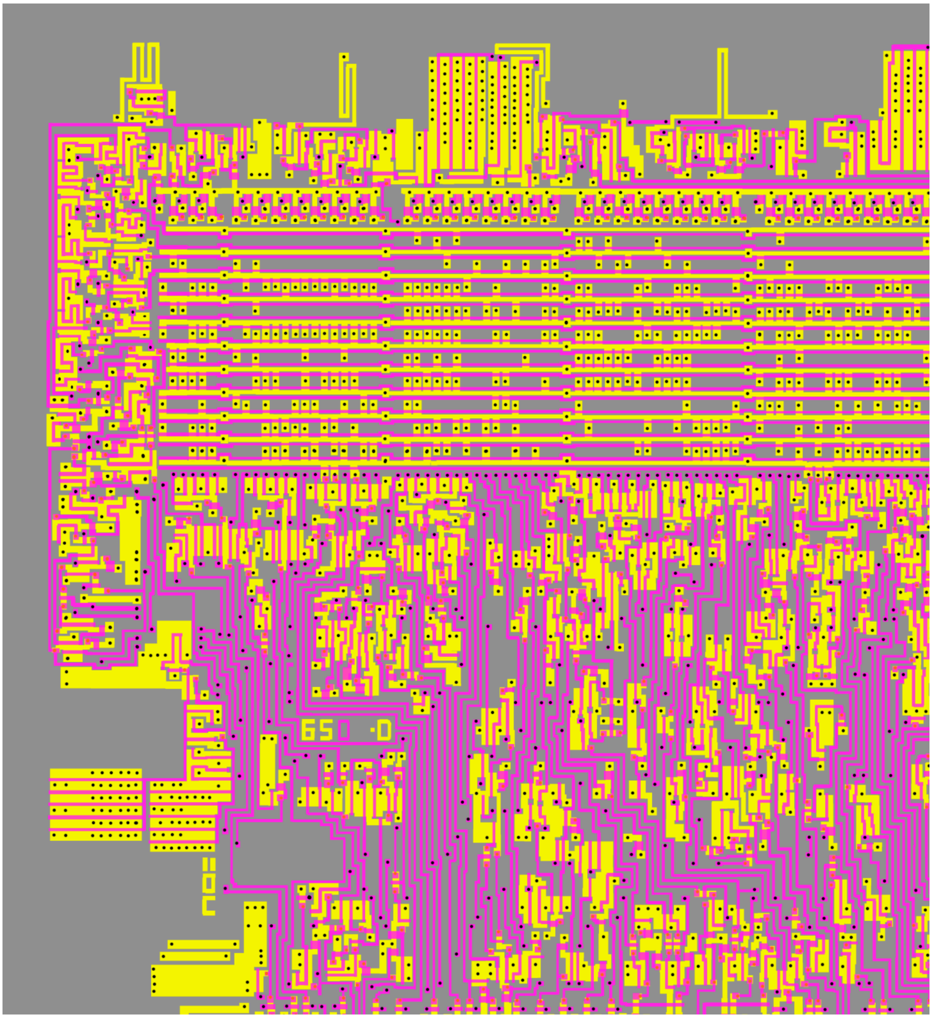
For convenience, the logical variant of DLatch has two outputs (`out` and `/out`), since the current value of DLatch (out) is often used as an input of a NOR operation.

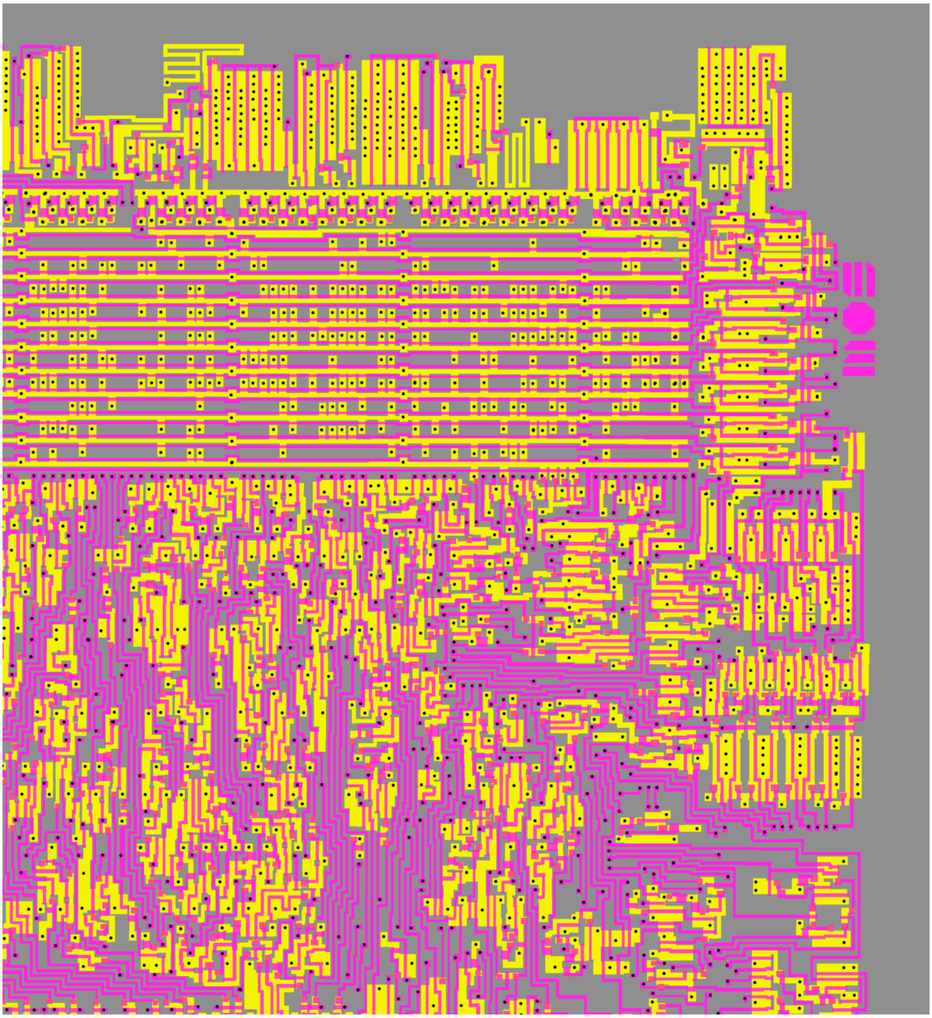

6502-0

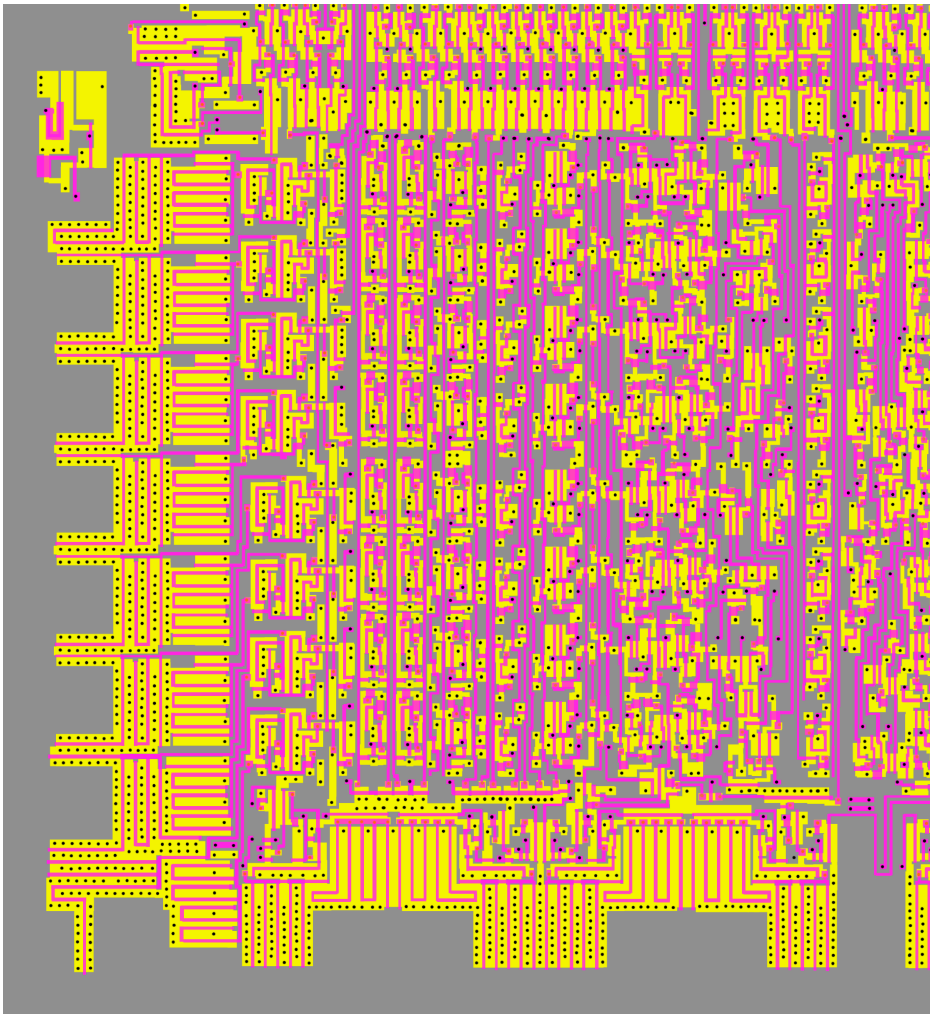


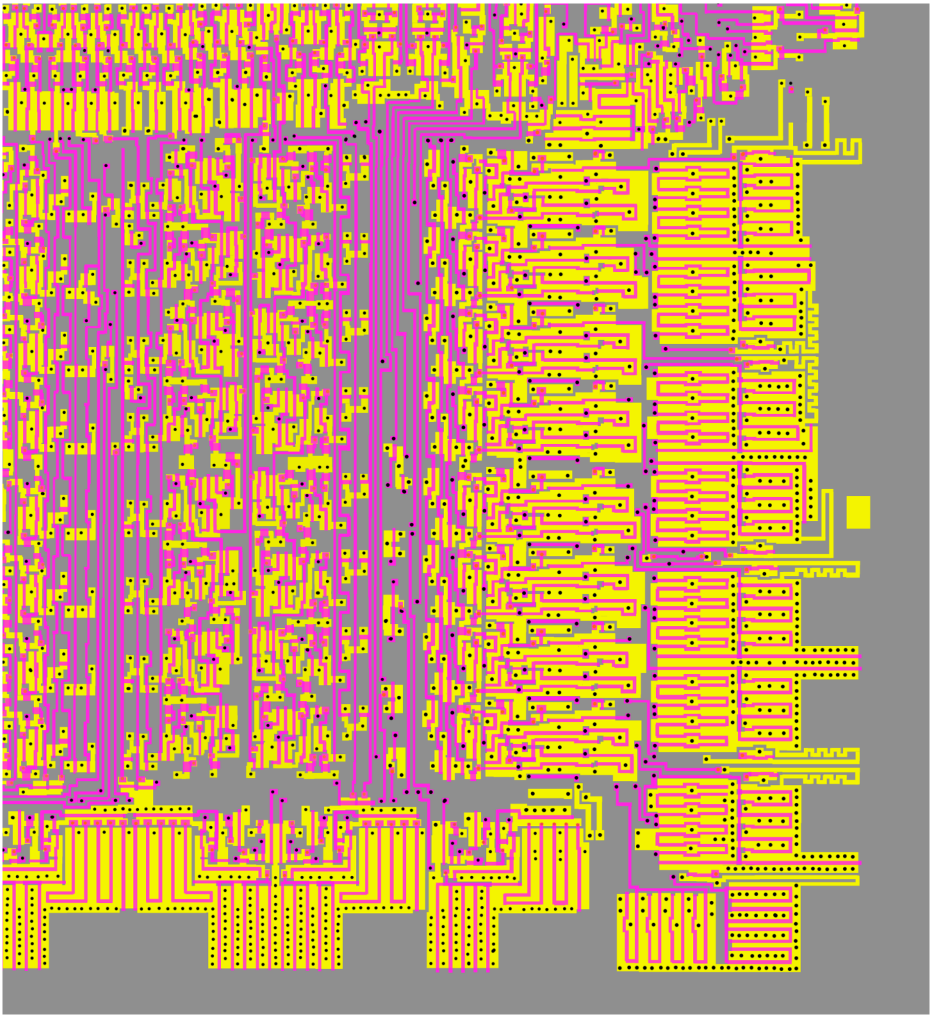





650 ·D

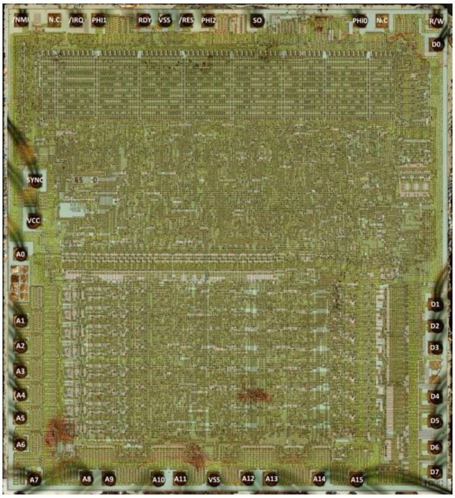


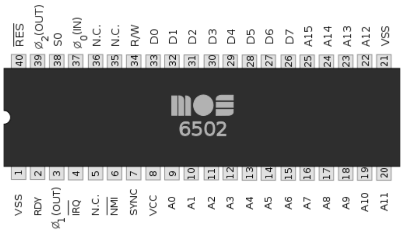




Pinout

The study of any integrated circuit begins with the pinout.





Name	Direction	Description
VCC	=> 6502	Power +5 V
VSS	6502 =>	Ground
/NMI	=> 6502	Non-maskable interrupt signal, active low
/IRQ	=> 6502	Maskable interrupt signal, active low
/RES	=> 6502	Reset signal, active low
PHI0	=> 6502	Reference clock signal
PHI1	6502 =>	First half-cycle, processor in writing mode
PHI2	6502 =>	Second half-cycle, processor in read mode
RDY	=> 6502	Processor Ready (1: ready)
SO	=> 6502	Forced setting of the overflow flag (V)
R/W	6502 =>	Data bus direction (R/W=1: processor reads data, R/W=0: processor writes)
SYNC	6502 =>	The processor is on cycle T1 (opcode fetch)
A0-A15	6502 =>	Address bus
D0-D7	6502 <=>	Data bus (bidirectional)
N.C.	--	Not connected

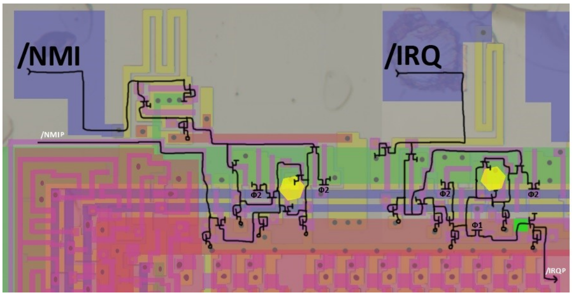
Vcc/Vss

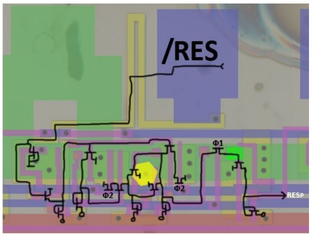
From the official datasheet we know that the operating range of Vcc = +5.0 volts +/- 5%.

Clock Generator

The clock signals are described in a separate section (see *clock generator*).

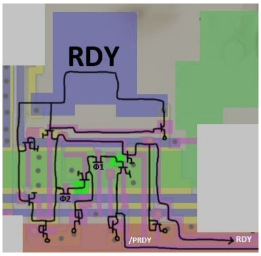
/NMI, /IRQ, /RES

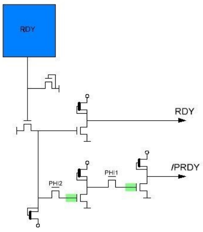




Each contact contains a FF where the interrupt arrival event is stored. The FF value corresponds to the control signals `/NMIP`, `/IRQP` and `RESP` (the value from FF for contact /RES is output as direct value). The "P" in the name of the control signals stands for "Pad" (contact).

RDY

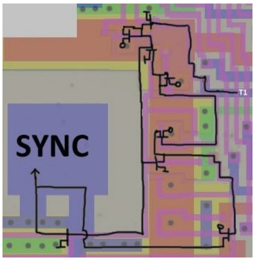


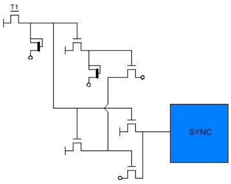


The RDY pin goes to the internal `RDY` signal and also through the DLATCH delay chain as the `/PRDY` ("Previous Ready") signal. /PRDY goes to the *decoder* input `Branch T0`.

The RDY pin can be used to temporarily suspend the processor, e.g. while an external device performs a DMA.

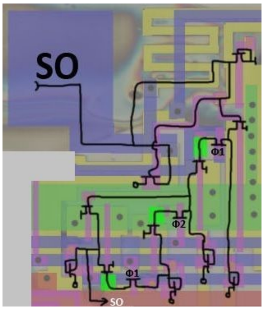
SYNC





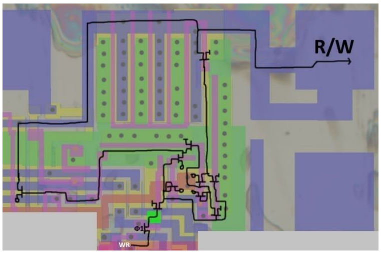
The SYNC signal comes from the internal T1 signal (opcode fetch).

SO



The internal signal `SO` is fed to the *flag V* input to process the control signal `1/V`.

R/W



The `WR` signal comes from *dispatcher* and defines the operating mode of the processor (WR:1 - processor writes data, WR:0 - processor reads data).

Address Bus

See *Address Bus*.

Data Bus

See *Data Bus*.

Notes in the margins for future revisions of the book.

Clock Generator

The 6502 includes two clock reference circuits: an external and an internal one.

The processor inputs one clock signal, `PHI0`, and outputs two clock signals, `PHI1` and `PHI2`.

This principle is based on the fact that each clock cycle of the processor consists of two "modes" (or "states"): write mode and read mode.

During write mode the `PHI1` signal is high. During this time, external devices can use the address set on the external address bus of the processor.

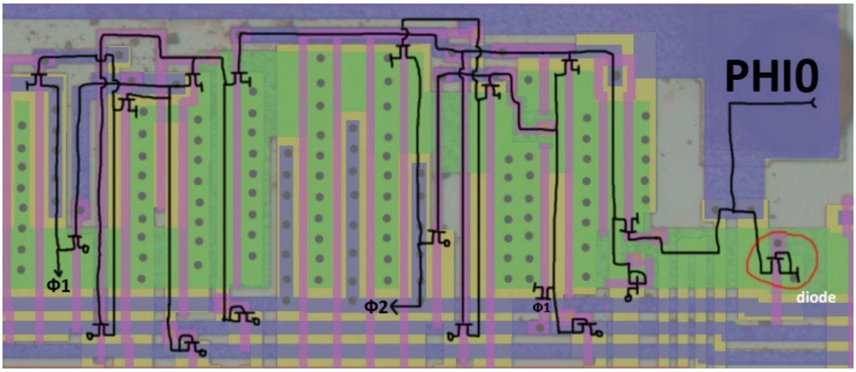
During read mode the signal `PHI2` is high. During this time external devices can write data to the processor's data bus so that the processor can use it for its own purposes.

The signals `PHI1` and `PHI2` are called half-cycles and are derived from the original clock signal `PHI0` as follows:

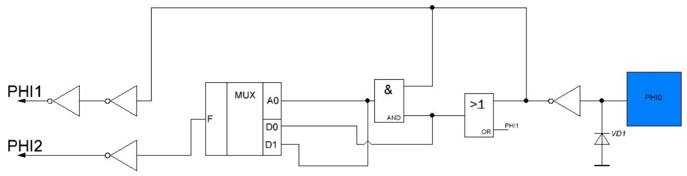
- When `PHI0` is low - the processor is in write mode and the `PHI1` signal is high
- When `PHI0` is high - the processor is in read mode and the `PHI2` signal is also high

PHI0	PHI1	PHI2
0	1	0
1	0	1

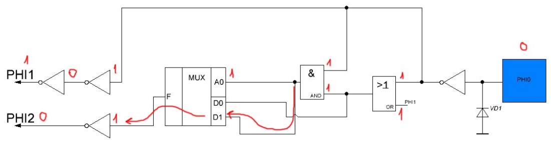
Internal Clock

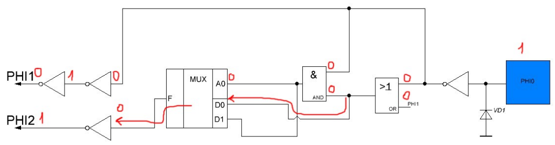


The circuit is quite complicated, because it is not quite "digital". The numerous transistors that act as inverters slightly delay the PHI0 signal, so the PHI1 and PHI2 signals going inside the processor are a bit "laggy". Here is the logical representation of the circuit:



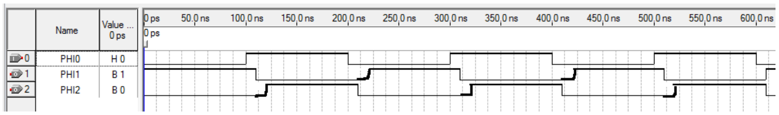
Logical analysis:





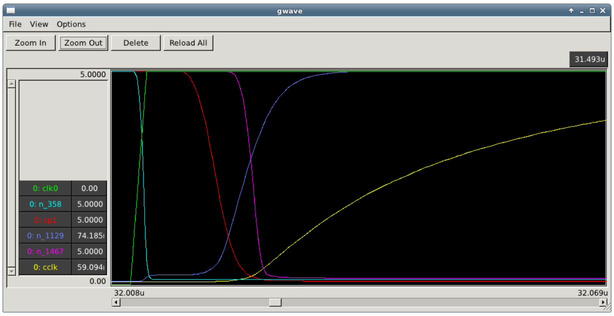
The layout of the clock signals should be about the following:

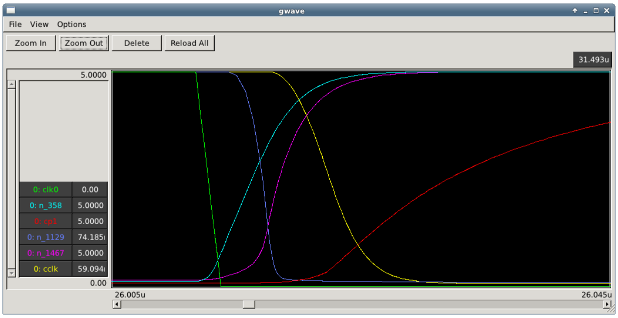
- PHI1/PHI2 are slightly lagging relative to PHI0
- The lower level of PHI1/PHI2 is slightly longer than the upper level, so that both signals are guaranteed not to have a high level



The simulation in Altera Quartus shows "lag", but does not show the elongated lower level (it is hand-drawn in the picture above).

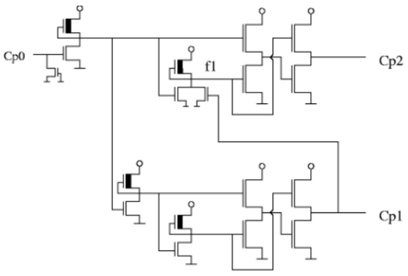
BigEd from the 6502.org forum suggested that he ran a simulation on the 6502 FPGA netlist and got the following sweeps:





The signal designations are as follows: clk0 = PHI0, cp1 = PHI1, cclk = PHI2 (according to the netlist with Visual6502)

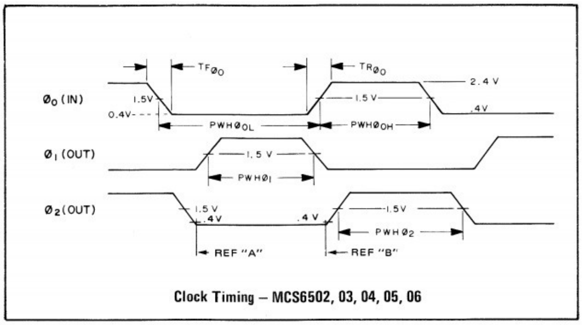
The schematic on which his simulation was based corresponds to the one in Balasz's documentation:



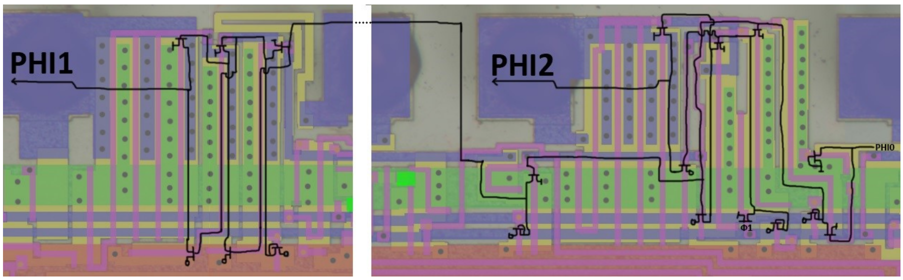
http://forum.6502.org/viewtopic.php?f=8&t=2208&start=195

It turns out that because of the asymmetrical inverter stage the rising edge is delayed, so the lower level is as if "delayed".

The official documentation gives the following diagram:



External Clock



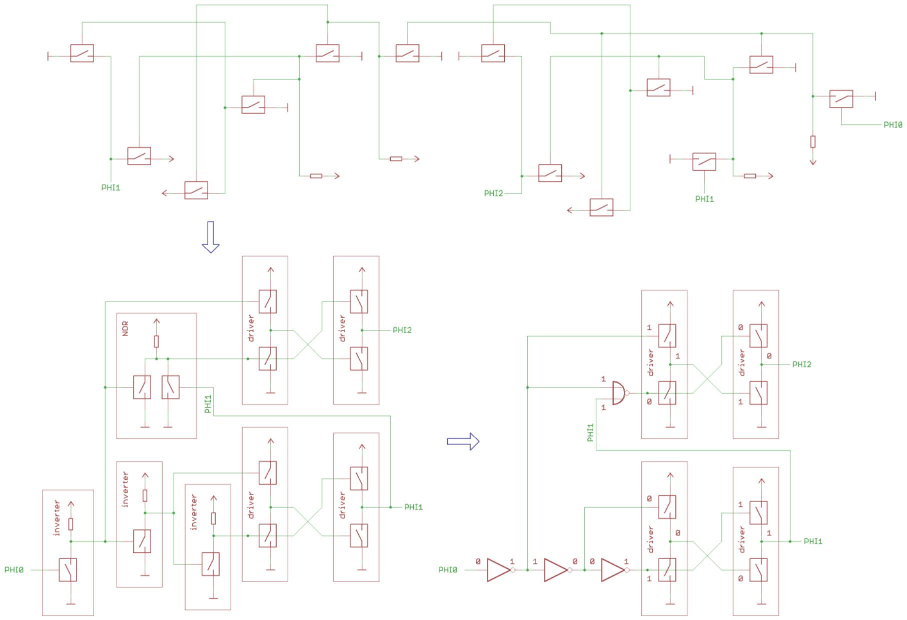
The PHI1/PHI2 reference signals are also output to the outside for consumers.

The logic circuit of the external wiring of the clock signals does not differ from the internal wiring circuit, except that the outputs of PHI1/PHI2 go to the same contacts through the "comb" of powerful transistors.

Why PHI

In the official 6502 datasheet the half-cycles are called "phases", respectively the name of these signals is Φ1 and Φ2. For unification we use the designations PHI1 and PHI2.

Optimized Schematics

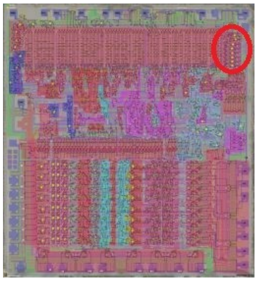


https://github.com/emu-russia/breaks/blob/master/BreakingNESWiki_DeepL/6502/clock.md
8:https://github.com/emu-russia/breaks/blob/master/BreakingNESWiki/imgstore/clock_internal.jpg

Notes in the margins for future revisions of the book.

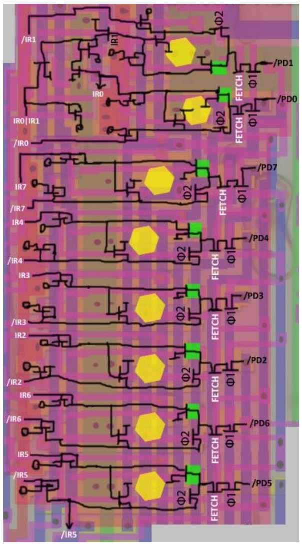
TOP PART

Instruction Register



The Instruction Register (IR) stores the current operation code, for processing on *decoder*. The operation code is loaded into the IR from *predecode logic*.

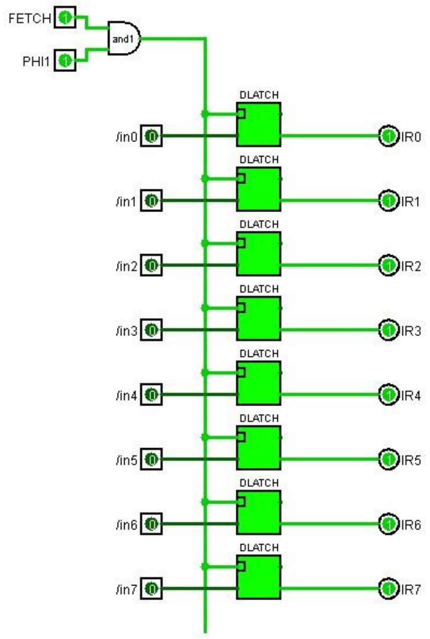
Transistor Circuit



The outputs in the schematic are on the left because the decoder is topologically located on the left side.

- IR0 and IR1 are combined into one common line `IR01` to save lines
- IR0 is used only for the 128th decoder line (IMPL) (this operation with IR0 is part of the random logic)
- /IR5 goes additionally to flags and is used in set/clear flags instructions (SEI/CLI, SED/CLD, SEC/CLC)

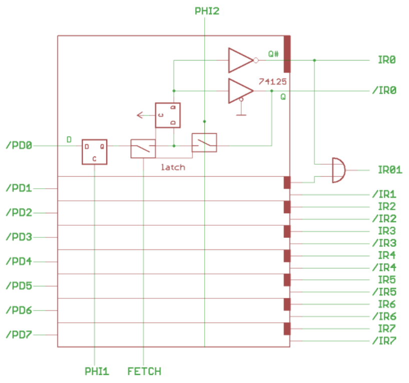
Logic



- During PHI1 the IR value is overloaded from the *Predecode (PD)* latch, but only if the `FETCH` command is active
- During PHI2 the IR is "refreshed" (this is not shown in logic circuit)

It should be noted that an inverted operation code (PD) value is fed to the IR input and is also stored on the latch in an inverted form.

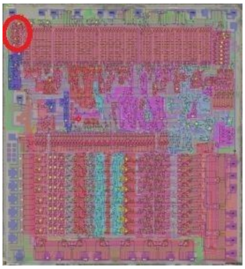
Optimized Schematics



https://github.com/emu-russia/breaks/blob/master/BreakingNESWiki_DeepL/6502/ir.md
18:https://github.com/emu-russia/breaks/blob/master/BreakingNESWiki/imgstore/ir_logic.jpg

Notes in the margins for future revisions of the book.

Extended Cycle Counter

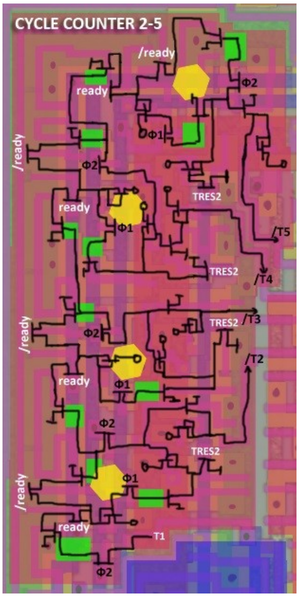


The 6502 has 3 cycle counters:

- The base counter, used for short instructions (Counts T0-T1 cycles)
- Extended counter (which we will talk about here) used for long instructions (Counts cycles T2-T5)
- Counter for very long instructions (Counts cycles RMW T6-T7)

One cycle (T) refers to two consecutive half-cycles (PHI1 + PHI2) of the processor.

Transistor Circuit



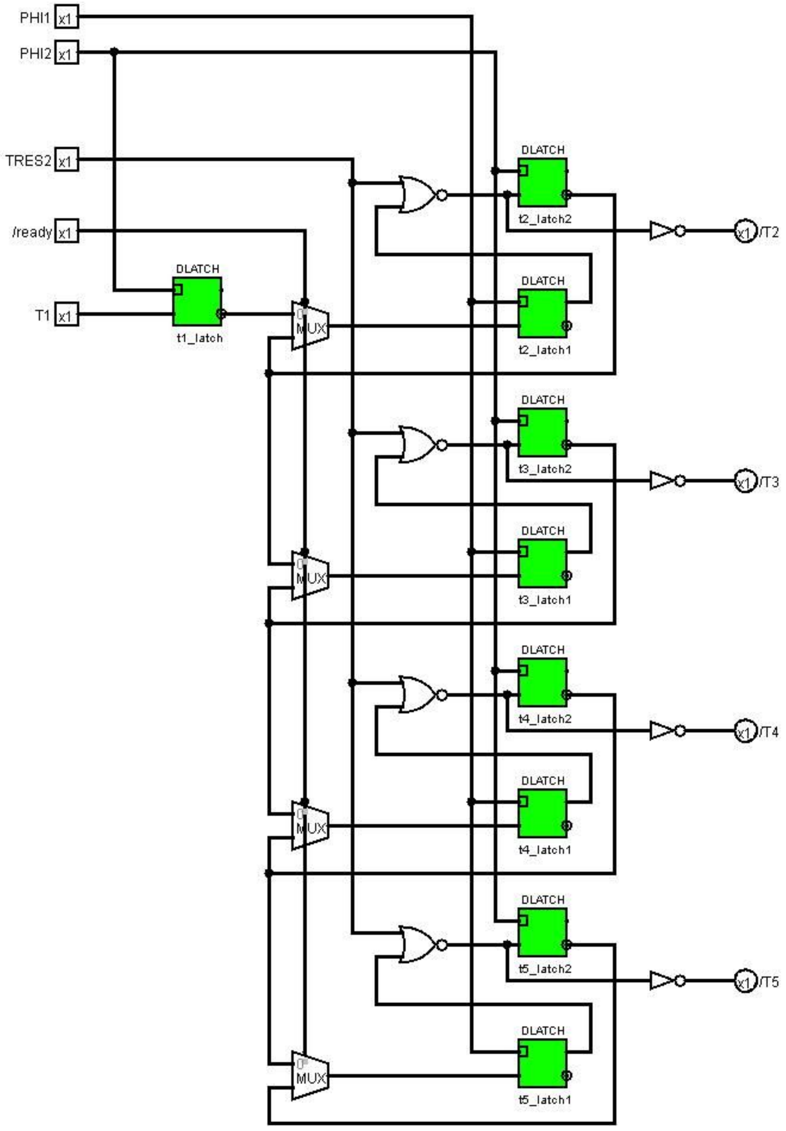
The whole circuit is a shift register, with a control signal `T1` as its input. While the shift register is running, the value of T1 is shifted and goes to the output of `/T2`, then to `/T3` and so on. The /T2-/T5 outputs are in inverse logic.

The shift register is used as a counter for easy transfer of the current cycle (/T2-/T5) to the decoder input.

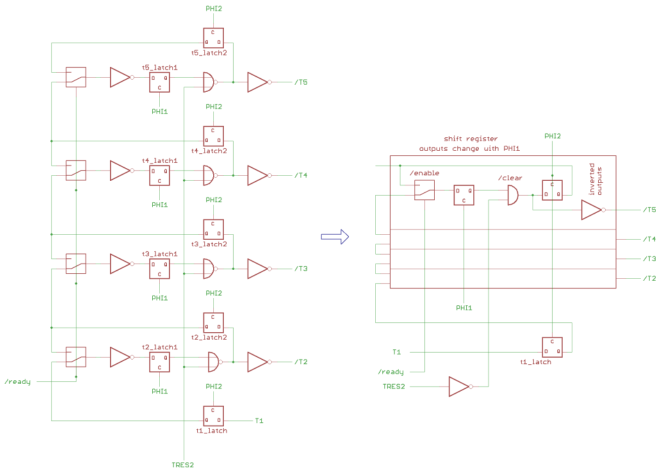
The register is reset by the `TRES2` command and is done after the instruction has been processed.

The circuit includes multiplexers on the `/ready` signal. This is done so that when the processor is not ready (ready=0) - shift register remains in the current state.

Logic


PHI1
x1
PHI2
x1
TRES2
x1
/ready
x1
T1
x1
DLATCH
t1_latch
MUX
DLATCH
t2_latch2
DLATCH
t2_latch1
DLATCH
t3_latch2
DLATCH
t3_latch1
DLATCH
t4_latch2
DLATCH
t4_latch1
DLATCH
t5_latch2
DLATCH
t5_latch1
x1
/T2
x1
/T3
x1
/T4
x1
/T5

Optimized Schematics



https://github.com/emu-russia/breaks/blob/master/BreakingNESWiki_DeepL/6502/extra_counter.md
14:https://github.com/emu-russia/breaks/blob/master/BreakingNESWiki/imgstore/extended_cycle_counter_logic.jpg

Notes in the margins for future revisions of the book.

Decoder

The decoder 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.

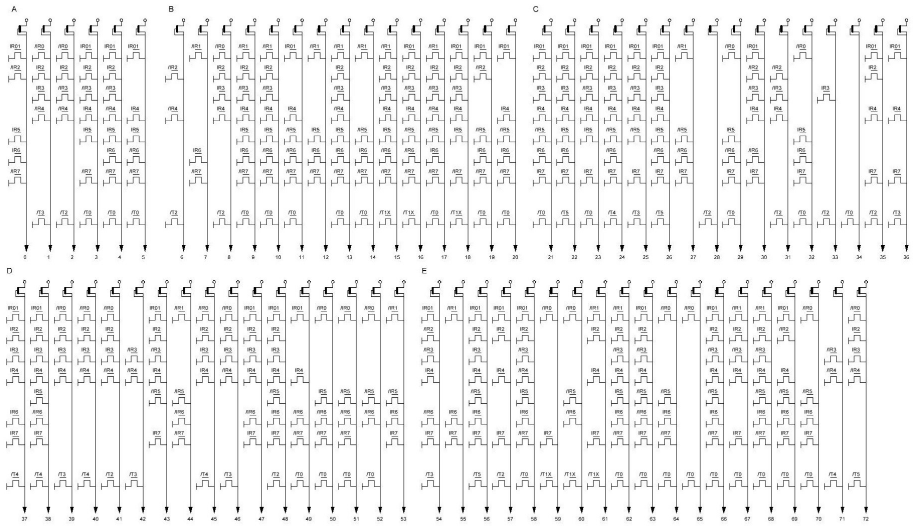
Topologically, the decoder is divided by ground lines into several groups, so we'll stick to the same division, for convenience.

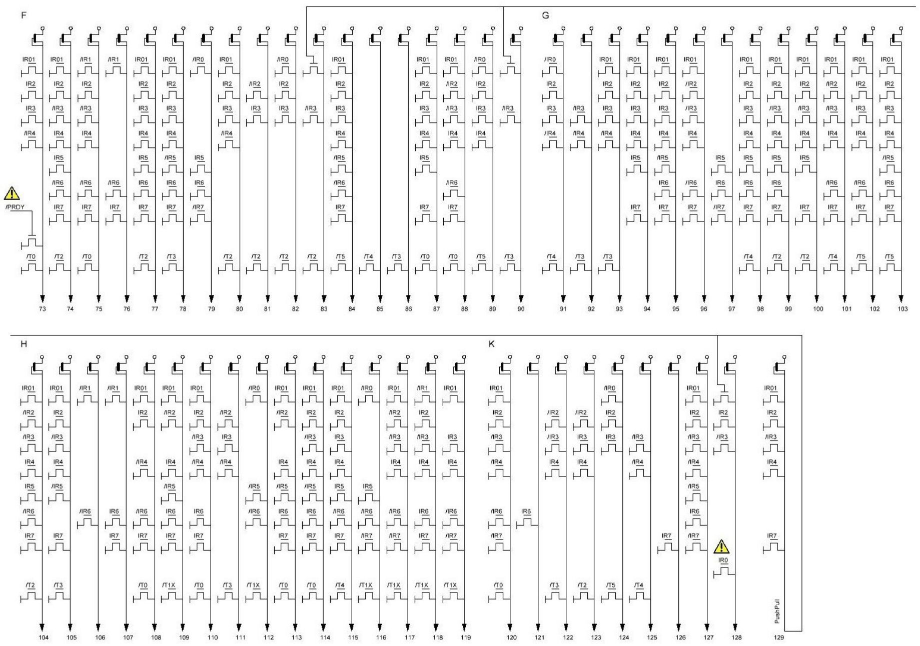
The input signals are:

- /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.





