6502

Integration

Integrations, the description of their mechanism and layouts for demonstration can be found here from every part of the pattern.

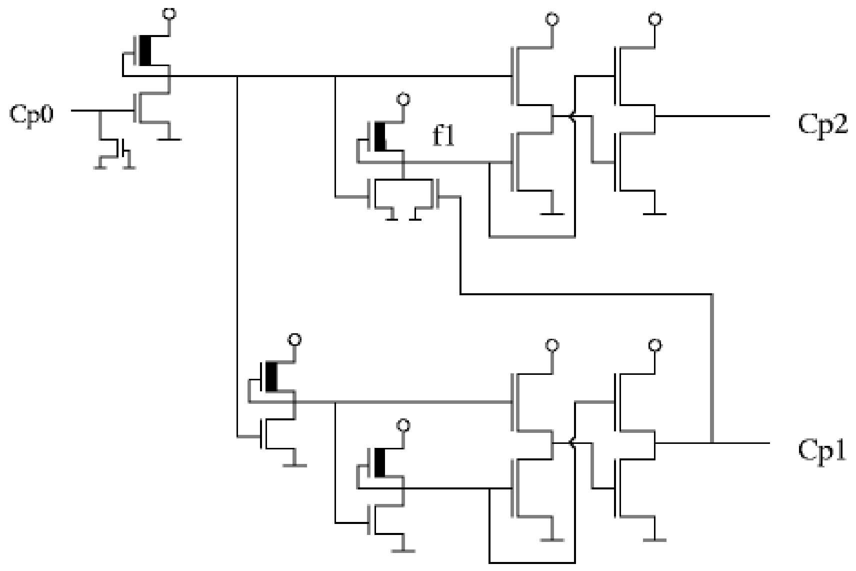
Clock signal generator

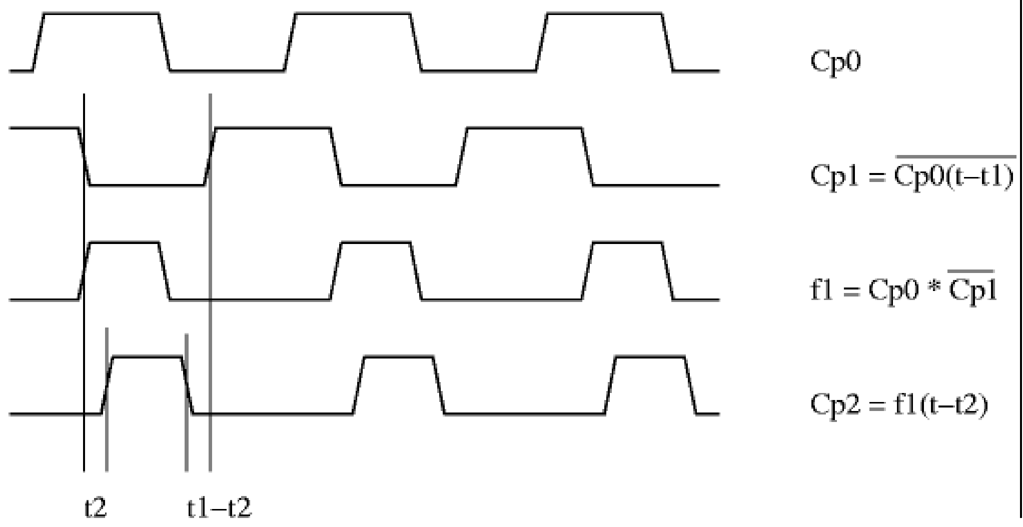
From input clock signal Cp0 (980Hz) the processor generates phase signals Cp1 and Cp2, which do not overlap, therefo they sharply divide the clock cycles into two time slices.

This is useful not only when writing the registers, but external units can also use it; consequently it is lead forth in two pi

The point in the functioning of the clock signal generator is two push-pull inverter pairs, which provide great power, si phase signals must be passed everywhere, and which are slow, therefore, can be used as retarders to slide the phases in ti See illustration below.

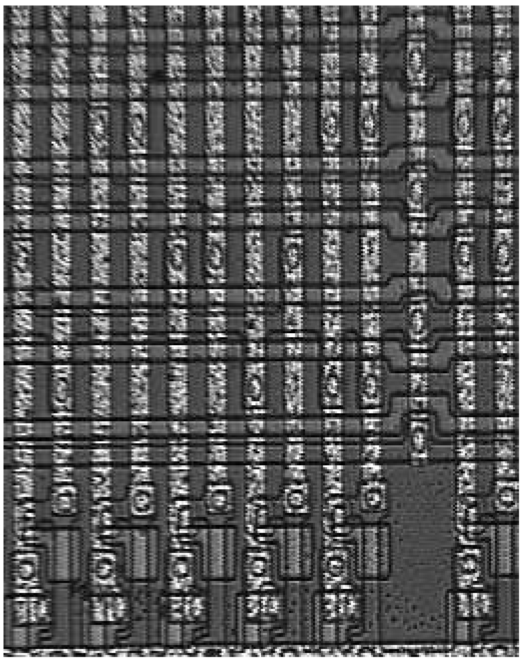
The integration itself is quite simple, it uses few transistors; on the other hand, it occupies lots of space on the silicon, si to bus driver circuits.





