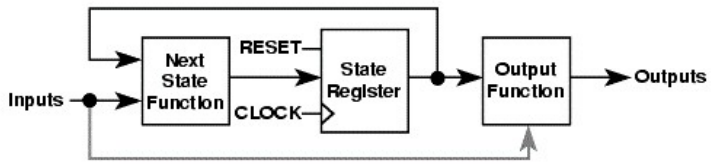
Relay computer technology V1.7 20160412 RJH

State machine intro

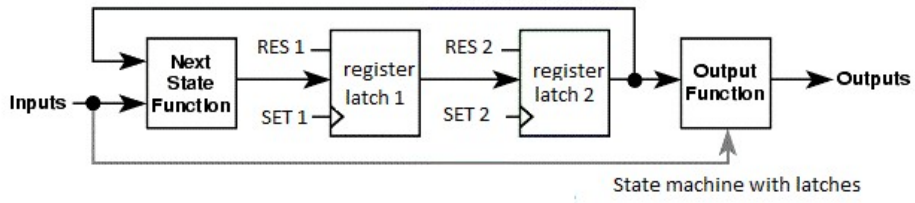
The design that I have in mind is based upon a "finite state machine", known from the literature:



Classic state machine

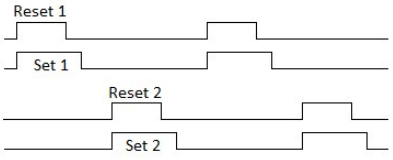
What does this state machine do ? It is characterized by the number of states that can be stored in the state register, and by the "Next State Function". If it is an 8 bit register, the state machine has 256 states. If we forget the inputs for a moment, the "Next State Function" takes the current state as input and gives the next state as an output. When the state register gets its clock signal, the state machine "goes" to this next state.

When we want to implement this in relay logic, we encounter the problem that the "State register" is not easily built, because this must be an edge-triggered flipflop. However, we can solve this by replacing the edge-triggered flipflop by two latches.



State machine with latches

A single relay can act as a latch. The latches must be driven with a correct sequence of set and reset signals:

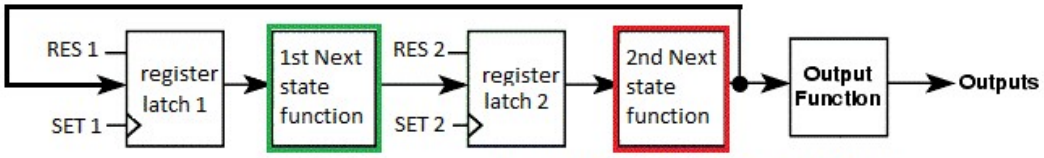


The Reset will set the latch to 0. The Set will set it to 1 if the input data (in the diagram the line with the arrow) is also 1. The effect is that after the Reset and Set signals, the latch has the same logic state as the input data.

Note that RESET can partially overlap the SET signal, also for relay circuits. The RESET must be long enough for the relay to switch off. For the SET signal, it must of course be long enough for the relay to switch on, and at the end of the SET signal you must be sure that RESET has ended, so the hold contact of the relay will keep the relay in ON position when the SET signal is gone.

Splitting the "Next state" function in two sections

The "Next state function" will probably introduce some delay, that makes that to achieve the fastest speed, the time between SET1 and SET2 must be different than the time between SET2 and SET1. But for the system that sends the set- and reset signals to this circuit, it may be practical to have these times equal. This suggests to split the next state function in two sections that have an (almost) equal delay:



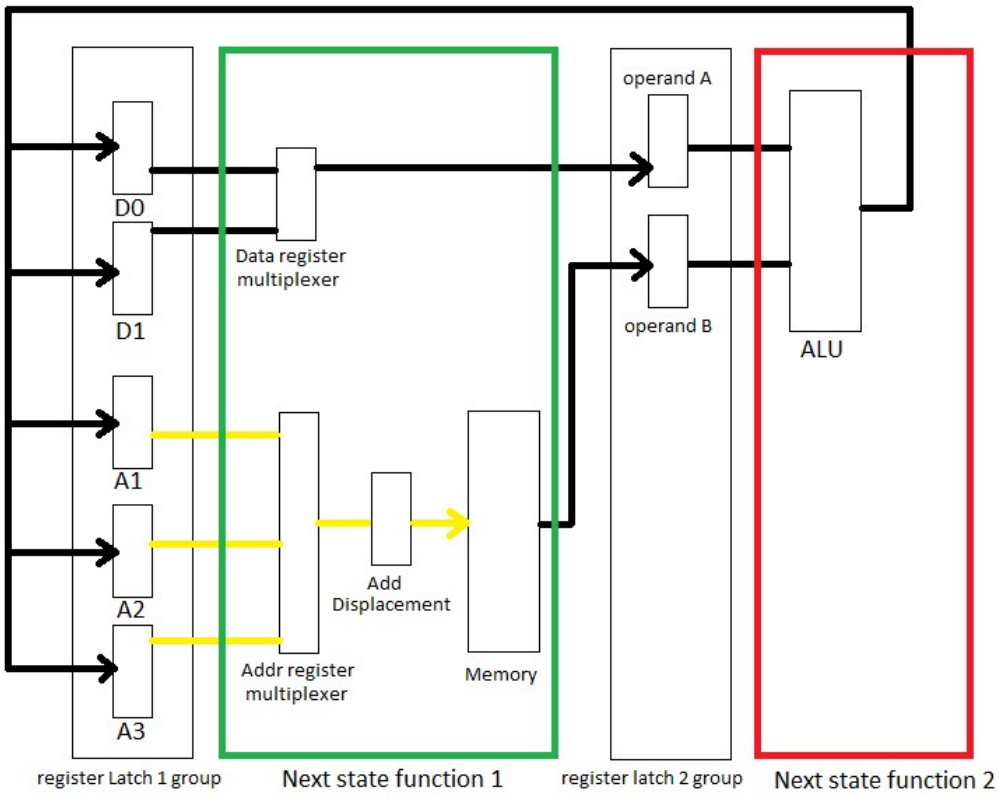
Splitting the "Next state function"in two