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. Appears on the schematic in duplicate, for different parts of the decoder.
- /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

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

C						
C01	21	011001010101010100000	00100000	T0	JSR	Register control
C02	22	000101010101010100001	00000000	T5	BRK	Register control; Auxiliary signal BRK5
C03	23	010100011001010100000	0X001000	T0	Push	Register control
C04	24	001010010101010100010	01100000	T4	RTS	Register control
C05	25	001000011001010100100	0X101000	T3	Pull	Register control
C06	26	000110010101010100001	01000000	T5	RTI	Register control; Auxiliary signal RTI/5
C07	27	001010000000010010000	011XXX1X	TX	ROR	To obtain an auxiliary /ROR signal for the ADD/SB7 circuit
C08	28	000000000000000001000	XXXXXXXX	T2	T2 ANY	Auxiliary signal T2 (processor is in cycle T2)
C09	29	010110000000011000000	010XXXX1	T0	EOR	ALU Control
C10	30	000010101001010100000	01X01100	TX	JMP (excluder for C11)	ALU Control
C11	31	000000101001000001000	XXX011XX	T2	ALU absolute	ALU Control
C12	32	010101000000011000000	000XXXX1	T0	ORA	ALU Control
C13	33	000000000100000001000	XXXX0XXX	T2	The entire "left" half of the opcode table (values X0-X7)	ALU Control
C14	34	010000000000000000000	XXXXXXXX	T0	T0 ANY	ALU Control
C15	35	000000010001010101000	0XX0X000	T2	BRK JSR RTI RTS Push/pull - stack operations on T2	Regs Control, ALU Control; Auxiliary signal STK2
C16	36	000000000001010100100	0XX0XX00	T3	BRK JSR RTI RTS Push/pull + BIT JMP	ALU Control

D						
D01	37	000001010101010100010	00X00000	T4	BRK JSR	ALU Control
D02	38	000110010101010100010	01000000	T4	RTI	ALU Control
D03	39	000000010101001000100	XXX000X1	T3	OP X, ind	ALU Control
D04	40	000000010110001000010	XXX100X1	T4	OP ind, Y	ALU Control
D05	41	000000010110001001000	XXX100X1	T2	OP ind, Y	ALU Control
D06	42	000000001010000000100	XXX11XXX	T3	RIGHT ODD	ALU Control
D07	43	001000011001010100000	0X101000	TX	Pull	ALU Control
D08	44	001010000000100010000	111XXX1X	TX	INC NOP	ALU Control
D09	45	000000010101001000010	XXX000X1	T4	OP X, ind	ALU Control; Bus Control (DL/DB)
D10	46	000000010110001000100	XXX100X1	T3	OP ind, Y	Bus Control (DL/DB)
D11	47	000010010101010100000	01X00000	TX	RTI RTS	Bus Control (DL/DB); Auxiliary signal RET
D12	48	001001010101010101000	00100000	T2	JSR	Auxiliary signal JSR2
D13	49	010010000001100100000	11X0XX00	T0	CPY CPX INY INX	ALU Control
D14	50	010110000000101000000	110XXXX1	T0	CMP	ALU Control
D15	51	011010000000101000000	111XXXX1	T0	SBC	ALU Control; Auxiliary signal SBC0
D16	52	011010000000001000000	X11XXXX1	T0	ADC SBC	ALU Control
D17	53	001001000000010010000	001XXX1X	TX	ROL	ALU Control

E						
E01	54	000010101001010100100	01X01100	T3	JMP ind	ALU Control
E02	55	000001000000010010000	00XXXX1X	TX	ASL ROL	Bus Control
E03	56	001001010101010100001	00100000	T5	JSR	Auxiliary signal JSR/5
E04	57	000000010001010101000	0XX0X000	T2	BRK JSR RTI RTS Push/	Bus Control
E05	58	010101011010100100000	10011000	T0	TYA	Bus Control
E06	59	100000000000011000000	0XXXXXX1	T1	UPPER ODD	Bus Control
E07	60	101010000000001000000	X11XXXX1	T1	ADC SBC	Bus Control
E08	61	100000011001010010000	0XX0101X	T1	ASL ROL LSR ROR	Bus Control
E09	62	010101011001100010000	1000101X	T0	TXA	Bus Control
E10	63	011010011001010100000	01101000	T0	PLA	Bus Control
E11	64	011001000000101000000	101XXXX1	T0	LDA	Bus Control
E12	65	010000000000001000000	XXXXXXX1	T0	ALL ODD	Bus Control
E13	66	011001011001100100000	10101000	T0	TAY	Bus Control
E14	67	010000011001010010000	0XX0101X	T0	ASL ROL LSR ROR	Bus Control
E15	68	011001011001100010000	1010101X	T0	TAX	Bus Control
E16	69	011001100001010100000	0010X100	T0	BIT0	ALU Control (AND)
E17	70	011001000000011000000	001XXXX1	T0	AND0	ALU Control (AND)
E18	71	000000001010000000010	XXX11XXX	T4	OP abs,XY	Bus Control (ADL/ABL)
E19	72	000000010110001000001	XXX100X1	T5	OP ind,Y	Bus Control (ADL/ABL)

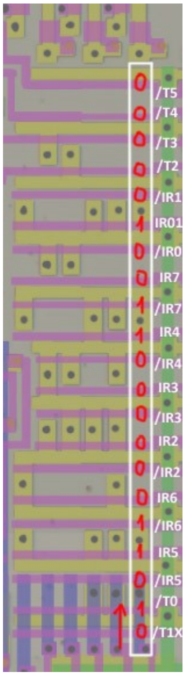
F						
F01	73	010000010110000100000	XXX10000	T0	<- Branch, additionally affected by the /PRDY line (from the RDY contact), immediately on the spot	Auxiliary signal BR0
F02	74	000110011001010101000	01001000	T2	PHA	Bus Control (AC/DB)
F03	75	010010011001010010000	01X0101X	T0	LSR ROR	ALU Control (SR)
F04	76	000010000000010010000	01XXXX1X	TX	LSR ROR	ALU Control (SR)
F05	77	000101010101010101000	00000000	T2	BRK	PC Control
F06	78	001001010101010100100	00100000	T3	JSR	PC Control
F07	79	000101000000101000000	100XXXX1	TX	STA	Auxiliary signal STA
F08	80	000000010110000101000	XXX10000	T2	BR2 (Branch T2)	Схема управления PC и схема инкремента PC
F09	81	000000100100000001000	XXXX01XX	T2	zero page	Bus Control (DL/ADL)
F10	82	000000010100001001000	XXXX00X1	T2	ALU indirect	Bus Control (DL/ADL)
F11	83	000000001000000001000	XXXX1XXX	T2	The entire "right" half of the opcode table (X8-XF values). The Push/Pull opcode exclusion operation is additionally applied to this line, right in place	Auxiliary signal ABS/2
F12	84	001010010101010100001	01100000	T5	RTS	Auxiliary signal RTS/5
F13	85	000000000000000000010	XXXXXXXX	T4	T4 ANY	Bus Control (NOADL)
F14	86	000000000000000000100	XXXXXXXX	T3	T3 ANY	Bus Control (NOADL)
F15	87	010100010101010100000	0X000000	T0	BRK RTI	Bus Control (NOADL)
F16	88	010010101001010100000	01X01100	T0	JMP	Bus Control (NOADL)
F17	89	000000010101001000001	XXX000X1	T5	OP X, ind	Bus Control (NOADL, IND)
F18	90	000000001000000000100	XXXX1XXX	T3	The entire "right" half of the opcode table (X8-XF values). The Push/Pull opcode exclusion operation is additionally applied to this line, right in place	Bus Control (IND)

G						
G01	91	000000010110001000010	XXX100X1	T4	OP ind, Y	Cycle Counter Reset, Bus Control (IND)
G02	92	000000001010000000100	XXX11XXX	T3	RIGHT ODD	Cycle Counter Reset
G03	93	000000010110000100100	XXX10000	T3	BR3 (Branch T3)	PC control circuit and PC increment circuit
G04	94	000100010101010100000	0X000000	TX	BRK RTI	PC Control (JB)
G05	95	001001010101010100000	00100000	TX	JSR	PC Control (JB)
G06	96	000010101001010100000	01X01100	TX	JMP	PC Control (JB), ENDX (Long instruction completion)
P/P	129	000000011001010100000	0XX01000	TX	<- Push/pull opcodes, used as an exclusive for F11 & F18	
G07	97	000101000000100000000	100XXXXX	TX	STORE	For RW Control and to obtain an auxiliary STOR signal
G08	98	000101010101010100010	00000000	T4	BRK	RW Control, !POUT (flags control)
G09	99	000101011001010101000	00001000	T2	PHP	!POUT (flags control)
G10	100	000100011001010101000	0X001000	T2	Push	RW Control, ENDX (Long instruction completion)
G11	101	000010101001010100010	01X01100	T4	JMP ind	ENDX, Bus Control; Auxiliary signal JMP/4
G12	102	000010010101010100001	01X00000	T5	RTI RTS	ENDX (Long instruction completion)
G13	103	001001010101010100001	00100000	T5	JSR	ENDX (Long instruction completion)

H						
H01	104	000110101001010101000	01001100	T2	JMP abs	ENDX (Long instruction
H02	105	001000011001010100100	0X101000	T3	Pull	ENDX (Long instruction
H03	106	000010000000000010000	X1XXXX1X	TX	LSR ROR DEC INC DEX NOP (4x4	Cycle Counter 5-6
H04	107	000001000000010010000	00XXXX1X	TX	ASL ROL	Cycle Counter 5-6, flags
H05	108	010010011010010100000	01X11000	T0	CLI SEI	flags control
H06	109	101001100001010100000	0010X100	T1	BIT	flags control
H07	110	010001011010010100000	00X11000	T0	CLC SEC	flags control
H08	111	000000100110000000100	XXX101XX	T3	Memory zero page X/Y	MemOP
H09	112	101010000000001000000	X11XXXX1	T1	ADC SBC	flags control
H10	113	011001100001010100000	0010X100	T0	BIT	flags control
H11	114	011001011001010100000	00101000	T0	PLP	flags control
H12	115	000110010101010100010	01000000	T4	RTI	flags control
H13	116	100110000000101000000	110XXXX1	T1	CMP	flags control
H14	117	100010101001100100000	11X01100	T1	CPY CPX abs	flags control
H15	118	100001011001010010000	00X0101X	T1	ASL ROL	flags control
H16	119	100010000101100100000	11X00X00	T1	CPY CPX zpg/immed	flags control

K						
P/P	129	000000011001010100000	0XX01000	TX	<- Push/pull opcodes, used as an exclusive for K09	
K01	120	010010011010100100000	11X11000	T0	CLD SED	flags control
K02	121	000001000000000000000	X0XXXXXX	TX	/IR6	Branch Logic
K03	122	000000101001000000100	XXX011XX	T3	Memory absolute	MemOP
K04	123	000000100101000001000	XXX001XX	T2	Memory zero page	MemOP
K05	124	000000010100001000001	XXXX00X1	T5	Memory indirect	MemOP
K06	125	000000001010000000010	XXX11XXX	T4	Memory absolute X/Y	MemOP
K07	126	000000000000010000000	0XXXXXXX	TX	/IR7	Branch Logic
K08	127	001001011010100100000	10111000	TX	CLV	flags control
K09	128	000000011000000000000	XXXX10X0	TX	IMPL. The Push/Pull opcode exclusion operation is additionally applied to this line, right on the spot. Also, the mask for this line does not take into account the & ~IR0 operation	Bus Control (DL/DB)

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: 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.

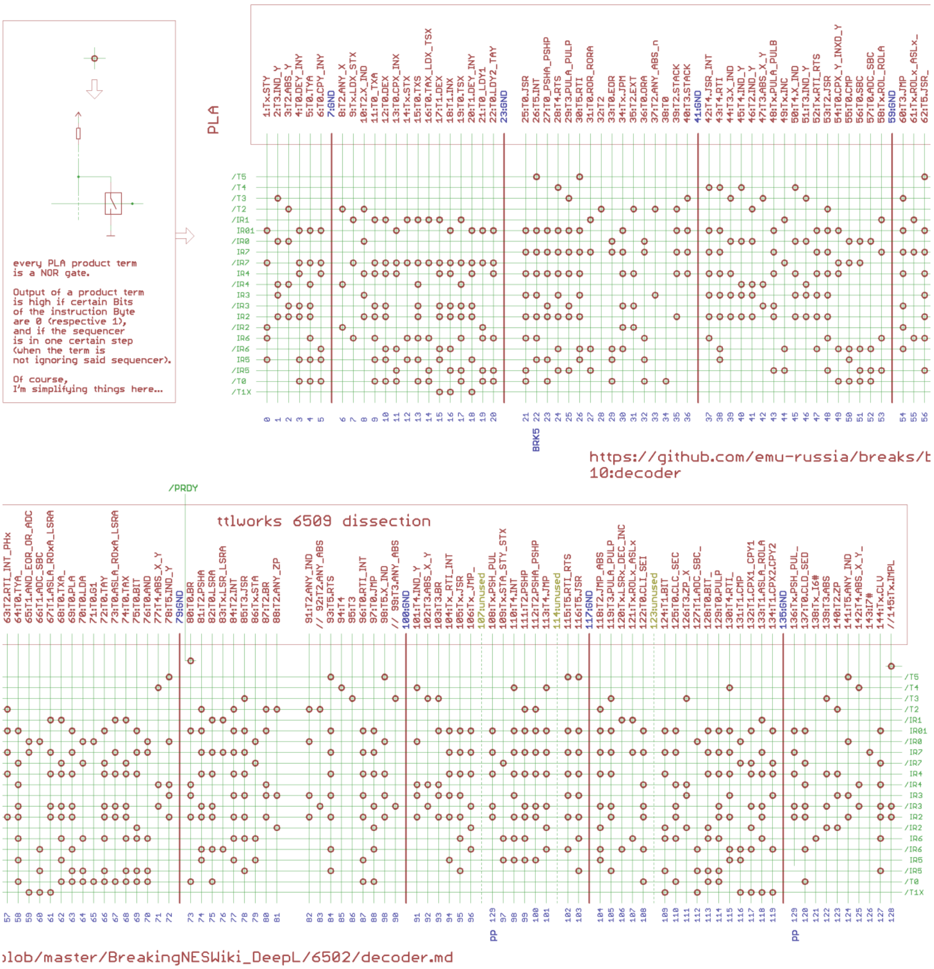
Why the decoder is so big and scary

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.

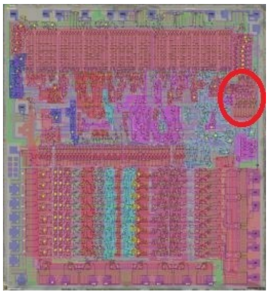
In other words - it is not random logic that adjusts to decoder, but vice versa. The impression that the decoder is "more important" is formed simply because it is above random logic.

Decoder by ttlworks



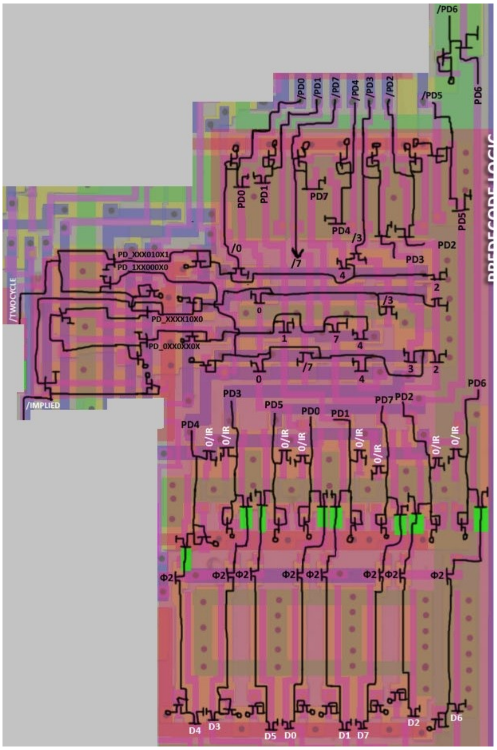
Notes in the margins for future revisions of the book.

Predecode



The circuit is designed to define the "class" of an instruction:

- A short instruction which is executed in 2 clock cycles (`TWOCYCLE`)
- An instruction of type `IMPLIED` which has no operands (takes 1 byte in memory)



The operation code received from the external data bus (D0...D7) is stored on the PREDECODE latch (PD) during PHI2 (in inverted form), after which the precoding logic immediately determines the instruction class (the circuit is combinatorial).

The output `/TWOCYCLE` is used by a short cycle counter. The output `/IMPLIED` is used by the PC increment logic.

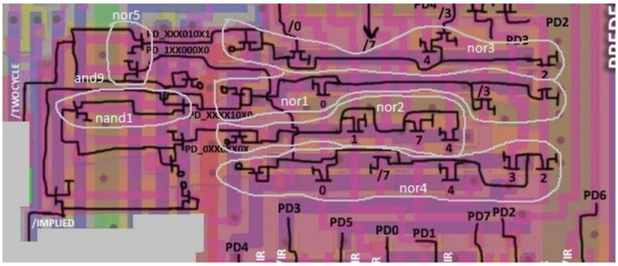
The PD latch value is fed to the *instruction register* input in inverted form.

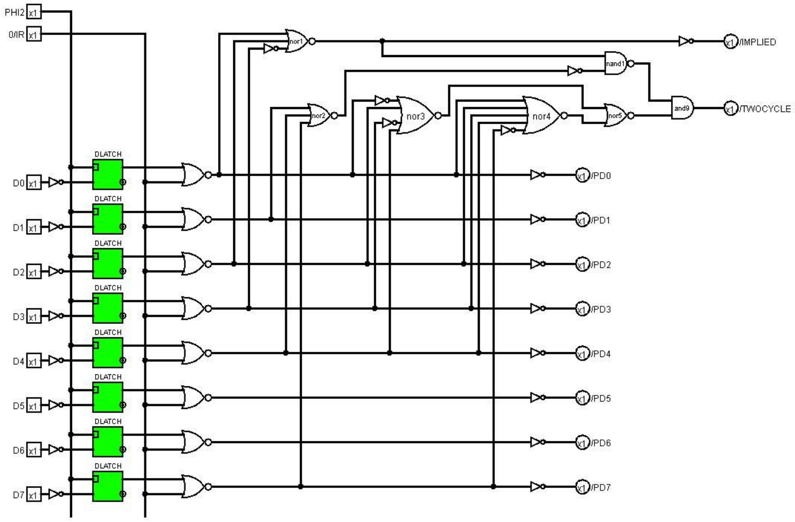
Also the control line `0/IR` is fed to the Predecode logic input which "injects" the BRK operation into the instruction stream. This occurs during interrupt processing, to initialize the BRK sequence (all interrupts simply mimic the BRK instruction, with slight modifications).

The pre-decode circuit works closely with the *dispatcher*, all control signals go there.

Logic

The corresponding gates are marked on the transistor schematic:

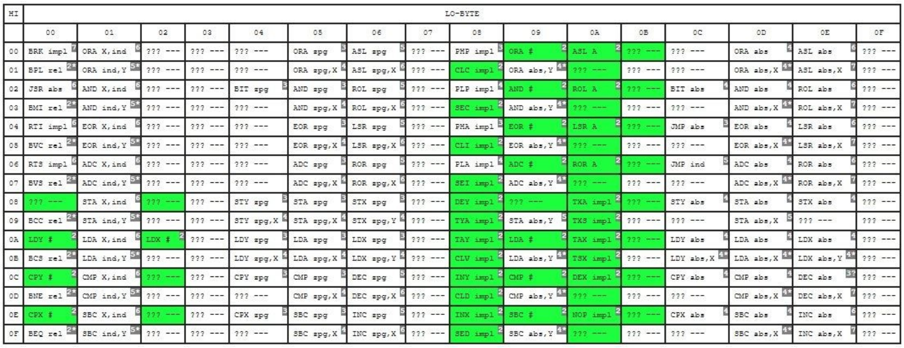




The predecoding logic is self-descriptive:

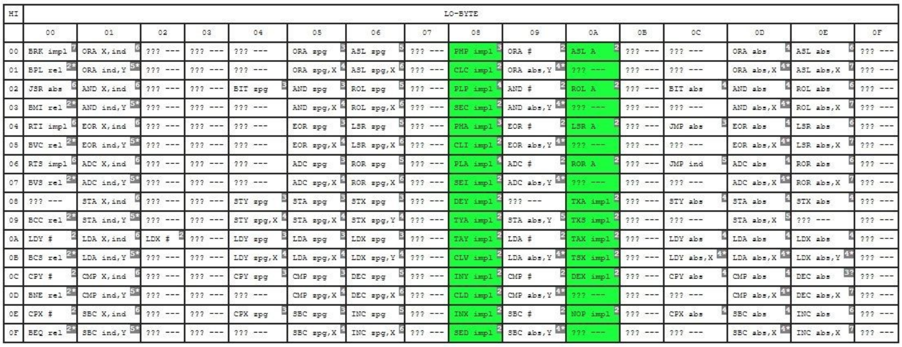
- 2-cycle instructions are: Direct operand instructions OR all single-byte instructions EXCEPT push/pull instructions (specified by mask XXX010X1 + 1XX000X0 + XXXX10X0 - 0XX0XX0X)
- Single-byte instructions are set by mask XXXX10X0

TWOCYCLE instructions:



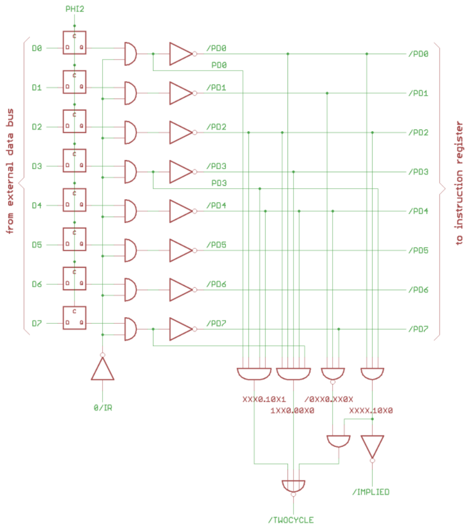
HI	LO-BYTE															
	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
00	BRK impl 7	ORA X,ind 6	??? ---	??? ---	??? ---	ORA zpg 3	ASL zpg 5	??? ---	PHP impl 3	🟩 ORA # 2	🟩 ASL A 2	🟩 ??? ---	??? ---	ORA abs 4	ASL abs 6	??? ---
01	BPL rel 2*	ORA ind,Y 5*	??? ---	??? ---	??? ---	ORA zpg,X 4	ASL zpg,X 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	??? ---

IMPLIED instructions:



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	??? ---

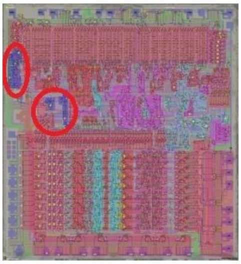
Optimized Schematics



https://github.com/emu-russia/breaks/blob/master/BreakingNESWiki_DeepL/6502/predecode.md
24:https://github.com/emu-russia/breaks/blob/master/BreakingNESWiki/imgstore/predecode_logic.jpg

Notes in the margins for future revisions of the book.

Interrupt Processing



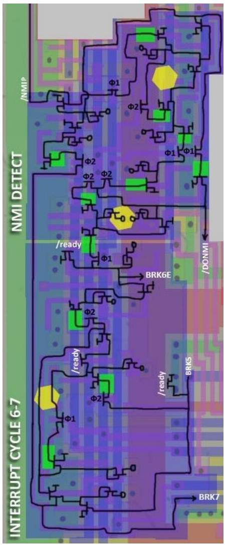
Interrupt processing includes the following circuits:

- NMI edge detection
- Cycle counter 6-7 for interrupt handling
- Setting the low-order bits of the interrupt vector address (ADL0-3)
- Circuit for issuing internal signal `DORES`.
- B Flag

Three signals `/NMIP`, `/IRQP` and `RESP` come to the input of the circuits from the corresponding input pads.

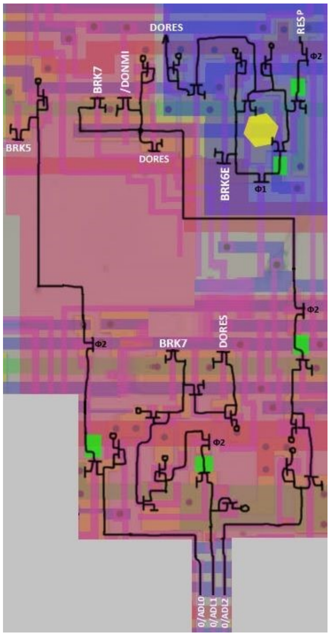
NMI Processing

Transistor circuit (includes cycle counter 6-7 and NMI edge detector):



Interrupt vector address and Reset FF

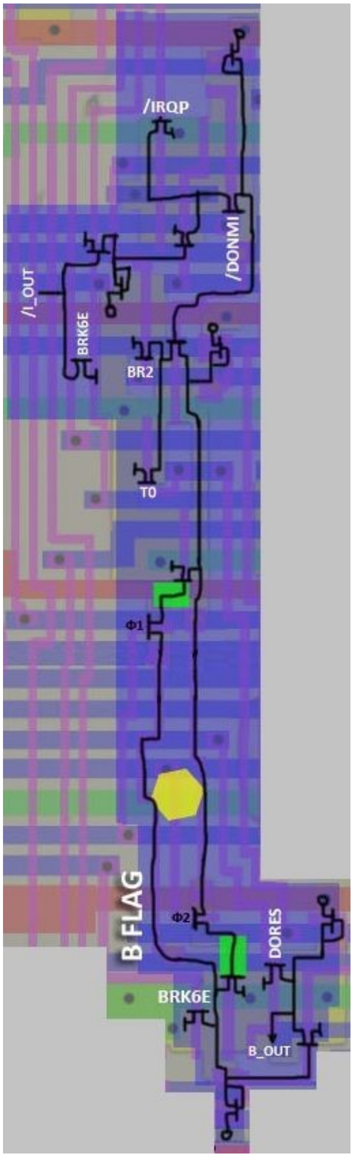
Transistor circuit:



The circuit for getting the control signal `DORES` ("Do Reset") (which is binned to all other internals) is combined here with the interrupt vector setting circuit to save space.

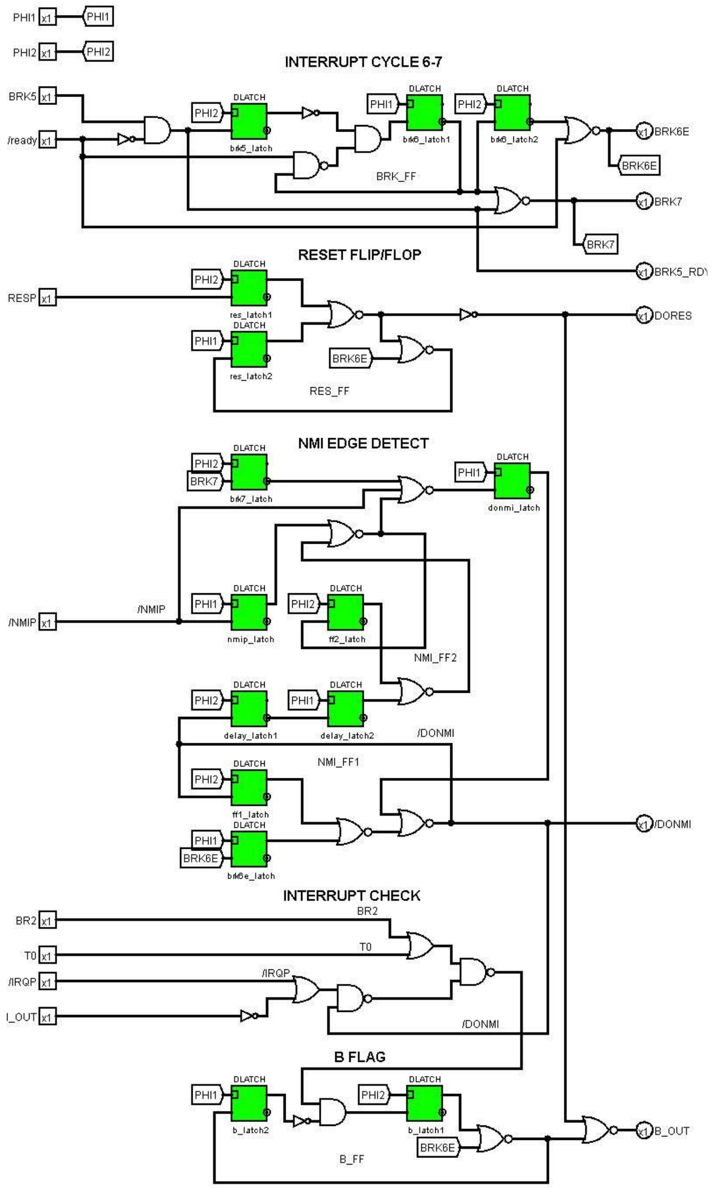
B Flag

Transistor circuit:



Logic

Interrupt handling schematic:



To handle interrupts an additional circuit is required to generate cycles 6 and 7 (because they do not come from the decoder) (control signals `BRK6E` and `BRK7`). And the control signal BRK6E ("Break Cycle 6 End") starts during PHI2 of cycle 6 and ends during PHI1 of cycle 7. This is done to determine the edge of the /NMI signal.

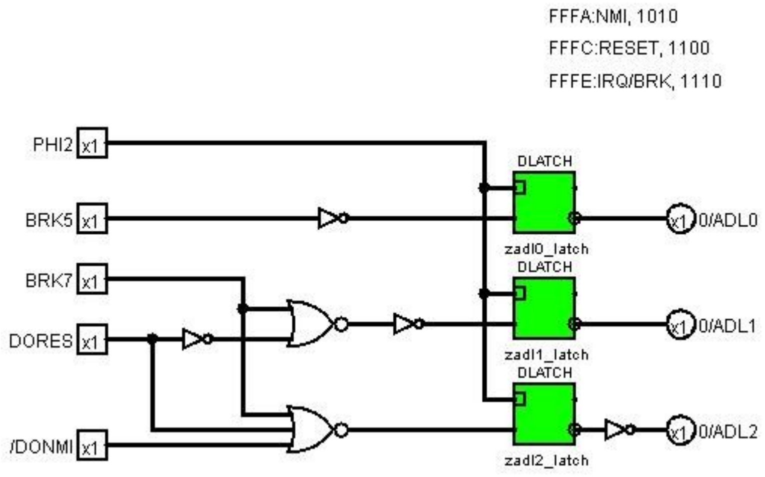
The detection of the /NMI edge is done by a classic edge detection circuit based on two RS triggers.

The /RES signal is additionally stored on RESET FLIP/FLOP, because it is required for other random logic circuits (particularly for special control of the R/W pin).

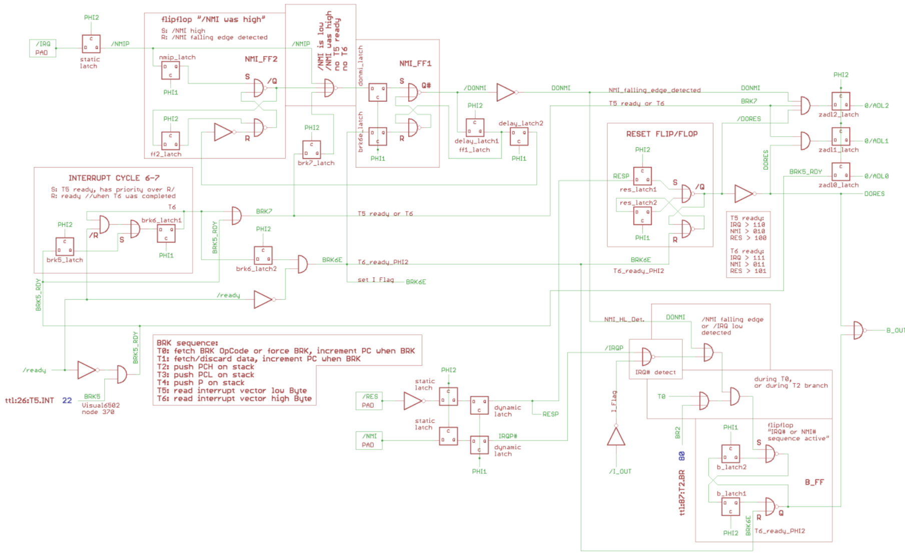
The arrival of any interrupt is reflected on flag B, the output of which (`B_OUT`) forces the processor to execute a BRK instruction (operation code 0x00). This way the developers have unified the handling of all interrupts.

The last small circuit forms the address (or vector) of the interrupts (control signals 0/ADL0, 0/ADL1 and 0/ADL2), which control the lowest 3 bits of the address bus.

Schematic for setting the address of the interrupt handler:



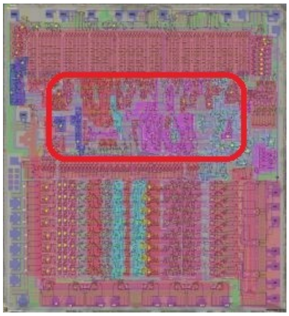
Optimized Schematics



https://github.com/emu-russia/breaks/blob/master/BreakingNESWiki_DeepL/6502/interrupts.md
17:interrrupt_logic

Notes in the margins for future revisions of the book.

Random Logic



The name has nothing to do with random numbers, it simply reflects the essence of randomly scattered circuits here and there.

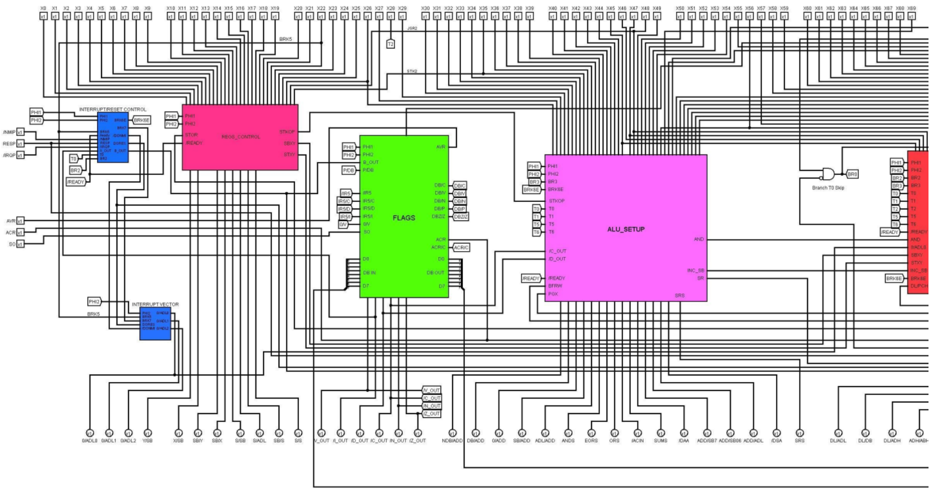
This logic is the thinking organ of the processor and completely determines its behavior when processing and executing instructions.

From the hardware point of view, the random logic is a "handmade" product of MOS engineers, which is a mess of transistors and wires. Therefore, it would be more correct to use the name "chaotic logic" instead of random logic.

There is no need to give a full-size transistor circuit here, because it will be easier to master it by component parts.

Below you can see all the function blocks of the random logic:

- Register control
- ALU control
- Program counter (PC) control
- Bus control
- Execution logic (dispatch)
- Flags control logic
- Flags
- Conditional branch logic

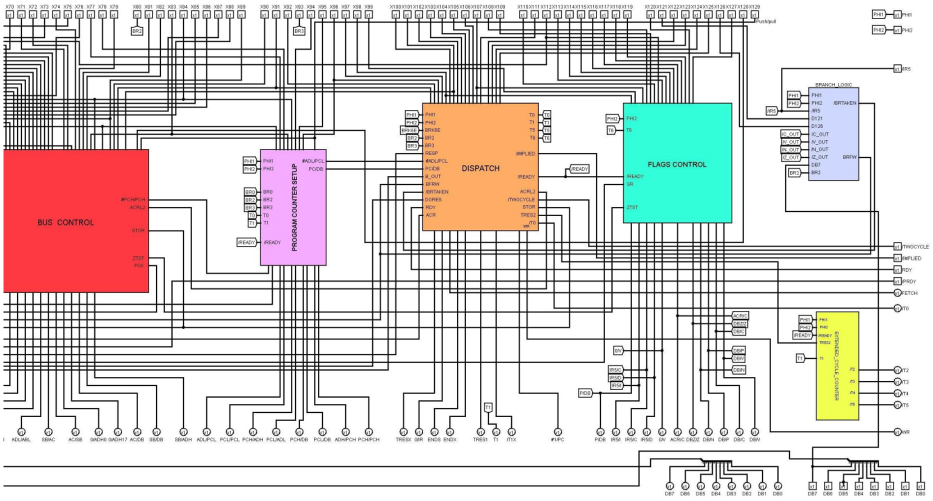


Principle of Operation

In general, the operation of the logic is quite complex (did you think I would say simple again? :smiley:):

- The execution logic (dispatch) conducts the work of the entire processor. It determines when to terminate an instruction and also controls the PC increment and the cycle counter. Additionally it includes a processor readiness circuit (RDY) which is controlled by the RDY pin.
- After the execution logic has started executing the next instruction - the code of that instruction as well as the current cycle is fed to the decoder
- Depending on the results of decoding the control circuitry of registers, ALU, PC and buses give outward to the lower part special *control commands*
- Additionally, the behavior of the processor is affected by its flags as well as interrupt handling logic. And flags are also affected by executable instructions.

All this is closely coupled to control the lower part of the processor, where its context (registers), ALU and communication with the outside world via buses are located.



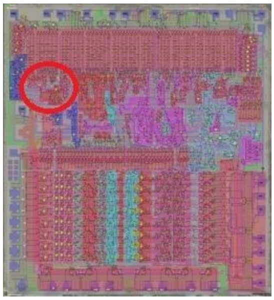
Notes in the margins for future revisions of the book.

Auxiliary Signals

This section contains a table of auxiliary signals exchanged between all parts of the random logic (for reference):

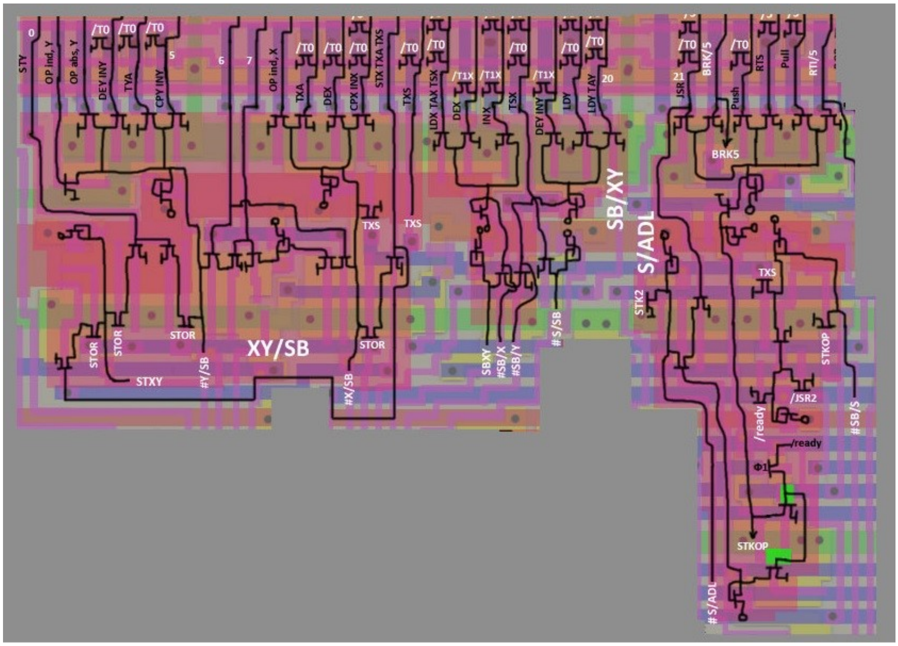
Name	From	To	Description
ACRL1	Dispatch	Dispatch	One of the ACR Latch outputs
ACRL2	Dispatch	Bus Control	One of the ACR Latch outputs
AND	ALU Control	Bus Control	Used when forming an ALU ANDS command
BR2	Decoder	PC Control, PC Increment	Branch T2
BR3	Decoder	PC Control, PC Increment	Branch T3
BRFW	Branch Logic, ALU Control	PC Control	Branch forward (whenever taken)
BRK5	Decoder	Interrupts, Regs Control	Used to obtain the STKOP signal and also goes into the *interrupt handling* circuit
BRK6E	Interrupts	ALU Control, Bus Control	BRK6 (cycle 6 of the interrupt sequence), during the half-step PHI2
/BRTAKEN	Branch Logic	PC Control	Branch taken
C_OUT	Flags	ALU Control	Flag C value
/C_OUT	Flags	ALU Control	Flag C value (inverted)
DL/PCH	PC Control	Bus Control	Intermediate signal
D_OUT	Flags	ALU Control	Flag D value
JSR2	Decoder	Regs Control, ALU Control, Bus Control	To obtain the JSXY signal and other bus control circuits
/JSR2	Bus Control	Regs Control	Intermediate signal, JSR2 inversion
IMPL	Decoder	ALU Control	Decoder X128. Additionally modified with Push/Pull (X129) and IR0 signals.
INC_SB	ALU Control	Bus Control	Intermediate signal ("Increment SB")
NOADL	Bus Control	ALU Control	Intermediate signal ("No ADL")
PC/DB	PC Control	Dispatch	Auxiliary output signal for the RW Control circuit that is part of the dispatcher
PGX	Bus Control	ALU Control	Intermediate signal ("Page X")
/ready	Dispatch	All	Global internal processor readiness signal
RTI/5	Decoder	Regs Control, ALU Control	Used to obtain STKOP and NOADL signals
SBXY	Regs Control	Bus Control	Intermediate signal ("SB Bus X,Y")
STK2	Decoder	Regs Control, ALU Control	Auxiliary signal from decoder (X35)
STKOP	Regs Control	ALU Control	Intermediate signal ("Stack Operation")
STOR	Dispatcher	Regs Control, ALU Control, RW Control	Intermediate signal
STXY	Regs Control	Bus Control	Intermediate signal ("Store X,Y")
T0	Short Cycle Counter	All	Processor in the T0 instruction execution cycle
T1	PC Control	All	Processor in the T1 cycle
T2	Decoder	All	Processor in the T2 cycle
T5	Long Cycle Counter	All	Processor in cycle RMW T6 (the name \`T5\` is the old name of the signal, but we will not rename it anymore)
T6	Long Cycle Counter	All	Processor in cycle RMW T7 (the name \`T6\` is the old name of the signal, but we will not rename it anymore)
ZTST	Bus Control	Flags Control	Intermediate signal ("Z Test")

Registers Control



Most likely this control circuit will be observed first, so I will write here: be prepared to see a large number of intermediate signals in the control circuits, which can come sometimes from all other parts of the random logic. A summary table of all the intermediate signals can be found in the main section with the *random logic* overview.