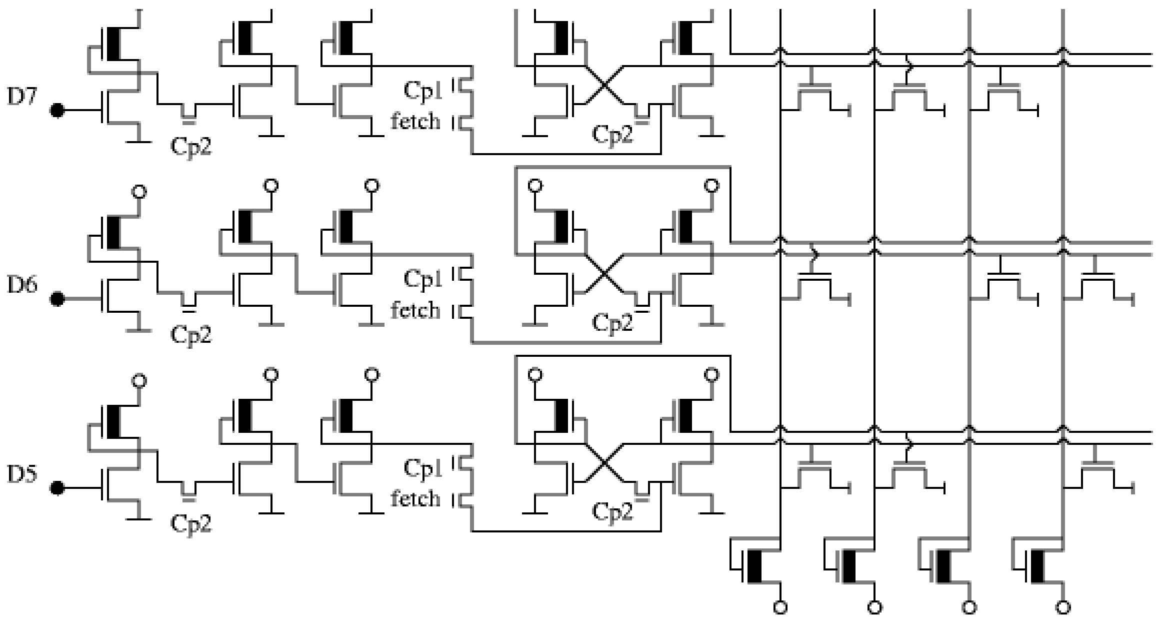
Instruction decoder

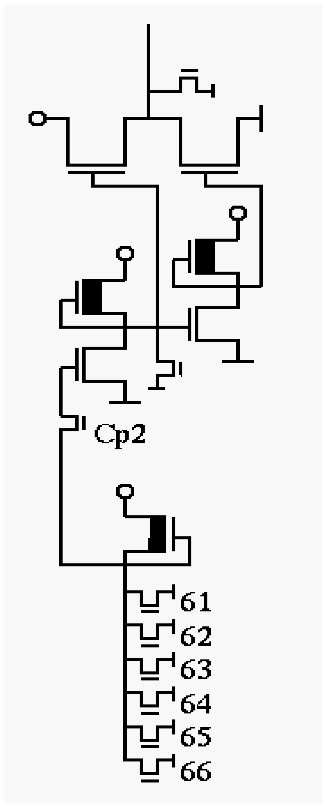
The most important part of it is the PLA, which consists of NOR ports, and is given inputs from the instruction register a timer. Its outputs directly drive the register block. Besides, its significant task is to draw the instruction line. This is im for giving out the fetch signal, for controlling the sync part, and for the interrupt logic (intermission possible only instruction line). The time of loading a new opcode (fetch) depends on the previous opcode, since the major part o instructions requires a given clock cycle, independent of whether there is BCD correction or not.



Potential cases of control transmission are exceptional; in these either control transmission or fetch follows, depending fulfilment of conditions.

The columns in the layout are the outputs of the NOR ports; the lines are the ponated and negated parts of the inputs. very practical because one contact can carry even two bits of information: the negated input of one of the lines an ponated input of the line below. By the way, the 8-databit-input is not in a line but it is immethodical. There might be c that are not contacted anywhere since they do not belong to the instruction decoder; they carry the IRQ request.





The PLA outputs are received by a combination network, which works not only from PLA but also uses several of internal control signals. The internal Ready signal, the Sync signal, and one of the timer outputs is of this sort. Through c ports the PLA outputs get to push-pull inverters, which directly influence the writing / reading of registers, the functio ALU, and the content of bus drive circuits, address and data cables by driving a long, polysilicon cable and 8 transfer gat

There is a simple case illustrated on the left; the controlling of battery writing. From PLA outputs No. 61-66 the foll operations are ported: writing X or Y into A, logic procedures, addition – subtraction, bit tracking of batteries, and loading from stack. Of course, there are far more complicated integrations, too; they also control the transfer gates o register block. For instance, when the content of one track must be written onto another; where the ALU gets its inputs f and so on.

Ports with several inputs process those outputs of PLA, which are not entirely specified; where a couple of the operation
bits are tied down neither to 0 nor to 1.
Due to this, unused opcodes can perform quite interesting operation. Since the conformance of the opcode is not monit
something is to happen by all means. Some outputs of the PLA are activated, and consequently, make the register bloc
something. The so-called "forbidden codes" can be put into three categories:

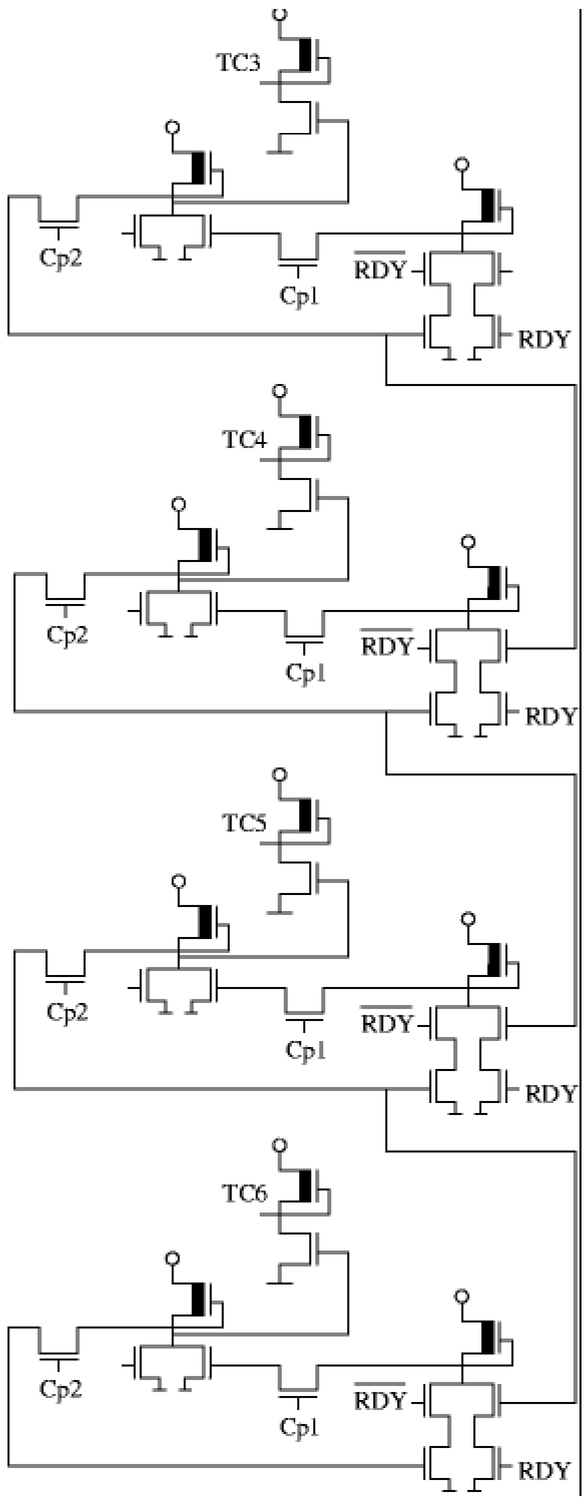
* Empty instruction, such as the NOP: it reads the operand, writes it out on a track, and with this ends its functionin
result is forgotten and left there. Some later instruction will overwrite it, and thus it perishes.

* Rational instruction, which can be complex, and can perform several operations, such as the ANE, where A= (A | #$
& X & #byte. The built-in constant is not guaranteed.

* Instruction that causes freezing. When the content of several registers are written on the same track, or of the
functions counted in one course more than one is forced on the output, the logic ports can work against one another. Ne
sharp level 0 nor level 1 evolves; and though ports accept it and use it, they switch rather slowly. Prescribed timings fa
and the processor becomes unable to function.

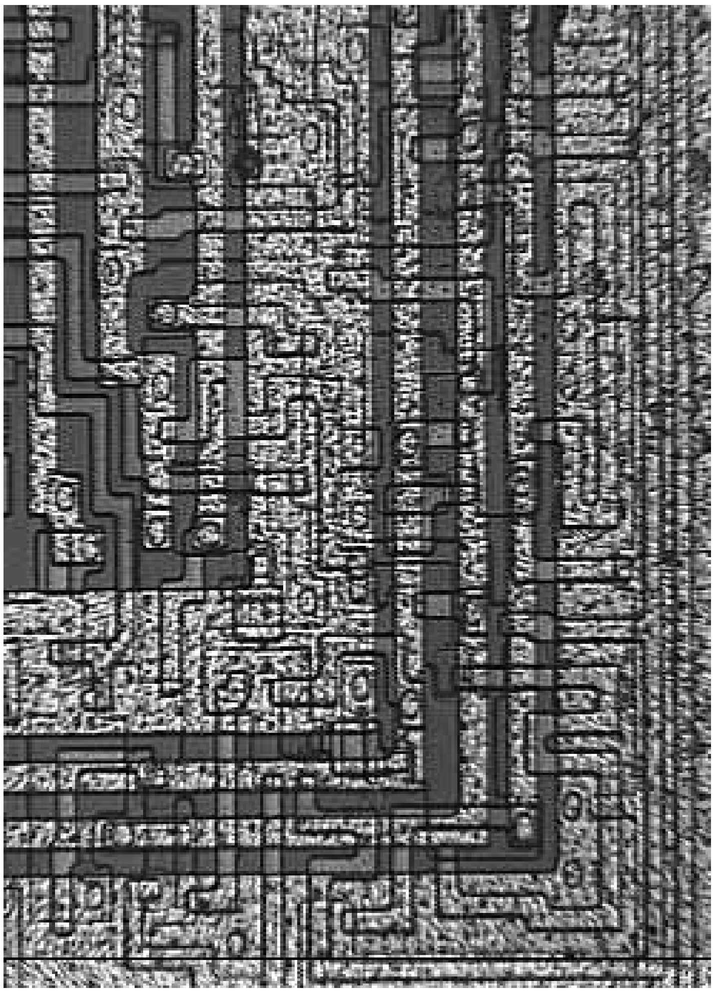
By the way, forbidden codes had an important application in program protection: it was one of the methods used by soft
writers to render the breaking of codes more difficult. In the beginning, disassemblers did not know the forbidden code
showed them as invalid opcodes; therefore, instruction lines were determined badly; the program written with the forb
codes seemed to be a confused byte set, which was difficult to alter.