The input has been left out of the drawing because it is not essential. The both function halves are marked green and red. Note that both register latches are not restricted to 8 bits, they can contain as many bits as you want.

Especially when we use DPDT registers for the latches, another benefit of this split system is, that it may make better use of the relay contacts, because these contacts can be a part of the function that follows the latch.

In the intended computer, all computer registers are in the "Register latch 1" box, and the input registers for the ALU are in the "register latch 2" box. Suppose that one of our data registers is called D0, and our ALU is in the 2nd state function, this system gives the possibility that the D0 register is input to the ALU, the ALU does a calculation and the result is sent to the same register D0.

Basic architecture of a simple CPU



This diagram shows the most important parts of the CPU. It has the following functionality:

- The ALU can accept input A from a register and input B from memory (or immediate, not shown), and write the result to the the same (or another) register
- The ALU can ignore the A input and put B at its output, to load a register from memory.
- A register can be stored to memory (not shown).

The sequence of operation is:

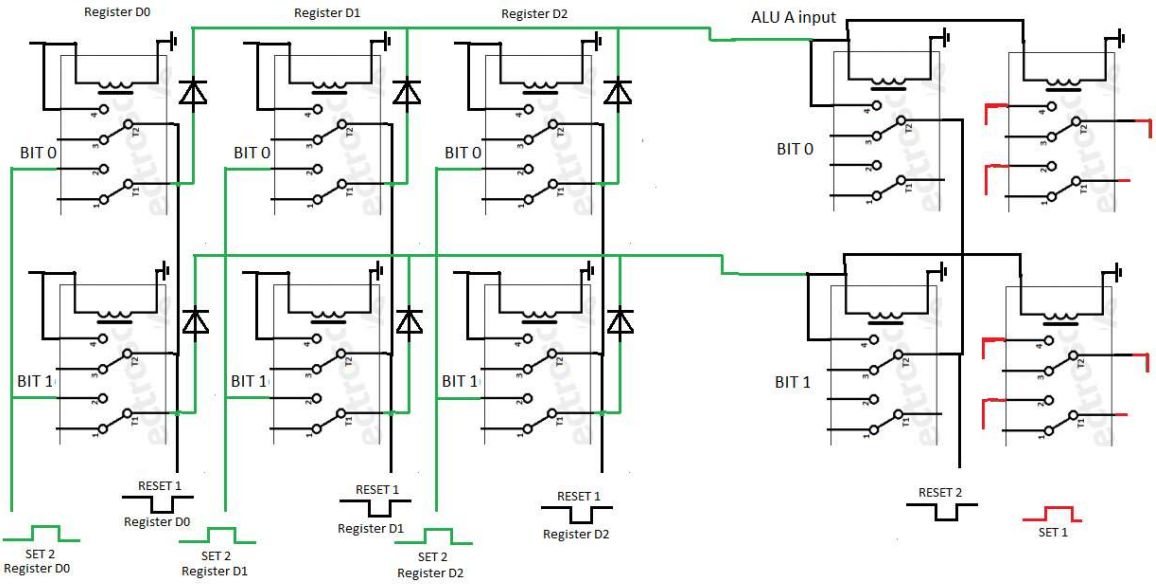
- The operation starts when new data has just been written to the registers of the "Latch 1" group.
- The "Next state function 1" selects operand A, selects an address register, adds a displacement to the address, read data from memory as operand B.
- Operands A and B are written to the registers in the Latch 2 group
- The "Next state function 2" calculates the ALU function
- The result of the ALU function is written to a register in the "Latch 1" group

This solution supposes that the memory read is fast enough. If the memory is slow compared to the relay actions, another solution must be found.