The register control circuit is responsible for generating *control commands* to exchange registers with the internal buses.



Inputs:

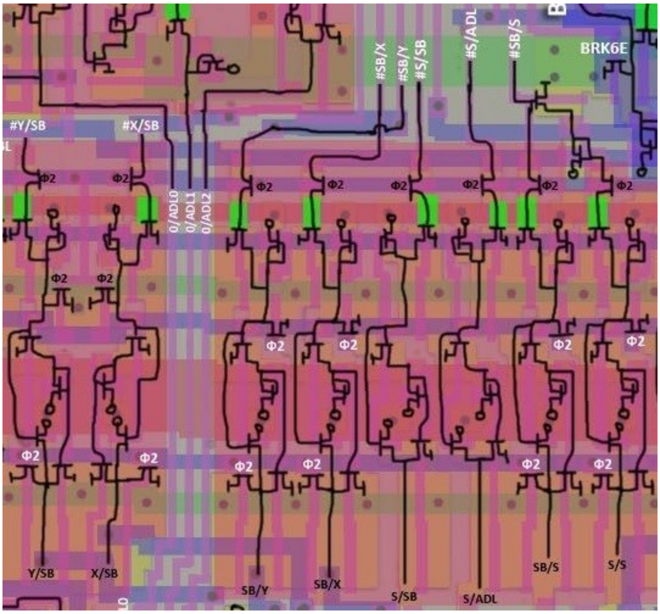
Signal	Description
X0-X26	Outputs from the decoder
/JSR2	Intermediate signal from the bus control circuit
STK2	Just an auxiliary signal from another part of the decoder (X35)
STOR	Auxiliary signal from the *dispatcher* circuit
/ready	Global processor readiness signal

Outputs:

Signal	Description
SBXY	Intermediate signal for *bus control circuitry.* This signal is actually in inverse logic (#SBXY)
STXY	Intermediate signal for bus control circuitry
STKOP	Intermediate signal ("Stack Operation") for the *ALU control circuit*
#Y/SB	Intermediate signal to latch, to obtain a Y/SB command
#X/SB	Intermediate signal to latch, to obtain a X/SB command
#SB/X	Intermediate signal to latch, to obtain a SB/X command
#SB/Y	Intermediate signal to latch, to obtain a SB/Y command
#S/SB	Intermediate signal to latch, to obtain a S/SB command
#S/ADL	Intermediate signal to latch, to obtain a S/ADL command
#SB/S	Intermediate signal to latch, to obtain a SB/S command
BRK5	Output X22 from decoder. Used to obtain the STKOP signal and also goes to the *interrupt circuitry*
RTI/5	Output X26 from decoder. Used to obtain STKOP and NOADL signals

The `TXS` (X13) signal is used within this circuit and does not go outside.

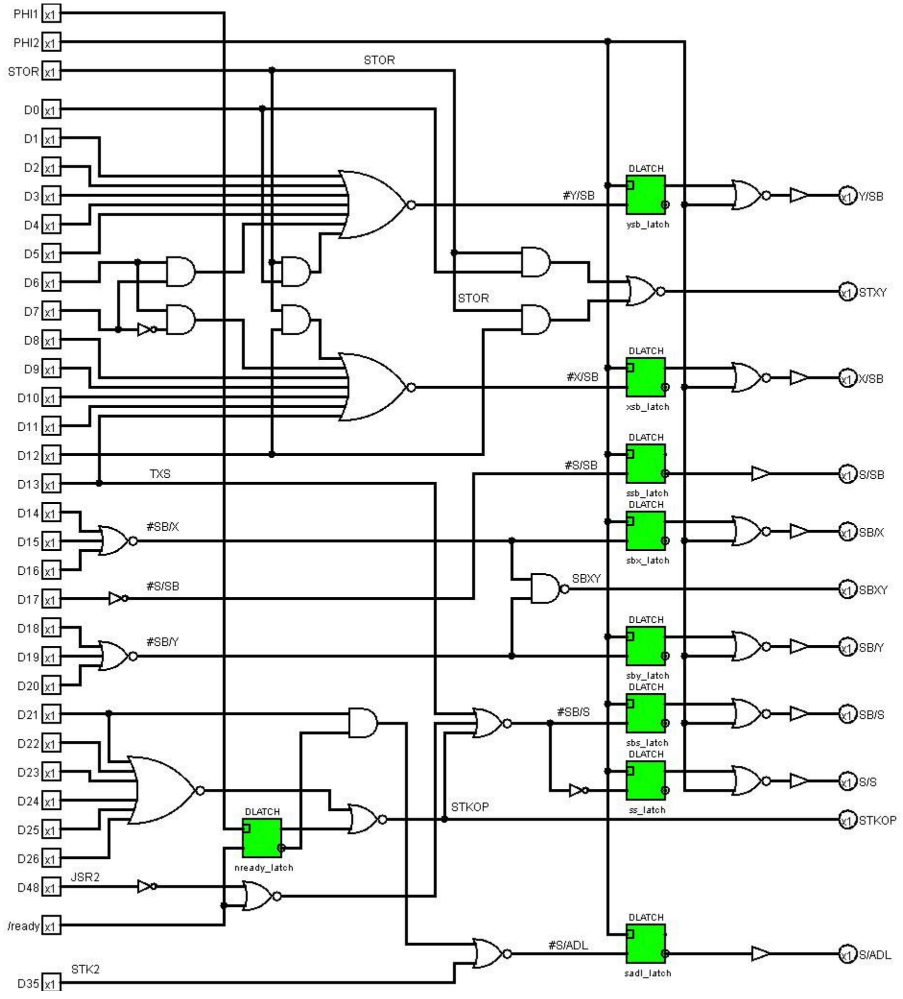
The intermediate signals from the register control circuitry go to the input of the control command latches:



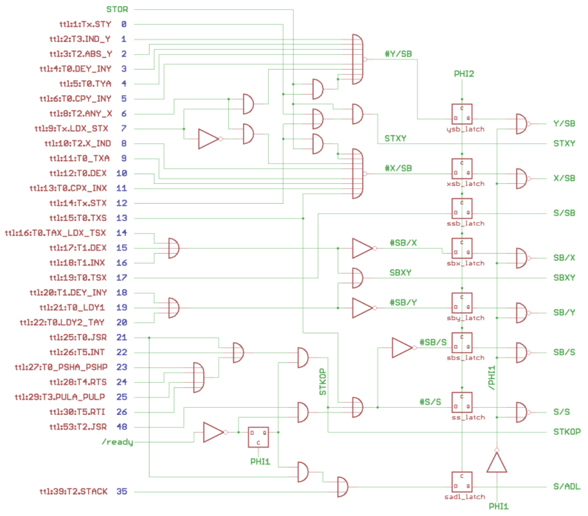
Register control commands:

Command	Description
X/SB	Place the value of register X on the SB bus
Y/SB	Place the value of register Y on the SB bus
SB/X	Place the SB bus value on the X register
SB/Y	Place the SB bus value on the Y register
S/SB	Place the value of register S on the SB bus
S/ADL	Place the value of register S on the ADL bus
SB/S	Place the SB bus value on the S register
S/S	Refresh the value of the S register. The S/S control command is obtained by a complement of the SB/S signal (active when the SB/S command is inactive)

Logic


PHI1
PHI2
STOR
D0
D1
D2
D3
D4
D5
D6
D7
D8
D9
D10
D11
D12
D13
D14
D15
D16
D17
D18
D19
D20
D21
D22
D23
D24
D25
D26
D48
/ready
D35
x1
STOR
#Y/SB
DLATCH
ysb_latch
Y/SB
STXY
#X/SB
xsb_latch
X/SB
TXS
#S/SB
ssb_latch
S/SB
#SB/X
sbx_latch
SB/X
SBXY
#SB/Y
sby_latch
SB/Y
#SB/S
sbs_latch
SB/S
ss_latch
S/S
STKOP
nready_latch
JSR2
#S/ADL
sadl_latch
S/ADL
STK2

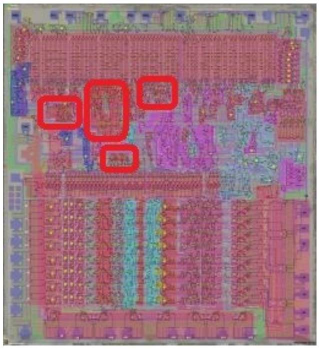
Optimized Schematics



https://github.com/emu-russia/breaks/blob/master/BreakingNESWiki_DeepL/6502/regs_control.md
26:https://github.com/emu-russia/breaks/blob/master/BreakingNESWiki/imgstore/logisim/regs_control_logisim.jpg

Notes in the margins for future revisions of the book.

ALU Control



The *ALU* control is designed to generate ALU *control commands*.

Intermediate Signals

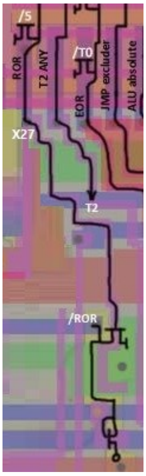
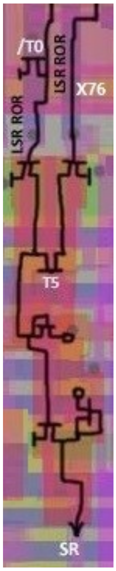
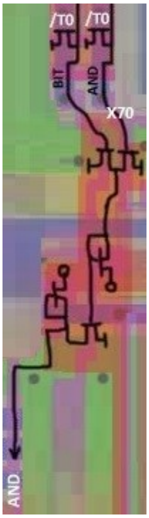
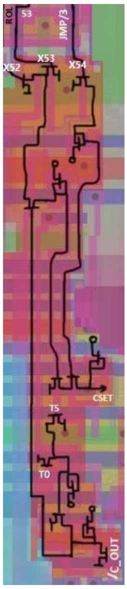
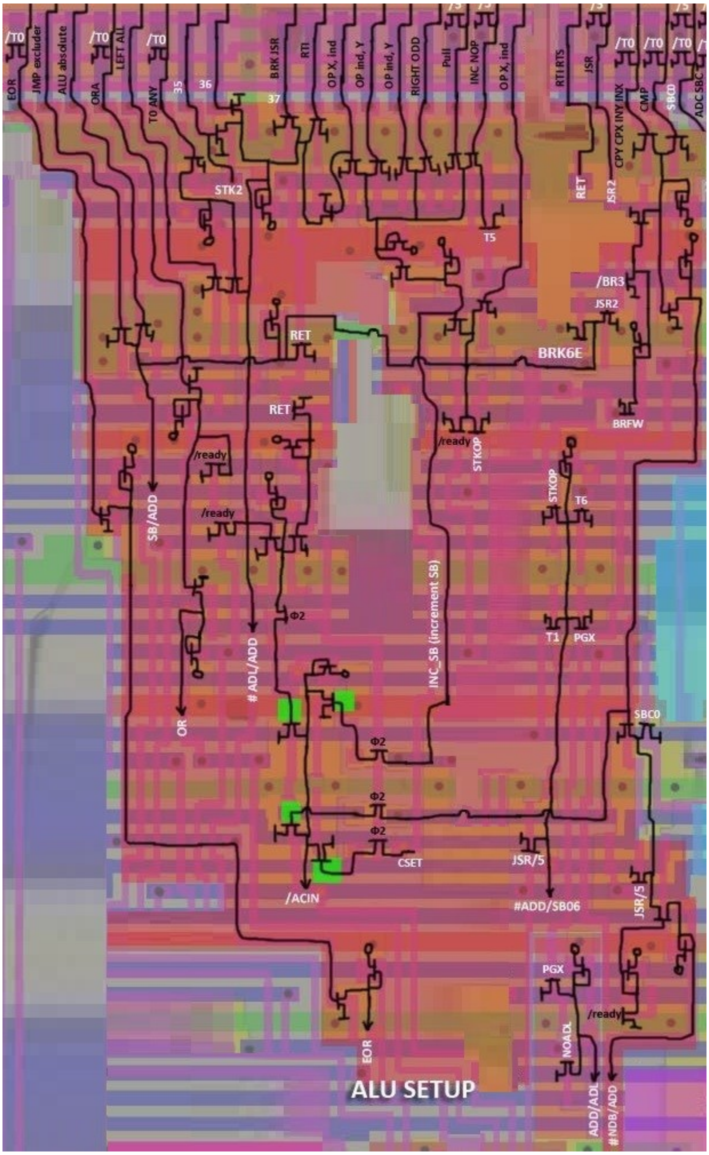
/ROR	SR	AND	CSET
 /5 ROR T2 ANY /T0 EOR JMP excluder ALU absolute X27 T2 /ROR			 ROL 53 JMP/3 X52 X53 X54 CSET T5 T0 /C_OUT

Table of auxiliary and intermediate signals, which are found further in the schematics:

Signal	Description
/ROR	Intermediate signal, used in the ADD/SB7 circuit
SR	Intermediate signal
AND	Intermediate signal
T0	Comes from the cycle counter of short instructions
T5	Comes from the cycle counter of long instructions
/C_OUT	*Flag* C value (inverted value)
CSET	Intermediate signal ("Carry Set"), used in the main ALU control circuit
STK2	Decoder X35
RET	Decoder X47
SBC0	Decoder X51
JSR2	Decoder X48
/BR3	Decoder X93 (inverted value). The inversion circuit was lost somewhere in the optimization process.
BRK6E	Comes from the *interrupts processing* circuit
STKOP	Comes from *register control* circuitry
/ready	Global internal processor readiness signal
INC_SB	Intermediate signal ("Increment SB"), used in the main control circuitry as well as in the *bus control* circuitry
JSR/5	Decoder X56
PGX	Comes from the bus control circuitry
NOADL	Comes from the bus control circuitry
BRFW	Comes from the conditional *branch logic*
T1	Comes from the PC increment circuit (see *dispatcher*)
T6	Comes from the cycle counter of long instructions
D_OUT	Flag D value
C_OUT	Flag C value

ALU Control (Main Part)

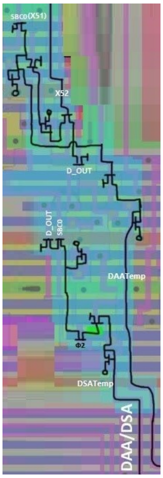
The circuit is a mess of gates and 4 latches to generate the input carry for the ALU (control signal /ACIN).



BCD Correction Control

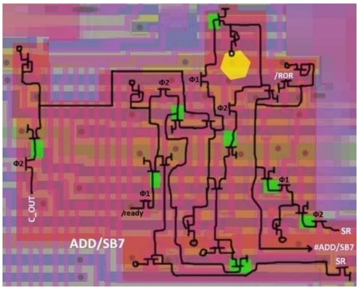
BCD correction is applied in the following cases:

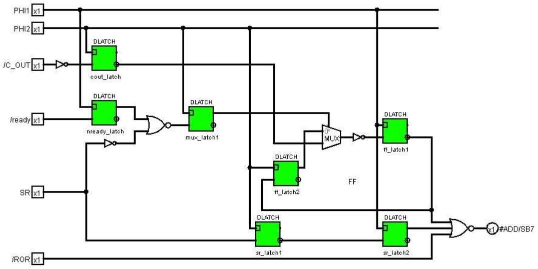
- If the BCD mode is enabled with flag D and the current instruction `SBC` (control signal DSATemp)
- If the BCD mode is enabled with flag D and the current instruction `ADC` (control signal DAATemp)



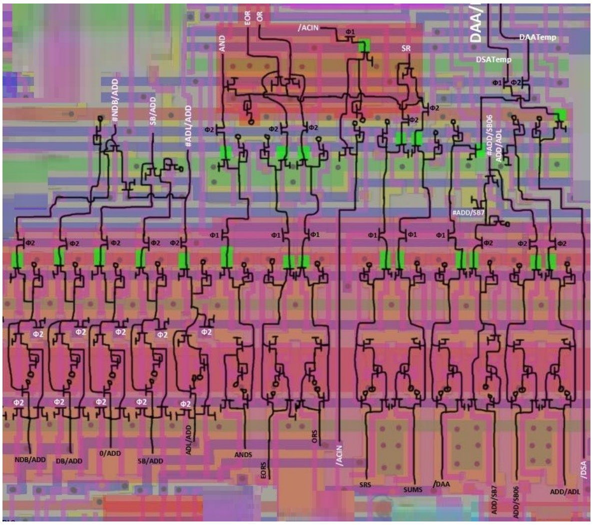
ADD/SB7

The attentive reader will notice that the processor has support for bit rotation instructions (ROL/ROR). The additional processing associated with these instructions is just handled by this circuit.



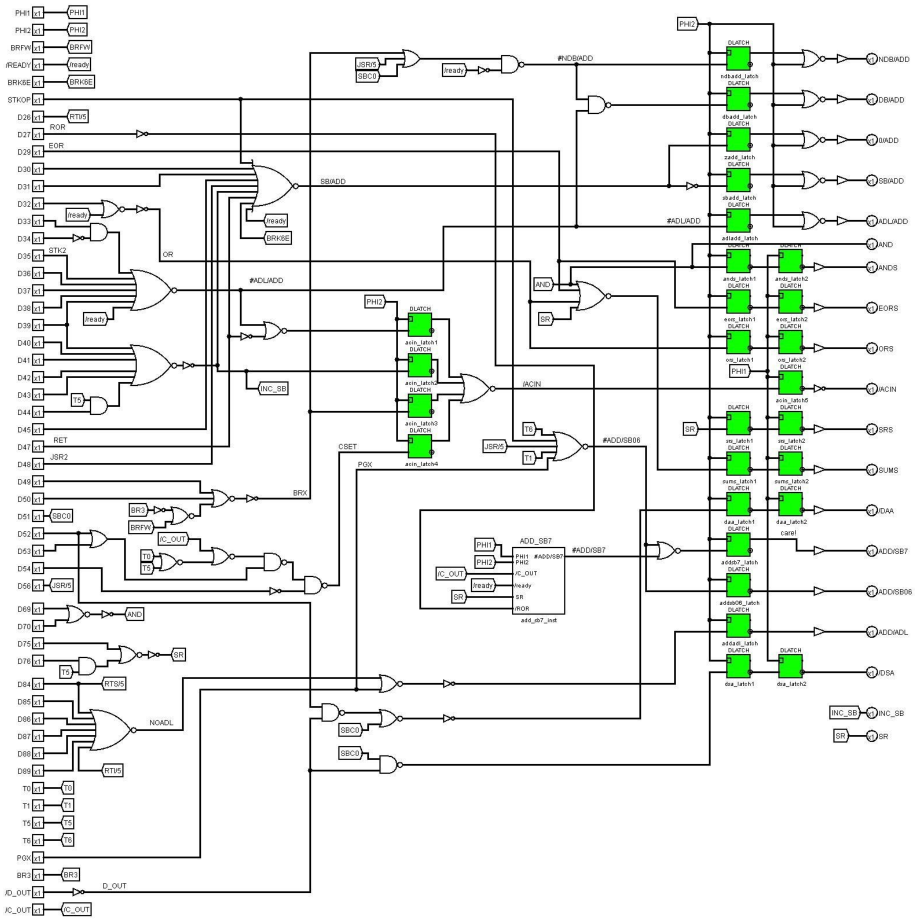


ALU Control Commands

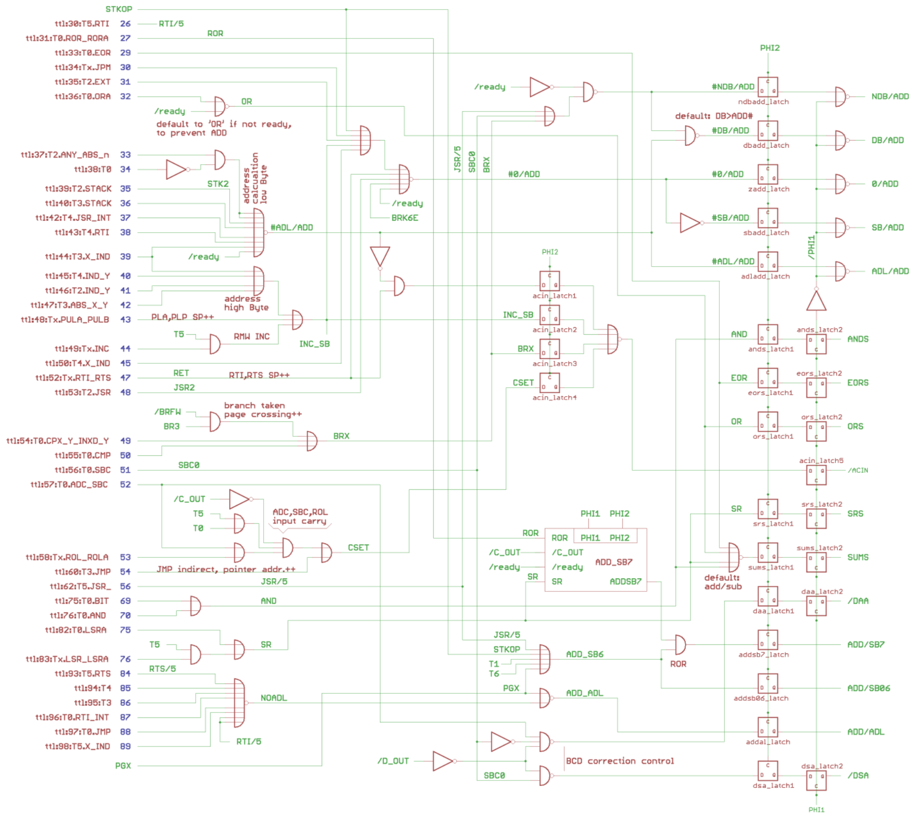


Command	Description
Setting the ALU input values	
NDB/ADD	Load inverse value from DB bus to the BI latch
DB/ADD	Load direct value from DB bus to the BI latch
0/ADD	Write 0 to the AI latch
SB/ADD	Load a value from the SB bus to the AI latch
ADL/ADD	Load a value from the ADL bus to the BI latch
ALU operation commands	
ANDS	Logical AND operation (AI & BI)
EORS	Logical XOR operation (AI ^ BI)
ORS	Logical OR operation (AI | BI)
SRS	Shift Right
SUMS	Summation (AI + BI)
Control commands of the intermediate ALU result	
ADD/SB06	Place the value of the ADD latch on the SB bus (bits 0-6)
ADD/SB7	Place the value of the ADD latch on the SB bus (bit 7)
ADD/ADL	Place the ADD latch value on the ADL bus
Additional signals	
/ACIN	Input carry
/DAA	Perform correction after addition
/DSA	Perform correction after subtraction

Logic



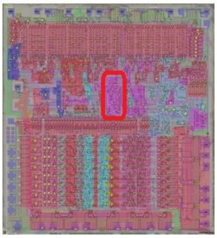
Optimized Schematics



https://github.com/emu-russia/breaks/blob/master/BreakingNESWiki_DeepL/6502/alu_control.md
28:https://github.com/emu-russia/breaks/blob/master/BreakingNESWiki/imgstore/logisim/alu_control_logisim.jpg

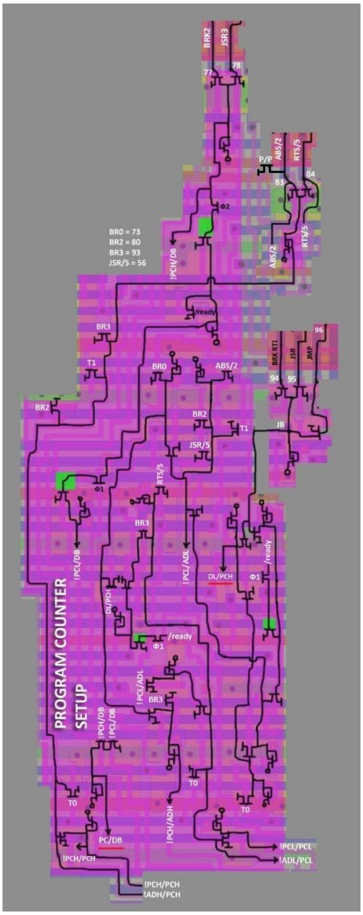
Notes in the margins for future revisions of the book.

Program Counter Control

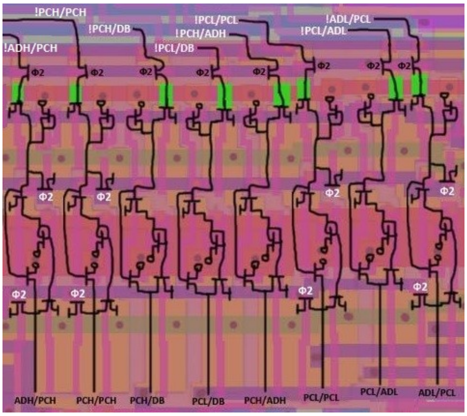


The program counter (PC) control circuitry is designed to generate *control commands* to exchange the PC value and the internal buses ADL, ADH and DB.
Nearby is the PC increment circuit, which is discussed in another section on *dispatcher*.

Transistor circuit for obtaining intermediate signals:



Output latches and control commands:



Inputs:

Signal	Description
BR0	Decoder X73. Additionally modified with the /PRDY signal
BR2	Decoder X80
BR3	Decoder X93
T0	Comes from the cycle counter of short instructions
T1	Comes from the PC increment circuit (see dispatcher)
ABS/2	Decoder X83. Additionally modified with Push/Pull signal (X129)
RTS/5	Decoder X84
JSR/5	Decoder X56
/ready	Global internal processor readiness signal

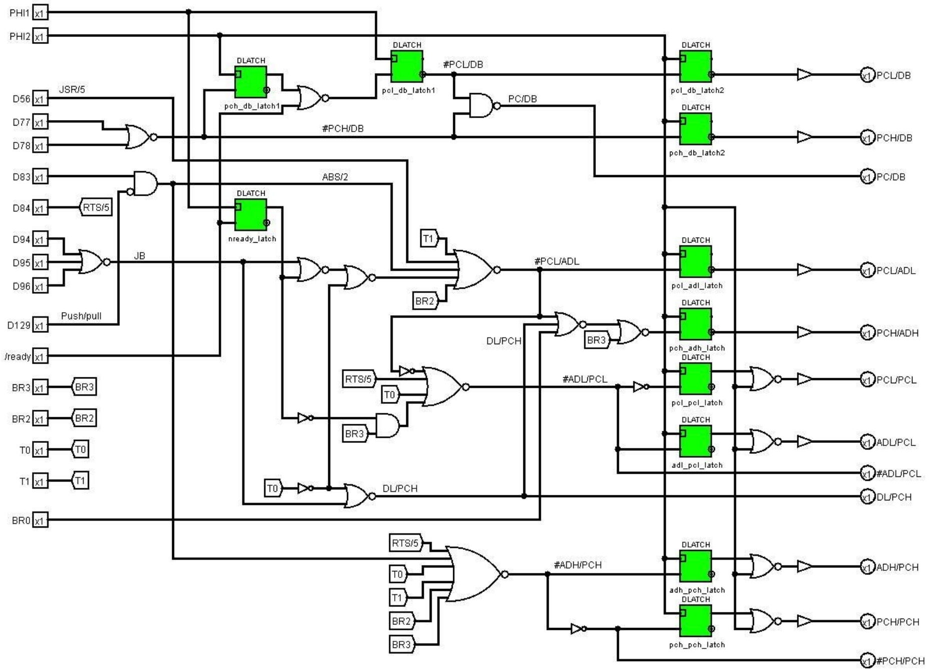
Outputs:

Signal	Description
DL/PCH	Auxiliary output signal for DL/ADH *bus control* circuitry
PC/DB	Auxiliary output signal for the RW Control circuit that is part of the dispatcher

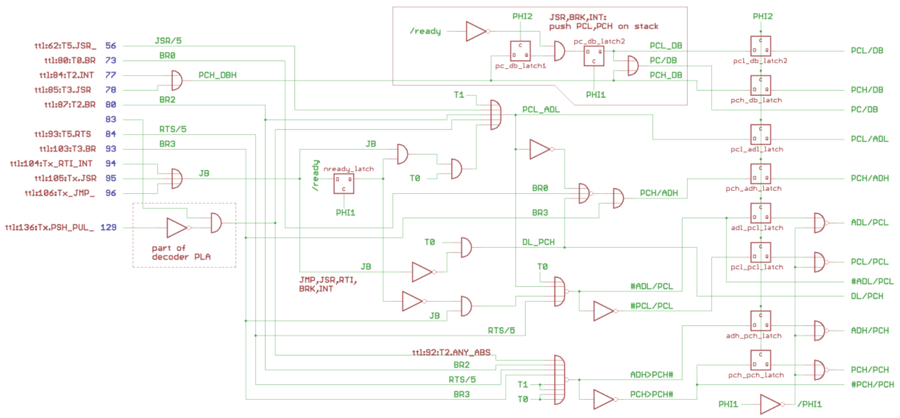
Control commands:

Command	Description
ADH/PCH	Load ADH bus value into the PCHS latch
PCH/PCH	If ADH/PCH is not running, this command is executed (refresh PCH)
PCH/ADH	Write the PCH register value to the ADH bus
PCH/DB	Write the PCH register value to the DB bus
ADL/PCL	Load the ADL bus value into the PCLS latch
PCL/PCL	If ADL/PCL is not running, this command is executed (refresh PCL)
PCL/ADL	Write the PCL register value to the ADL bus
PCL/DB	Write the PCL register value to the DB bus

Logic


PHI1
PHI2
D56
JSR/5
D77
D78
D83
D84
RTS/5
D94
D95
D96
JB
D129
Push/pull
/ready
BR3
BR2
T0
T1
BR0
DLATCH
pch_db_latch1
pcl_db_latch1
#PCL/DB
PC/DB
#PCH/DB
ABS/2
nready_latch
#PCL/ADL
DL/PCH
#ADL/PCL
#ADH/PCH
pcl_db_latch2
pch_db_latch2
pcl_adl_latch
pch_adh_latch
pcl_pcl_latch
adl_pcl_latch
adh_pch_latch
pch_pch_latch
PCL/DB
PCH/DB
PC/DB
PCL/ADL
PCH/ADH
PCL/PCL
ADL/PCL
#ADL/PCL
DL/PCH
ADH/PCH
PCH/PCH
#PCH/PCH

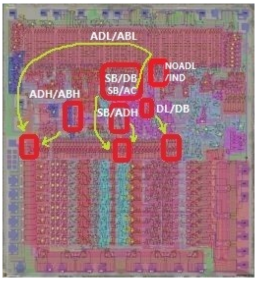
Optimized Schematics



https://github.com/emu-russia/breaks/blob/master/BreakingNESWiki_DeepL/6502/pc_control.md
23:https://github.com/emu-russia/breaks/blob/master/BreakingNESWiki/imgstore/logisim/pc_control_logisim.jpg

Notes in the margins for future revisions of the book.

Bus Control



Bus control is most of all "scattered" around the processor surface. It is easiest to describe all the bus *control commands* first, and then to look at the corresponding circuits individually.

Bus control commands:

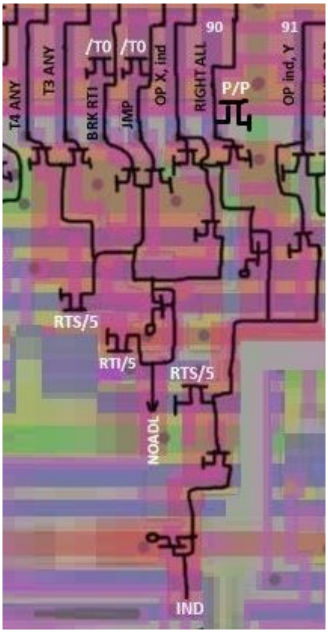
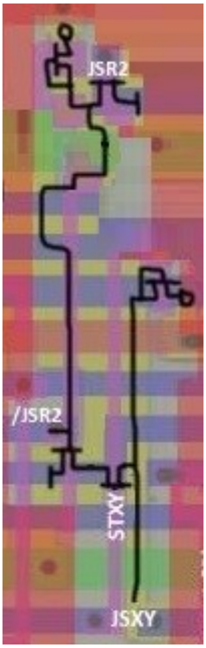
Command	Description
External address bus control	
ADH/ABH	Set the high 8 bits of the external address bus, in accordance with the value of the internal bus ADH
ADL/ABL	Set the low-order 8 bits of the external address bus, in accordance with the value of the internal bus ADL
ALU connection to SB, DB buses	
AC/DB	Place the AC value on the DB bus
SB/AC	Place the value from the SB bus/BCD correction circuit into the accumulator
AC/SB	Place the AC value on the SB bus
Control of the SB, DB and ADH internal buses	
SB/DB	Connect the SB and DB buses
SB/ADH	Connect SB and ADH buses
0/ADH0	Forced to clear the ADH[0] bit
0/ADH17	Forced to clear the ADH[1-7] bits
External data bus control	
DL/ADL	Write the DL value to the ADL bus
DL/ADH	Write the DL value to the ADH bus
DL/DB	Exchange the value of the DL and the internal bus DB. The direction of the exchange depends on the operating mode of the external data bus (read/write)

The motive of all the circuits is roughly as follows:

- The control circuits get a lot of input from the decoder and other auxiliary signals
- All circuits are mostly combinatorial (no triggers, just a mess of gates)
- The outputs from the control circuits go to the output latches of the commands to control the lower part of the processor.

Auxiliary Signals

Circuits for obtaining auxiliary signals:

NOADL и IND	JSXY
 T4 ANY T3 ANY /T0 /T0 BRK RTI JMP OP X, ind 90 RIGHT ALL P/P 91 OP ind, Y RTS/5 RTI/5 RTS/5 NOADL IND	

In the `IND` circuit the decoder output X90 is additionally modified by the Push/Pull signal (X129).

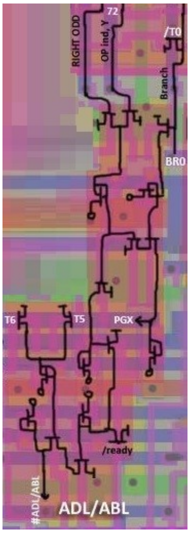
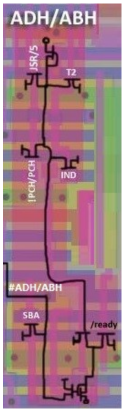
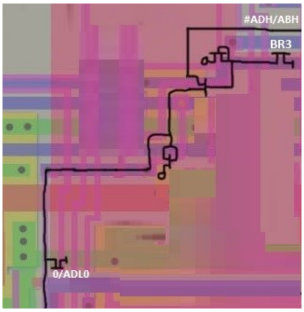
The other auxiliary and intermediate signals that can be found in the schematics in this section:

Signal	Description
RTS/5	Decoder X84
RTI/5	Decoder X26
STXY	Comes from *register control* circuitry
BR0	Decoder X73. Additionally modified with the /PRDY signal
T5	Comes from the cycle counter of long instructions
T6	Comes from the cycle counter of long instructions
PGX	Output signal from ADL/ABL circuit
JSR/5	Decoder X56
T2	Decoder X28
!PCH/PCH	Comes from the *PC control* circuitry
SBA	The signal comes out of the #SB/ADH circuit, used in the #ADH/ABH circuit
/ready	Global internal processor readiness signal
BR3	Decoder X93
0/ADL0	Comes from the interrupt vector setting circuit
AND	Comes from the *ALU control* circuit
STA	Decoder X79
STOR	Intermediate signal from the dispatcher
SBXY	Comes from a register control circuit (not to be confused with STXY)
T1	Comes from the PC increment circuit (see dispatcher)
BR2	Decoder X80
ZTST	Output signal for *flags control* from SB/DB circuit
ACRL2	One of the ACR Latch outputs
T0	Comes from the cycle counter of short instructions
ABS/2	Decoder X83. Additionally modified with Push/Pull signal (X129)
JMP/4	Decoder X101
IMPL	Decoder X128. Additionally modified with Push/Pull (X129) and IR0 signals.
JSR2	Decoder X48
/JSR	Inversion of JSR2 for the register control circuit
BRK6E	Comes from the *interrupts processing* circuit
INC_SB	Comes from the ALU control circuit
DL/PCH	Comes from the PC control circuitry

The signals are arranged in the order they appear in the schematics.

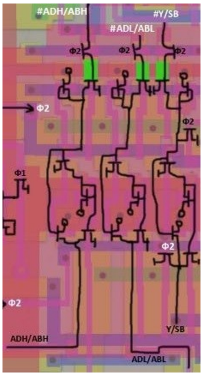
External Address Bus Control

Circuits for the generation of intermediate signals:

#ADL/ABL	#ADH/ABH (1)	#ADH/ABH (2)
		

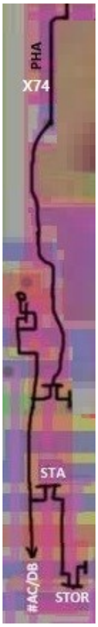
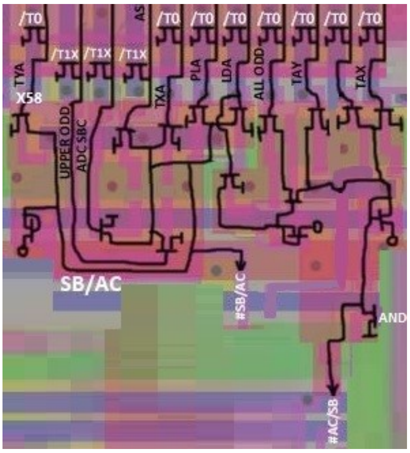
The first piece of the #ADH/ABH circuit is to the right of flag B, the second piece is in the interrupt address generation circuitry. The #ADH/ABH signal connects directly between these two pieces.