Timing control



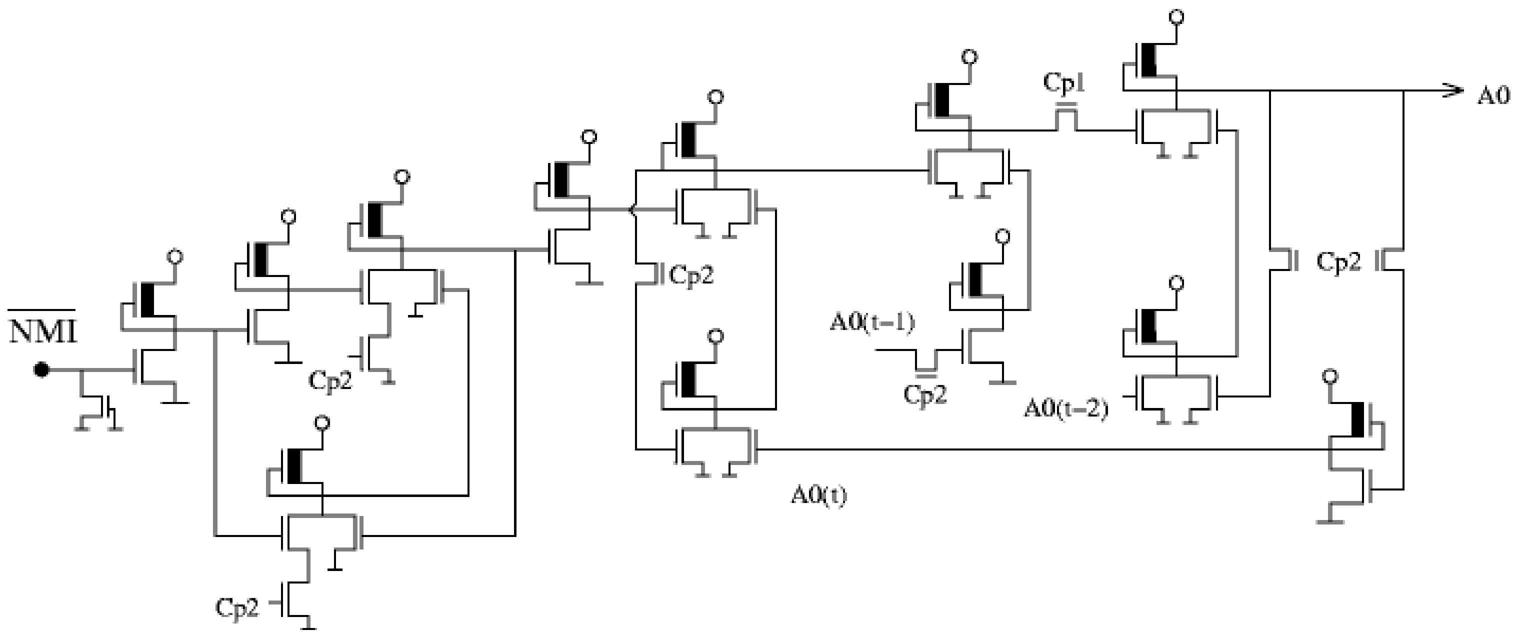
Its main task is to inform the instruction decoder about the clock cycle in which the instruction is during the exec
Instructions need at least 2, max. 7 hours to terminate, the first cycle of which is always a fetch (that is, the loading
operation code to the instruction register).

The timing control gives out the information of 6 cycles through the outputs, which get directly to the PLA inputs. The l
cycles are carried out with a shift register, in which the information is meant by 1 bit that is going along, and in RDY=
the shift register works as a 4 bit storage unit. Apart from this, the register also has a setback input, which is controlled b
instruction decoder.