In the diagram, some essential CPU parts are omitted, like the program counter, instruction register, writing to memory, using immediate values, and all control circuits.

Selecting register output

Now we will propose some details of the described principle.



This picture has 3 data registers (for each register, only two bits are shown). The relays at the right side represent the A input of an ALU (also two bits shown, ALU function not shown).

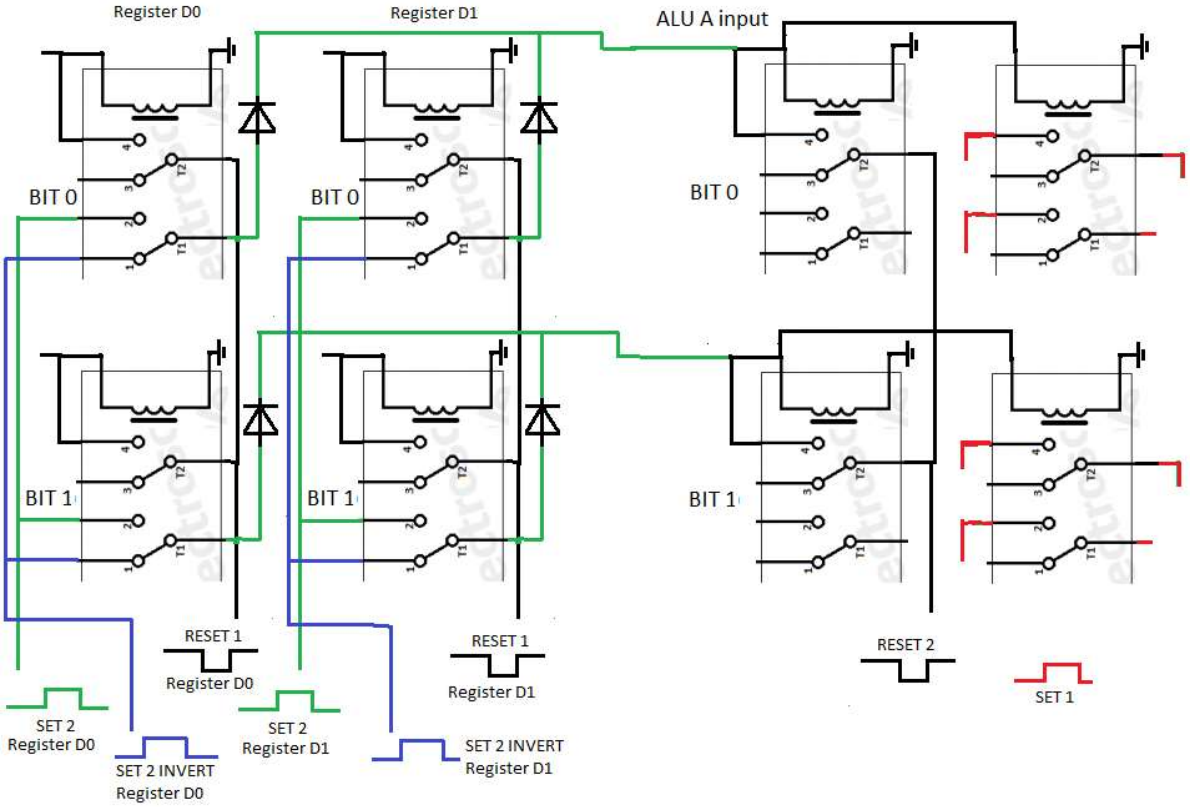
We assume that the three dataregisters are already filled with data (writing to these registers is not in this diagram).

To write to the ALU input, we activate the RESET 2 signal to reset the ALU input, and then activate one of the three SET 2 inputs to write the data of the corresponding register to the ALU input.

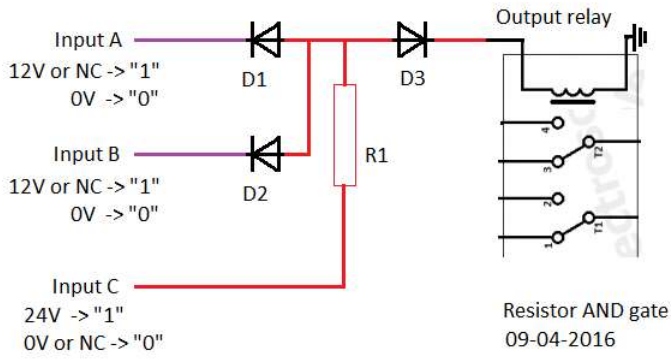
Note that we can also force the A input of the ALU to zero, by not selecting any register.

Note that in this description, the reverse diodes that are needed to prevent high voltages during relay-switch-off are not shown, to keep the diagrams as simple as possible.