The output latches of the ADL/ABL and ADH/ABH control commands:

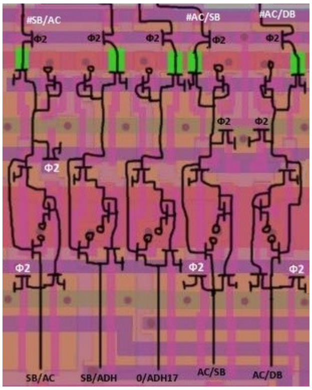


ALU Connection to SB, DB

Circuits for the generation of intermediate signals:

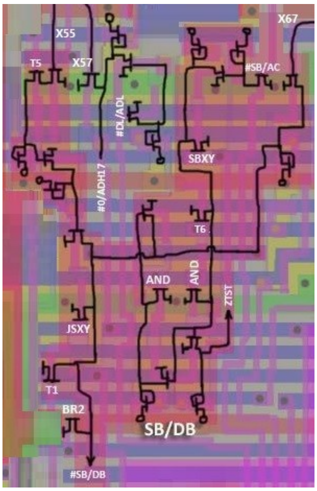
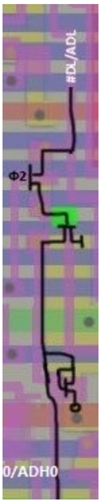
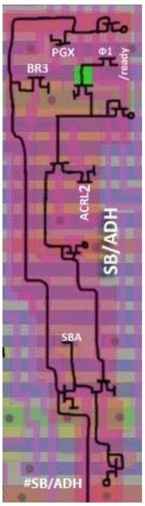
#AC/DB	#SB/AC, #AC/SB
	 /T0 AS /T0 /T0 /T0 /T0 /T0 /T0 /T0 /T1X /T1X /T1X TYA X58 TXA PLA LDA ALL ODD TAY TAX UPPER ODD ADC SBC SB/AC #SB/AC AND #AC/SB

AC/DB, SB/AC, AC/SB control command output latches:

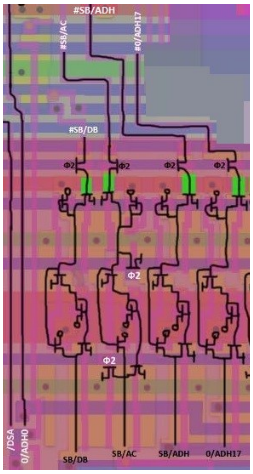


SB, DB, ADH Control

Circuits for generating intermediate signals (for 0/ADH0 you get the control command at once):

#SB/DB (also #0/ADH17)	0/ADH0	#SB/ADH
 X55 X67 T5 X57 #SB/AC #DL/ADL SBXY #0/ADH17 T6 AND AND ZTST JSXY T1 BR2 SB/DB #SB/DB		 PGX Φ1 BR3 /ready ACRL2 SB/ADH SBA #SB/ADH

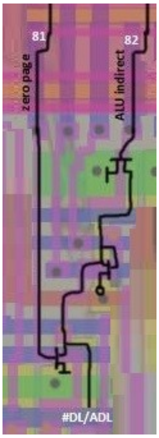
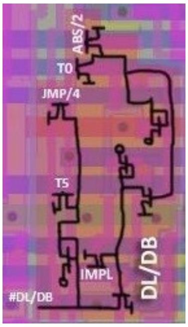
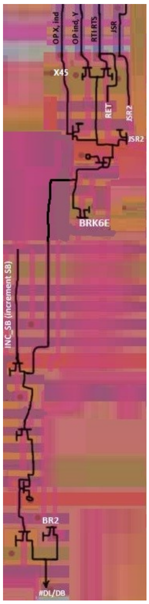
SB/DB, SB/ADH, 0/ADH17 control command output latches:



(0/ADH0 above)

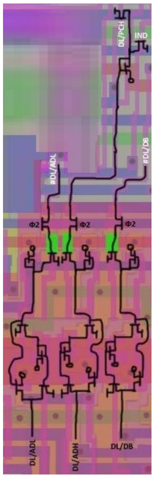
External Data Bus Control

Circuits for the generation of intermediate signals:

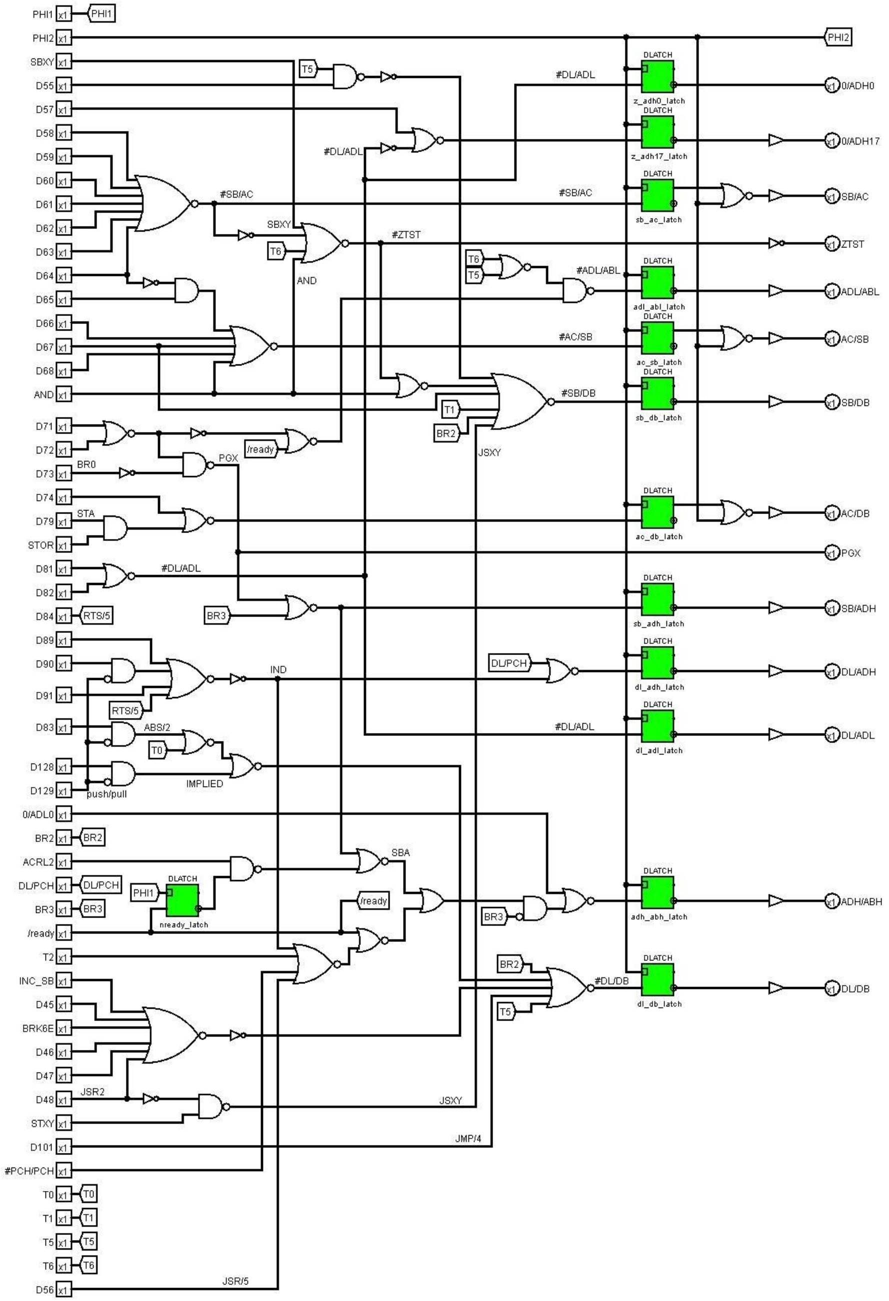
#DL/ADL	#DL/DB (1)	#DL/DB (2)
		 OP X, ind OP ind, Y RTI RTS JSR X45 RET JSR2 JSR2 BRK6E INC_SB (increment SB) BR2 #DL/DB

The first piece of #DL/DB circuitry is next to the ACR Latch, the second piece is right inside the ALU control circuitry. The #DL/DB signal connects directly between these two pieces.

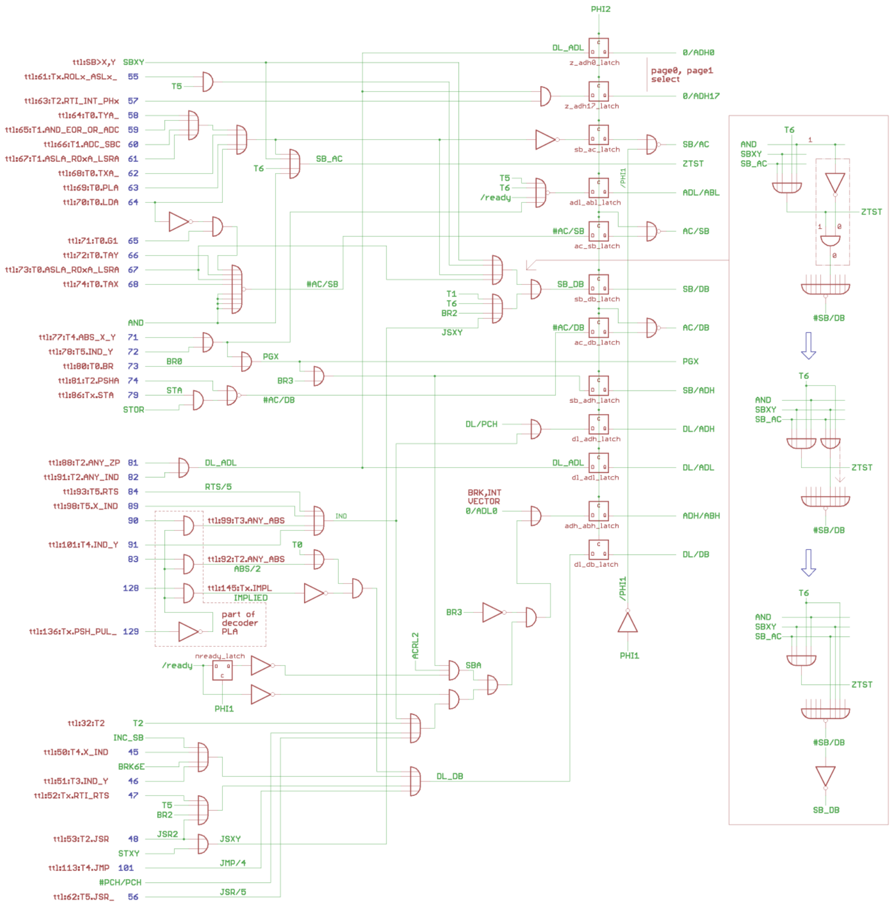
DL/ADL, DL/ADH, DL/DB control command output latches:



Logic


PHI1
PHI2
SBXY
D55
D57
D58
D59
D60
D61
D62
D63
D64
D65
D66
D67
D68
AND
D71
D72
D73
D74
D79
STOR
D81
D82
D84
D89
D90
D91
D83
D128
D129
0/ADL0
BR2
ACRL2
DL/PCH
BR3
/ready
T2
INC_SB
D45
BRK6E
D46
D47
D48
STXY
D101
#PCH/PCH
T0
T1
T5
T6
D56
#SB/AC
#DL/ADL
#ZTST
#ADL/ABL
#AC/SB
#SB/DB
#DL/DB
JSXY
PGX
BR0
STA
IND
ABS/2
IMPLIED
push/pull
SBA
JSR2
JMP/4
JSR/5
RTS/5
nready_latch
z_adh0_latch
z_adh17_latch
sb_ac_latch
adl_abl_latch
ac_sb_latch
sb_db_latch
ac_db_latch
sb_adh_latch
dl_adh_latch
dl_adl_latch
adh_abh_latch
dl_db_latch
DLATCH
0/ADH0
0/ADH17
SB/AC
ZTST
ADL/ABL
AC/SB
SB/DB
AC/DB
PGX
SB/ADH
DL/ADH
DL/ADL
ADH/ABH
DL/DB

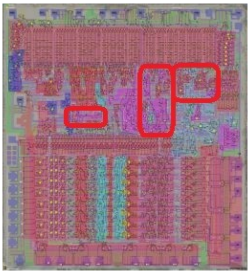
Optimized Schematics



https://github.com/emu-russia/breaks/blob/master/BreakingNESWiki_DeepL/6502/bus_control.md
7:https://github.com/emu-russia/breaks/blob/master/BreakingNESWiki/imgstore/logisim/bus_control_logisim.jpg

Notes in the margins for future revisions of the book.

Dispatcher



The execution logic (dispatcher) is the key mechanism of the processor that "directs" the execution of instructions.

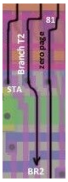
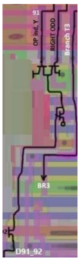
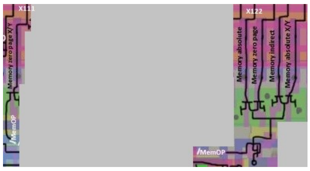
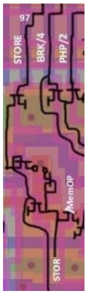
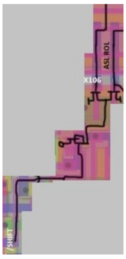
The execution logic consists of the following circuits:

- Intermediate signals
- Processor readiness control
- R/W pad control
- Short instruction cycle counter (T0-T1)
- Cycle counter for very long instructions (RMW T6-T7)
- Instruction completion circuit
- ACR latch
- Program counter (PC) increment circuit
- Opcode fetch circuit (Fetch)

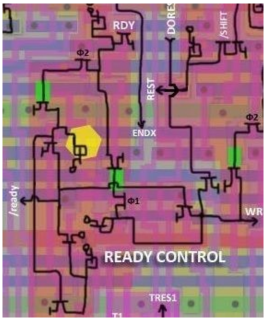
Long instruction cycle counter (T2-T5) is discussed in the corresponding section.

Intermediate Signals

Intermediate signals are obtained from the decoder outputs without any regularity. It was very difficult to separate them from the intermediate signals of the other control circuits, because of the chaotic connections.

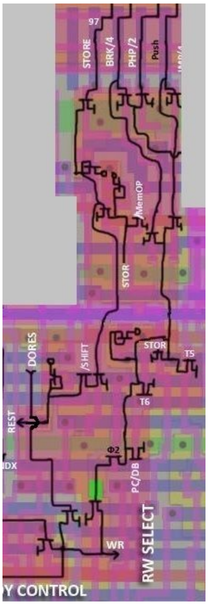
BR2	BR3, D91_92	/MemOP	STORE, STOR	/SHIFT
		 X111 Memory zero page X/Y /MemOP X122 Memory absolute Memory zero page Memory indirect Memory absolute X/Y /MemOP		

Processor Readiness



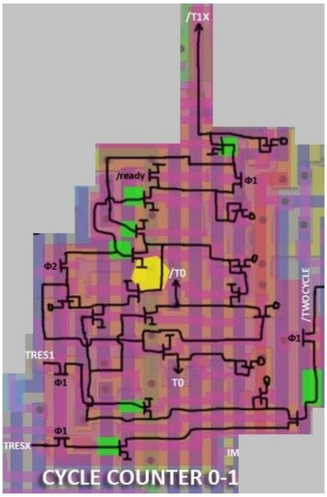
The /ready is the global ready signal of the processor, derived from the RDY input signal which comes from the corresponding contact.

R/W Control



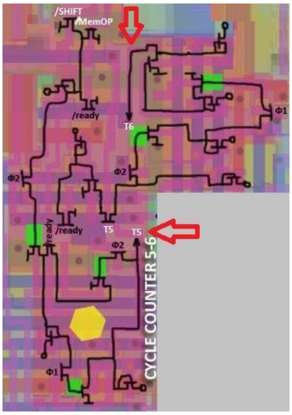
- REST: Reset cycle counters
- WR: The processor is in write mode

Short Cycle Counter



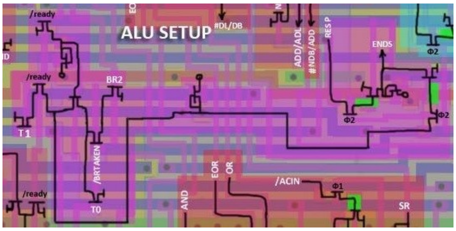
- T0: Internal signal (processor in T0 cycle)
- /T0, /T1X: Coming to *decoder* input

Very Long Cycle Counter

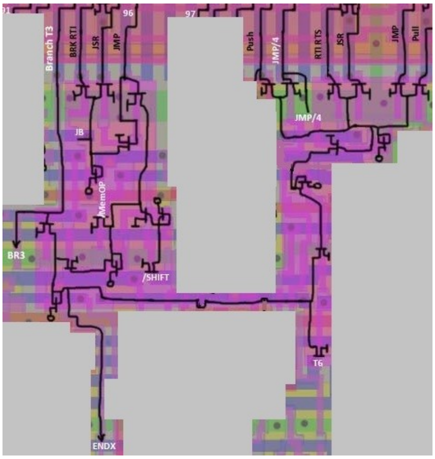


- T5, T6: The processor is in the RMW cycle T6/T7 (the signal names T5/T6 are old, but we will not rename them anymore)

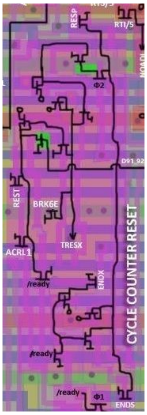
Instruction Completion



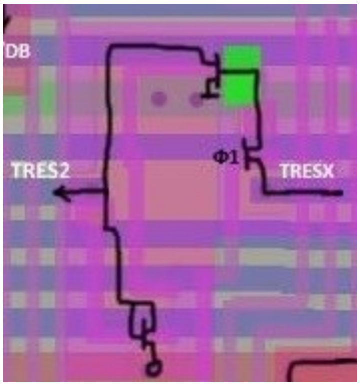
ENDS: Complete the short instructions



ENDX: Complete long instructions

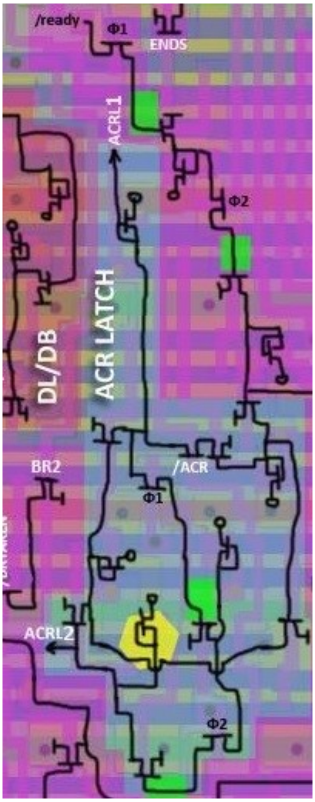


TRESX: Reset Cycle Counters



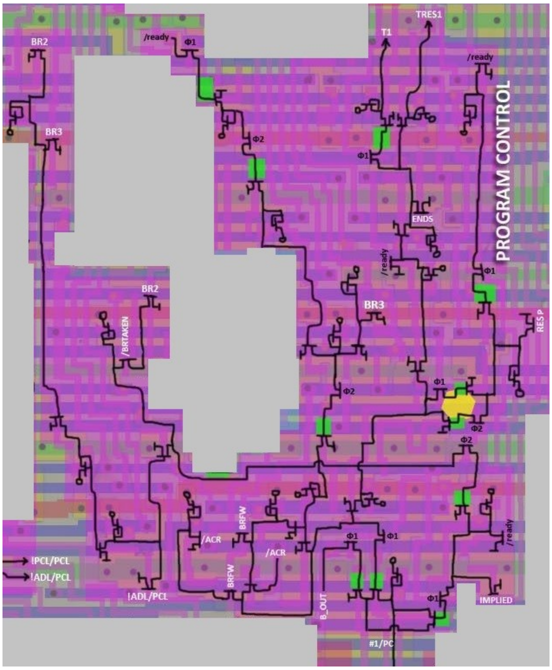
TRES2: Reset *extra instruction counter*

ACR Latch



Outputs 2 internal intermediate signals: ACRL1 and ACRL2.

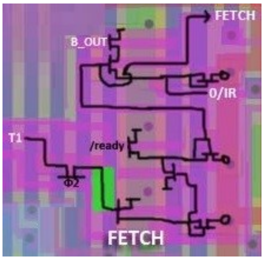
Increment PC



The circuit contains 3 "branches" of combinatorial logic, which finally form the *control command* to increment PC (`#1/PC`).
The circuit also generates the following signals:

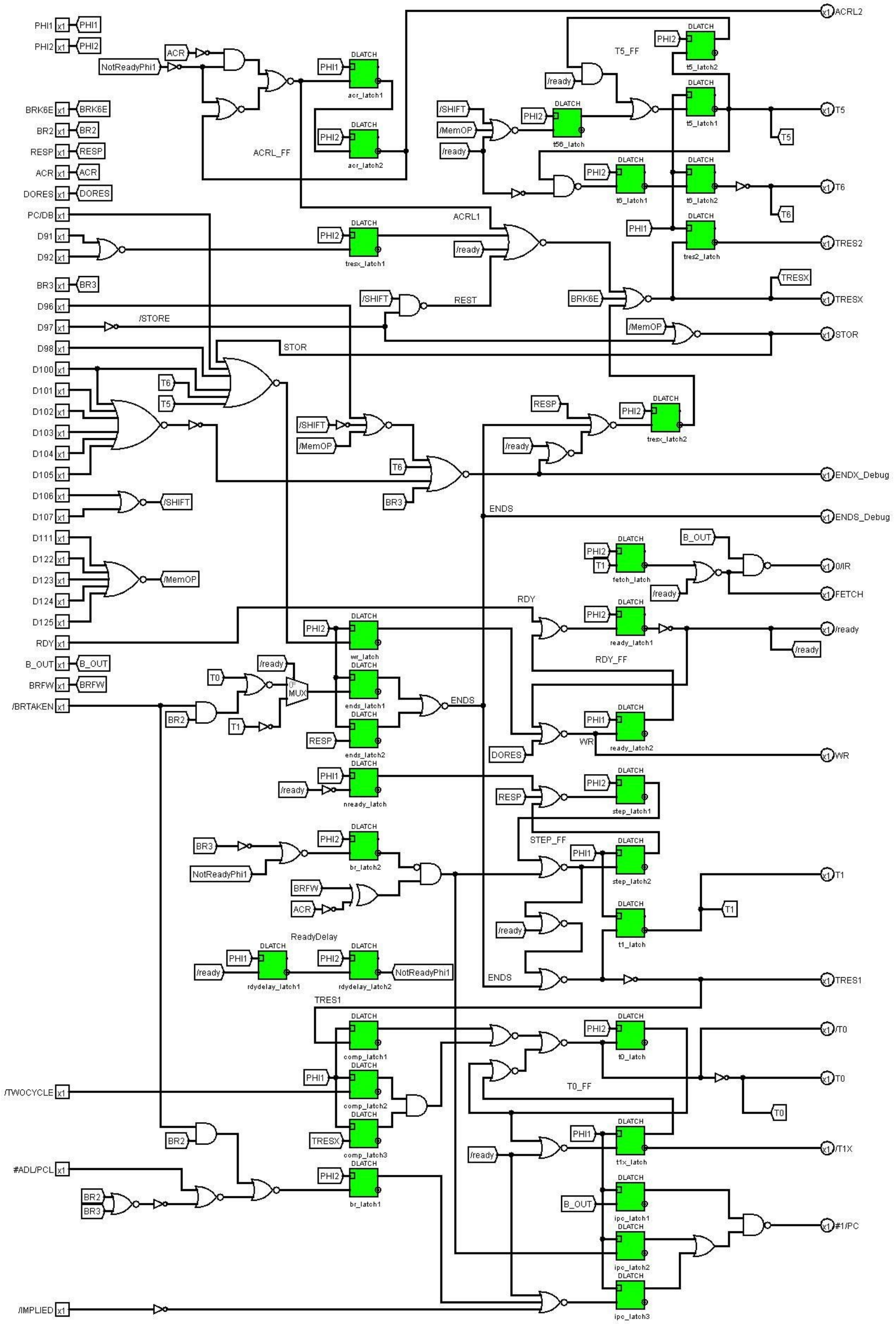
- T1: Processor in cycle T1
- TRES1: Reset short instruction cycle counter

Opcode Fetch



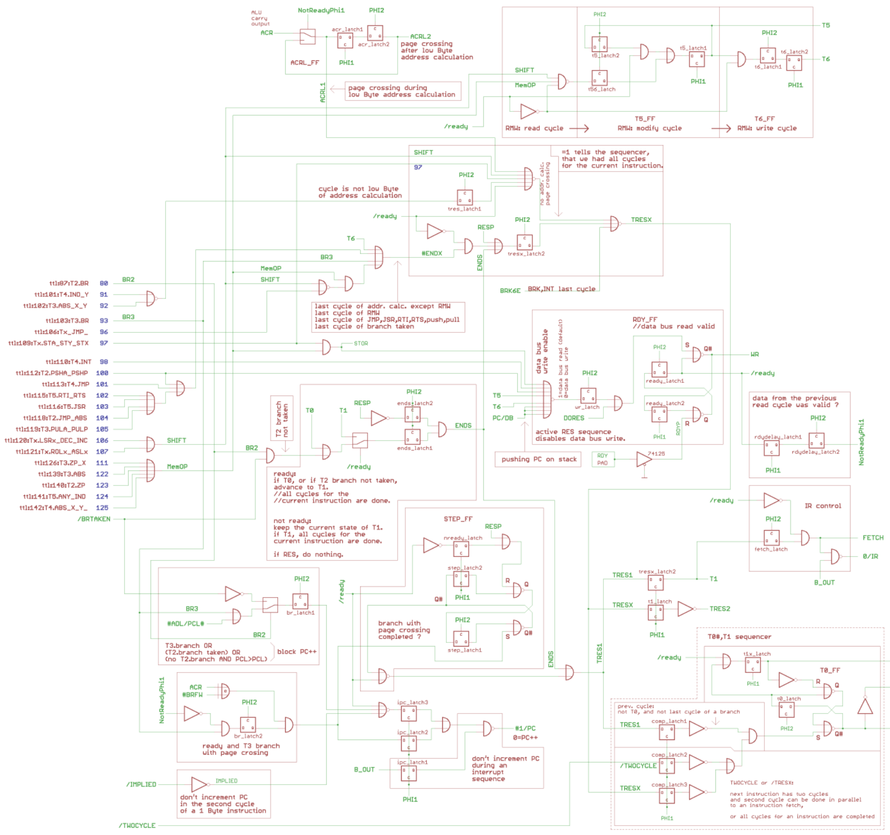
- FETCH: Fetch opcode to *instruction register*
- 0/IR: Inject `BRK` operation code, for *interrupt handling*

Logic



PHI1
PHI2
BRK6E
BR2
RESP
ACR
DORES
PC/DB
D91
D92
BR3
D96
D97
D98
D100
D101
D102
D103
D104
D105
D106
D107
D111
D122
D123
D124
D125
RDY
B_OUT
BRFW
/BRTAKEN
/TWOCYCLE
#ADL/PCL
/IMPLIED
NotReadyPhi1
ACRL_FF
acr_latch1
acr_latch2
ACRL2
T5_FF
t5_latch2
t5_latch1
t56_latch
t6_latch1
t6_latch2
tres2_latch
tresx_latch1
tresx_latch2
ACRL1
REST
/STORE
STOR
/SHIFT
/MemOP
/ready
T5
T6
TRES2
TRESX
ENDX_Debug
ENDS_Debug
ENDS
fetch_latch
0/IR
FETCH
RDY
ready_latch1
RDY_FF
ready_latch2
WR
wr_latch
ends_latch1
ends_latch2
MUX
nready_latch
step_latch1
STEP_FF
step_latch2
t1_latch
T1
br_latch2
ReadyDelay
rdydelay_latch1
rdydelay_latch2
TRES1
comp_latch1
comp_latch2
comp_latch3
t0_latch
/T0
T0
T0_FF
t1x_latch
/T1X
br_latch1
ipc_latch1
ipc_latch2
ipc_latch3
#1/PC
DLATCH

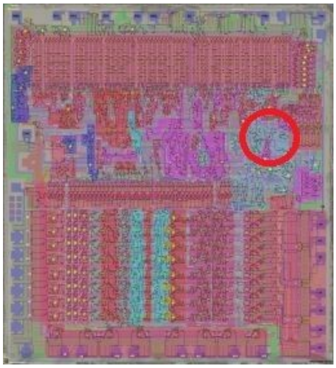
Optimized Schematics



https://github.com/emu-russia/breaks/blob/master/BreakingNESWiki_DeepL/6502/dispatch.md
13:https://github.com/emu-russia/breaks/blob/master/BreakingNESWiki/imgstore/logisim/dispatcher_logisim.jpg

Notes in the margins for future revisions of the book.

Flags Control

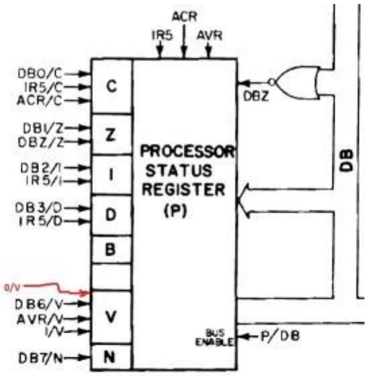


The flag control circuits are divided into two parts for convenience:

- Intermediate control signals from the decoder (opcode selection)
- Flag control signals

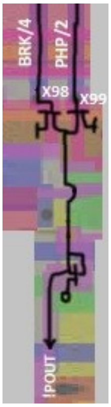
As you can guess, the purpose of the circuit is to control *processor flags*, depending on the currently executed instruction.

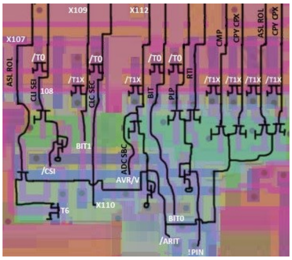
I think it makes sense to show here the relevant part of the wonderful 6502 circuit made by Donald F. Hanson:

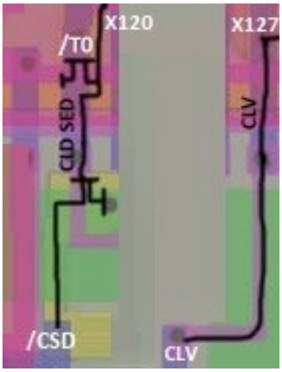


(The missing `0/V` signal has been corrected in the schematic)

Opcode Selection







Input signals:

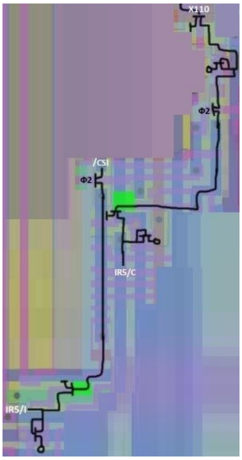
- /T0: Processor executes cycle T0 of the current instruction
- /T1X: Processor executes T1 cycle of the current instruction
- T6: Processor executes T6 cycle of current instruction

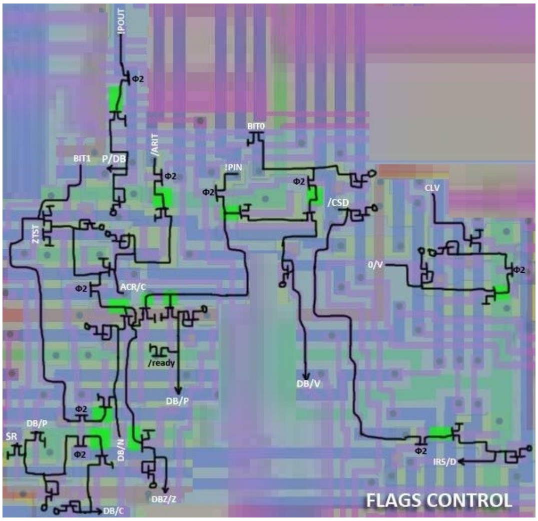
Output signals:

Signal	Decoder outputs	Dedicated instructions
!POUT	98,99	Working with flags outward (saving context after interrupt, `PHP` instruction)
/CSI	108	Instructions `CLI`, `SEI`
BIT1	109	Instruction `BIT`, cycle T1
X110	110	The 110th decoder output (instructions `CLC`, `SEC`), for convenience is left in these circuits. It just goes on to the main flag control circuitry.
AVR/V	112	Instructions `ADC`, `SBC`. This signal is the control signal for flag V
/ARIT	107,112,116-119	Matrix of comparison (`CMP`, `CPX`, `CPY`) and shift instructions (`ASL`, `ROL`) where flags are used
BIT0	113	Instruction `BIT`, cycle T0
!PIN	114,115	Working with flags inside (context loading after `RTI`, instruction `PLP`)
/CSD	120	Instructions `CLD`, `SED`
CLV	127	Instruction `CLV`

All of these control signals (except `AVR/V`) are intermediate signals and are not used anywhere else except for the flag control circuitry.

Flags Control





Input signals:

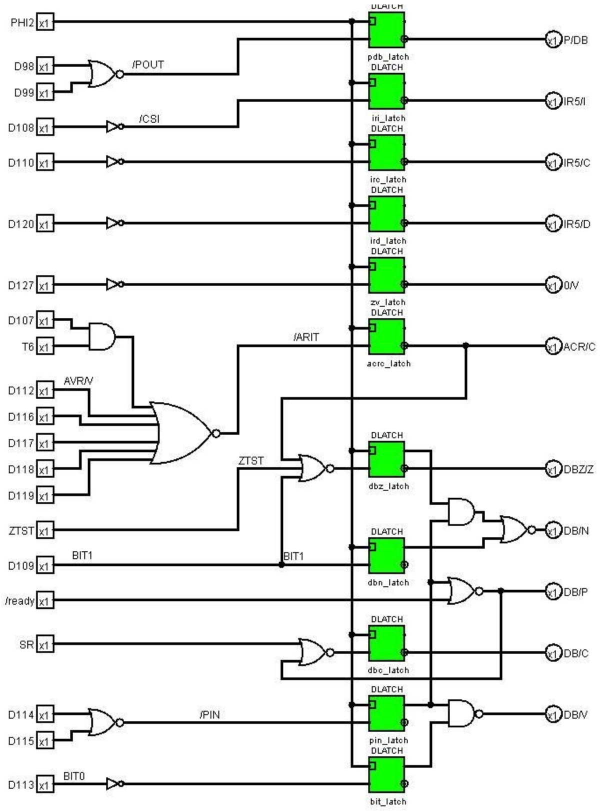
Signal	Purpose
/CSI	see above
X110	see above
!POUT	see above
BIT1	see above
ZTST	Comes from SB/DB *bus control* circuitry
/ARIT	see above
SR	Shift instruction from *ALU control* logic
/ready	Global internal processor readiness signal
!PIN	see above
BIT0	see above
/CSD	see above
CLV	see above

Output *Flag* control signals:

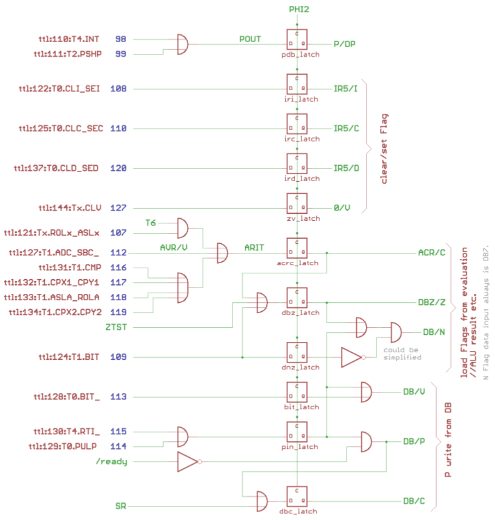
Signal	Flag	Purpose
IR5/C	C	Change the value of the flag according to the IR5 bit
ACR/C	C	Change the value of the flag according to the ACR value
DB/C	C	Change the value of the flag according to DB0 bit
DBZ/Z	Z	Change the value of the flag according to the /DBZ value
IR5/I	I	Change the value of the flag according to the IR5 bit
IR5/D	D	Change the value of the flag according to the IR5 bit
DB/V	V	Change the value of the flag according to DB6 bit
0/V	V	Clear flag V
DB/N	N	Change the value of the flag according to DB7 bit
P/DB	All	Place the value of the flags register P on the DB bus
DB/P	All	Place the DB bus value on the flag register P

The control signal `1/V` is obtained by the input contact `SO` and is not shown here.

Logic



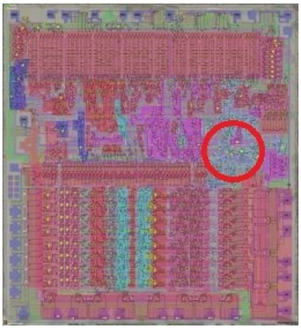
Optimized Schematics



https://github.com/emu-russia/breaks/blob/master/BreakingNESWiki_DeepL/6502/flags_control.md
16:https://github.com/emu-russia/breaks/blob/master/BreakingNESWiki/imgstore/logisim/flags_control_logisim.jpg

Notes in the margins for future revisions of the book.

Flags

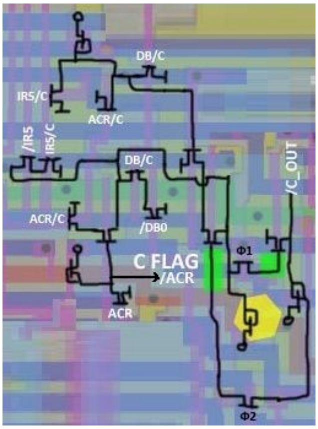


The flags (bits of the P register) are in "scattered" form, as several circuits of the upper part of the processor.

The flags are controlled by the *flags control* circuit.

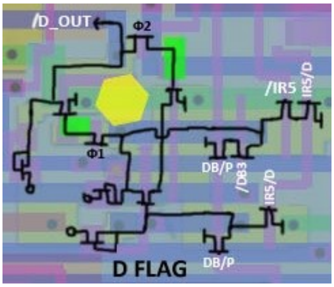
Flag B is treated separately in the section on *interrupt handling*. Topologically it is also located in another part of the processor.

C Flag



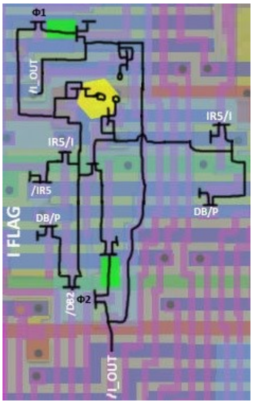
- IR5/C: Change the flag value according to the IR5 bit (applies during execution of the `SEC` and `CLC` instructions)
- ACR/C: Change the flag value according to the ACR value
- DB/C: Change the value of the flag according to the bit DB0
- /IR5: Inverted IR5 value
- /DB0: Input value from DB bus, in inverted form
- ACR: Result of a carry from the ALU (/ACR: in inverted form for dispatcher)
- /C_OUT: Output value of flag C, in inverted form

D Flag



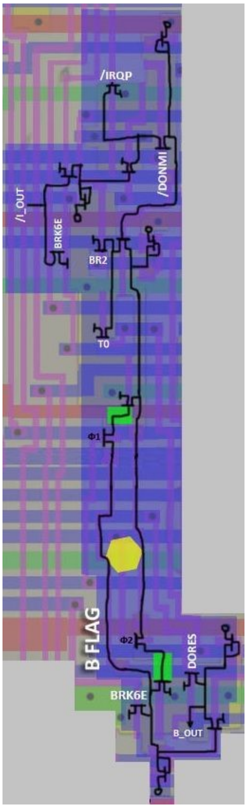
- Change the flag value according to the IR5 bit (applied during execution of `SED` and `CLD` instructions)
- DB/P: Common control signal, place the DB bus value on the flag register P
- /IR5: IR5 bit value, in inverted form
- /DB3: Input value from the DB bus, in inverted form
- /D_OUT: Output value of flag D, in inverted form

`I` Flag

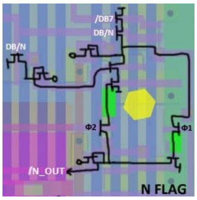


- IR5/I: Change the flag value according to the IR5 bit (applied during execution of `SEI` and `CLI` instructions)
- DB/P: Common control signal, place the DB bus value on the flag register P
- /IR5: IR5 bit value, in inverted form
- /DB2: Input value from the DB bus, in inverted form
- /I_OUT: Output value of flag I, in inverted form. This signal goes to two places: to the interrupt processing circuit and to the circuit for exchanging flag register values with the DB bus (below).

The `/I_OUT` signal is further modified by the `BRK6E` signal in flag B circuitry:

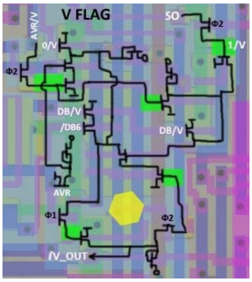


N Flag



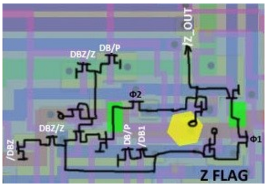
- DB/N: Change the flag value according to DB7
- /DB7: Input value from DB bus, in inverted form
- /N_OUT: Output value of flag N, in inverted form

V Flag



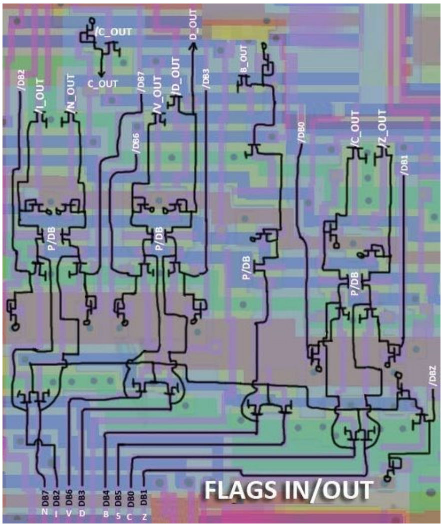
- 0/V: Clear flag V (applies during execution of `CLV` instructions)
- 1/V: Set flag V. Forced flag setting is done by the `SO` pin.
- AVR/V: Change the value of the flag according to the AVR value
- DB/V: Change the flag value according to DB6
- AVR: Overflow result from the ALU
- SO: Input value from pin `SO`
- /DB6: Input value from DB bus, in inverted form
- /V_OUT: Output value of flag V, in inverted form

Z Flag



- DBZ/Z: Change the flag value according to the /DBZ value
- DB/P: Common control signal, place the DB bus value on the flag P register
- /DBZ: Control signal from the flag exchange circuit with the DB bus (check that all bits of the DB bus are 0)
- /DB1: Input value from the DB bus, in inverted form
- /Z_OUT: Output value of flag Z, in inverted form

Flags I/O



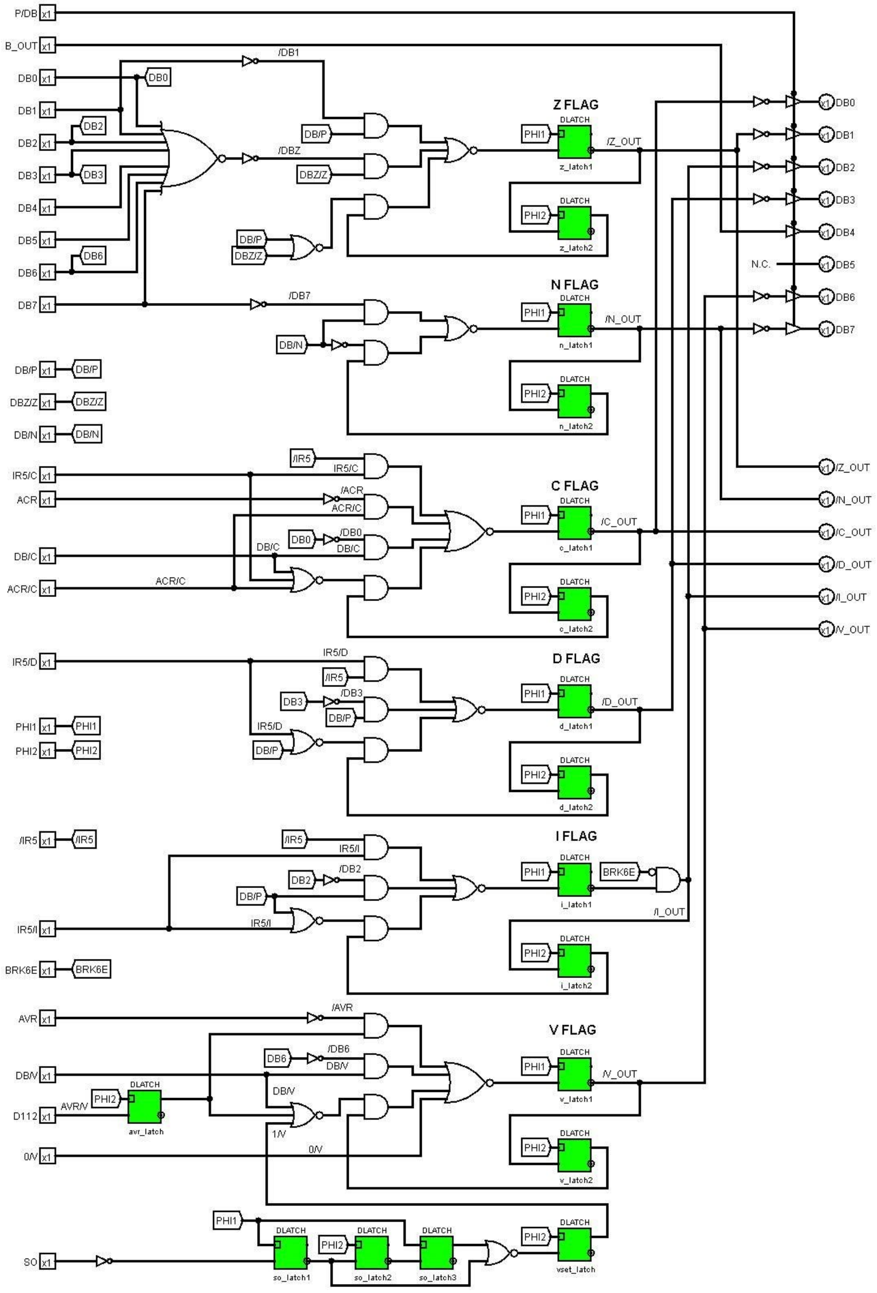
- C_OUT: Flag C value in direct form, used in *ALU control* circuit (in the circuit to form the `ADD/SB7` signal)
- D_OUT: Flag D value in direct form, used in the ALU control circuit (to form BCD correction signals DAA/DSA)
- P/DB: Place the P flag register value on the DB bus
- /DB0-7: The value of the DB bus bits, in inverted form. It is fed to the input of the corresponding bits of the P register.
- /DBZ: Check that all DB bus bits are 0 (i.e. checking the value to 0). It is used by the Z flag.