Interrupt logic

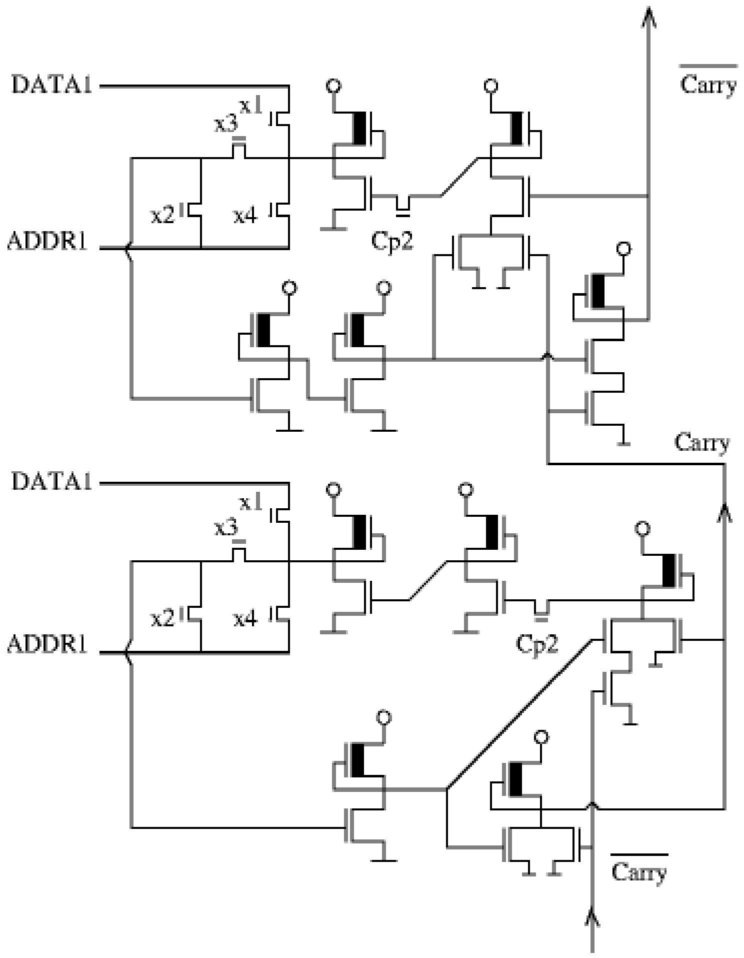
Its task is to receive the interrupt requests coming in via IRQ and NMI lines, to settle permission, and the control giving determined address by the appropriate vector, if the processor reaches the boundary of instruction; and in case of RES the load of the reset vector. All three cases are indirect jump, which need memory reading cycles. In case of IRQ and N stacks are needed, too, and the flags need to be saved together with the instruction counter. When accepting the interrup the interrupt logic stimulates a BRK (software interrupt) instruction to the instruction decoder, the only difference amo events being the address of the vector. When setting the address of the vector, the processor directly drives the address p To complete this, it reserves clock cycles, and it delays the other parts of the processor with an internal READY signal. Rdy0 on the schematic.) The other blocks detect this as if the RDY pin of the processor was pulled down by an external device, which demands extra clock cycles. The three vectors, the NMI, the RESET, and the IRQ are the following: FF FFFC, FFFE. Consequently, when loading a vector, the upper 13 address bit is 1, and the lower three depend on the ev type occurring. The lower bit is always zero, which serves not only to drive the address pins, but also to signal tha interrupt execution is in progress. The following circuit uses this signal, and its variants delayed by some cycles to r clock cycles to accomplish the request:

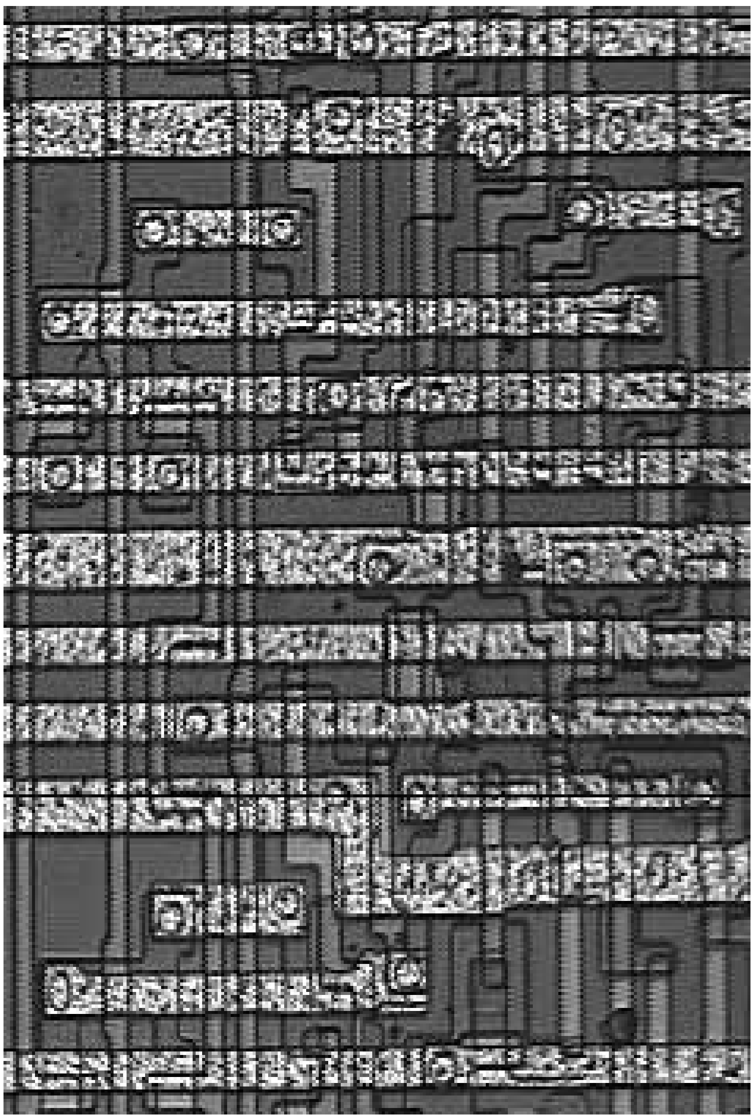


Seemingly it is only the Schmitt trigger input of the NMI that is directly connected to the above mentioned schematic works the same way in case of IRQ and RESET, since the lowest address pin is zero in all three cases, and the sig delayed by clock-cycle 1 and 2 drive this schematic

Register block, instruction counter

A 16-bit register within an 8-bit processor: it was implemented in two 8-bit pieces, two clock-cycles are needed for access. That kind of address method is recommended, which works on the zero page (the lower 256 bytes of the memo because this way we can spare one clock-cycle. The most common operation done by the instruction counter is the incre for this a separate circuit is built in, so that there is no need for the ALU, and the increase goes on independently fro other parts of the processor. The containers are such inverter pairs, which have a feedback on the increaser; if the car zero, the original bit is written back, if it is one, then its inverse.