The previous example is now expanded with an extra function. We can also load the ALU with the complement of the data register contents (with the blue control signals), with no extra relays or contacts per bit. Only a new control line is needed for each register. (Only two registers shown this time).



Diode - Resistor AND gate



In the following parts of this story, it is supposed that 12 Volt relays are used and that the resistors have the same resistance as the relay.

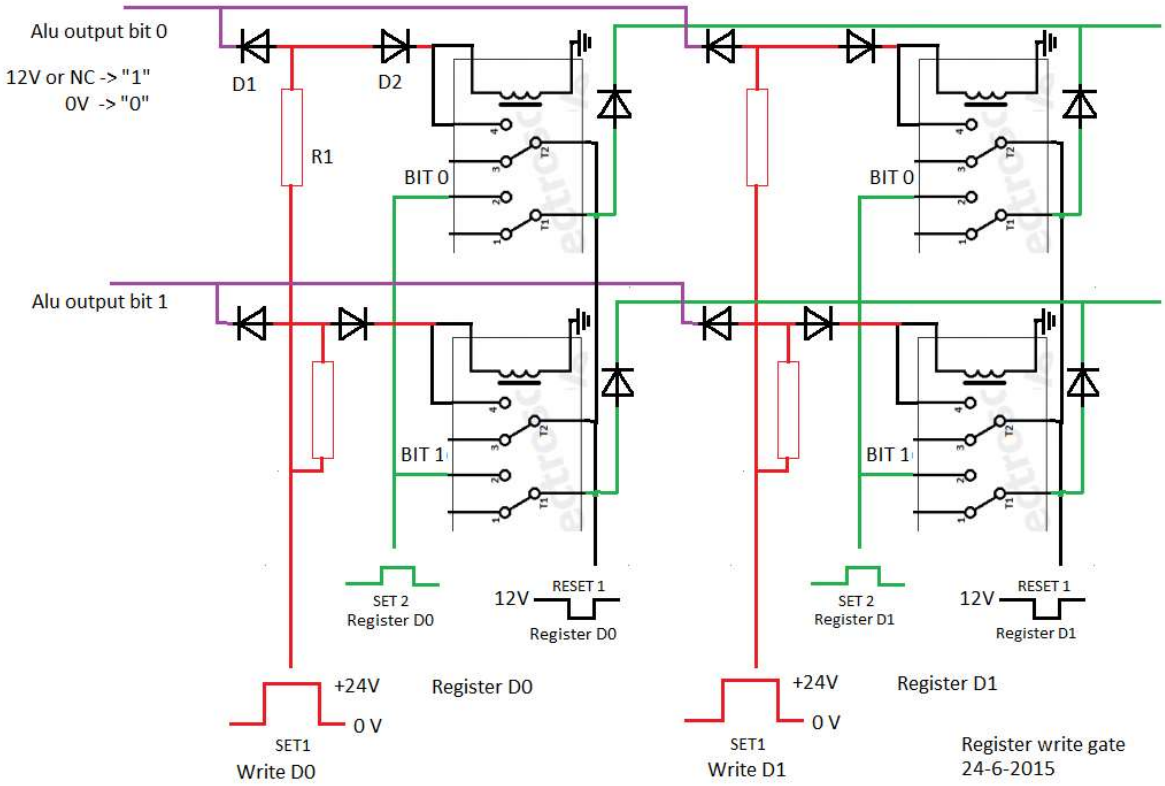
It is clear that, as long as input C is not connected or connected to ground, the relay can not be activated by input A or B, because the diodes do not allow that. But when input C has 24 volt, the relay will activate, UNLESS input A or B has a zero voltage, that pulls the upper end of the resistor to ground.

In other words, this circuit is a 3-input AND gate. If we use 24V for "1"and 0V for "0', the relay will only be activated if all 3 inputs are logic "1".

We can easily add more inputs with just one diode extra for each extra input.

It has just one drawback.... When input C is "1" but one or more of the other inputs is "0", the resistor has almost 24 Volts across it, that means that it will use 4 times as much energy as a relay in the "ON" position. OK... but if we had wanted to built a really energy efficient design we would have used CMOS....

Writing to a register



The output of the ALU is connected to several registers. We need a method to write this output to a certain register without affecting the contents of the other registers.

There is a way to do this, without the need for a relay contact for each bit. There are only relay contacts needed that are in common for all register bits. This is an application of the diode-resistor AND gate of the previous section.