Correspondence of the bits of the DB bus and the flag register P:

DB Bit	Flag
0	C
1	Z
2	I
3	D
4	B
5	-
6	V
7	N

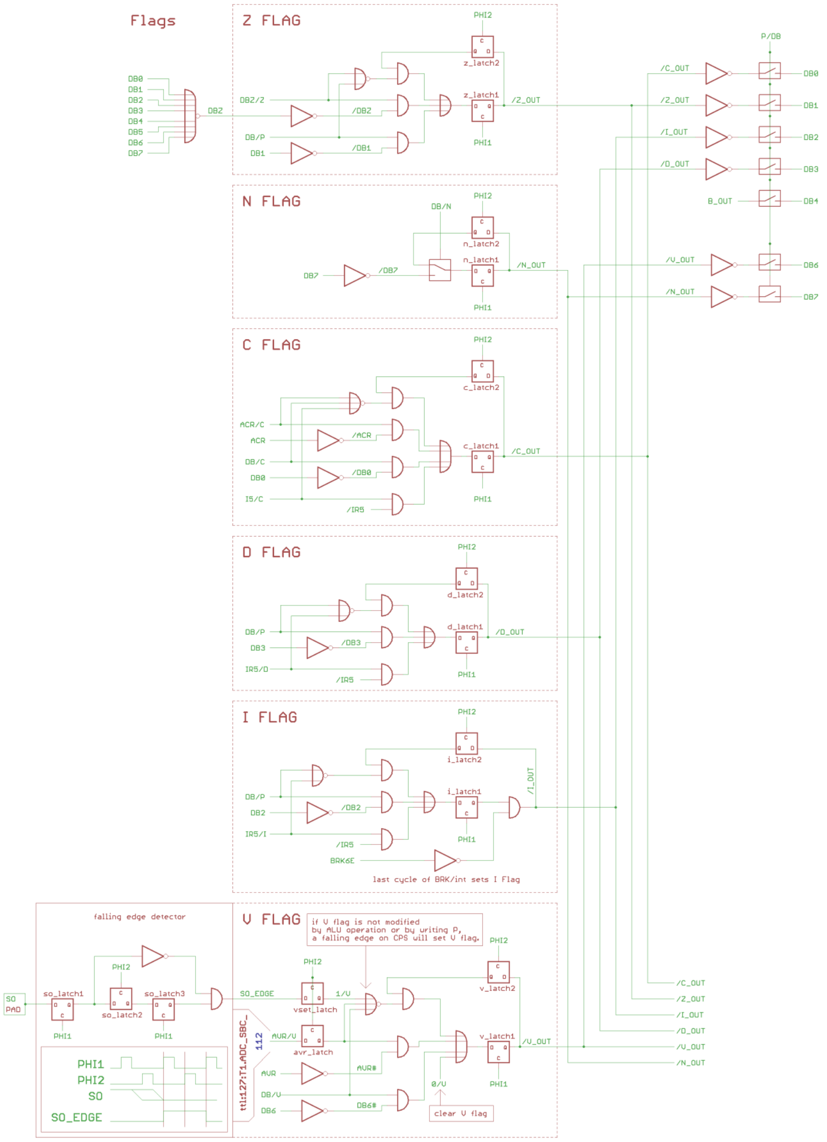
Flag 5 is not used. The DB5 bit is not changed (not connected) when saving the register P to the DB bus. However, the value of the DB5 bit is checked by the `/DBZ` control signal (to compare the value on the DB bus with zero).

Logic



P/DB
B_OUT
DB0
DB1
DB2
DB3
DB4
DB5
DB6
DB7
DB/P
DBZ/Z
DB/N
IR5/C
ACR
DB/C
ACR/C
IR5/D
PHI1
PHI2
/IR5
IR5/I
BRK6E
AVR
DB/V
D112
0/V
SO
/DB1
/DBZ
/DB7
/ACR
ACR/C
/DB0
IR5/D
/DB3
/DB2
/AVR
/DB6
AVR/V
1/V
Z FLAG
N FLAG
C FLAG
D FLAG
I FLAG
V FLAG
DLATCH
z_latch1
z_latch2
n_latch1
n_latch2
c_latch1
c_latch2
d_latch1
d_latch2
i_latch1
i_latch2
v_latch1
v_latch2
avr_latch
so_latch1
so_latch2
so_latch3
vset_latch
/Z_OUT
/N_OUT
/C_OUT
/D_OUT
/I_OUT
/V_OUT
N.C.
x1

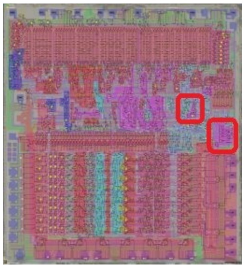
Optimized Schematics


Flags
Z FLAG
N FLAG
C FLAG
D FLAG
I FLAG
V FLAG
falling edge detector
if V flag is not modified by ALU operation or by writing P, a falling edge on CPS will set V flag.
last cycle of BRK/int sets I Flag
clear V flag
P/DB
SO PAD
PHI1
PHI2
SO
SO_EDGE

https://github.com/emu-russia/breaks/blob/master/BreakingNESWiki_DeepL/6502/flags.md
15:https://github.com/emu-russia/breaks/blob/master/BreakingNESWiki/imgstore/logisim/flags_logisim.jpg

Notes in the margins for future revisions of the book.

Branch Logic



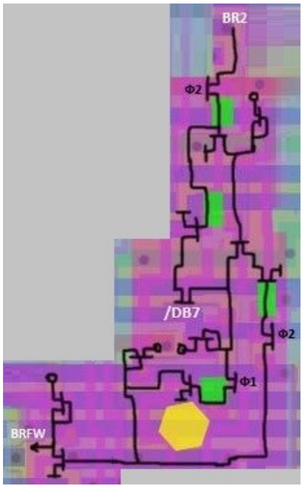
The logic of conditional branches determines:

- Whether the branch went forward or backward
- Whether a branch occurred at all

The branch direction is determined by the 7th bit of the branch instruction operand (relative offset) which is stored on the internal data bus (DB). If the 7th bit is 1, it means that branch is made "backwards" (PC = PC - offset).

The branch is checked according to the branch instruction (which differs by 6 and 7 bit of the operation code) as well as the flags: C, V, N, Z.

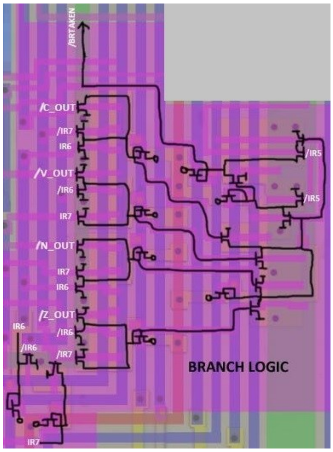
Branch Forward



The BRFW trigger is updated with the value D7 during BR3.PHI1. The rest of the time the trigger stores its current value. The value of the trigger is output as a `BRFW` control signal to the *Program Counter (PC) control* circuit.

The `BR2` is the X80 output of the decoder.

Branch Taken

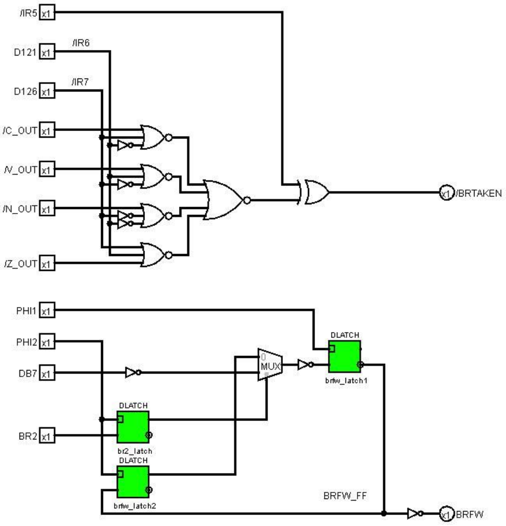


The combinatorial logic first selects by IR6/IR7 which group the branch instruction belongs to (i.e. which flag it checks) and the subsequent XOR selects how the branch instruction is triggered (flag set/reset). The output of `/BRTAKEN` is in inverse logic, that is, if branch is triggered, then /BRTAKEN = 0. The consumer of the /BRTAKEN signal is also the PC control circuit.

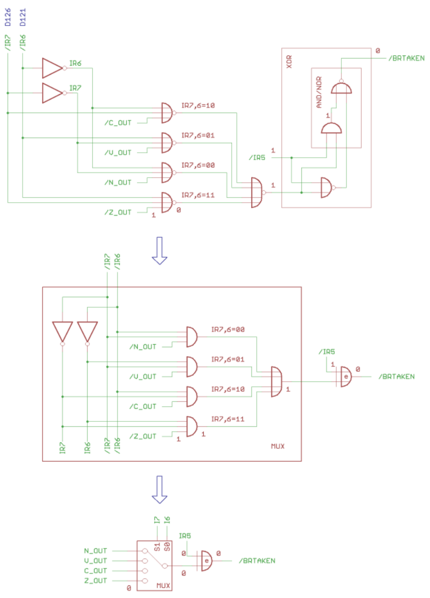
Inputs `/IR6` and `/IR7` are decoder outputs X121 and X126 respectively. The `/IR5` input comes directly from the *instruction register*.

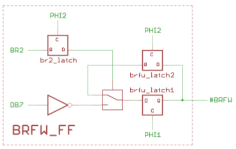
Note: The Branch Taken logic operates continuously and the value of the /BRTAKEN control line is updated every cycle, regardless of which instruction is being processed by the processor at the time.

Logic



Optimized Schematics

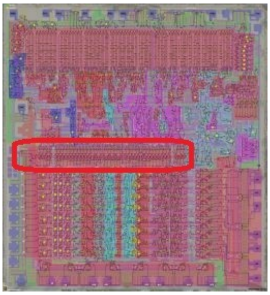




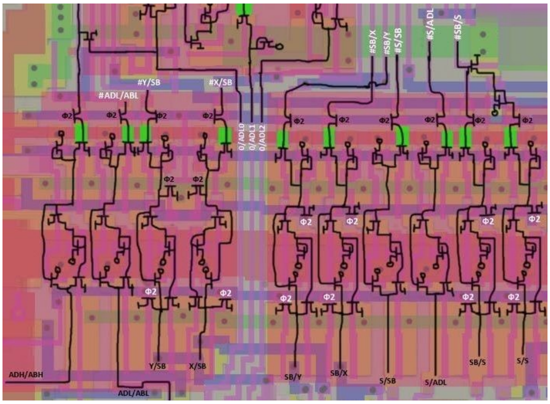
https://github.com/emu-russia/breaks/blob/master/BreakingNESWiki_DeepL/6502/branch_logic.md
6:https://github.com/emu-russia/breaks/blob/master/BreakingNESWiki/imgstore/logisim/branch_logic_logisim.jpg

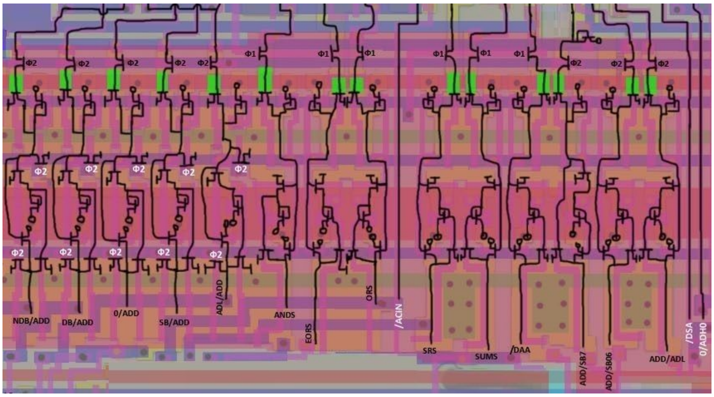
Notes in the margins for future revisions of the book.

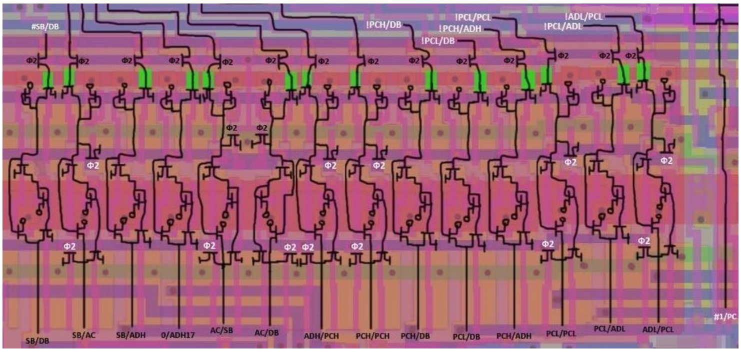
Control Commands

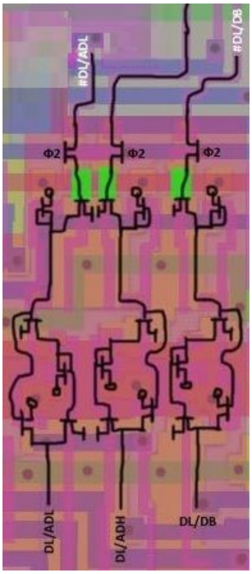


"Control Commands" is the conventional name for the large number of control signals that go from the top of the processor to the bottom and control the context (registers, buses, and ALU).









The control commands for the flag register are discussed in the corresponding section on *flag management*, since they do not go beyond the top of the processor.

Each control signal usually contains an output latch and sometimes a special "cutoff" transistor that turns the signal off at a certain half-cycle (usually some of the signals are turned off during PHI2). This is because the internal buses are pre-charged during PHI2, and the registers are usually "refreshed" at that time.

Most signals have names like `A/B` which means that the line "connects" `A` to `B`. For example SB/X means that the value from the internal bus SB is placed in register X.

List

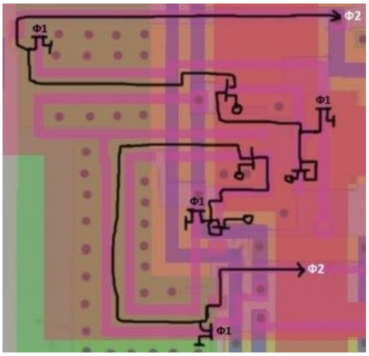
All commands are discussed in more detail in their respective sections. The summary table is just for reference.

Name	PHI1	PHI2	Description
Register control commands			
Y/SB	√		Y => SB
SB/Y	√		SB => Y
X/SB	√		X => SB
SB/X	√		SB => X
S/ADL	√	√	S => ADL
S/SB	√	√	S => SB
SB/S	√		SB => S
S/S	√		The S/S command is active if the SB/S command is inactive. This command simply "refreshes" the current state of the S register.
ALU control commands			
NDB/ADD	√		~DB => BI
DB/ADD	√		DB => BI
0/ADD	√		0 => AI
SB/ADD	√		SB => AI
ADL/ADD	√		ADL => BI
/ACIN	√	√	ALU input carry. The ALU also returns the result of carry (ACR) and overflow (AVR)
ANDS	√	√	AI & BI
EORS	√	√	AI ^ BI
ORS	√	√	AI
SRS	√	√	>>= 1
SUMS	√	√	AI + BI
/DAA	√	√	0: Perform BCD correction after addition
/DSA	√	√	0: Perform BCD correction after subtraction
ADD/SB7	√	√	ADD[7] => SB[7]. **Be careful, all output values are inverse latch values, except for `ADD/SB7`**.
ADD/SB06	√	√	ADD[0-6] => SB[0-6]
ADD/ADL	√	√	ADD => ADL
SB/AC	√		SB => AC
AC/SB	√		AC => SB
AC/DB	√		AC => DB

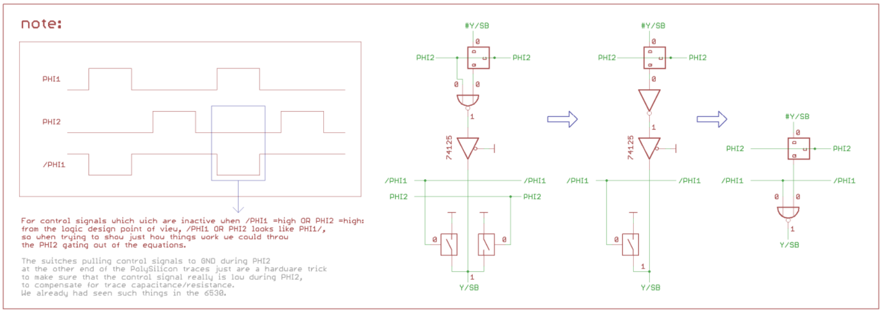
Program counter (PC) control commands			
#1/PC	√	√	0: Increment the program counter
ADH/PCH	√		ADH => PCH
PCH/PCH	√		If ADH/PCH is not performed, this command is performed (refresh PCH)
PCH/ADH	√	√	PCH => ADH
PCH/DB	√	√	PCH => DB
ADL/PCL	√		ADL => PCL
PCL/PCL	√		If ADL/PCL is not performed, this command is performed (refresh PCL)
PCL/ADL	√	√	PCL => ADL
PCL/DB	√	√	PCL => DB
Bus control commands			
ADH/ABH	√	√	ADH => ABH
ADL/ABL	√	√	ADL => ABL
0/ADL0, 0/ADL1, 0/ADL2	√	√	Reset some of the ADL bus bits. Used to set the interrupt vector.
0/ADH0, 0/ADH17	√	√	Reset some of the ADH bus bits
SB/DB	√	√	SB <=> DB, connect the two buses
SB/ADH	√	√	SB <=> ADH
DL latch control commands			
DL/ADL	√	√	DL => ADL
DL/ADH	√	√	DL => ADH
DL/DB	√	√	DL <=> DB

"Other" PHI2 (/PHI1)

On the left side is a small circuit to pull up PHI2 (which is used by a lot of cutoff transistors, so it must be quite powerful):



Optimized logic diagram with explanations:



For control signals which wich are inactive when /PHI1 =high OR PHI2 =high:
from the logic design point of view, /PHI1 OR PHI2 looks like PHI1/,
so when trying to show just how things work we could throw
the PHI2 gating out of the equations.

The switches pulling control signals to GND during PHI2
at the other end of the PolySilicon traces just are a hardware trick
to make sure that the control signal really is low during PHI2,
to compensate for trace capacitance/resistance.
We already had seen such things in the 6530.

https://github.com/emu-russia/breaks/blob/master/BreakingNESWiki_DeepL/6502/context_control.md
9:context_control_gating

Command Priority

Although in a real processor all commands are "executed" at the same time, it is still possible to outline some priority that the developers have laid down.

The commands on the bottom of the 6502, in order of execution:

PHI1 "Set Address and R/W Mode":

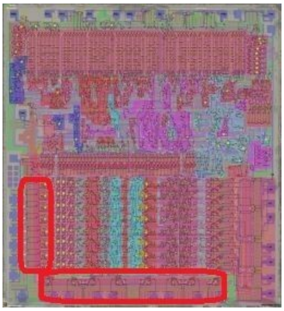
- Loading on the bus from DL: DL_DB, DL_ADL, DL_ADH
- Registers to the SB bus: Y_SB, X_SB, S_SB
- Saving flags on the DB bus: P_DB
- ADD saving on SB/ADL: ADD_SB7, ADD_SB06, ADD_ADL
- Saving AC: AC_SB, AC_DB
- Saving of old stack pointer value to ADL bus: S_ADL
- Increment PC: n_1PC
- Saving PC to bus: PCL_ADL, PCH_ADH, PCL_DB, PCH_DB
- Bus multiplexing: SB_DB, SB_ADH
- Constant generator: Z_ADL0, Z_ADL1, Z_ADL2, Z_ADH0, Z_ADH17
- Loading ALU operands: NDB_ADD, DB_ADD, Z_ADD, SB_ADD, ADL_ADD
- BCD correction via SB bus: SB_AC
- Loading flags: DB_P, DBZ_Z, DB_N, IR5_C, DB_C, IR5_D, IR5_I, DB_V, Z_V, ACR_C, AVR_V
- Loading registers: SB_X, SB_Y, SB_S / S_S
- Load PC from bus or keep old value: ADH_PCH/PCH_PCH, ADL_PCL/PCL_PCL
- Saving DB to DOR
- Set external bus address: ADH_ABH, ADL_ABL

PHI2 "Read/Write Data":

- Loading the DL with a value from the external data bus
- Registers on SB bus: S_SB
- Saving flags to the DB bus: P_DB
- ALU operation: ANDS, EORS, ORS, SRS, SUMS, n_ACIN, n_DAA, n_DSA
- ADD saving on SB/ADL: ADD_SB7, ADD_SB06, ADD_ADL
- Saving old stack pointer value to ADL bus: S_ADL
- Increment PC: n_1PC (PC is incremented in this half-cycle)
- Saving PC to bus: PCL_ADL, PCH_ADH, PCL_DB, PCH_DB
- Bus multiplexing: SB_DB, SB_ADH
- Constant generator: Z_ADL0, Z_ADL1, Z_ADL2, Z_ADH0, Z_ADH17
- Loading flags: DB_P, DBZ_Z, DB_N, IR5_C, DB_C, IR5_D, IR5_I, DB_V, Z_V, ACR_C, AVR_V
- Setting external data bus from DOR: If WR = 1

BOTTOM PART

Address Bus

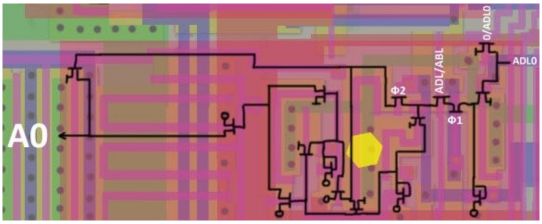


Although the 6502 communicates with the outside world on a 16-bit address bus, but because the processor is 8-bit in nature, the address bus is internally divided into two 8-bit halves: an upper (ADH) and a lower (ADL).

The internal ADH/ADL address bus connects to the external 16-bit bus (pins A0-A15) through registers ABH/ABL, which contain the last written value (address that has been set).

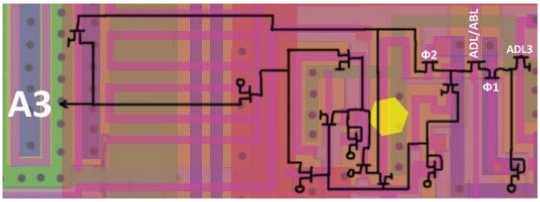
The address bus is unidirectional. It can only be controlled by the 6502.

Transistor circuit of the lower bits of the ABL (0-2):

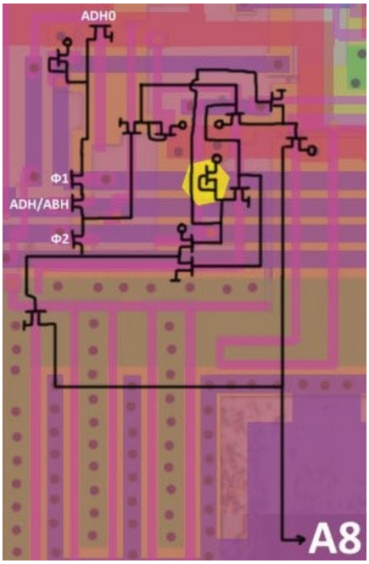


(The schematic is the same for ABL1 and ABL2 bits)

The remaining ABL bits (3-7):



ABH bits:

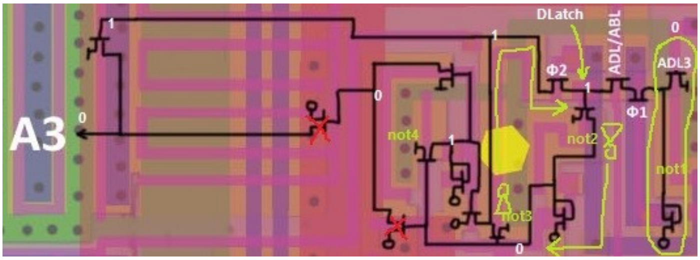


Control commands:

- 0/ADL0, 0/ADL1, 0/ADL2: The lower 3 bits of the ADL bus can be forced to zero by commands when setting *interrupts vector*
- ADL/ABL: Place the value of the internal ADL bus on the ABL register
- ADH/ABH: Place the ADH internal bus value on the ABH register

Circuit Flow

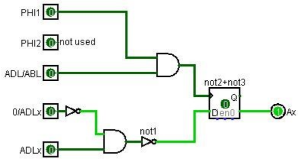
Consider the behavior of the circuit when ADL = 0:

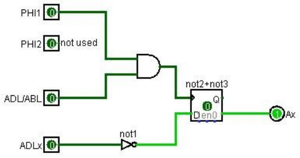


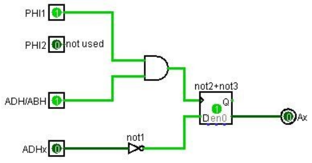
- The flip/flop of the ABL bit is organized on two inverters (not2 and not3) with not2 acting simultaneously as a DLatch (whose input Enable is connected to PHI2)
- PHI2: FF is "refreshed" in this half-step.
- PHI1: In this half-step the old FF value is "cut off" by the PHI2 tristate (located to the left of not2) and the new FF value is loaded from the ADL bus (inverted, see not1) but only if an ADL/ABL command is active
- The output from not2 organizes the final generation of the output value for the external address bus. This part of the circuit contains an inverter not3 to form the FF and also an inverter not4 which controls the amplifier "comb" of the Ax contacts

Logic

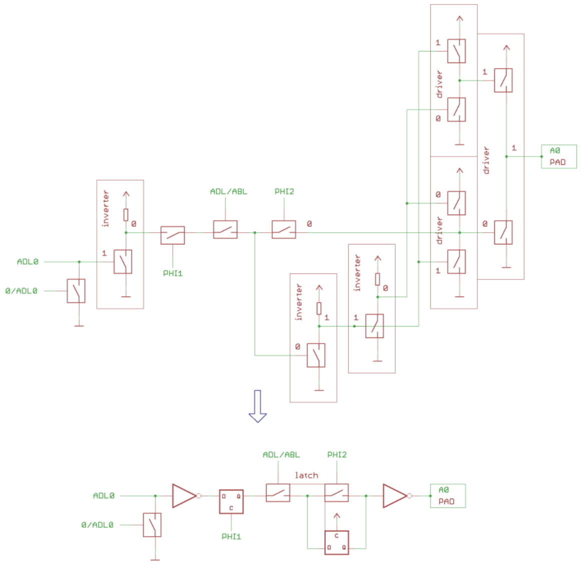
On the logic circuits PHI2 is not used, and FF organized on two inverters is replaced by a regular trigger.







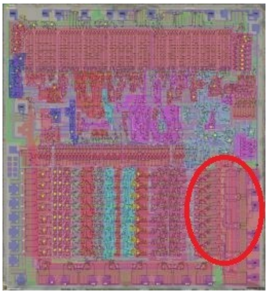
Optimized Schematics



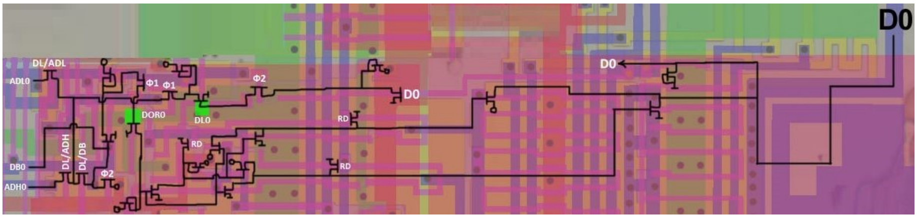
https://github.com/emu-russia/breaks/blob/master/BreakingNESWiki_DeepL/6502/address_bus.md
0:https://github.com/emu-russia/breaks/blob/master/BreakingNESWiki/imgstore/abl02_tran.jpg

Notes in the margins for future revisions of the book.

Data Bus



The circuits for working with the external data bus consist of 8 identical pieces:



(The circuit is shown for bit 0, the rest are the same)

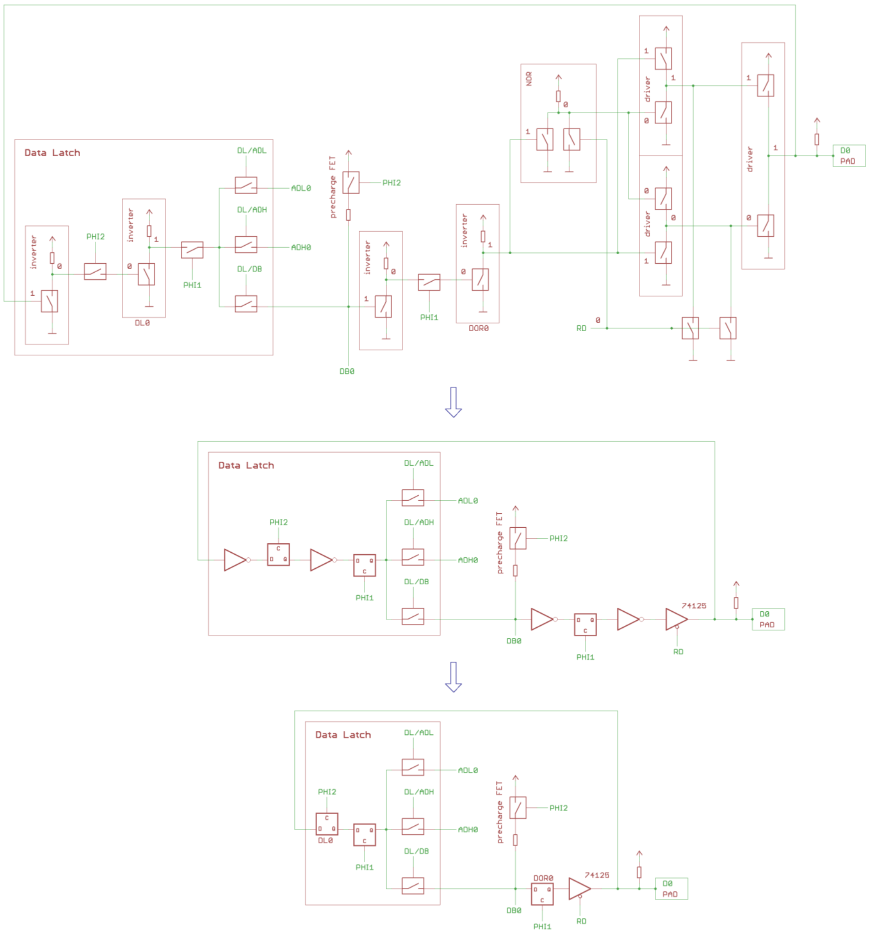
- DOR: The DOR latch stores the output value to be placed on the D0-D7 bus pins. If RD=1 the complementary output lines with DOR are cut off, so the whole output part becomes floating.
- DL: The DL latch stores the input value
- Next to the control signal DL/DB you can see the precharge transistor for the internal bus DB

Control signals:

- DL/ADL: Place the DL latch value on the internal ADL bus
- DL/ADH: Place the DL latch value on the internal ADH bus
- DL/DB: In read mode (RD=1), the value from the DL latch is placed on the internal DB bus. In write mode (RD=0) the value from the DB bus is placed on the DOR latch

The external data bus (pins D0-D7) is also directly connected to the input of the *predecode* circuit.

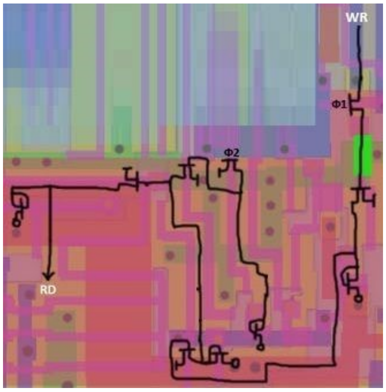
Optimized schematics:



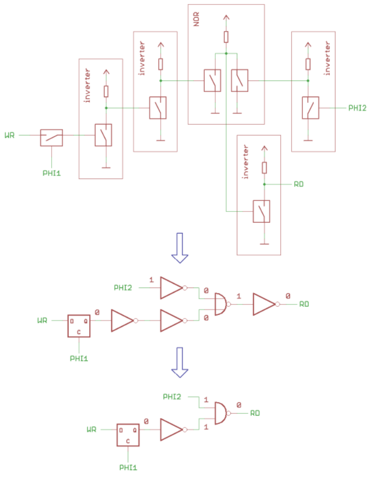
https://github.com/emu-russia/breaks/blob/master/BreakingNESWiki_DeepL/6502/data_bus.md
11:https://github.com/emu-russia/breaks/blob/master/BreakingNESWiki/imgstore/6502_data_bit_tran.jpg

WR Latch

From the R/W control circuit, the latch circuit receives a control signal `WR`. The circuit outputs a control signal `RD` which controls the direction of the external data bus.



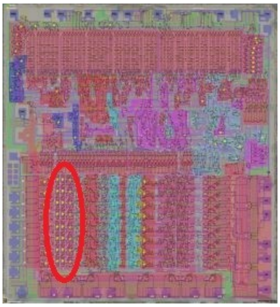
Optimized schematics:



https://github.com/emu-russia/breaks/blob/master/BreakingNESWiki_DeepL/6502/data_bus.md
12:https://github.com/emu-russia/breaks/blob/master/BreakingNESWiki/imgstore/6502_wr_latch_tran.jpg

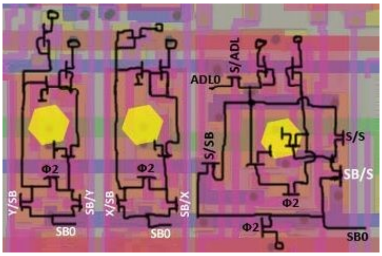
Notes in the margins for future revisions of the book.

Registers



The X and Y registers are used for index addressing. Register S is a stack pointer and the stack is located at addresses 0x100 ... 0x1FF (on the first page).

Schematically the X, Y and S registers consist of 8 identical chunks (bits):



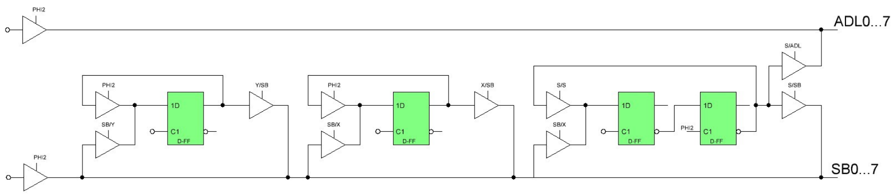
(In the schematic above, replace SB0 and ADL0 with SBx and ADLx for the remaining register bits)

Each register bit is based on a trigger, loading and unloading of values on the buses is done by *control signals*:

- Y/SB: Place the value of the register Y on the SB bus
- SB/Y: Load the Y register value from the SB bus
- X/SB: Place the value of the register X on the SB bus
- SB/X: Load the X register value from the SB bus
- S/ADL: Place the old S register value on the ADL bus
- S/SB: Place the old S register value on the SB bus
- SB/S: Load the new S register value from the SB bus
- S/S: Refresh S register, active when SB/S = 0

So the registers can only connect to two buses: SB and ADL.

Logic



- During PHI1 the X and Y registers output their value to the SB bus / are overloaded with new values from the SB bus.
- The S register has an input latch and an output latch. During PHI1 the value from the output latch is placed on the SB or ADL buses and the input latch is either loaded with a new value from the SB bus or refreshed from the output latch (S/S).
- During PHI2 the X and Y registers "store" their old value as the control signals disconnect them from the bus.
- The S register simply outputs its value to the SB or ADL bus during PHI2. The input latch is overridden because the exchange commands are disabled during PHI2.

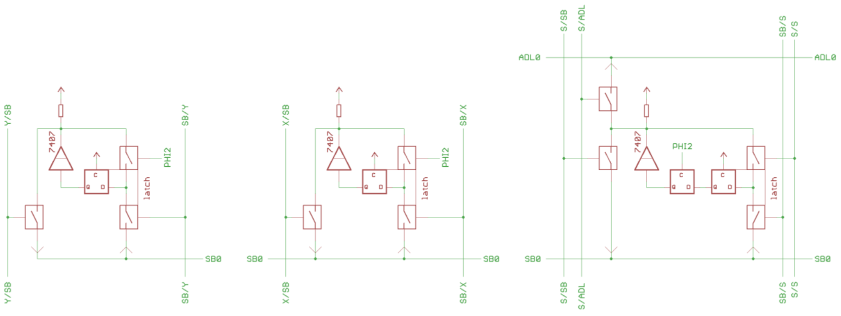
The SB and ADL buses are precharged during PHI2. This is done because it takes longer to "charge" the bus than to "discharge" it. Therefore, when the bus is not needed - it is precharged, so that it does not have "floating" values. If the value placed on the bus is 1, then the bus is already prepared ("charged") in advance. If the value placed on the bus is 0, then the bus is "discharged" to ground.

In modern processors the task of precharging the bus is done by dedicated standard cells called Bus Keeper.

In the transistor schematic above you can only see the transistors to charge the SB bus (located in the circuit for the S register bits). The transistors to precharge the ADL bus are scattered next to the program counter (PC).

⚠ Pay special attention to the design of the S register. It has an input latch (to load a new value) and an output latch (to save the old value). Loading the new value (SB/S) and saving the old value (S/ADL) can happen simultaneously.

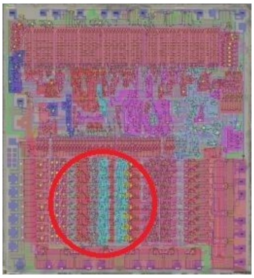
Optimized Schematics



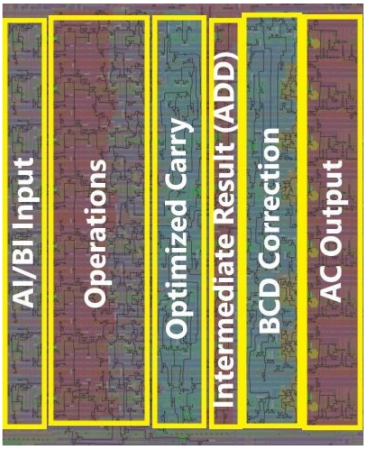
https://github.com/emu-russia/breaks/blob/master/BreakingNESWiki_DeepL/6502/regs.md
25:https://github.com/emu-russia/breaks/blob/master/BreakingNESWiki/imgstore/regs_logic.jpg

Notes in the margins for future revisions of the book.

ALU



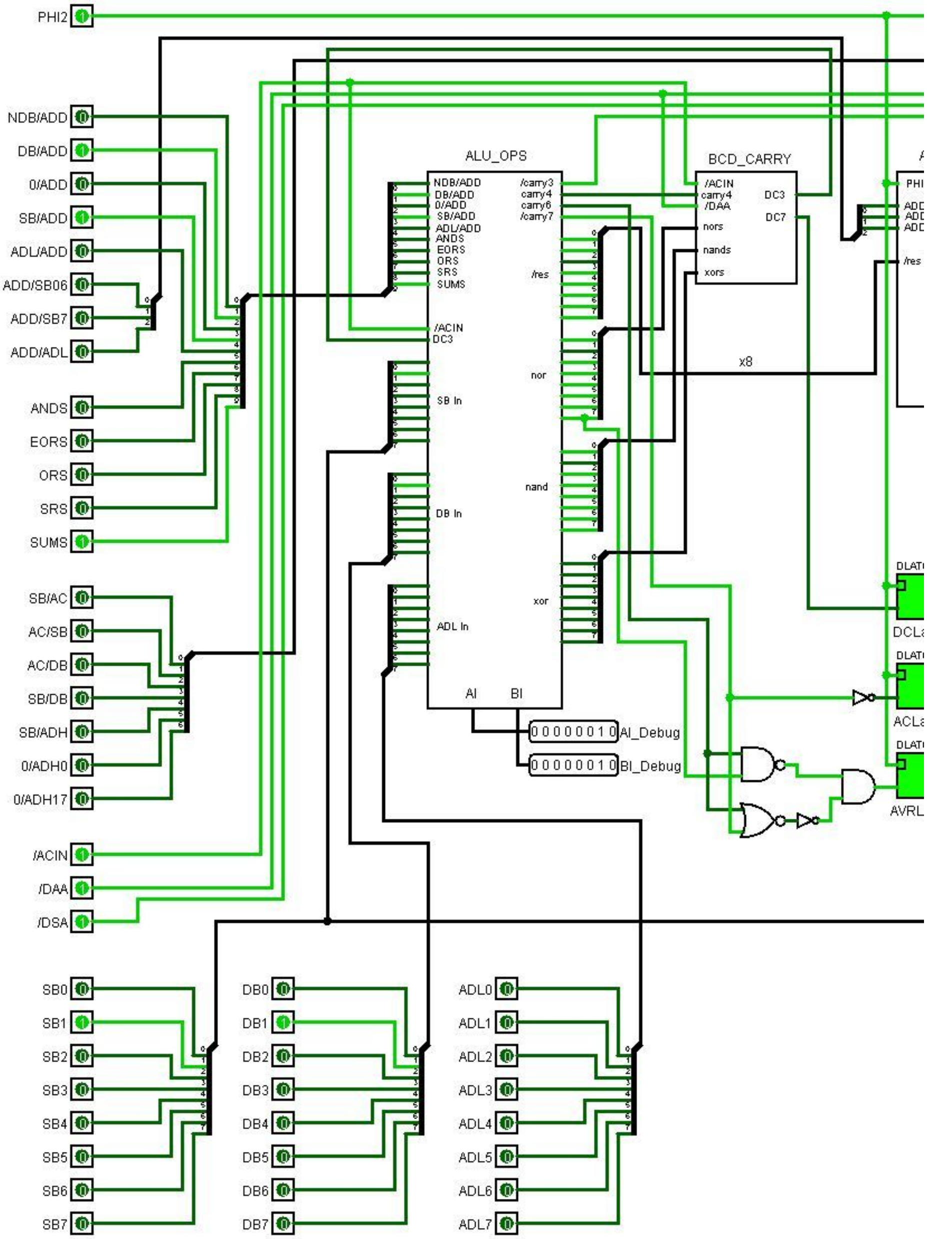
It is not possible to show the whole ALU circuit, so let's saw it into its component parts and consider each one separately.