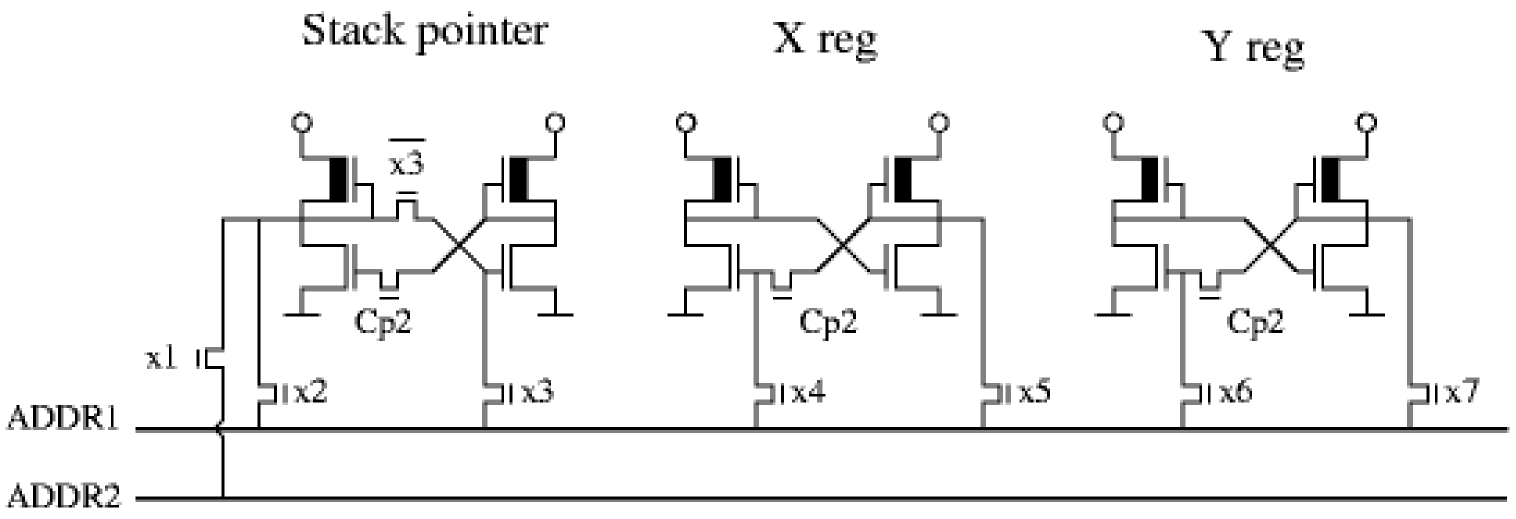
The function of the transfer gates is the following:

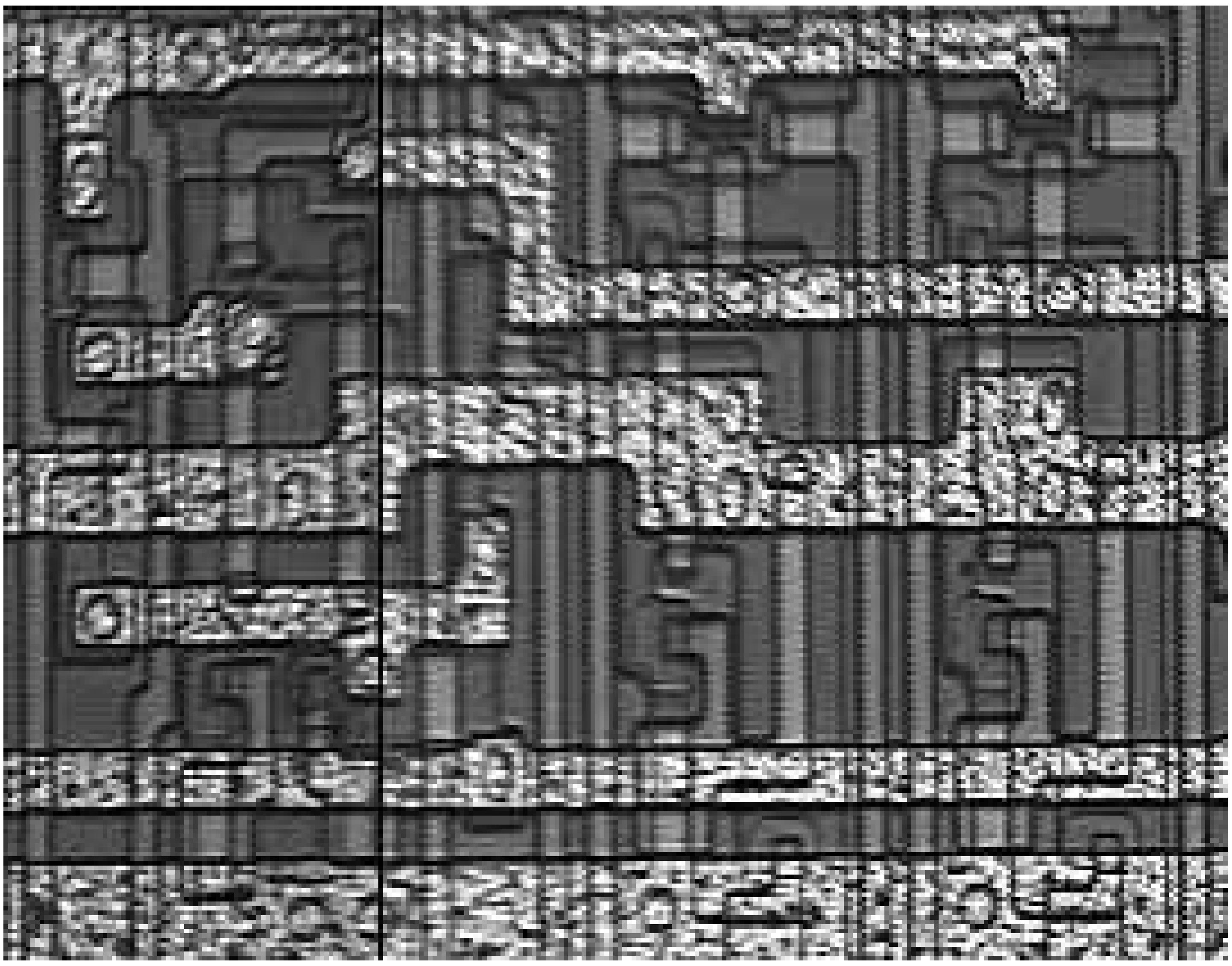
* x1: writing on bus DATA 1, from where one 8-bit half of the IP can get into the ALU or the data pin
* x2: IP initialization from ADDR1 (in case the carry = 0)
* x3: feedback
* x4: writing on ADDR1 (towards ALU or address pin)