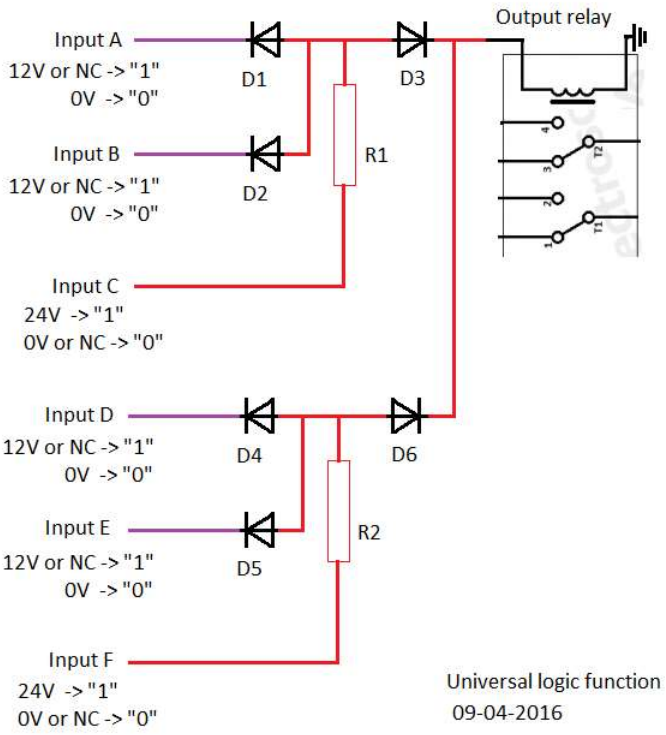
Let's use bit 0 of register D0 as an example. Suppose the relay is off (RESET 1 was activated) and the ALU output is 0. Now when SET1 for D0 becomes active, the lower connection of the resistor R1 is 24 V, but the other side can not rise higher than the 0.7 volt of diode D1, because the other side of the diode is connected to 0 volt ALU output. So, the relay will not switch.

Now suppose the relay is off and the ALU output is 1. Now when SET1 for D0 becomes active, the lower connection of the resistor R1 is 24 V, and the other side will rise in voltage because diode D1 has 12 volt from the ALU output at the other side. So, the voltage will be equally divided between the relay coil and the resistor, and the relay will switch.

So, by applying the SET1 / RESET1 pair to the correct register, we write to this register.

When the SET1 voltage is not present (or 0 volts), the two diodes D1 and D2 will isolate the register from the ALU output.

Universal diode-resistor logic



The AND function can be combined with an OR function. The circuit above shows two 3-input AND gates, the outputs of the AND gates are inputs to a 2-input OR gate (built with D3 and D6). It gives the following logic function:

Output = (A and B and C) or (D and E and F)

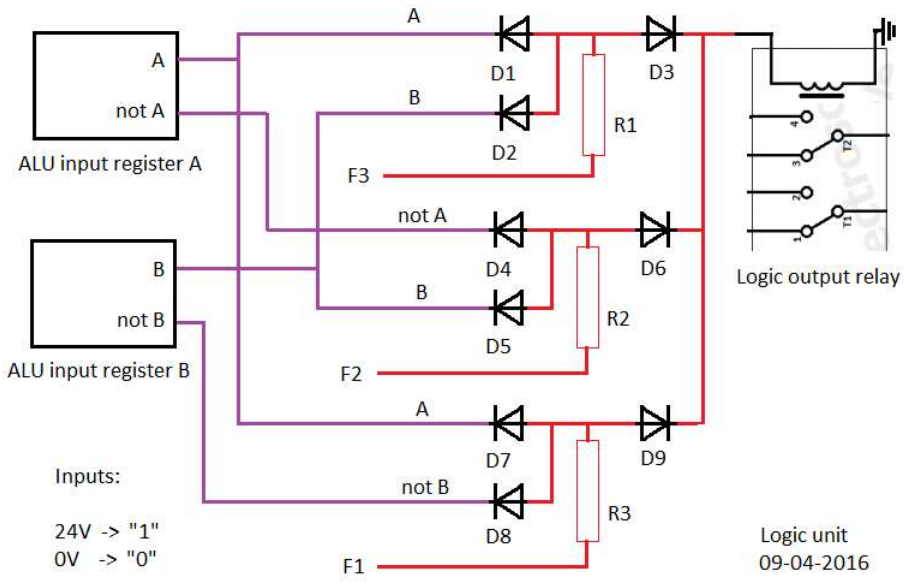
If all inputs are also available in an inverted version, it can be shown that with this kind of solution you can create ANY logic function (as long as it does not contain a memory function).

Creating several logic functions is just what we need for the logic part of the Arithmetic Logic Unit (ALU)....

Note that when the voltage levels are reversed, you can use the same circuit, but with the diodes reversed and the output relay connected to 24V instead of ground.

Complete logic function ALU

The next diagram shows the logic functions of the ALU (only a single bit shown):



From the previous section we know, that this function calculates the logic value:

Output = (F3 and A and B) or (F2 and (not A) and B) or (F1 and A and (not B)).

The inputs F1, F2 and F3 are the same for every bit in the ALU. These bits are derived from the instruction register and control the logic function of the ALU.

Suppose F3= 1, F2= 0, F1= 0, this simplifies the output function to:

Output = A and B. So this clearly is the AND function.