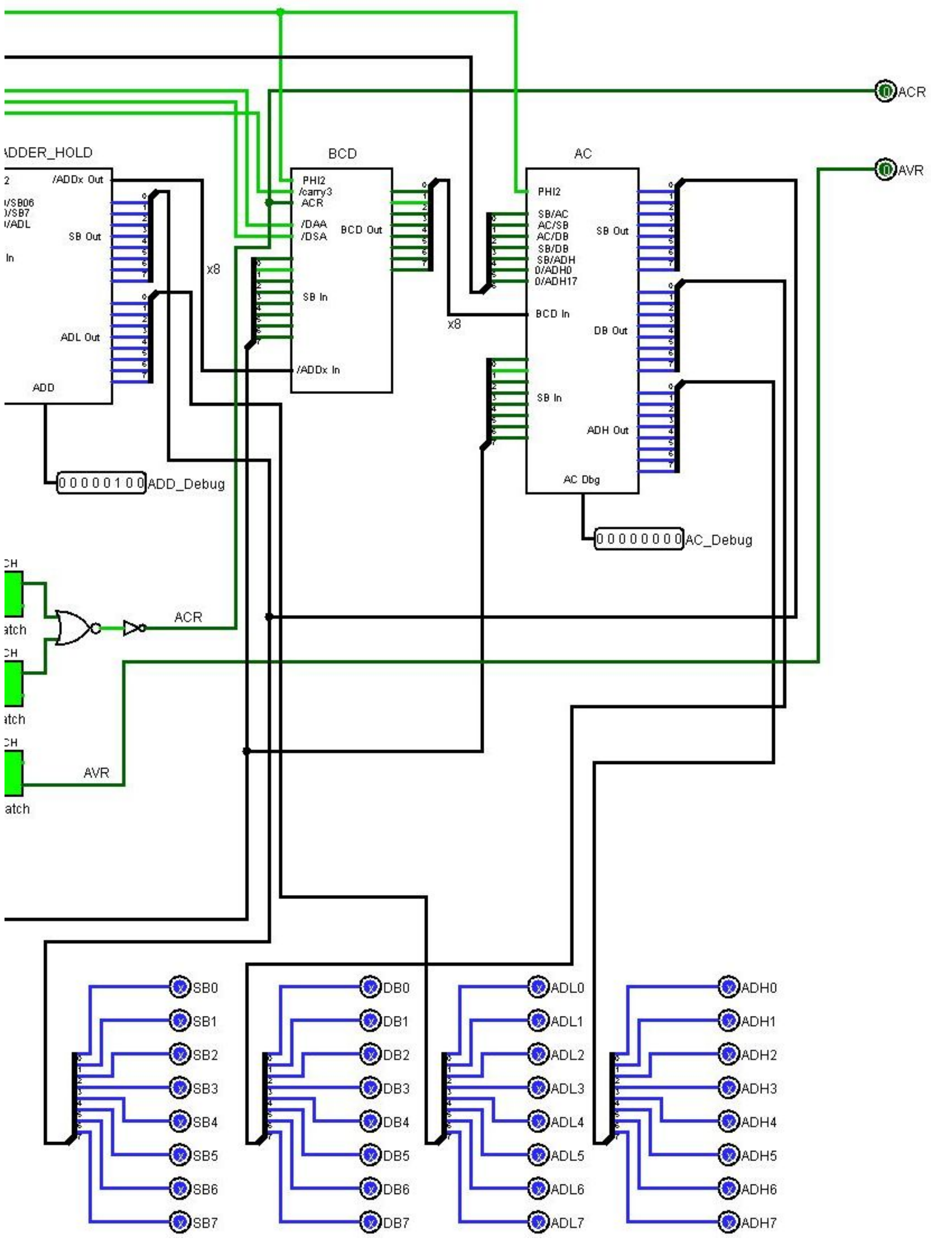


The ALU consists of the following components:

- Input circuits for AI/BI latch loading
- The main computational part (Operations)
- A fast carry calculation circuit for the BCD
- Intermediate result (ADD latch)
- BCD correction circuit
- Accumulator (AC)

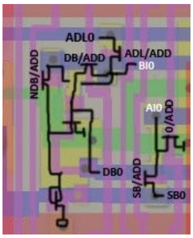
Generally speaking the ALU is a mess of transistors and wires, but its workings are not very complicated, as you can see later.


PHI2
NDB/ADD
DB/ADD
0/ADD
SB/ADD
ADL/ADD
ADD/SB06
ADD/SB7
ADD/ADL
ANDS
EORS
ORS
SRS
SUMS
SB/AC
AC/SB
AC/DB
SB/DB
SB/ADH
0/ADH0
0/ADH17
/ACIN
/DAA
/DSA
ALU_OPS
/carry3
carry4
carry6
/carry7
/res
nor
nand
xor
SB In
DB In
ADL In
/ACIN
DC3
AI
BI
0 0 0 0 0 0 1 0 AI_Debug
0 0 0 0 0 0 1 0 BI_Debug
BCD_CARRY
/ACIN
carry4
/DAA
DC3
DC7
nors
nands
xors
x8
PHI
/res
DCL
ACL
AVRL
SB0
SB1
SB2
SB3
SB4
SB5
SB6
SB7
DB0
DB1
DB2
DB3
DB4
DB5
DB6
DB7
ADL0
ADL1
ADL2
ADL3
ADL4
ADL5
ADL6
ADL7


ADDER_HOLD
/ADDx Out
SB Out
ADL Out
ADD
0 0 0 0 0 1 0 0 ADD_Debug
x8
BCD
PHI2
/carry3
ACR
/DAA
/DSA
BCD Out
SB In
/ADDx In
x8
AC
PHI2
SB/AC
AC/SB
AC/DB
SB/DB
SB/ADH
0/ADH0
0/ADH17
BCD In
SB In
SB Out
DB Out
ADH Out
AC Dbg
0 0 0 0 0 0 0 0 AC_Debug
ACR
AVR
ACR
AVR
SB0
SB1
SB2
SB3
SB4
SB5
SB6
SB7
DB0
DB1
DB2
DB3
DB4
DB5
DB6
DB7
ADL0
ADL1
ADL2
ADL3
ADL4
ADL5
ADL6
ADL7
ADH0
ADH1
ADH2
ADH3
ADH4
ADH5
ADH6
ADH7

AI/BI Latches

The input circuits consist of 8 identical chunks, which are designed to load input values on the AI and BI latches:



Control signals:

- DB/ADD: Load direct value from DB bus to the BI latch
- NDB/ADD: Load inverse value from DB bus to the BI latch
- ADL/ADD: Load a value from the ADL bus to the BI latch
- SB/ADD: Load a value from the SB bus to the AI latch
- 0/ADD: Write 0 to the AI latch

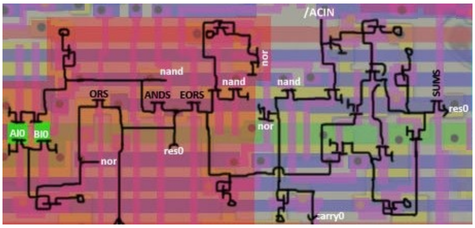
(The picture shows the circuit for bit 0, the rest are the same)

Computational Part

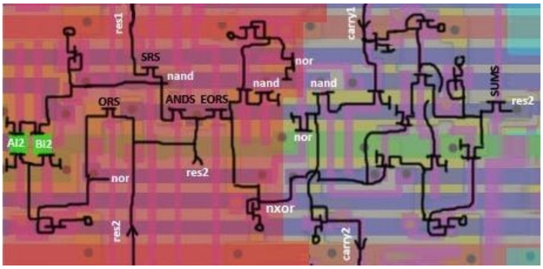
The ALU uses an inverted carry chain, so the even and odd bit circuits alternate.

Bit 0 is slightly different from the other even bits because it has an input carry (`/ACIN`) and no `SRS` input.

Schematic for bit 0:

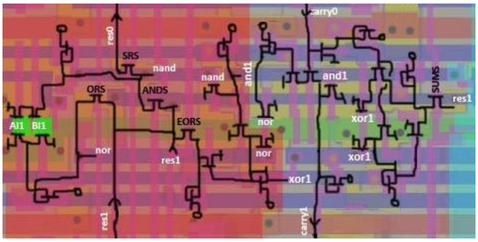


Schematics for bits 2, 4, 6:



(The circuit for bit 2 is shown, the rest are the same)

Schematics for bits 1, 3, 5, 7:



(The circuit for bit 1 is shown, the rest are the same)

Anatomically, the left side deals with logical operations, the right side is the adder (Full Adder), and in the middle is the carry chain.

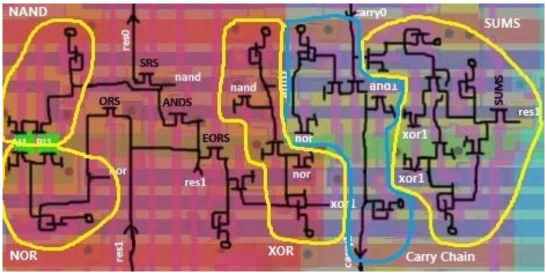
Control signals for ALU operations:

- ORS: Logical OR operation (AI | BI)
- ANDS: Logical AND operation (AI & BI)
- EORS: Logical XOR operation (AI ^ BI)
- SRS: Shift Right. For this the result of the current nand operation is stored as the result of the previous bit.
- SUMS: Summation (AI + BI)

Notations on the schematics:

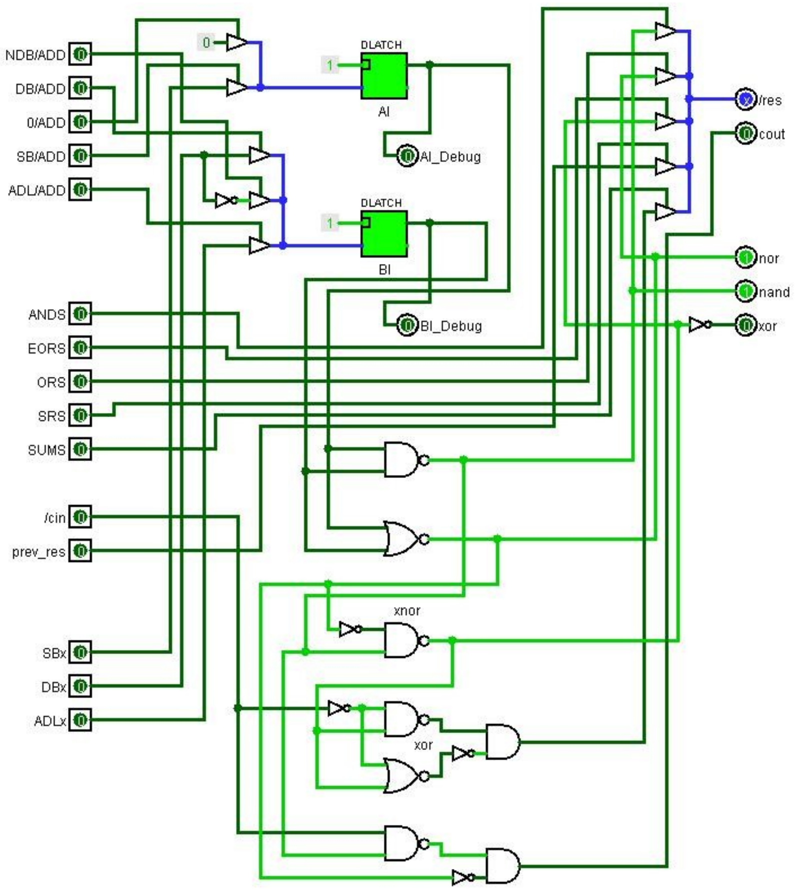
- nand: intermediate result of NAND operation for the selected bit
- and: intermediate result of AND operation for the selected bit (obtained by `nand` inversion)
- nor: intermediate result of NOR operation for a selected bit
- xor: intermediate result of EOR operation for the selected bit
- nxor: intermediate result of an ENOR operation for the selected bit
- carry: the result of a carry operation. The carry chain is inverted every bit, but for simplicity all `carry` names do not consider value inversion.
- res: the result of a logical operation or the result of an adder which is then stored on the ADD latch. The result of an operation in inverted form.

To make it clearer how the intermediate results are obtained, all the main motifs are marked in the image below:

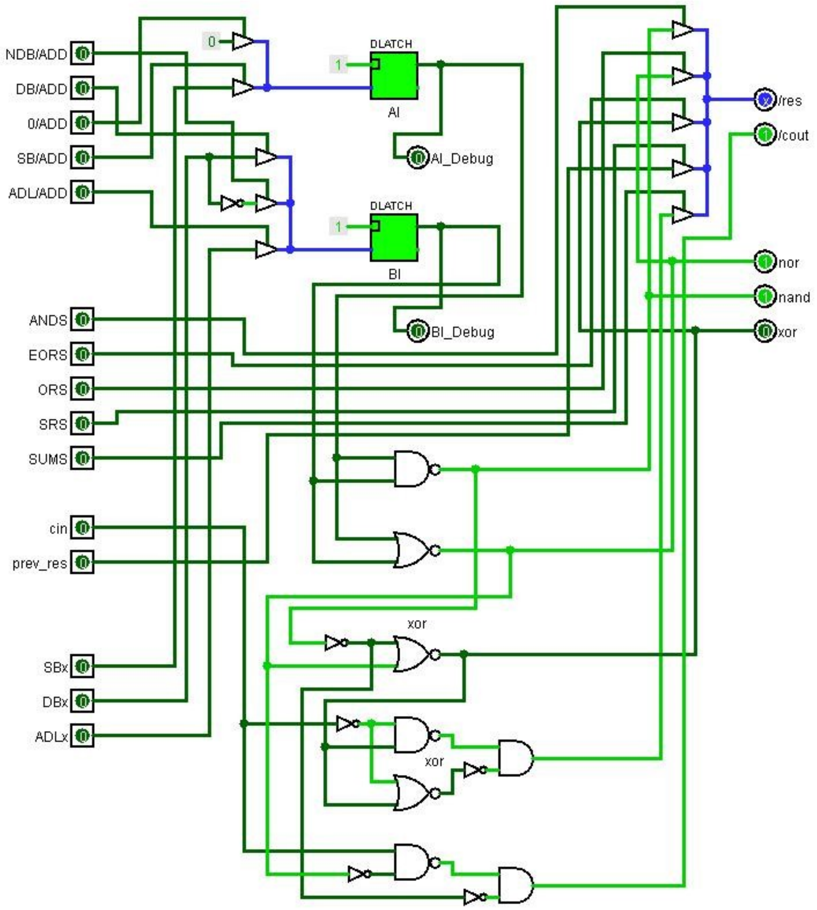


(Bit 1 is shown, for the other bits the motif looks similar)

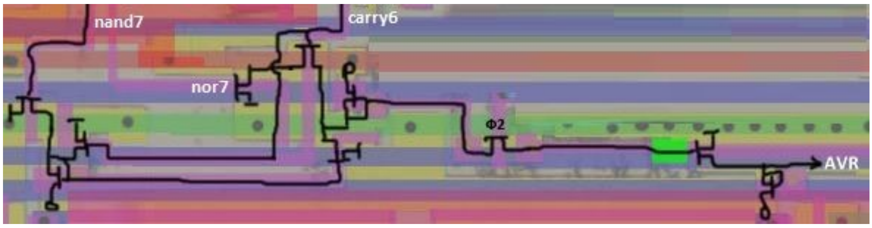
Logic for even bits:



Logic for odd bits:

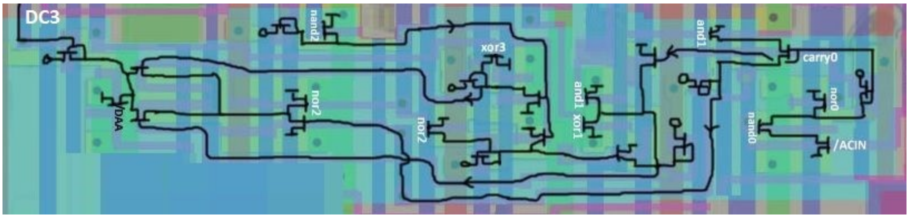


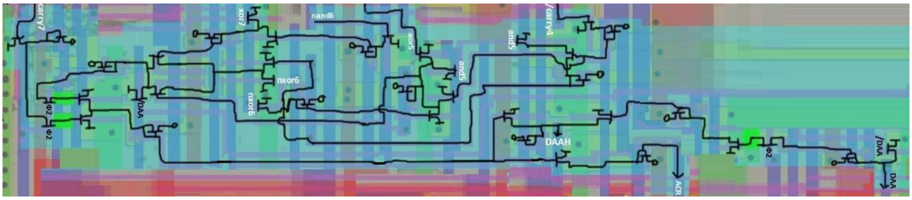
Overflow calculation (control signal `AVR`):



Fast BCD Carry

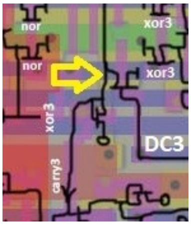
This is the circuit that appears in patent US 3991307 (https://patents.google.com/patent/US3991307A).





The schematics are "layered on the side" for easy perception.

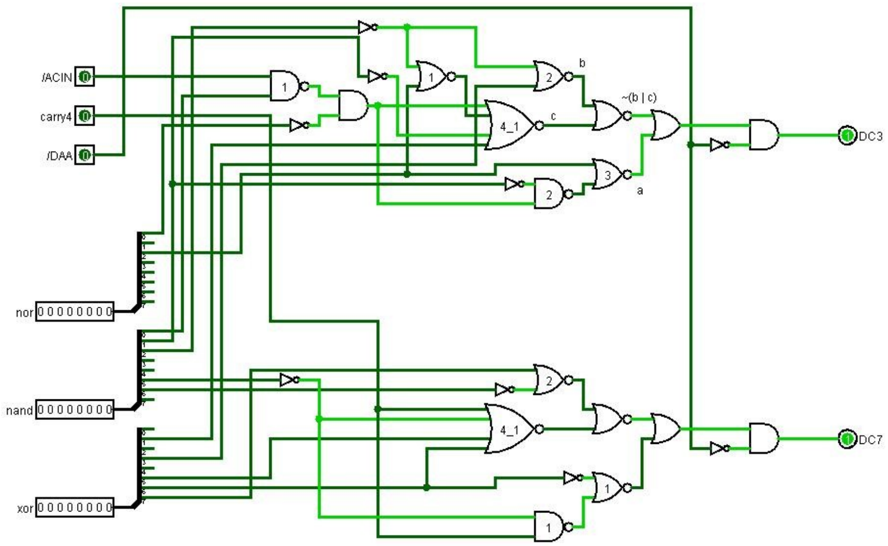
DC3 output is connected to the carry chain as follows:



How exactly this circuit works is written in the patent, I have nothing much to add. Just a mish-mash of logic gates - do the same and it will work.

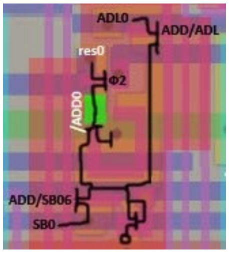
Besides calculating the carry for BCD the circuit also generates the ACR (ALU carry for flags) and DAAH control signals for the BCD correction circuit.

Logic:



Intermediate Result (ADD)

The intermediate result is stored on the ADD latch (stored in inverted form, output to the buses in direct form). The ADD latch circuit consists of 8 identical pieces:



(The circuit is shown for bit 0, the others are the same)

- ADD/SB06: Place the value of the ADD latch on the SB bus. The control signal `ADD/SB7` is used instead of ADD/SB06 for bit 7.
- ADD/ADL: Place the ADD latch value on the ADL bus

BCD Correction

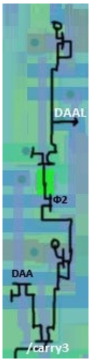
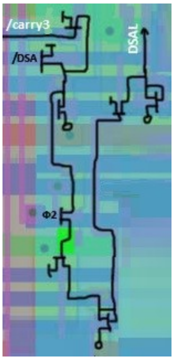
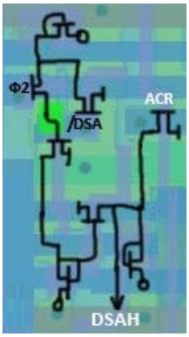
The BCD correction circuit is controlled by two signals: `/DAA` (perform correction after addition) and `/DSA` (perform correction after subtraction).

The outputs of the circuit are connected to the accumulator inputs (AC) and the circuit takes into account the ALU operation when the BCD mode is disabled.

Some of the accumulator inputs are connected directly to the SB bus and do not participate in BCD correction (bits 0 and 4).

The circuit uses 4 auxiliary internal signals in its operation: DAAL, DAAH, DSAL and DSAH. The "L" in the name stands for the lower part of the bits (0-3), the "H" stands for the higher part of the bits (4-7).

Circuits for obtaining auxiliary signals:

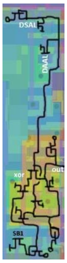
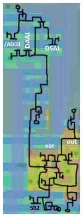
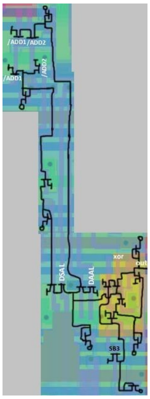
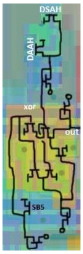
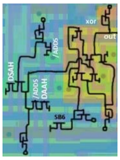
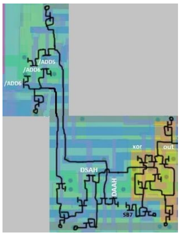
DAAL	DSAL	DSAH
		

The `DAAH` circuit is in the carry circuit.

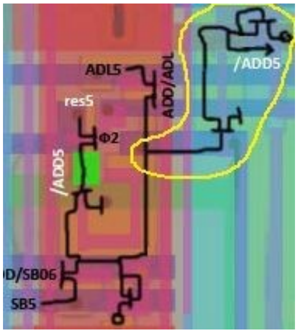
The correction circuits use a common motif:

- The input combinatorial circuits, in various combinations accounting for the 4 auxiliary signals and the bits of the intermediate result (ADD latches)
- Output xor, one of the inputs of which is a bit of the bus SB, and the second of the above combinatorial circuits

Sawed schematics:

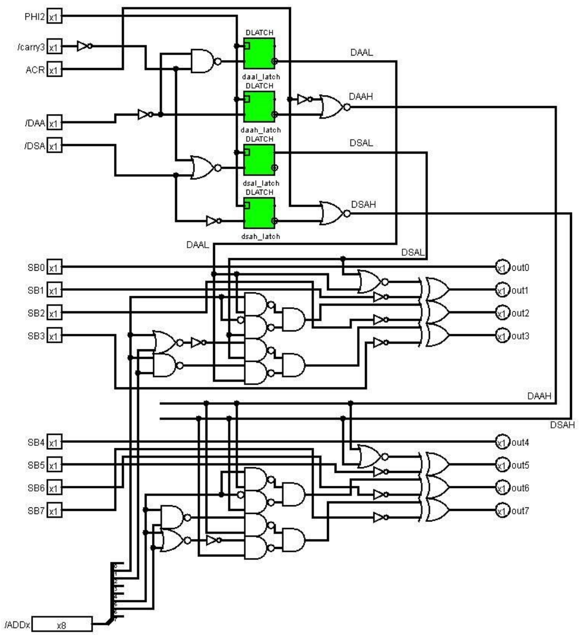
Bit 1	Bit 2	Bit 3	Bit 5	Bit 6	Bit 7
		 /ADD1 /ADD2 /ADD1 /ADD2 DSAL DAAL xor out SB3			 /ADD5 /ADD6 /ADD6 xor out DSAH DAAH SB7

The auxiliary signals /ADDx on the BCD correction circuits are derived from the values of the ADD latch bits as follows:



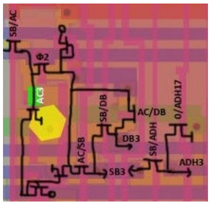
(Using `/ADD5` as an example)

Logic:



Accumulator (AC)

The accumulator consists of eight identical pieces:



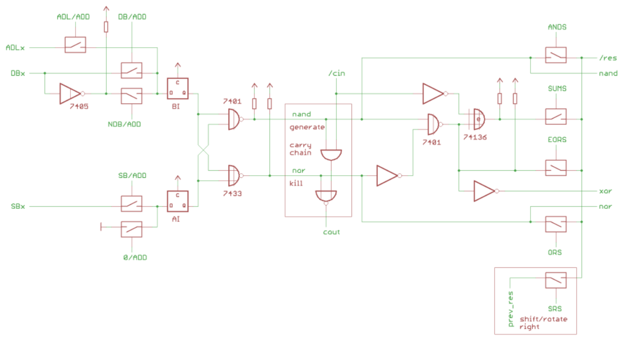
(The circuit for bit 3 is shown, the others are the same)

The accumulator inputs a value from the BCD correction circuit (bits 1-3, 5-7) or directly from the SB bus (bits 0 and 4).

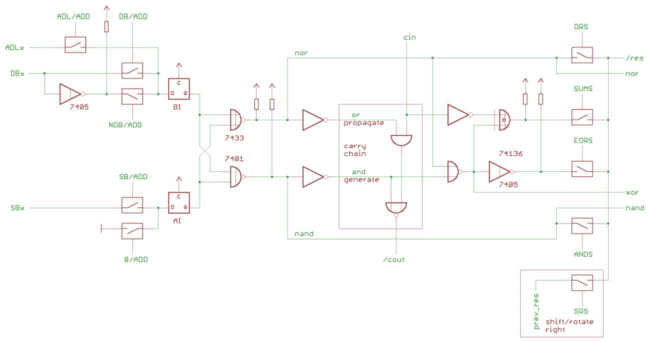
In addition to directly outputting the accumulator to the SB and DB buses, other bus operations are also performed at this point, so they are also discussed in this section.

- SB/AC: Place the value from the SB bus/BCD correction circuit into the accumulator
- AC/SB: Place the AC value on the SB bus
- AC/DB: Place the AC value on the DB bus
- SB/DB: Connect the SB bus to DB bus
- SB/ADH: Connect the SB bus to ADH bus
- 0/ADH17: Forced write 0 to ADH bits 1-7. The control signal `0/ADH0` is used for bit 0 instead of 0/ADH17.

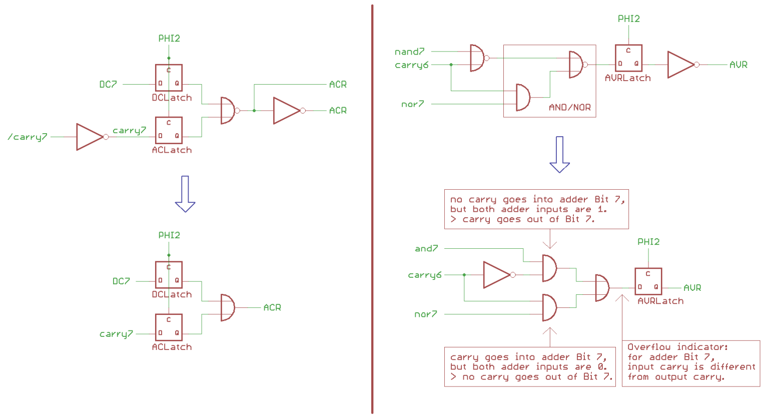
Optimized Schematics



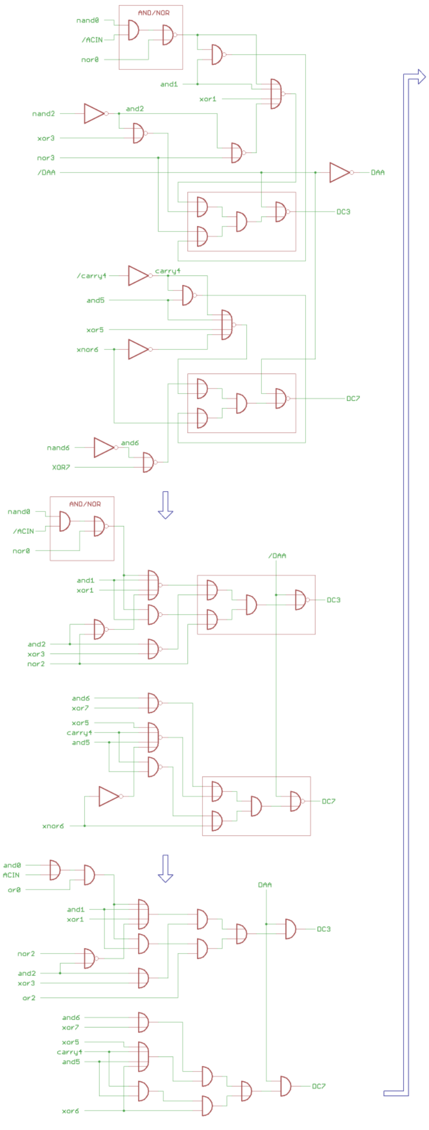
https://github.com/emu-russia/breaks/blob/master/BreakingNESWiki_DeepL/6502/alu.md
1:https://github.com/emu-russia/breaks/blob/master/BreakingNESWiki/imgstore/logisim/alu_even_bit_logisim.jpg

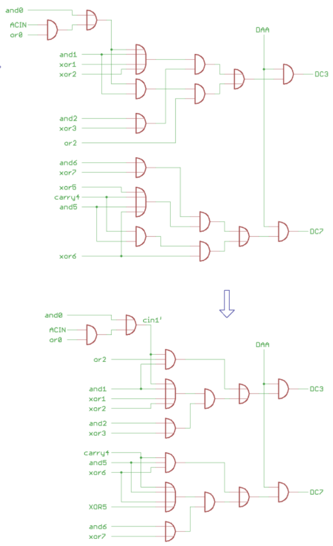


https://github.com/emu-russia/breaks/blob/master/BreakingNESWiki_DeepL/6502/alu.md
2:https://github.com/emu-russia/breaks/blob/master/BreakingNESWiki/imgstore/logisim/alu_odd_bit_logisim.jpg

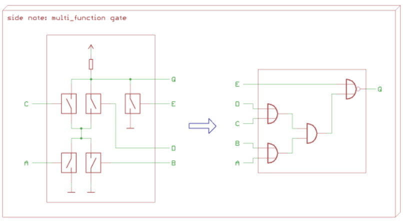

PHI2
DC7
DCLatch
ACR
ACR
/carry7
carry7
ACLatch
PHI2
DC7
DCLatch
ACR
carry7
ACLatch
nand7
carry6
nor7
PHI2
AVRLatch
AVR
AND/NOR
no carry goes into adder Bit 7, but both adder inputs are 1. > carry goes out of Bit 7.
and7
carry6
nor7
PHI2
AVR
AVRLatch
carry goes into adder Bit 7, but both adder inputs are 0. > no carry goes out of Bit 7.
Overflow indicator: for adder Bit 7, input carry is different from output carry.

https://github.com/emu-russia/breaks/blob/master/BreakingNESWiki_DeepL/6502/alu.md
5:carry_plus_overflow_evaluation

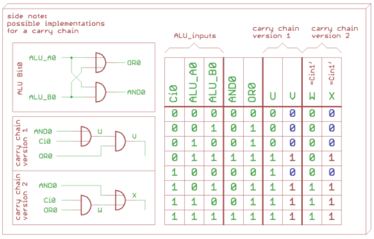




side note: multi_function gate

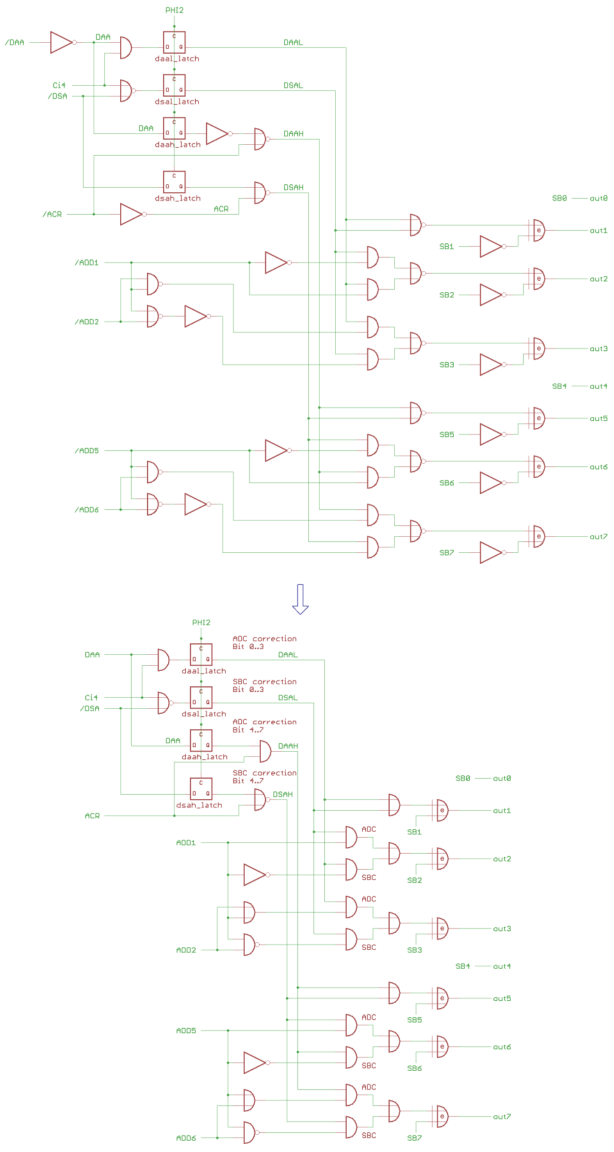


side note:
possible implementations
for a carry chain



ALU_inputs					carry chain version 1		carry chain version 2	
Ci0	ALU_A0	ALU_B0	AND0	OR0	U	V	W =Cin1'	X =Cin1'
0	0	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0
0	1	0	0	1	0	0	0	0
0	1	1	1	1	1	1	0	1
1	0	0	0	0	1	0	0	0
1	0	1	0	1	1	1	1	1
1	1	0	0	1	1	1	1	1
1	1	1	1	1	1	1	1	1

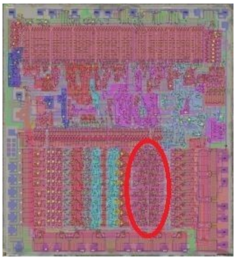
https://github.com/emu-russia/breaks/blob/master/BreakingNESWiki_DeepL/6502/alu.md
3:https://github.com/emu-russia/breaks/blob/master/BreakingNESWiki/imgstore/logisim/alu_bcd_carry_logisim.jpg



https://github.com/emu-russia/breaks/blob/master/BreakingNESWiki_DeepL/6502/alu.md
4:https://github.com/emu-russia/breaks/blob/master/BreakingNESWiki/imgstore/logisim/alu_bcd_logisim.jpg

Notes in the margins for future revisions of the book.

Program Counter (PC)

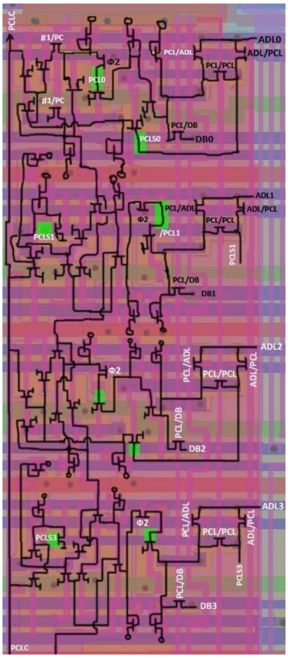
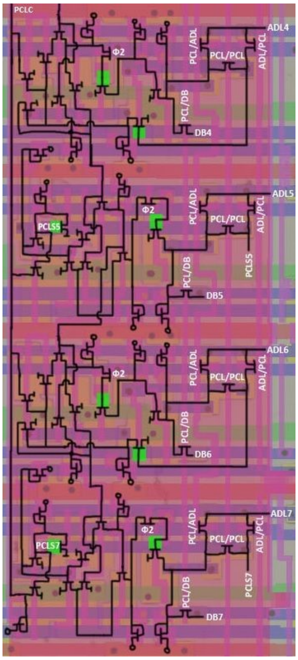


Because of the 8-bit nature of the processor its instruction counter is divided into two 8-bit halves: PCL (Program Counter Low) and PCH (Program Counter High).

The PCH is also divided into two halves: the low part of the bits (0-3) and the high part (4-7).

PCL

Represents the low 8 least significant bits of PC.

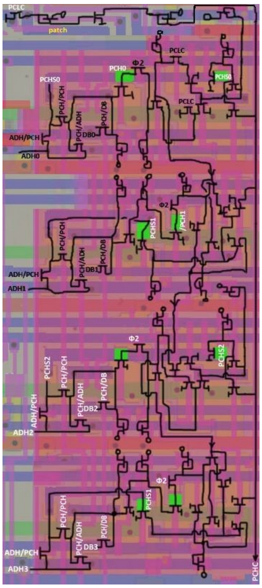
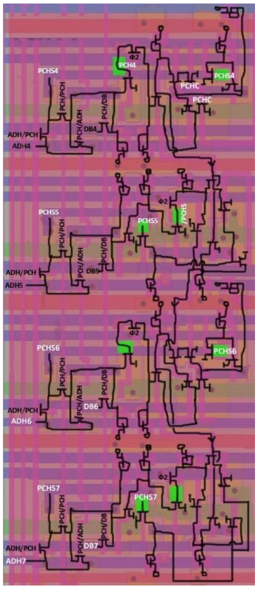
PCL 0-3	PCL 4-7
 PCLC #1/PC Φ2 PCL0 PCL/ADL ADL0 ADL/PCL PCL/PCL #1/PC PCL/DB PCLS0 DB0 ADL1 PCL/ADL ADL/PCL Φ2 PCL/PCL PCLS1 /PCL1 PCLS1 PCL/DB DB1 ADL2 PCL/ADL Φ2 PCL/PCL ADL/PCL PCL/DB DB2 ADL3 PCL/ADL Φ2 PCLS3 PCL/PCL ADL/PCL PCLS3 PCL/DB DB3 PCLC	 PCLC ADL4 Φ2 PCL/ADL PCL/PCL ADL/PCL PCL/DB DB4 ADL5 PCL/ADL Φ2 PCLS5 PCL/PCL ADL/PCL PCLS5 PCL/DB DB5 ADL6 PCL/ADL Φ2 PCL/PCL ADL/PCL PCL/DB DB6 ADL7 PCL/ADL Φ2 PCLS7 PCL/PCL ADL/PCL PCLS7 PCL/DB DB7

- The circuits alternate for even and odd bits because an optimization known as an inverted carry chain is used
- The control signal `#1/PC` (0: perform PC increment) comes to the PCL0 bit
- PCLC (PCL Carry): Carry from the lowest 8 bits (PC[0-7]) to the highest (PC[8-15])
- PCL connects to two buses: ADL and DB
- PCL/PCL is used when PCL is not connected to any bus (to maintain the current state)
- Each bit contains two latches (input latch `PCLSx` and output latch `PCLx`) which implement the counter logic

PCH

Represents the top 8 most significant bits of PC.

The circuits for the even bits (0, 2, ...) of the PCH repeat the circuits for the odd bits (1, 3, ...) of the PCL. Similarly, circuits for odd bits (1, 3, ...) of PCH repeat circuits for even bits (0, 2, ...) of PCL.

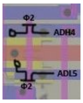
PCH 0-3	PCH 4-7
	

The circuit marked as "patch" to form the `PCHC` is actually between the `ADL/PCL` and `#1/PC` control outputs.