Stack pointer, X and Y index-registers

The storing in these registers is done by cross connected inverter pair, which is cut off by a transfer gate controlled by a p signal. When Cp1 is active, the loading of stray capacitance, when Cp2 is active, feedback and the holding of inform takes place. A stack pointer has more functions than X and Y because its content can be written on the address pins, too.

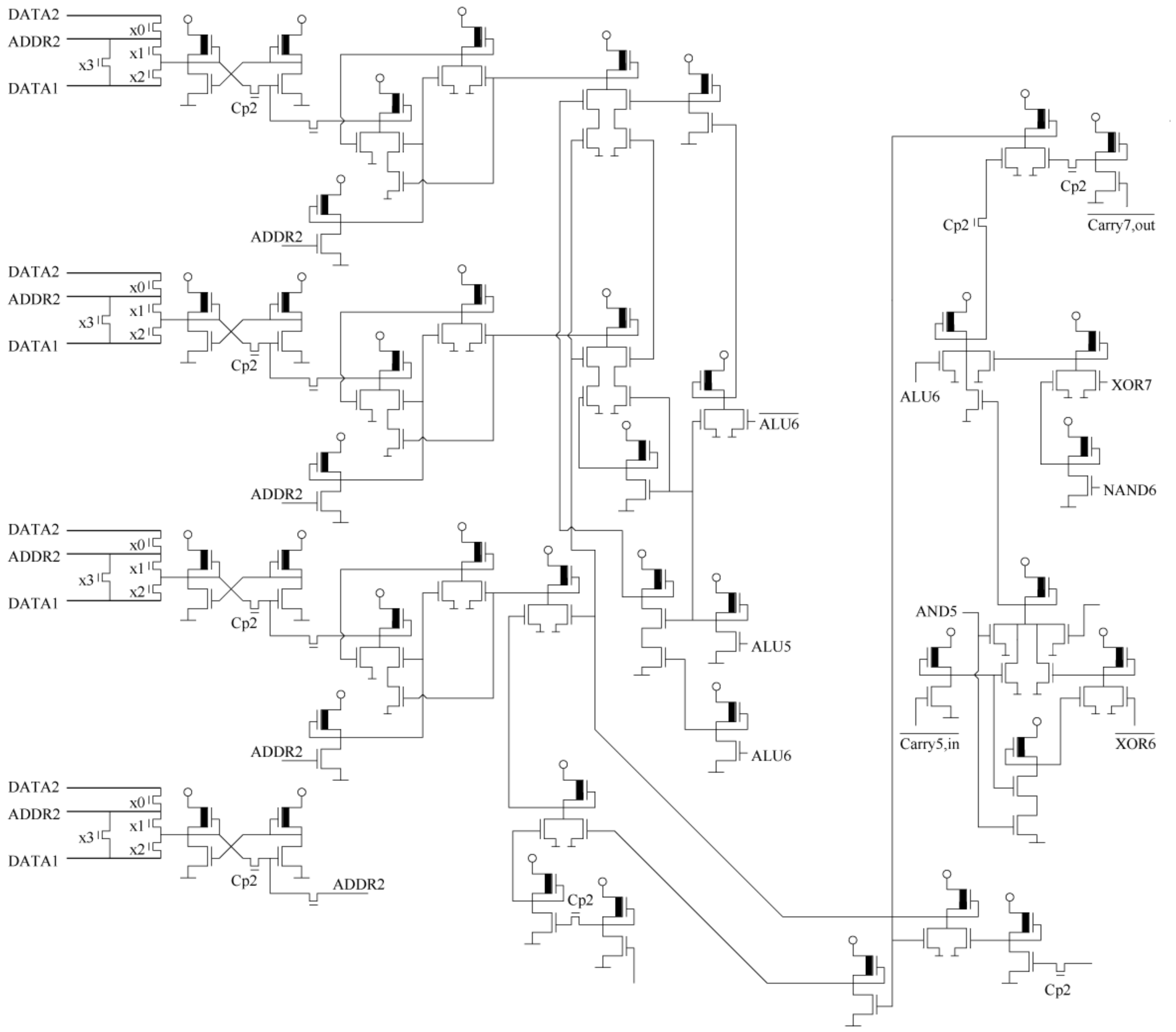
* x1: writing stack pointer on ADDR2 (and from there to the address output)
* x2, x5, x7: writing on ADDR1 (writing other registers or ALU input)
* x3, x4, x6: reading from ADDR1