Now suppose F3= 0, F2= 1 , F1= 1, this gives:

Output = ((not A) and B) or (A and (not B)). This is the XOR function.

Now suppose F3= 1, F2= 1 , F1= 0, this gives:

Output = (A and B) or ((not A) and B).

Suppose now also A = 1, it gives **(A and B)**, since A=1 this is **B**. But if A = 0, it gives **((not A) and B)**, and since now A=0, this is **((not "0") and B)** = **("1" and B)** is also **B**. Apparently, for **[**F3= 1, F2= 1 , F1= 0] the value of A doesn't matter, the output is always **B**. So this can be used for the LOAD instruction (LD).

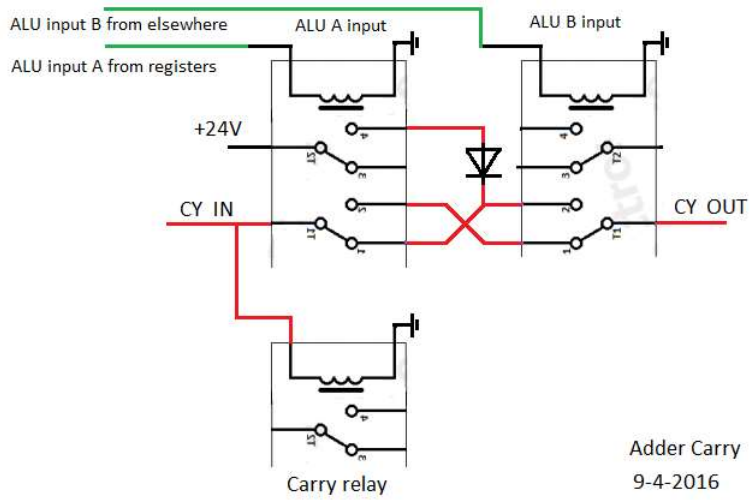
The last useful setting is F3= 1, F2= 1 , F1= 1, this gives:

Output = (A and B) or ((not A) and B) or (A and (not B)).

This is the same as **(A or B)** , so it is the OR function.

Adder carry circuit

The next part of the ALU is the adder. This is the carry circuit (only one bit shown):



The carry is defined as active "1" if it is connected to 24V. A "0" is 0V or not connected (NC).

If both ALU inputs are "0" (not active), the carry output must be "0". This is the situation as shown, it is clear that the CY_OUT is not connected.

If only one of the ALU inputs is "1" (so both inputs are different), the carry output must be equal to the carry input. It can be seen in the diagram that the CY_OUT is connected to CY_IN when both relays are not in the same state.

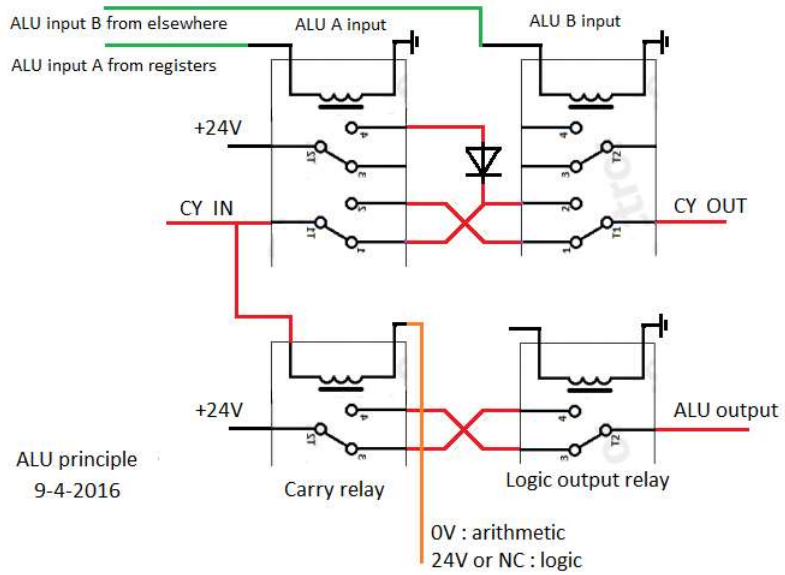
Finally, when both ALU inputs are "1", the carry must also be "1". This is accomplished by the diode, that will put 24V on the carry output if both relays are on.

Note that at every bit position in the ALU there is a carry relay. In some situations, the current through a single such diode has to power all these carry relays in the ALU (and also relays that are used for decimal correction), so this is something to bear in mind when the diode is chosen.

Note that in this circuit, the diode is not really needed and could be replaced by a short. However, in the final ALU there are also other circuits connected to the upper contact of input relay A, that makes the diode necessary.

Relay adders are also explained in the excellent pages of Dieter Mueller:
http://www.6502.org/users/dieter/a3/a3_1.htm

Complete ALU circuit



The logic output relay is added to the diagram. The input circuit to this relay is not shown here, but is comparable to the logic function unit. discussed before.