- The basic principles of PCH are the same as PCL, but PCH is divided into two halves: the lower half (PCH0-3) and the higher half (PCH4-7)
- PCHC (PCH Carry): Carry from the lowermost to the highestermost PCH half
- The PCH connects to two buses: ADH and DB
- PCH/PCH is used when the PCH is not connected to any bus (to maintain the current state)

ADL/ADH Precharge

In between the PC bits you can find transistors for precharge of the ADL and ADH buses:

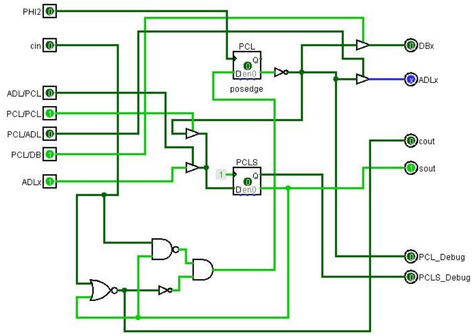


(The image shows the precharge transistors for ADH4 and ADL5. The others are similar)

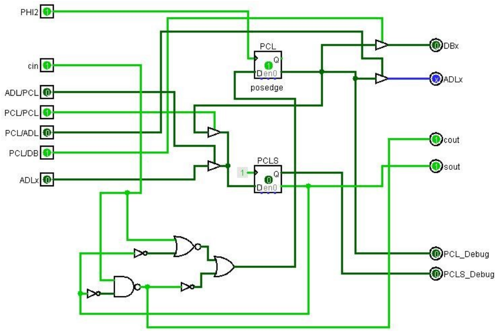
Logic

It makes sense to show only the bit schematics (the circuitry alternates between even and odd PCL/PCH bits).

This circuit is used, for example, in PCL0:

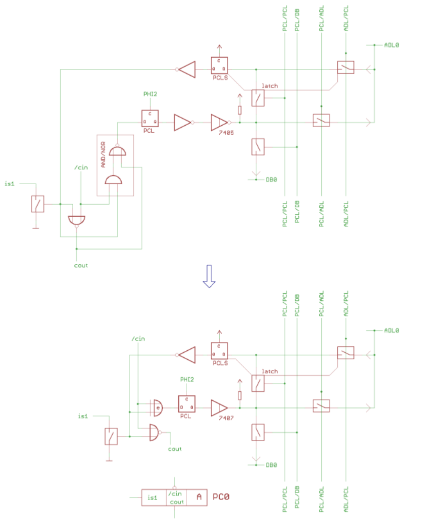


This circuit is used, for example, in PCL1:



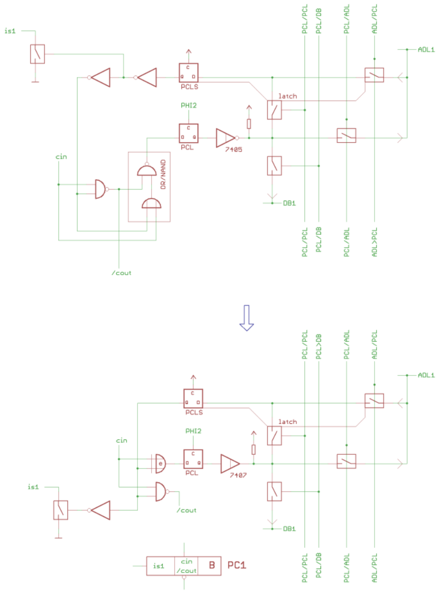
For these circuits to work correctly in the simulator, FF uses a posedge trigger for the PCL/PCH register.

Optimized schematics (Even):



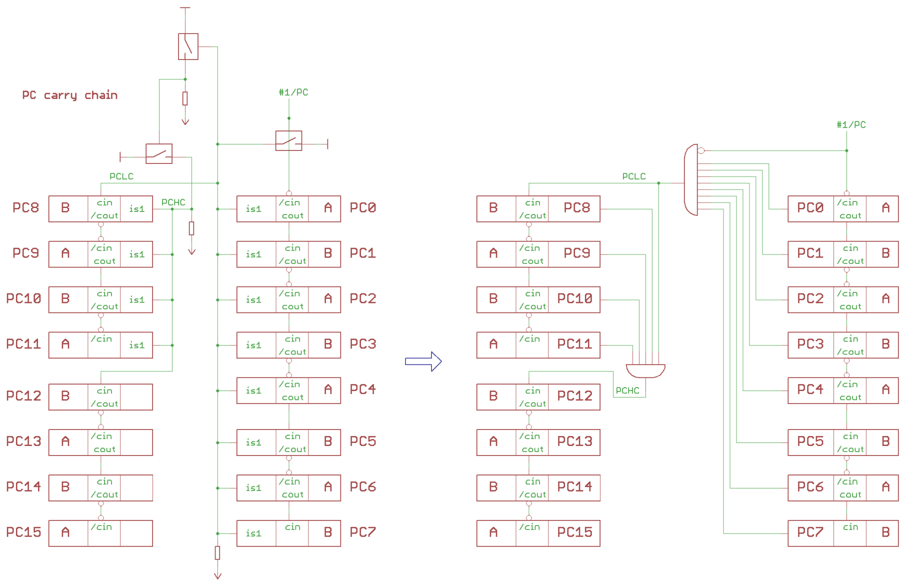
https://github.com/emu-russia/breaks/blob/master/BreakingNESWiki_DeepL/6502/pc.md
20:https://github.com/emu-russia/breaks/blob/master/BreakingNESWiki/imgstore/pc_even_bit_logisim.jpg

Optimized schematics (Odd):



https://github.com/emu-russia/breaks/blob/master/BreakingNESWiki_DeepL/6502/pc.md
21:https://github.com/emu-russia/breaks/blob/master/BreakingNESWiki/imgstore/pc_odd_bit_logisim.jpg

Optimized logic circuit for the carry chain:



https://github.com/emu-russia/breaks/blob/master/BreakingNESWiki_DeepL/6502/pc.md
22:PC_carry_chain

Notes in the margins for future revisions of the book.

6502 Operations

This section describes how the processor performs various operations:

- Processor Reset, Interrupts and BRK Sequence
- A description of the operation of the various instructions
- Reaction to external signals (RDY, SO pins)
- Undefined Behavior (execution of opcodes not provided by processor's developers)

Operation description motif:

- The operation is divided into cycles (T0, T1, and so on)
- Each cycle is considered in two half-cycles (PHI1 / PHI2)
- The state of processor internals (signals, registers, buses) is described by tables at each half-cycle

Instruction operation is considered in context between two `nop` operations:

```
nop
<instr>
nop
```

This is because the overlap causes some instructions to "finish" their work at the first half-cycle (PHI1) of the next instruction. This is why a wrapper in the form of `nop` is used, as a maximum non-invasive variant of work.

This revision of the book covers only the following operations:

- BRK Sequence at Power Up
- JSR instruction
- LDA #imm instruction
- NOP instruction

In this way we clear the way for those who like to delve into the inner workings of circuits and study them in detail, by analogy.

You can find a description of the rest of the operations on the project Wiki on GitHub.

BRK Sequence

BRK-sequence is a unified mechanism of CPU reaction to external interrupt signals (`/NMI`, `/IRQ`, `/RES`) and also to execution of instruction `BRK` (0x00).

In the main part has already been mentioned how the developers have approached the unification of this mechanism (injection of the operation code 0x00 in the register of instructions, etc.), in this section is a more detailed analysis.

Further consideration of the state is made with the assumption that input `RDY` = 1 (processor ready).

BRK Software Model

General information for programmers, sufficient for a general understanding of the BRK-sequencing process.

Cycle	Operation
T0	Load BRK opcode (instruction) / inject BRK (interrupt). Increment PC if a BRK instruction is executed directly
T1	Load and discard data. Increment PC if a BRK instruction is executed directly
T2	Put PCH on the stack
T3	Put PCL on the stack
T4	Put P on the stack
T5	Read interrupt vector address (low byte)
T6	Read interrupt vector address (high byte)

Включение питания

При включении питания процессор выполняет особенную последовательность (Pre-BRK).

UB (0xFF), T1 (PHI1)

Top Part:

State	Note
Interrupt processing	
RESP=1	This is not an effect of /RES=0 because the input FF of the /RES pin is only updated during PHI2. This is the effect of the output `resp_latch`.
DORES=0	The input DORES_FF latch is only updated with the RESP signal during PHI2.
BRK6E=0	The value of the output latch is undefined, so through the inverter the value of BRK6E takes the value 0.
B_OUT=0	Although DORES = 0, the output latch value of the B flag is undefined and BRK6E = 0, so B_OUT = 0.
Dispatcher	
/ready = 0	The value of the output /ready latch is updated during PHI2. At the time of power up the value of the latch is undefined, as a result /ready takes the value 0.
WR=1	WR is generated by a 3-NOR operation whose inputs (/ready, DORES, wr_latch) take the value 0. As a result WR = 1
FETCH=0	The value of the output latch of the FETCH circuit is not yet defined (updated during PHI2).
0/IR=1	Since FETCH = 0 and B_OUT = 0.
ENDS = 1	The values of the output latches of the ENDS circuit are undefined.
TRES1 = 1	Since ENDS = 1
TRESX=0	The TRESX circuit includes a latch whose value is not yet defined (PHI2). And through the inverter and NOR - TRESX as a result takes the value 0.
/TWOCYCLE=1	PD=0x00
TRES2=1	Since TRESX = 0
/T0 = 1	According to the circuit
/T1X = 0	According to the circuit
/T2-/T5 = 1	TRES2 = 1
Decoder	
44: INC NOP (TX), 60: ADC SBC (T1), 106: LSR ROR DEC INC DEX NOP (4x4 bottom right) (TX), 112: ADC SBC (T1)	Executes 0xFF/T1. The operation makes no sense because all random logic output latches are only updated during PHI2.
PD=0x00	As a result of 0/IR=1 the value of PD = 0x00
IR=0xFF	The value of the IR register is not updated (FETCH = 0), and it is arranged so that the decoder gets the value 0xFF

Bottom Part:

As a result of the fact that the output commands are only loaded during PHI2 - their output values immediately after power up are undefined.

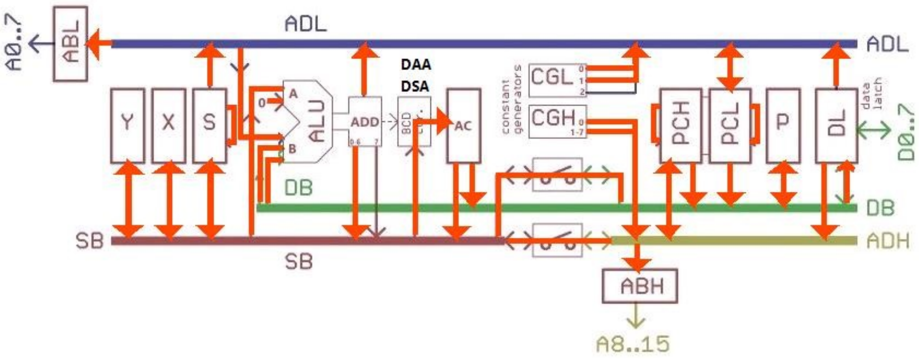
This causes almost all commands of the bottom part to be active (the lack of charge on the output latches' gates causes them to have 1 on the output). In this case the bottom part "goes crazy".

State	Note
ADH/ABH, ADL/ABL	The address bus takes the value 0x0000
0/ADL0	1
0/ADL1	1
0/ADL2	0
0/ADH0, 0/ADH17	Both active commands cause the ADH bus to have a value of 0x00.
Y/SB, SB/Y, X/SB, SB/X, S/ADL, S/SB, SB/S, S/S	Y/SB, X/SB, SB/S, S/S have no effect, because the register is updated only during PHI2. As a result, the current value of registers X, Y, S simultaneously placed on the bus SB (for the register S - also on the bus ADL command S/ADL). The peculiarity of the register S is that the value from the output latch is output in inverted form. That is, on the bus SB and ADL placed value 0xFF (S = 0). But since the X/Y registers have already put value 0x00 on the bus before that, the ground wins and the SB bus takes value 0x00
NDB/ADD, DB/ADD, 0/ADD, SB/ADD, ADL/ADD, ADD/SB06, ADD/ADL, SB/AC, AC/SB, AC/DB, SB/DB, SB/ADH	AI: The 0/ADD, SB/ADD commands have the effect of loading 0 on the AI latch and simultaneously "grounding" the SB bus (SB/ADD opens the SB bus and 0/ADD zeroes it). But this makes no sense because the SB is already grounded by register operations. BI: There is no point in considering (?). ALU Output: There is no point in considering (?). ADD/SB7 = 0 because of the nature of its output latch (but it makes no sense now either)
ALU Operations	All disabled
/ACIN	TBD
/DAA	TBD
/DSA	TBD
#1/PC	TBD
ADH/PCH, PCH/PCH, PCH/ADH, PCH/DB, ADL/PCL, PCL/PCL, PCL/ADL, PCL/DB	TBD
RD = 0	According to WR = 1
DL/ADL, DL/ADH, DL/DB	Setting the DL/ADL and DL/ADH commands simultaneously causes the ADL/ADH buses to be shorted, causing them both to become 0x00 (the ADH is already grounded by the 0/ADH0, 0/ADH17 commands). DL/DB also causes the DB bus to be grounded. The DOR latch = 0x00.
SB	0x00
DB	0x00
ADL	0x00
ADH	0x00

Phenomenon: All parts of the processor "go crazy", but miraculously all the operations cause the processor to write 0x00 to address 0x0000.

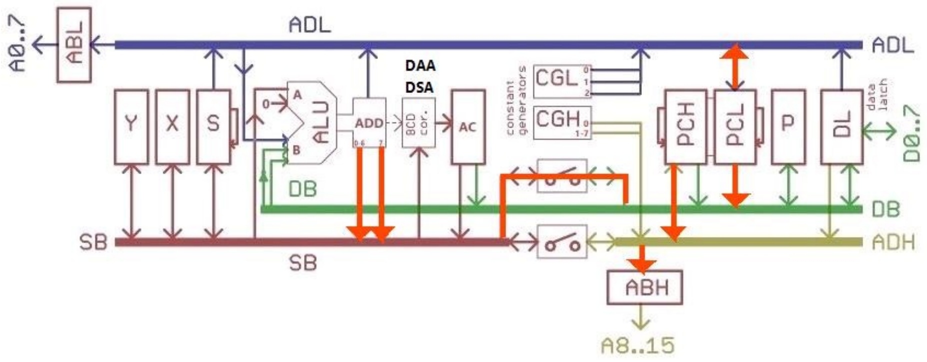
The image below is a schematic representation of the connections of the lower part of the processor. The currently active commands of the lower part are highlighted.

Similar images will be found further on to explain each half-cycle of the operation in progress.



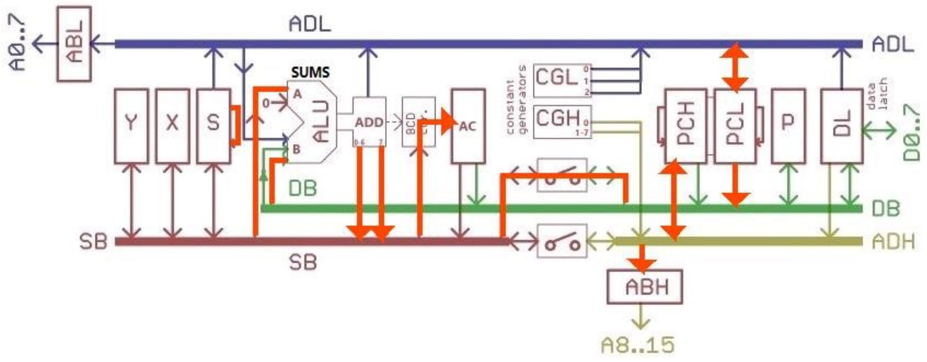
UB (0xFF), T1 (PHI2)

Component/	State
Dispatcher	T0: 0, /T0: 1, /T1X: 0, 0/IR: 1, FETCH: 1, /ready: 0, WR: 0, ACRL1: 1, ACRL2: 1, T5: 0, T6: 1, ENDS: 0,
Interrupts	/NMIP: 1, /IRQP: 1, RESP: 1, BRK6E: 0, BRK7: 1, DORES: 1, /DONMI: 0
Extra Cycle	T1: 1, TRES2: 1, /T2: 1, /T3: 1, /T4: 1, /T5: 1
Decoder	44: INC NOP (TX), 60: ADC SBC (T1), 106: LSR ROR DEC INC DEX NOP (4x4 bottom right) (TX), 112:
Commands	ADD_SB7, ADD_SB06, PCH_ADH, PCL_ADL, PCL_DB, ADH_ABH, SB_DB, DBZ_Z, DB_N, ACR_C, AVR_V
ALU Carry In	0
DAA	1
DSA	1
Increment PC	0
Regs	IR = 0xFF, PD = 0x00, Y = 0x00, X = 0x00, S = 0x00, AI = 0x00, BI = 0xFC, ADD = 0xFF, AC = 0x0A
PCL	0x00
PCH	0x00
ABL	0x00
ABH	0x00
DL	0x00
DOR	0x00
Flags	C: 0, Z: 0, I: 0, D: 0, B: 0, V: 0, N: 0
Buses	SB = 0xFF, DB = 0x00, ADL = 0x00, ADH = 0x00



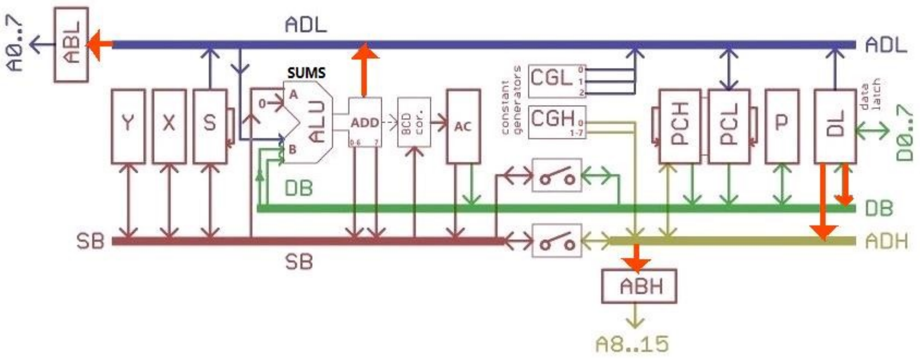
PreBRK (0x00), T0 (PHI1)

Component/	State
Dispatcher	T0: 1, /T0: 0, /T1X: 1, 0/IR: 1, FETCH: 1, /ready: 0, WR: 0, ACRL1: 1, ACRL2: 1, T5: 0, T6: 0, ENDS: 0,
Interrupts	/NMIP: 1, /IRQP: 1, RESP: 1, BRK6E: 0, BRK7: 1, DORES: 1, /DONMI: 0
Extra Cycle	T1: 0, TRES2: 1, /T2: 1, /T3: 1, /T4: 1, /T5: 1
Decoder	34: T0 ANY, 87: BRK RTI (T0), 94: BRK RTI (TX), 121: /IR6, 126: /IR7
Commands	S_S, DB_ADD, SB_ADD, SUMS, ADD_SB7, ADD_SB06, SB_AC, ADH_PCH, PCH_ADH, ADL_PCL,
ALU Carry In	0
DAA	0
DSA	0
Increment PC	0
Regs	IR = 0x00, PD = 0x00, Y = 0x00, X= 0x00, S = 0x00, AI = 0xFF, BI = 0x00, ADD = 0xFF, AC = 0xAA
PCL	0x00
PCH	0x00
ABL	0x00
ABH	0x00
DL	0x00
DOR	0x00
Flags	C: 1, Z: 1, I: 0, D: 0, B: 0, V: 0, N: 0
Buses	SB = 0x00, DB = 0x00, ADL = 0x00, ADH = 0x00



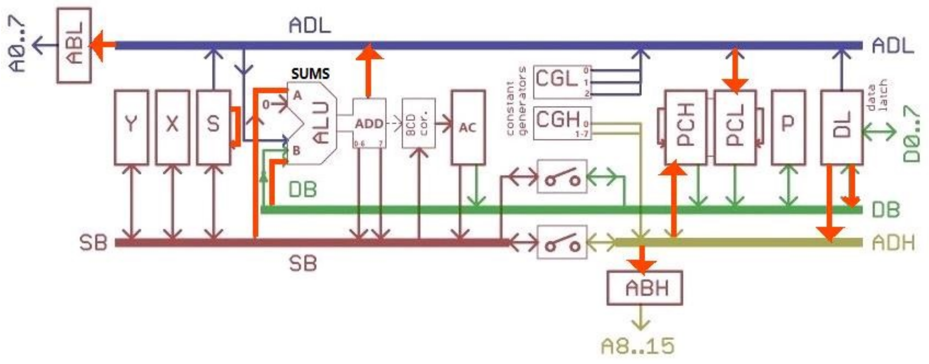
PreBRK (0x00), T0 (PHI2)

Component/Signal	State
Dispatcher	T0: 1, /T0: 0, /T1X: 1, 0/IR: 1, FETCH: 0, /ready: 0, WR: 0, ACRL1: 0, ACRL2: 1, T5: 0, T6:
Interrupts	/NMIP: 1, /IRQP: 1, RESP: 1, BRK6E: 0, BRK7: 1, DORES: 1, /DONMI: 0
Extra Cycle Counter	T1: 0, TRES2: 1, /T2: 1, /T3: 1, /T4: 1, /T5: 1
Decoder	34: T0 ANY, 87: BRK RTI (T0), 94: BRK RTI (TX), 121: /IR6, 126: /IR7
Commands	SUMS, ADD_ADL, ADH_ABH, ADL_ABL, DL_ADH, DL_DB
ALU Carry In	0
DAA	0
DSA	0
Increment PC	0
Regs	IR = 0x00, PD = 0x00, Y = 0x00, X = 0x00, S = 0x00, AI = 0xFF, BI = 0x00, ADD = 0xFF, AC = 0xAA
PCL	0x00
PCH	0x00
ABL	0x00
ABH	0x00
DL	0x00
DOR	0x00
Flags	C: 1, Z: 1, I: 0, D: 0, B: 0, V: 0, N: 0
Buses	SB = 0xFF, DB = 0xFF, ADL = 0xFF, ADH = 0xFF



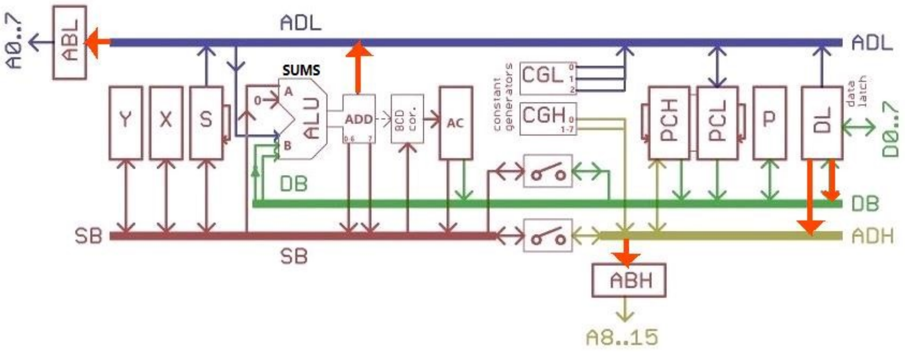
PreBRK (0x00), T01 (PHI1)

Component/Signal	State
Dispatcher	T0: 1, /T0: 0, /T1X: 0, 0/IR: 1, FETCH: 0, /ready: 0, WR: 0, ACRL1: 0, ACRL2: 0, T5: 0, T6: 0, ENDS: 0, ENDX: 1, TRES1: 0, TRESX: 0
Interrupts	/NMIP: 1, /IRQP: 1, RESP: 1, BRK6E: 0, BRK7: 1, DORES: 1, /DONMI: 0
Extra Cycle Counter	T1: 0, TRES2: 1, /T2: 1, /T3: 1, /T4: 1, /T5: 1
Decoder	34: T0 ANY, 87: BRK RTI (T0), 94: BRK RTI (TX), 121: /IR6, 126: /IR7
Commands	S_S, DB_ADD, SB_ADD, SUMS, ADD_ADL, ADH_PCH, ADL_PCL, ADH_ABH, ADL_ABL, DL_ADH, DL_DB
ALU Carry In	0
DAA	0
DSA	0
Increment PC	0
Regs	IR = 0x00, PD = 0x00, Y = 0x00, X = 0x00, S = 0x00, AI = 0xFF, BI = 0x00, ADD = 0xFF, AC = 0xAA
PCL	0xFF
PCH	0x00
ABL	0xFF
ABH	0x00
DL	0x00
DOR	0x00
Flags	C: 1, Z: 1, I: 0, D: 0, B: 0, V: 0, N: 0
Buses	SB = 0xFF, DB = 0x00, ADL = 0xFF, ADH = 0x00



PreBRK (0x00), T01 (PHI2)

Component/Signal	State
Dispatcher	T0: 1, /T0: 0, /T1X: 0, 0/IR: 1, FETCH: 0, /ready: 0, WR: 0, ACRL1: 0, ACRL2: 0, T5: 0, T6: 0,
Interrupts	/NMIP: 1, /IRQP: 1, RESP: 1, BRK6E: 0, BRK7: 1, DORES: 1, /DONMI: 0
Extra Cycle Counter	T1: 0, TRES2: 1, /T2: 1, /T3: 1, /T4: 1, /T5: 1
Decoder	34: T0 ANY, 87: BRK RTI (T0), 94: BRK RTI (TX), 121: /IR6, 126: /IR7
Commands	SUMS, ADD_ADL, ADH_ABH, ADL_ABL, DL_ADH, DL_DB
ALU Carry In	0
DAA	0
DSA	0
Increment PC	0
Regs	IR = 0x00, PD = 0x00, Y = 0x00, X = 0x00, S = 0x00, AI = 0xFF, BI = 0x00, ADD = 0xFF,
PCL	0xFF
PCH	0x00
ABL	0xFF
ABH	0x00
DL	0x00
DOR	0x00
Flags	C: 1, Z: 1, I: 0, D: 0, B: 0, V: 0, N: 0
Buses	SB = 0xFF, DB = 0xFF, ADL = 0xFF, ADH = 0xFF



Further, until /RES takes value 1 - processor will execute in PreBRK loop T0+T1.

Notes in the margins for future revisions of the book.

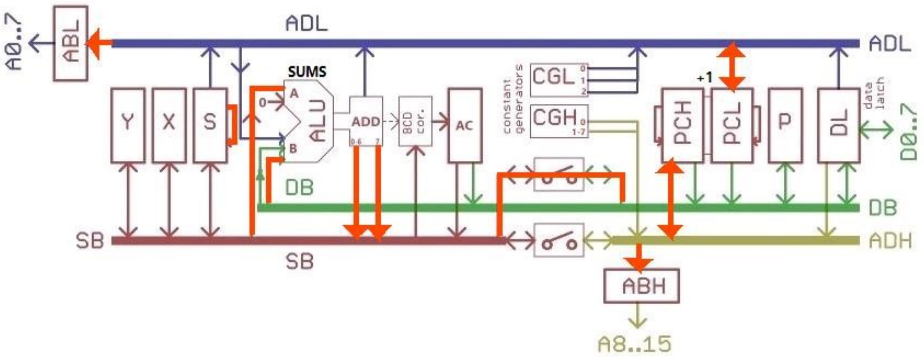
JSR (0x20)

Timing:

- T2: Read new PCL
- T3: Dummy read from stack
- T4: Write return PCH to stack
- T5: Write return PCL to stack
- T0: Read new PCH
- T1: Set new PC + Fetch next opcode

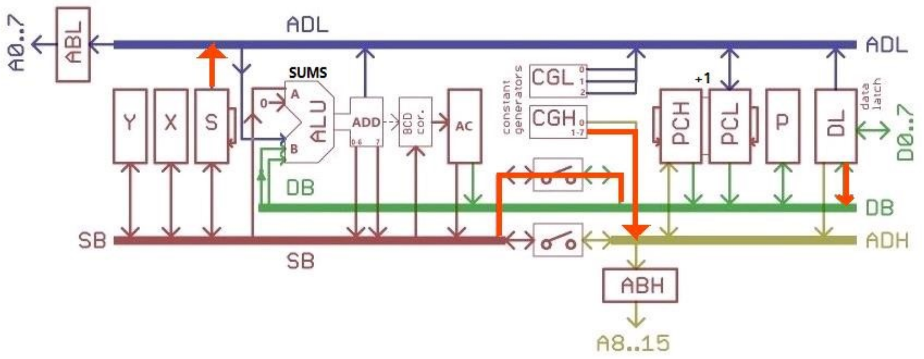
JSR (0x20), T2 (PHI1)

Component/ Signal	State
Dispatcher	T0: 0, /T0: 1, /T1X: 1, 0/IR: 0, FETCH: 1, /ready: 0, WR: 0, ACRL1: 1, ACRL2: 0, T5: 0, T6: 0, ENDS: 0,
Interrupts	/NMIP: 1, /IRQP: 1, RESP: 0, BRK6E: 0, BRK7: 1, DORES: 0, /DONMI: 1
Extra Cycle Counter	T1: 0, TRES2: 0, /T2: 0, /T3: 1, /T4: 1, /T5: 1
Decoder	28: T2, 33: LEFT_ALL (T2), 35: STK2, 48: JSR2, 57: BRK JSR RTI RTS Push/pull (T2), 95: JSR (TX),
Commands	S_S, DB_ADD, SB_ADD, SUMS, ADD_SB7, ADD_SB06, ADH_PCH, PCH_ADH, ADL_PCL, PCL_ADL,
ALU Carry In	0
DAA	0
DSA	0
Increment PC	1
Regs	IR=0x20, PD=0x20, Y=0x00, X=0x05, S=0xFD, AI=0x6B, BI=0x6B, ADD=0x6B, AC=0x6C
PCL	0x07
PCH	0xC0
ABL	0x07
ABH	0xC0
DL	0x20
DOR	0x6B
Flags	C: 1, Z: 0, I: 1, D: 0, B: 1, V: 0, N: 0
Buses	SB=0x6B, DB=0x6B, ADL=0x07, ADH=0xC0



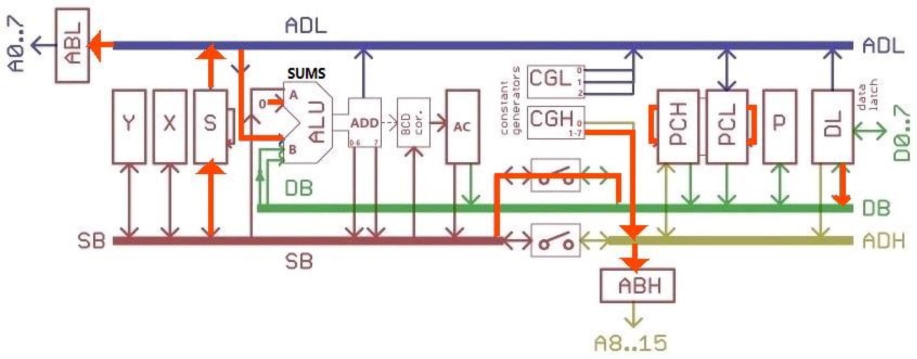
JSR (0x20), T2 (PHI2)

Component/ Signal	State
Dispatcher	T0: 0, /T0: 1, /T1X: 1, 0/IR: 1, FETCH: 0, /ready: 0, WR: 0, ACRL1: 1, ACRL2: 1, T5: 0, T6: 0, ENDS: 0, ENDX: 1, TRES1: 0, TRESX: 1
Interrupts	/NMIP: 1, /IRQP: 1, RESP: 0, BRK6E: 0, BRK7: 1, DORES: 0, /DONMI: 1
Extra Cycle Counter	T1: 0, TRES2: 0, /T2: 0, /T3: 1, /T4: 1, /T5: 1
Decoder	28: T2, 33: LEFT_ALL (T2), 35: STK2, 48: JSR2, 57: BRK JSR RTI RTS Push/pull (T2), 95: JSR (TX), 121: /IR6, 126: /IR7
Commands	S_ADL, SUMS, ADH_ABH, ADL_ABL, Z_ADH17, SB_DB, DL_DB
ALU Carry In	0
DAA	0
DSA	0
Increment PC	1
Regs	IR=0x20, PD=0x00, Y=0x00, X=0x05, S=0xFD, AI=0x6B, BI=0x6B, ADD=0xD6, AC=0x6C
PCL	0x08
PCH	0xC0
ABL	0x07
ABH	0xC0
DL	0x0E
DOR	0x6B
Flags	C: 1, Z: 0, I: 1, D: 0, B: 1, V: 0, N: 0
Buses	SB=0xFF, DB=0xFF, ADL=0xFD, ADH=0x01



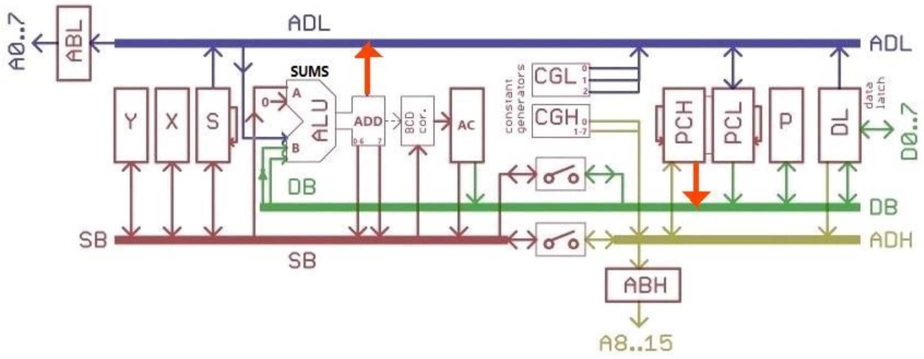
JSR (0x20), T3 (PHI1)

Component/ Signal	State
Dispatcher	T0: 0, /T0: 1, /T1X: 1, 0/IR: 1, FETCH: 0, /ready: 0, WR: 0, ACRL1: 0, ACRL2: 1, T5: 0, T6: 0, ENDS: 0, ENDX: 1, TRES1: 0, TRESX: 1
Interrupts	/NMIP: 1, /IRQP: 1, RESP: 0, BRK6E: 0, BRK7: 1, DORES: 0, /DONMI: 1
Extra Cycle Counter	T1: 0, TRES2: 0, /T2: 1, /T3: 0, /T4: 1, /T5: 1
Decoder	36: BRK JSR RTI RTS Push/pull + BIT JMP (T3), 78: JSR (T3), 86: T3 ANY, 95: JSR (TX), 121: /IR6, 126: /IR7
Commands	S_ADL, SB_S, Z_ADD, ADL_ADD, SUMS, PCH_PCH, PCL_PCL, ADH_ABH, ADL_ABL, Z_ADH17, SB_DB, DL_DB
ALU Carry In	0
DAA	0
DSA	0
Increment PC	0
Regs	IR=0x20, PD=0x00, Y=0x00, X=0x05, S=0xFD, AI=0x00, BI=0xFD, ADD=0xD6, AC=0x6C
PCL	0x08
PCH	0xC0
ABL	0xFD
ABH	0x01
DL	0x0E
DOR	0x0E
Flags	C: 1, Z: 0, I: 1, D: 0, B: 1, V: 1, N: 0
Buses	SB=0x0E, DB=0x0E, ADL=0xFD, ADH=0x01



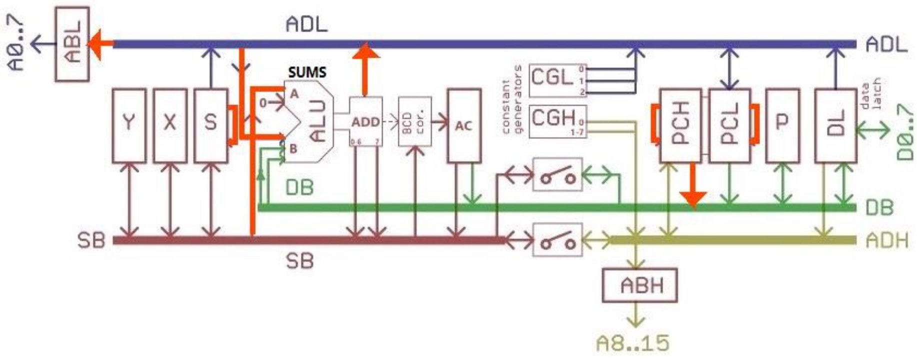
JSR (0x20), T3 (PHI2)

Component/ Signal	State
Dispatcher	T0: 0, /T0: 1, /T1X: 1, 0/IR: 1, FETCH: 0, /ready: 0, WR: 1, ACRL1: 0, ACRL2: 0, T5: 0, T6: 0, ENDS: 0, ENDX: 1, TRES1: 0, TRESX: 1
Interrupts	/NMIP: 1, /IRQP: 1, RESP: 0, BRK6E: 0, BRK7: 1, DORES: 0, /DONMI: 1
Extra Cycle Coun-ter	T1: 0, TRES2: 0, /T2: 1, /T3: 0, /T4: 1, /T5: 1
Decoder	36: BRK JSR RTI RTS Push/pull + BIT JMP (T3), 78: JSR (T3), 86: T3 ANY, 95: JSR (TX), 121: /IR6, 126: /IR7
Commands	SUMS, ADD_ADL, PCH_DB, ADL_ABL
ALU Carry In	0
DAA	0
DSA	0
Increment PC	0
Regs	IR=0x20, PD=0x00, Y=0x00, X=0x05, S=0x0E, AI=0x00, BI=0xFD, ADD=0xFD, AC=0x6C
PCL	0x08
PCH	0xC0
ABL	0xFD
ABH	0x01
DL	0x00
DOR	0x0E
Flags	C: 1, Z: 0, I: 1, D: 0, B: 1, V: 1, N: 0
Buses	SB=0xFF, DB=0xC0, ADL=0xFD, ADH=0xFF



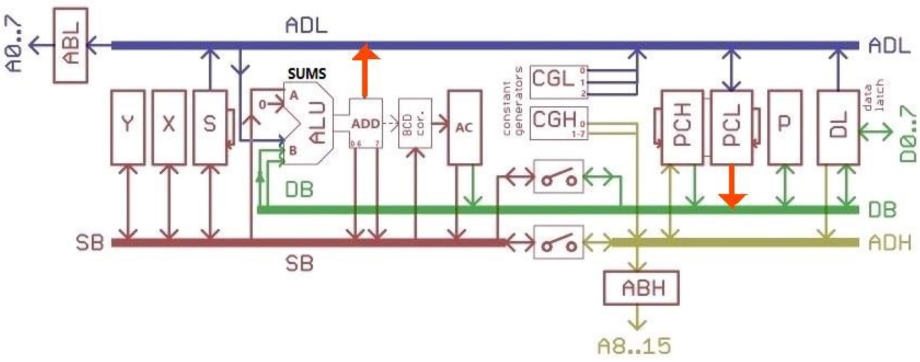
JSR (0x20), T4 (PHI1)

Component/Signal	State
Dispatcher	T0: 0, /T0: 1, /T1X: 1, 0/IR: 1, FETCH: 0, /ready: 0, WR: 1, ACRL1: 0, ACRL2: 0, T5: 0, T6: 0, ENDS: 0, ENDX: 1, TRES1: 0, TRESX: 1
Interrupts	/NMIP: 1, /IRQP: 1, RESP: 0, BRK6E: 0, BRK7: 1, DORES: 0, /DONMI: 1
Extra Cycle Counter	T1: 0, TRES2: 0, /T2: 1, /T3: 1, /T4: 0, /T5: 1
Decoder	37: BRK JSR (T4), 85: T4 ANY, 95: JSR (TX), 121: /IR6, 126: /IR7
Commands	S_S, SB_ADD, ADL_ADD, SUMS, ADD_ADL, PCH_PCH, PCH_DB, PCL_PCL, ADL_ABL
ALU Carry In	0
DAA	0
DSA	0
Increment PC	0
Regs	IR=0x20, PD=0x00, Y=0x00, X=0x05, S=0x0E, AI=0xFF, BI=0xFD, ADD=0xFD, AC=0x6C
PCL	0x08
PCH	0xC0
ABL	0xFD
ABH	0x01
DL	0x00
DOR	0xC0
Flags	C: 1, Z: 0, I: 1, D: 0, B: 1, V: 0, N: 0
Buses	SB=0xFF, DB=0xC0, ADL=0xFD, ADH=0xFF



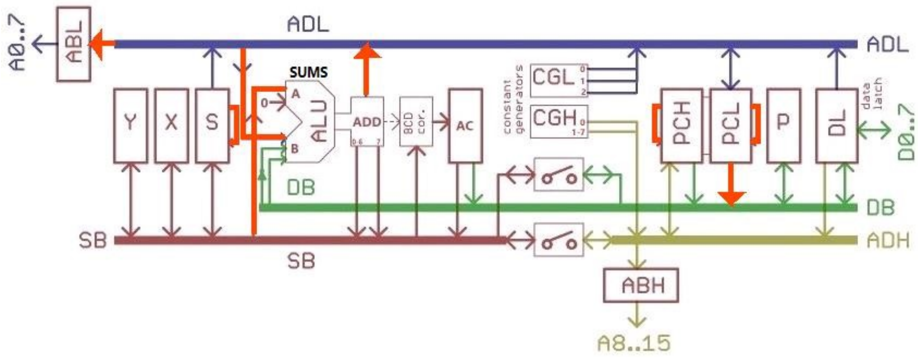
JSR (0x20), T4 (PHI2)

Component/Signal	State
Dispatcher	T0: 0, /T0: 1, /T1X: 1, 0/IR: 1, FETCH: 0, /ready: 0, WR: 1, ACRL1: 0, ACRL2: 0, T5: 0, T6: 0, ENDS: 0, ENDX: 1, TRES1: 0, TRESX: 1
Interrupts	/NMIP: 1, /IRQP: 1, RESP: 0, BRK6E: 0, BRK7: 1, DORES: 0, /DONMI: 1
Extra Cycle Counter	T1: 0, TRES2: 0, /T2: 1, /T3: 1, /T4: 0, /T5: 1
Decoder	37: BRK JSR (T4), 85: T4 ANY, 95: JSR (TX), 121: /IR6, 126: /IR7
Commands	SUMS, ADD_ADL, PCL_DB, ADL_ABL
ALU Carry In	0
DAA	0
DSA	0
Increment PC	0
Regs	IR=0x20, PD=0x00, Y=0x00, X=0x05, S=0x0E, AI=0xFF, BI=0xFD, ADD=0xFC, AC=0x6C
PCL	0x08
PCH	0xC0
ABL	0xFD
ABH	0x01
DL	0x00
DOR	0xC0
Flags	C: 1, Z: 0, I: 1, D: 0, B: 1, V: 0, N: 0
Buses	SB=0xFF, DB=0x08, ADL=0xFC, ADH=0xFF