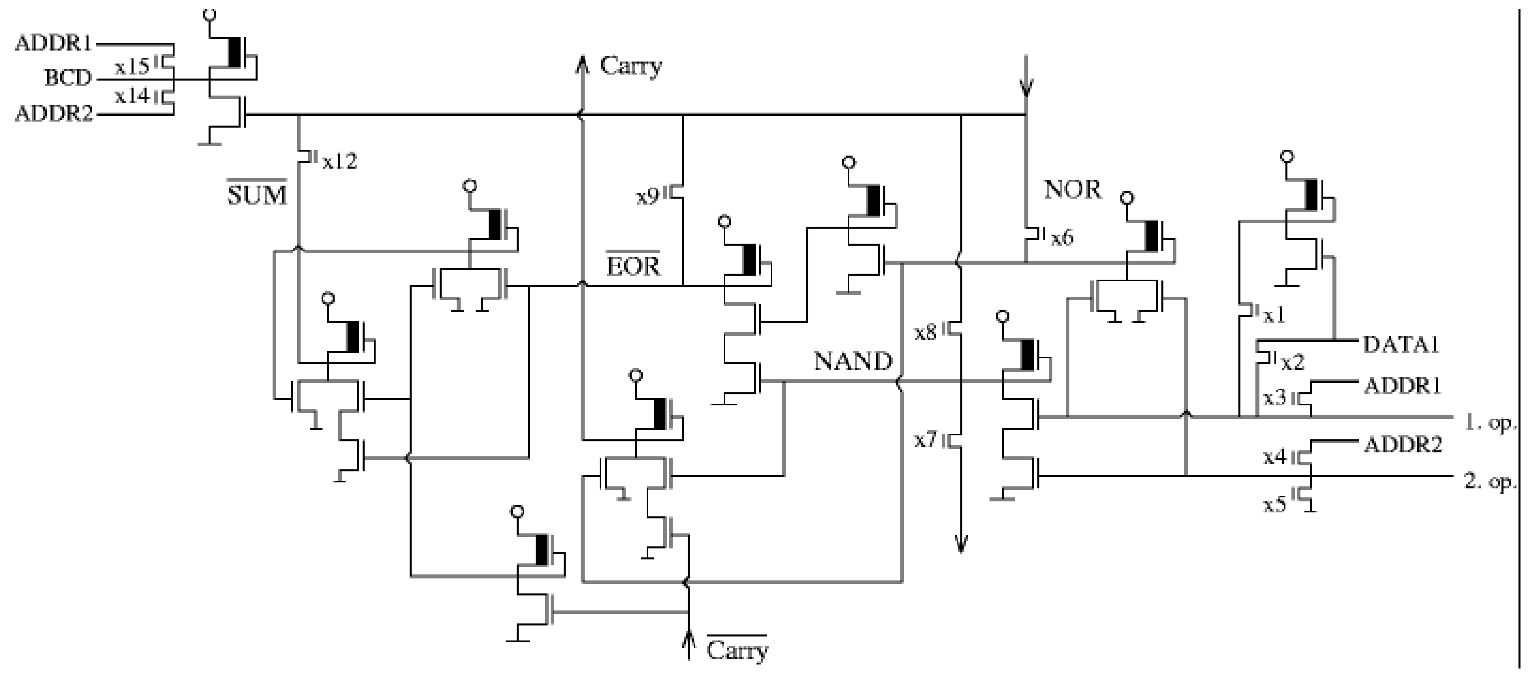
Accumulator, BCD correction

Content to the accumulator can be given through a BCD correction schematic. (the Binary Coded Decimal is a nu representation, in which the byte 33 –hexadecimal 21h– is worked up as decimal 21, therefore, the two 4-bit parts of the contain one decimal digit). When making an addition or a subtraction, the results have to be corrected because the makes a binary addition. Both digits have to be checked; if they are bigger than 9, we have to add 6 (thus decima becomes hexadecimal 10) The following correction circuit makes the addition in a way that it does not use a carry ru along every bit because that is a slow method, which takes 1 clock cycle. Instead, it counts the result of the additi advance from the logical functions of the ALU. The right column of the schematic does the comparison the second c from the right does the increasing. If the digit is bigger than 9 and the BCD is permitted. ON the left can be seen the registers and the adders.



The arithmetical-logical unit

The arithmetical-logical unit is a most compact schematic. It can be found in the register block and it uses the same in buses as the registers. It has two input operands; and on the right side of the circuit diagram can be seen 5 transfer gates. these can be chosen whether the three buses, the negated of one of them or 0 should be the input. The ALU creates all logical functions, the carry, and the sum; independent of the current command. Among these the transfer gates choose to be put on the output.



- Addition: with x12 it gets to the output, needs also carry, which goes on the 8 bit with alternate logic.
- Subtraction: the inverse of the subtracted is added; at the lower place value carry = 1.
- Comparison: same as subtraction, but x12 does not open
- Increase: one of the outputs is 0, addition set to carry = 1
- AND, OR, EOR: x8, x6, x9 switches

* Bit shift to the right: the output of the NAND gate gets by x7 to the negated line of the ALU output of the lower bit. W
needed to this is the same inputs (i.e., the operands have to be available on two buses).

* Bit shift to the left: this is a multiplication by two, which is addition to two identical inputs.

Source URL: http://impulzus.sch.bme.hu/6502/6502/6502_kapcs.php3

Translated by Dia Czipor - dia.czipor@gmail.com: 6-20-8

Owner: BEREGNYEI Balazs - balazs.beregnyei@gmail.com

Thanks to both for all their help and work on this project!