For addition, the logic function must be set to F3= 0, F2= 1 , F1= 1, giving **Logic output = A xor B**.

When the orange wire (common for all ALU bits) is connected to ground, the carry relay and logic output relay together form an XOR circuit, so the output is, as required for addition:

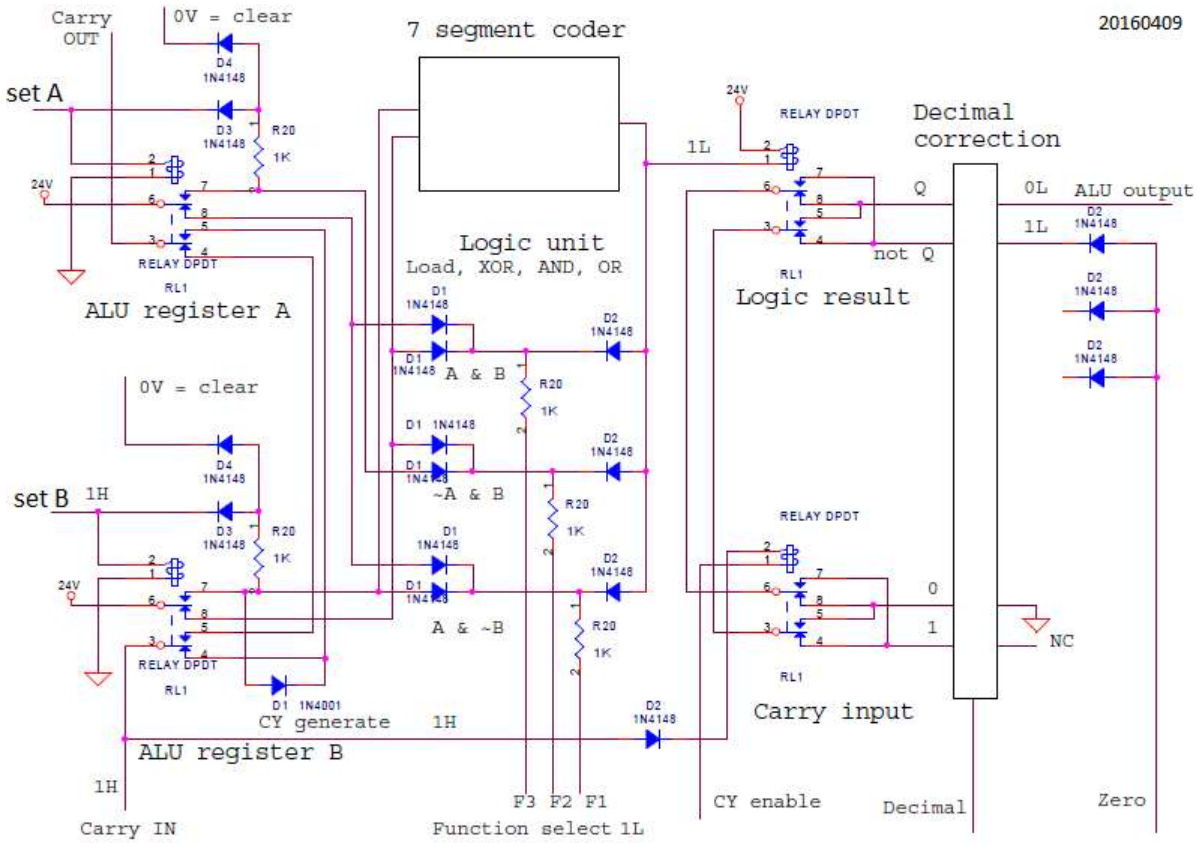
(A xor B) xor CY_IN.

Note that when the orange wire is not connected to ground, the carry relay will never be energized. In that case the ALU output is equal to the logic function as discussed before.

All the ALU functions have been discussed now (except BCD and 7 segment functions). An overview is in the following table:

F3	F2	F1	logic mode	OUTPUT
0	1	1	0	ADD
0	1	1	1	XOR
1	0	0	1	AND
1	1	0	1	LD
1	1	1	1	OR

ALU circuit schematic



In the schematic, codes are used: 0L means logic "0" is connect to low voltage level (pulldown to ground), 1L means logic "1" is pulldown to ground, 1H means logic "1" is connect to high voltage (pull up to 12V or 24V) .

This schematic differs in a few points from the discussed ALU.

- The logic unit works with inverted logic levels, because the outputs of registers A and B can only pull up to 24V (and not pull down to 0 volt).

- The ALU also has an inverted output, that is needed to detect if the ALU output is zero.

- The output of the ALU, that connects to the registers, must be capable of pulling the output line down to 0 volt, so in this design the logic is reversed, and the ALU can connect the output to ground to indicate logic "0".