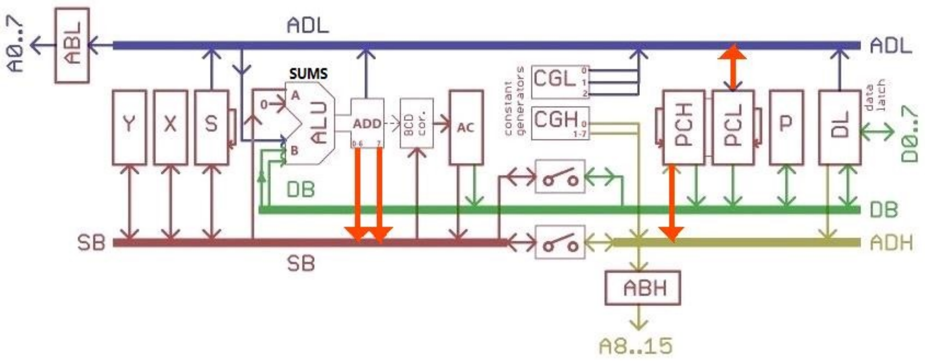
JSR (0x20), T5 (PHI1)

Component/Signal	State
Dispatcher	T0: 0, /T0: 1, /T1X: 1, 0/IR: 1, FETCH: 0, /ready: 0, WR: 1, ACRL1: 1, ACRL2: 0, T5: 0, T6: 0, ENDS: 0, ENDX: 0, TRES1: 0, TRESX: 1
Interrupts	/NMIP: 1, /IRQP: 1, RESP: 0, BRK6E: 0, BRK7: 1, DORES: 0, /DONMI: 1
Extra Cycle Counter	T1: 0, TRES2: 0, /T2: 1, /T3: 1, /T4: 1, /T5: 0
Decoder	56: JSR/5, 95: JSR (TX), 103: JSR (T5), 121: /IR6, 126: /IR7
Commands	S_S, SB_ADD, ADL_ADD, SUMS, ADD_ADL, PCH_PCH, PCL_PCL, PCL_DB, ADL_ABL
ALU Carry In	0
DAA	0
DSA	0
Increment PC	0
Regs	IR=0x20, PD=0x00, Y=0x00, X=0x05, S=0x0E, AI=0xFF, BI=0xFC, ADD=0xFC, AC=0x6C
PCL	0x08
PCH	0xC0
ABL	0xFC
ABH	0x01
DL	0x00
DOR	0x08
Flags	C: 1, Z: 0, I: 1, D: 0, B: 1, V: 0, N: 0
Buses	SB=0xFF, DB=0x08, ADL=0xFC, ADH=0xFF



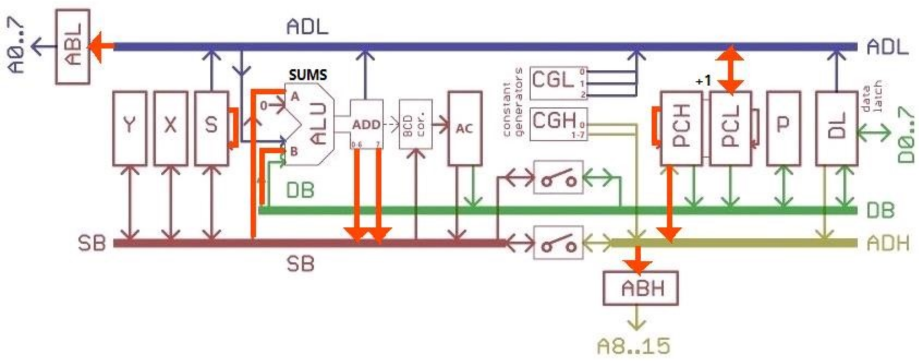
JSR (0x20), T5 (PHI2)

Component/Signal	State
Dispatcher	T0: 0, /T0: 1, /T1X: 1, 0/IR: 1, FETCH: 0, /ready: 0, WR: 0, ACRL1: 1, ACRL2: 1, T5: 0, T6: 0, ENDS: 0, ENDX: 0, TRES1: 0, TRESX: 0
Interrupts	/NMIP: 1, /IRQP: 1, RESP: 0, BRK6E: 0, BRK7: 1, DORES: 0, /DONMI: 1
Extra Cycle Counter	T1: 0, TRES2: 0, /T2: 1, /T3: 1, /T4: 1, /T5: 0
Decoder	56: JSR/5, 95: JSR (TX), 103: JSR (T5), 121: /IR6, 126: /IR7
Commands	SUMS, ADD_SB7, ADD_SB06, PCH_ADH, PCL_ADL, ADH_ABH, ADL_ABL
ALU Carry In	0
DAA	0
DSA	0
Increment PC	0
Regs	IR=0x20, PD=0x00, Y=0x00, X=0x05, S=0x0E, AI=0xFF, BI=0xFC, ADD=0xFB, AC=0x6C
PCL	0x08
PCH	0xC0
ABL	0xFC
ABH	0x01
DL	0xC0
DOR	0x08
Flags	C: 1, Z: 0, I: 1, D: 0, B: 1, V: 0, N: 0
Buses	SB=0xFB, DB=0xFF, ADL=0x08, ADH=0xC0



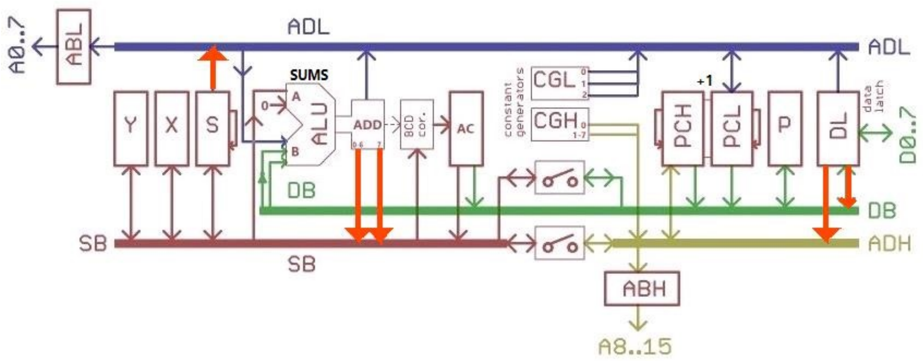
JSR (0x20), T0 (PHI1)

Component/Signal	State
Dispatcher	T0: 1, /T0: 0, /T1X: 1, 0/IR: 1, FETCH: 0, /ready: 0, WR: 0, ACRL1: 1, ACRL2: 1, T5: 0, T6: 0, ENDS: 0, ENDX: 1, TRES1: 0, TRESX: 0
Interrupts	/NMIP: 1, /IRQP: 1, RESP: 0, BRK6E: 0, BRK7: 1, DORES: 0, /DONMI: 1
Extra Cycle Counter	T1: 0, TRES2: 1, /T2: 1, /T3: 1, /T4: 1, /T5: 1
Decoder	21: JSR (T0), 34: T0 ANY, 95: JSR (TX), 121: /IR6, 126: /IR7
Commands	S_S, NDB_ADD, SB_ADD, SUMS, ADD_SB7, ADD_SB06, PCH_PCH, PCH_ADH, ADL_PCL, PCL_ADL, ADH_ABH, ADL_ABL
ALU Carry In	0
DAA	0
DSA	0
Increment PC	1
Regs	IR=0x20, PD=0x00, Y=0x00, X=0x05, S=0x0E, AI=0xFB, BI=0x00, ADD=0xFB, AC=0x6C
PCL	0x08
PCH	0xC0
ABL	0x08
ABH	0xC0
DL	0xC0
DOR	0xFF
Flags	C: 1, Z: 0, I: 1, D: 0, B: 1, V: 0, N: 0
Buses	SB=0xFB, DB=0xFF, ADL=0x08, ADH=0xC0



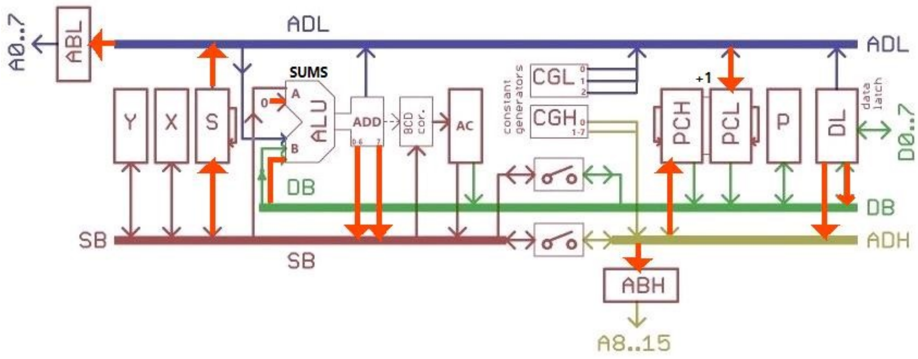
JSR (0x20), T0 (PHI2)

Component/Signal	State
Dispatcher	T0: 1, /T0: 0, /T1X: 1, 0/IR: 1, FETCH: 0, /ready: 0, WR: 0, ACRL1: 1, ACRL2: 1, T5: 0, T6: 0, ENDS: 1, ENDX: 1, TRES1: 1, TRESX: 0
Interrupts	/NMIP: 1, /IRQP: 1, RESP: 0, BRK6E: 0, BRK7: 1, DORES: 0, /DONMI: 1
Extra Cycle Counter	T1: 0, TRES2: 1, /T2: 1, /T3: 1, /T4: 1, /T5: 1
Decoder	21: JSR (T0), 34: T0 ANY, 95: JSR (TX), 121: /IR6, 126: /IR7
Commands	S_ADL, SUMS, ADD_SB7, ADD_SB06, ADH_ABH, ADL_ABL, DL_ADH, DL_DB
ALU Carry In	0
DAA	0
DSA	0
Increment PC	1
Regs	IR=0x20, PD=0x00, Y=0x00, X=0x05, S=0x0E, AI=0xFB, BI=0x00, ADD=0xFB, AC=0x6C
PCL	0x09
PCH	0xC0
ABL	0x08
ABH	0xC0
DL	0xC0
DOR	0xFF
Flags	C: 1, Z: 0, I: 1, D: 0, B: 1, V: 0, N: 0
Buses	SB=0xFB, DB=0xFF, ADL=0x0E, ADH=0xFF



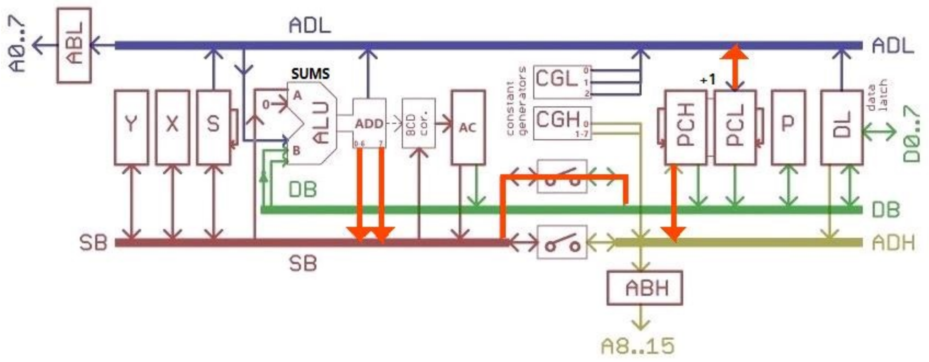
JSR (0x20), T1 (PHI1)

Component/Signal	State
Dispatcher	T0: 0, /T0: 1, /T1X: 0, 0/IR: 1, FETCH: 0, /ready: 0, WR: 0, ACRL1: 0, ACRL2: 1, T5: 0, T6: 0, ENDS: 1, ENDX: 1, TRES1: 1, TRESX: 0
Interrupts	/NMIP: 1, /IRQP: 1, RESP: 0, BRK6E: 0, BRK7: 1, DORES: 0, /DONMI: 1
Extra Cycle Counter	T1: 1, TRES2: 1, /T2: 1, /T3: 1, /T4: 1, /T5: 1
Decoder	95: JSR (TX), 121: /IR6, 126: /IR7
Commands	S_ADL, SB_S, DB_ADD, Z_ADD, SUMS, ADD_SB7, ADD_SB06, ADH_PCH, ADL_PCL, ADH_ABH, ADL_ABL, DL_ADH, DL_DB
ALU Carry In	0
DAA	0
DSA	0
Increment PC	1
Regs	IR=0x20, PD=0x00, Y=0x00, X=0x05, S=0x0E, AI=0x00, BI=0xC0, ADD=0xFB, AC=0x6C
PCL	0x09
PCH	0xC0
ABL	0x0E
ABH	0xC0
DL	0xC0
DOR	0xC0
Flags	C: 1, Z: 0, I: 1, D: 0, B: 1, V: 0, N: 0
Buses	SB=0xFB, DB=0xC0, ADL=0x0E, ADH=0xC0



JSR (0x20), T1 (PHI2)

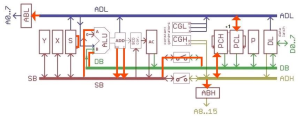
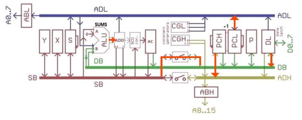
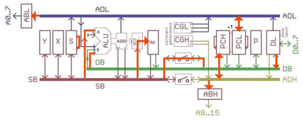
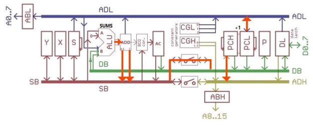
Component/Signal	State
Dispatcher	T0: 0, /T0: 1, /T1X: 0, 0/IR: 0, FETCH: 1, /ready: 0, WR: 0, ACRL1: 0, ACRL2: 0, T5: 0,
Interrupts	/NMIP: 1, /IRQP: 1, RESP: 0, BRK6E: 0, BRK7: 1, DORES: 0, /DONMI: 1
Extra Cycle Counter	T1: 1, TRES2: 1, /T2: 1, /T3: 1, /T4: 1, /T5: 1
Decoder	95: JSR (TX), 121: /IR6, 126: /IR7
Commands	SUMS, ADD_SB7, ADD_SB06, PCH_ADH, PCL_ADL, ADH_ABH, ADL_ABL, SB_DB
ALU Carry In	0
DAA	0
DSA	0
Increment PC	1
Regs	IR=0x20, PD=0x69, Y=0x00, X=0x05, S=0xFB, AI=0x00, BI=0xC0, ADD=0xC0,
PCL	0x0F
PCH	0xC0
ABL	0x0E
ABH	0xC0
DL	0x69
DOR	0xC0
Flags	C: 1, Z: 0, I: 1, D: 0, B: 1, V: 0, N: 0
Buses	SB=0xC0, DB=0xC0, ADL=0x0F, ADH=0xC0



Notes in the margins for future revisions of the book.

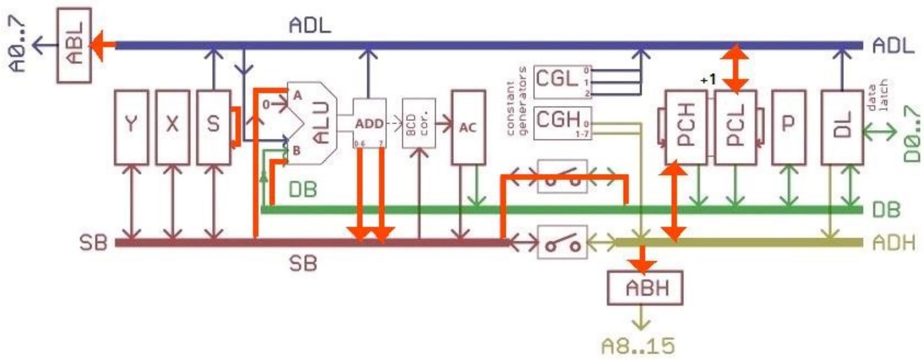
LDA imm (0xA9)

```
NOP
LDA    #A5
NOP
```

T	PHI1	PHI2	R/W
T0+2			Read
T1			Read

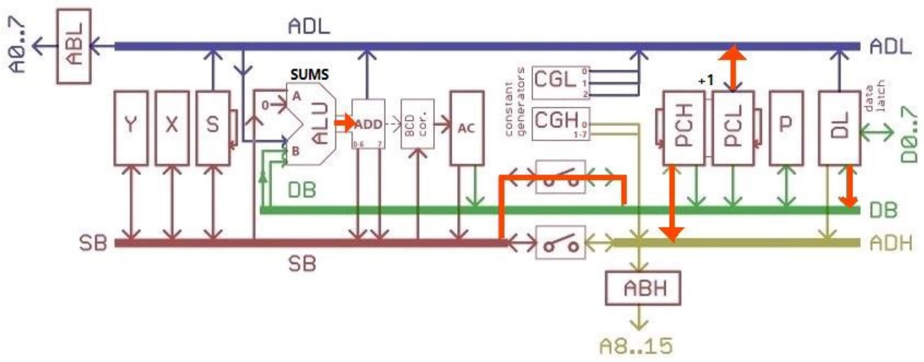
LDA (0xA9), T02 (PHI1)

Component/Signal	State
Dispatcher	T0: 1, /T0: 0, /T1X: 1, 0/IR: 0, FETCH: 1, /ready: 0, WR: 0, ACRL1: 1, ACRL2: 1, T5: 0, T6: 0, ENDS: 0, ENDX: 1, TRES1: 0, TRESX: 1
Interrupts	/NMIP: 1, /IRQP: 1, RESP: 0, BRK6E: 0, BRK7: 1, DORES: 0, /DONMI: 1
Extra Cycle Counter	T1: 0, TRES2: 0, /T2: 0, /T3: 1, /T4: 1, /T5: 1
Decoder	28: T2, 34: T0 ANY, 64: LDA (T0), 65: ALL ODD (T0), 83: ABS/2, 121: /IR6, 128: IMPL
Commands	S_S, DB_ADD, SB_ADD, SUMS, ADD_SB7, ADD_SB06, ADH_PCH, PCH_ADH, ADL_PCL, PCL_ADL, ADH_ABH, ADL_ABL, SB_DB
ALU Carry In	0
DAA	0
DSA	0
Increment PC	1
Regs	IR=0xA9, PD=0xA9, Y=0x00, X=0x00, S=0xFD, AI=0xFE, BI=0xFE, ADD=0xFE, AC=0x0A
PCL	0x02
PCH	0xC0
ABL	0x02
ABH	0xC0
DL	0xA9
DOR	0xFE
Flags	C: 0, Z: 0, I: 1, D: 0, B: 1, V: 0, N: 1
Buses	SB=0xFE, DB=0xFE, ADL=0x02, ADH=0xC0



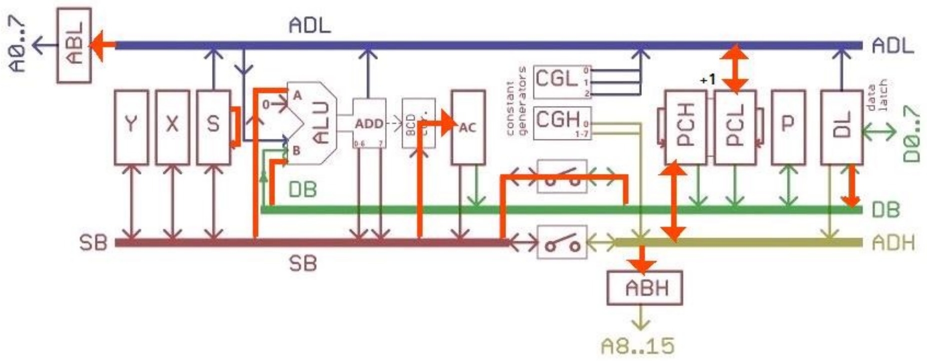
LDA (0xA9), T02 (PHI2)

Component/Signal	State
Dispatcher	T0: 1, /T0: 0, /T1X: 1, 0/IR: 1, FETCH: 0, /ready: 0, WR: 0, ACRL1: 1, ACRL2: 1, T5: 0, T6: 0, ENDS: 1, ENDX: 1, TRES1: 1, TRESX: 0
Interrupts	/NMIP: 1, /IRQP: 1, RESP: 0, BRK6E: 0, BRK7: 1, DORES: 0, /DONMI: 1
Extra Cycle Counter	T1: 0, TRES2: 0, /T2: 0, /T3: 1, /T4: 1, /T5: 1
Decoder	28: T2, 34: T0 ANY, 64: LDA (T0), 65: ALL ODD (T0), 83: ABS/2, 121: /IR6, 128: IMPL
Commands	SUMS, PCH_ADH, PCL_ADL, ADH_ABH, ADL_ABL, SB_DB, DL_DB, DBZ_Z, DB_N
ALU Carry In	0
DAA	0
DSA	0
Increment PC	1
Regs	IR=0xA9, PD=0x00, Y=0x00, X=0x00, S=0xFD, AI=0xFE, BI=0xFE, ADD=0xFC, AC=0x0A
PCL	0x03
PCH	0xC0
ABL	0x02
ABH	0xC0
DL	0x00
DOR	0xFE
Flags	C: 0, Z: 0, I: 1, D: 0, B: 1, V: 0, N: 1
Buses	SB=0xFF, DB=0xFF, ADL=0x03, ADH=0xC0



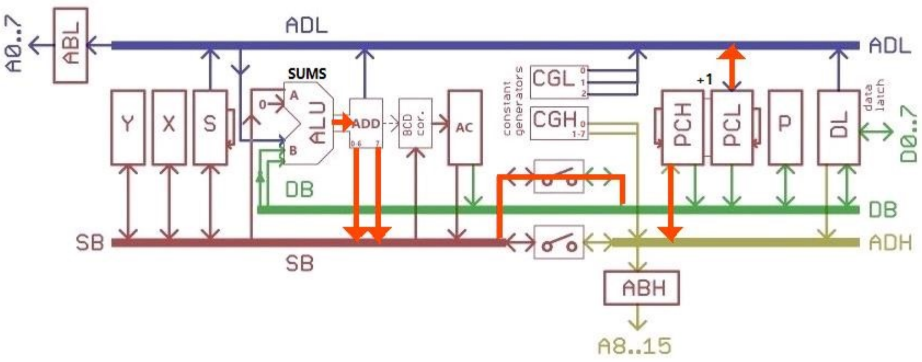
LDA (0xA9), T1 (PHI1)

Component/Signal	State
Dispatcher	T0: 0, /T0: 1, /T1X: 0, 0/IR: 1, FETCH: 0, /ready: 0, WR: 0, ACRL1: 1, ACRL2: 1, T5: 0, T6: 0, ENDS: 1, ENDX: 1, TRES1: 1, TRESX: 0
Interrupts	/NMIP: 1, /IRQP: 1, RESP: 0, BRK6E: 0, BRK7: 1, DORES: 0, /DONMI: 1
Extra Cycle Counter	T1: 1, TRES2: 1, /T2: 1, /T3: 1, /T4: 1, /T5: 1
Decoder	121: /IR6, 128: IMPL
Commands	S_S, DB_ADD, SB_ADD, SUMS, SB_AC, ADH_PCH, PCH_ADH, ADL_PCL, PCL_ADL, ADH_ABH, ADL_ABL, SB_DB, DL_DB, DBZ_Z, DB_N
ALU Carry In	0
DAA	0
DSA	0
Increment PC	1
Regs	IR=0xA9, PD=0x00, Y=0x00, X=0x00, S=0xFD, AI=0x00, BI=0x00, ADD=0xFC, AC=0x00
PCL	0x03
PCH	0xC0
ABL	0x03
ABH	0xC0
DL	0x00
DOR	0x00
Flags	C: 0, Z: 1, I: 1, D: 0, B: 1, V: 0, N: 0
Buses	SB=0x00, DB=0x00, ADL=0x03, ADH=0xC0



LDA (0xA9), T1 (PHI2)

Component/ Signal	State
Dispatcher	T0: 0, /T0: 1, /T1X: 0, 0/IR: 0, FETCH: 1, /ready: 0, WR: 0, ACRL1: 0, ACRL2: 1, T5: 0, T6: 0, ENDS: 0, ENDX: 1, TRES1: 0, TRESX: 1
Interrupts	/NMIP: 1, /IRQP: 1, RESP: 0, BRK6E: 0, BRK7: 1, DORES: 0, /DONMI: 1
Extra Cycle Counter	T1: 1, TRES2: 1, /T2: 1, /T3: 1, /T4: 1, /T5: 1
Decoder	121: /IR6, 128: IMPL
Commands	SUMS, ADD_SB7, ADD_SB06, PCH_ADH, PCL_ADL, ADH_ABH, ADL_ABL, SB_DB
ALU Carry In	0
DAA	0
DSA	0
Increment PC	1
Regs	IR=0xA9, PD=0xEA, Y=0x00, X=0x00, S=0xFD, AI=0x00, BI=0x00, ADD=0x00, AC=0x00
PCL	0x04
PCH	0xC0
ABL	0x03
ABH	0xC0
DL	0xEA
DOR	0x00
Flags	C: 0, Z: 1, I: 1, D: 0, B: 1, V: 0, N: 0
Buses	SB=0x00, DB=0x00, ADL=0x04, ADH=0xC0



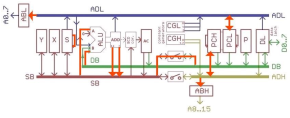
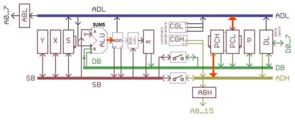
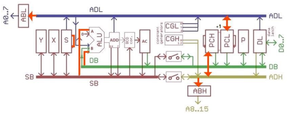
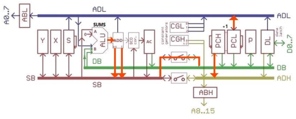
Next NOP T0+2 PHI1

Component/Signal	State
Dispatcher	T0: 1, /T0: 0, /T1X: 1, 0/IR: 0, FETCH: 1, /ready: 0, WR: 0, ACRL1: 0, ACRL2: 1, T5: 0, T6: 0, ENDS: 0, ENDX: 1, TRES1: 0, TRESX: 1
Interrupts	/NMIP: 1, /IRQP: 1, RESP: 0, BRK6E: 0, BRK7: 1, DORES: 0, /DONMI: 1
Extra Cycle Counter	T1: 0, TRES2: 0, /T2: 0, /T3: 1, /T4: 1, /T5: 1
Decoder	28: T2, 34: T0 ANY, 44: INC NOP (TX), 83: ABS/2, 106: LSR ROR DEC INC DEX NOP (4x4 bottom right) (TX), 128: IMPL
Commands	S_S, DB_ADD, SB_ADD, SUMS, ADD_SB7, ADD_SB06, ADH_PCH, PCH_ADH, ADL_PCL, PCL_ADL, ADH_ABH, ADL_ABL, SB_DB
ALU Carry In	0
DAA	0
DSA	0
Increment PC	0
Regs	IR=0xEA, PD=0xEA, Y=0x00, X=0x00, S=0xFD, AI=0x00, BI=0x00, ADD=0x00, AC=0x00
PCL	0x04
PCH	0xC0
ABL	0x04
ABH	0xC0
DL	0xEA
DOR	0x00
Flags	C: 0, Z: 1, I: 1, D: 0, B: 1, V: 0, N: 0
Buses	SB=0x00, DB=0x00, ADL=0x04, ADH=0xC0

Notes in the margins for future revisions of the book.

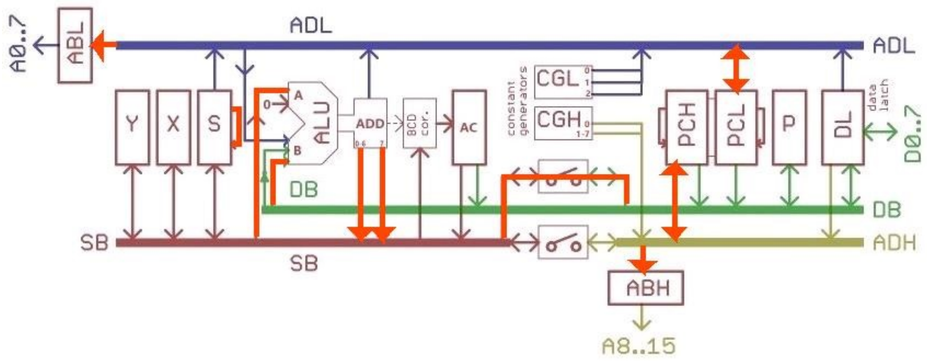
NOP (0xEA)

```
nop
nop
nop
```

T	PHI1	PHI2	R/W
T0+2			Read
T1			Read

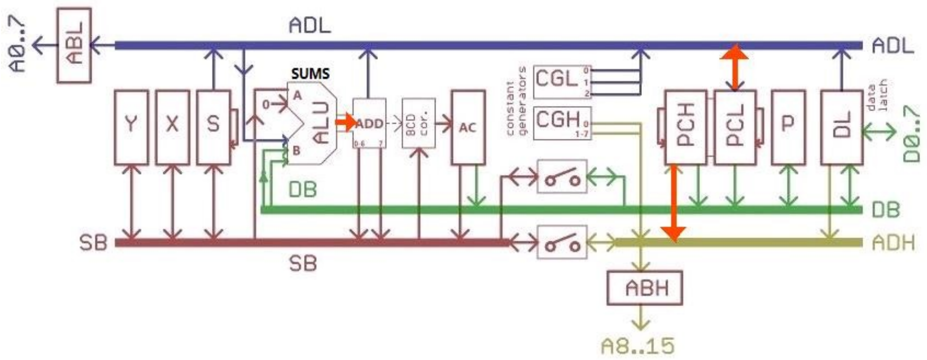
NOP (0xEA), T02 (PHI1)

Component/Signal	State
Dispatcher	T0: 1, /T0: 0, /T1X: 1, 0/IR: 0, FETCH: 1, /ready: 0, WR: 0, ACRL1: 1, ACRL2: 0, T5: 0, T6: 0, ENDS: 0, ENDX: 1, TRES1: 0, TRESX: 1
Interrupts	/NMIP: 1, /IRQP: 1, RESP: 0, BRK6E: 0, BRK7: 1, DORES: 0, /DONMI: 1
Extra Cycle Counter	T1: 0, TRES2: 0, /T2: 0, /T3: 1, /T4: 1, /T5: 1
Decoder	28: T2, 34: T0 ANY, 44: INC NOP (TX), 83: ABS/2, 106: LSR ROR DEC INC DEX NOP (4x4 bottom right) (TX), 128: IMPL
Commands	S_S, DB_ADD, SB_ADD, SUMS, ADD_SB7, ADD_SB06, ADH_PCH, PCH_ADH, ADL_PCL, PCL_ADL, ADH_ABH, ADL_ABL, SB_DB
ALU Carry In	0
DAA	0
DSA	0
Increment PC	0
Regs	IR=0xEA, PD=0xEA, Y=0x00, X=0x00, S=0xFD, AI=0xFE, BI=0xFE, ADD=0xFE, AC=0x00
PCL	0x05
PCH	0xC0
ABL	0x05
ABH	0xC0
DL	0xEA
DOR	0xFE
Flags	C: 0, Z: 1, I: 1, D: 0, B: 1, V: 0, N: 0
Buses	SB=0xFE, DB=0xFE, ADL=0x05, ADH=0xC0



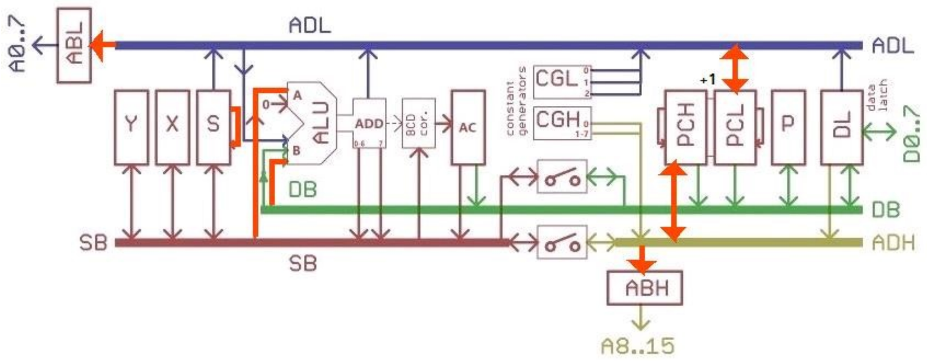
NOP (0xEA), T02 (PHI2)

Component/Signal	State
Dispatcher	T0: 1, /T0: 0, /T1X: 1, 0/IR: 1, FETCH: 0, /ready: 0, WR: 0, ACRL1: 1, ACRL2: 1, T5: 0, T6: 0, ENDS: 1, ENDX: 1, TRES1: 1, TRESX: 0
Interrupts	/NMIP: 1, /IRQP: 1, RESP: 0, BRK6E: 0, BRK7: 1, DORES: 0, /DONMI: 1
Extra Cycle Counter	T1: 0, TRES2: 0, /T2: 0, /T3: 1, /T4: 1, /T5: 1
Decoder	28: T2, 34: T0 ANY, 44: INC NOP (TX), 83: ABS/2, 106: LSR ROR DEC INC DEX NOP (4x4 bottom right) (TX), 128: IMPL
Commands	SUMS, PCH_ADH, PCL_ADL, ADH_ABH, ADL_ABL
ALU Carry In	0
DAA	0
DSA	0
Increment PC	0
Regs	IR=0xEA, PD=0x00, Y=0x00, X=0x00, S=0xFD, AI=0xFE, BI=0xFE, ADD=0xFC, AC=0x00
PCL	0x05
PCH	0xC0
ABL	0x05
ABH	0xC0
DL	0xEA
DOR	0xFE
Flags	C: 0, Z: 1, I: 1, D: 0, B: 1, V: 0, N: 0
Buses	SB=0xFF, DB=0xFF, ADL=0x05, ADH=0xC0



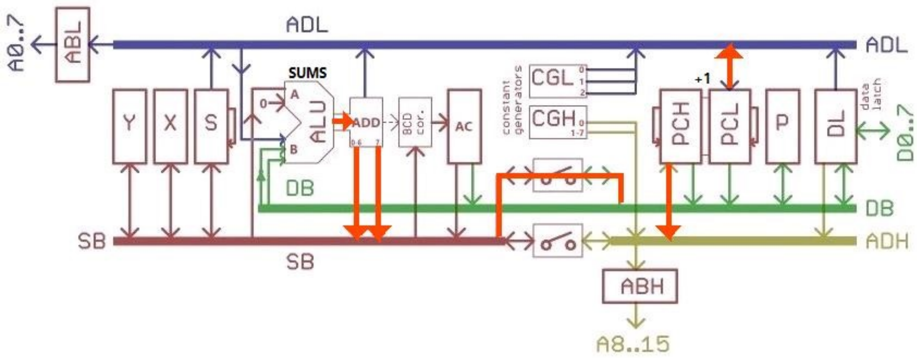
NOP (0xEA), T1 (PHI1)

Component/Signal	State
Dispatcher	T0: 0, /T0: 1, /T1X: 0, 0/IR: 1, FETCH: 0, /ready: 0, WR: 0, ACRL1: 1, ACRL2: 1, T5: 0, T6: 0, ENDS: 1, ENDX: 1, TRES1: 1, TRESX: 0
Interrupts	/NMIP: 1, /IRQP: 1, RESP: 0, BRK6E: 0, BRK7: 1, DORES: 0, /DONMI: 1
Extra Cycle Counter	T1: 1, TRES2: 1, /T2: 1, /T3: 1, /T4: 1, /T5: 1
Decoder	44: INC NOP (TX), 106: LSR ROR DEC INC DEX NOP (4x4 bottom right) (TX), 128: IMPL
Commands	S_S, DB_ADD, SB_ADD, SUMS, ADH_PCH, PCH_ADH, ADL_PCL, PCL_ADL, ADH_ABH, ADL_ABL
ALU Carry In	0
DAA	0
DSA	0
Increment PC	1
Regs	IR=0xEA, PD=0x00, Y=0x00, X=0x00, S=0xFD, AI=0xFF, BI=0xFF, ADD=0xFC, AC=0x00
PCL	0x05
PCH	0xC0
ABL	0x05
ABH	0xC0
DL	0xEA
DOR	0xFF
Flags	C: 0, Z: 1, I: 1, D: 0, B: 1, V: 0, N: 0
Buses	SB=0xFF, DB=0xFF, ADL=0x05, ADH=0xC0



NOP (0xEA), T1 (PHI2)

Component/Signal	State
Dispatcher	T0: 0, /T0: 1, /T1X: 0, 0/IR: 0, FETCH: 1, /ready: 0, WR: 0, ACRL1: 1, ACRL2: 1, T5: 0, T6: 0, ENDS: 0, ENDX: 1, TRES1: 0, TRESX: 1
Interrupts	/NMIP: 1, /IRQP: 1, RESP: 0, BRK6E: 0, BRK7: 1, DORES: 0, /DONMI: 1
Extra Cycle Counter	T1: 1, TRES2: 1, /T2: 1, /T3: 1, /T4: 1, /T5: 1
Decoder	44: INC NOP (TX), 106: LSR ROR DEC INC DEX NOP (4x4 bottom right) (TX), 128: IMPL
Commands	SUMS, ADD_SB7, ADD_SB06, PCH_ADH, PCL_ADL, ADH_ABH, ADL_ABL, SB_DB
ALU Carry In	0
DAA	0
DSA	0
Increment PC	1
Regs	IR=0xEA, PD=0xEA, Y=0x00, X=0x00, S=0xFD, AI=0xFF, BI=0xFF, ADD=0xFE, AC=0x00
PCL	0x06
PCH	0xC0
ABL	0x05
ABH	0xC0
DL	0xEA
DOR	0xFF
Flags	C: 0, Z: 1, I: 1, D: 0, B: 1, V: 0, N: 0
Buses	SB=0xFC, DB=0xFC, ADL=0x06, ADH=0xC0



Next NOP T0+2 PHI1

Component/Signal	State
Dispatcher	T0: 1, /T0: 0, /T1X: 1, 0/IR: 0, FETCH: 1, /ready: 0, WR: 0, ACRL1: 1, ACRL2: 1, T5: 0, T6: 0, ENDS: 0, ENDX: 1, TRES1: 0, TRESX: 1
Interrupts	/NMIP: 1, /IRQP: 1, RESP: 0, BRK6E: 0, BRK7: 1, DORES: 0, /DONMI: 1
Extra Cycle Counter	T1: 0, TRES2: 0, /T2: 0, /T3: 1, /T4: 1, /T5: 1
Decoder	28: T2, 34: T0 ANY, 44: INC NOP (TX), 83: ABS/2, 106: LSR ROR DEC INC DEX NOP (4x4 bottom right) (TX), 128: IMPL
Commands	S_S, DB_ADD, SB_ADD, SUMS, ADD_SB7, ADD_SB06, ADH_PCH, PCH_ADH, ADL_PCL, PCL_ADL, ADH_ABH, ADL_ABL, SB_DB
ALU Carry In	0
DAA	0
DSA	0
Increment PC	0
Regs	IR=0xEA, PD=0xEA, Y=0x00, X=0x00, S=0xFD, AI=0xFE, BI=0xFE, ADD=0xFE, AC=0x00
PCL	0x06
PCH	0xC0
ABL	0x06
ABH	0xC0
DL	0xEA
DOR	0xFE
Flags	C: 0, Z: 1, I: 1, D: 0, B: 1, V: 0, N: 0
Buses	SB=0xFE, DB=0xFE, ADL=0x06, ADH=0xC0

Notes in the margins for future revisions of the book.

Afterword

If you have read the whole book and even figured out the 6502, it does not mean that the processor has revealed all its secrets to you. There is an opinion that even the developers of the 6502 did not understand how their processor works. To be honest we don't understand how it works either :)

To be more exact, if we take every separate block it's quite clear how it works. But when we have to analyze work of the whole processor it becomes difficult to understand it. This is especially true for boundary conditions, such as RDY mode, instruction boundaries or interrupt overlapping.

The given examples of instructions work allows to learn a little more but in general it is possible to investigate details of 6502's work till the old age. And that's a good thing :)

In the process of preparing this revision of the book, auxiliary tools such as the 6502 simulator and a fully working 6502 circuit in Logisim were developed. The authors are confident that these tools will be useful to all 6502 fans, its researchers, or teachers of circuit engineering in educational institutions. All materials can be downloaded from the links below.

Links

- Visual6502.org
- 6502.org
- US3991307A - Integrated circuit microprocessor with parallel binary adder having on-the-fly correction to provide decimal results - Google Patents (https://patents.google.com/patent/US3991307A)
- emu-russia/breaks: Nintendo Entertainment System (NES) / Famicom / Dendy chip reversing (github.com) (https://github.com/emu-russia/breaks)
- Klaus2m5/6502_65C02_functional_tests: Tests for all valid opcodes of the 6502 and 65C02 processor (github.com) (https://github.com/Klaus2m5/6502_65C02_functional_tests)
- 6502.org • View topic - 6509 dissection: IDKFA (http://forum.6502.org/viewtopic.php?p=90782#p90782)
- 6502 Topology source (https://drive.google.com/drive/folders/1eDirYKJ6KSHD8MIrFj16GXP8oLAIRs9c?usp=sharing)
- Image of Bender © Fox Interactive

Author's Team:

andkorzh: Schematics for Logisim
HardWareMan: Technical advice, Verilog
org: Transistor circuits, simulation

Editor-in-Chief: org

Feedback: emu-russia Discord (https://discord.gg/WJcvqyCHkh), GitHub (https://github.com/emu-russia/breaks)

This book contains descriptions of all MOS 6502 circuits.

If you are good at digital circuitry, the 6502 processor will reveal all its secrets to you.

The online version of the book is free, the printed version can be ordered from various offices that print the pdf on paper.