- The input to the contacts of the "carry input" relay, that was +24V on the previous page, has been replaced by two signals that are each others complement. This is needed for the decimal correction. For non-decimal modes, these inputs are connected to "0" (ground) or "1" (not connected) as indicated.

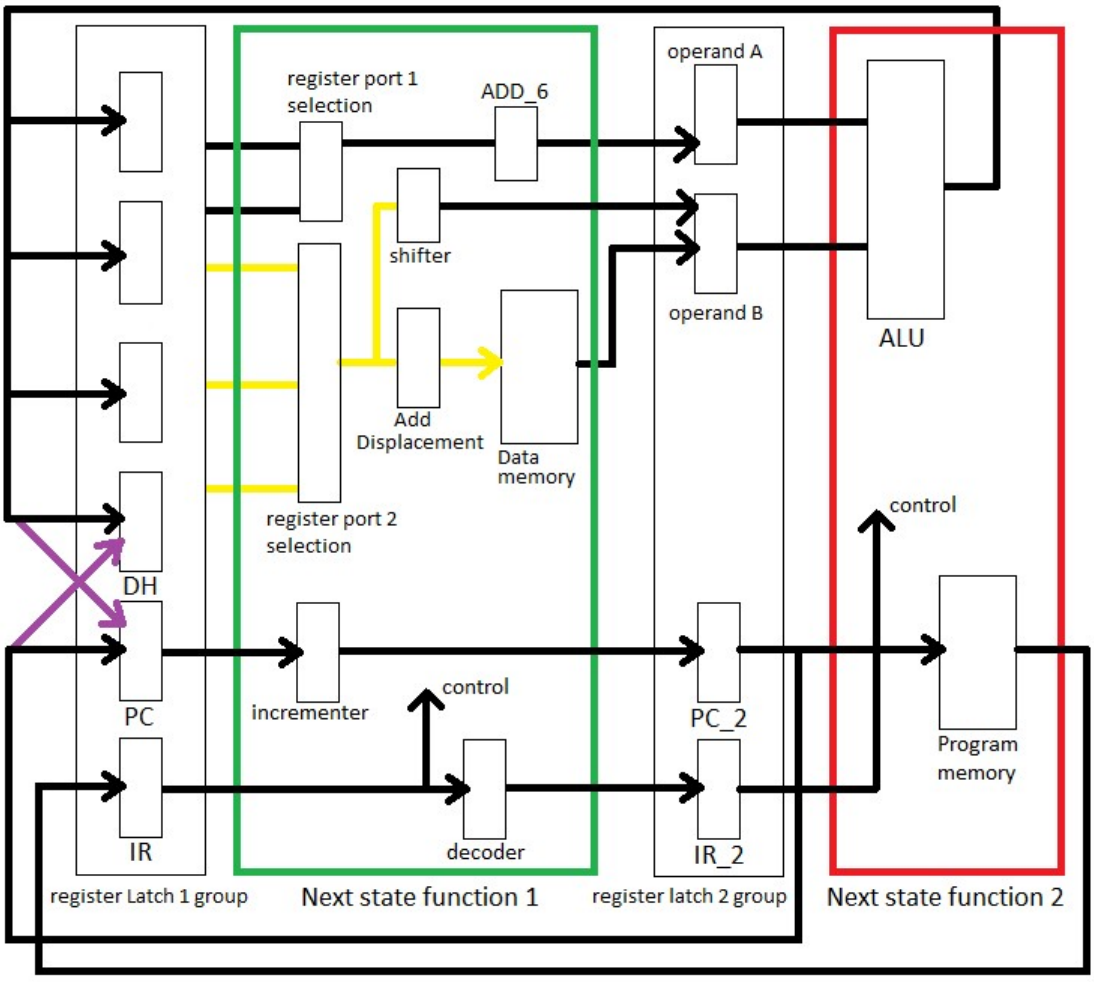
- There is a "7 segment coder", that converts BCD code to the 7 segments in a display. Its inputs are all ALU-register-B bits that belong to the same nibble. To be described later.

Generating control signals

The goal is to built a CPU without microcode. This means that the control signals must be generated directly from the instruction. Let's have a look at the instructions (from the ISA document):

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Zero page	**0**	**0**	**M**			**R**			**L**	**Z**						
Immediate	**0**	**1**	**M**			**R**			**Immediate data**							
Memory	**1**	**0**	**M**			**R**			**L**	**A**			**D**			
Register	**1**	**1**	**M**			**R**			**L**	**S**	**S**		**Q**			**S**

And have a look at the system diagram:



This diagram again does not show all details.

The instruction register will be written at the same time as the "register latch 1" group. The signals to control the "Next state function 1" must be generated with minimum delay. The signals that control the "Next state function 2" can tolerate more delay, and these control signals must also be latched in the latch 2 group (with IR_2), otherwise our state machine principles are violated.

The purple signal flow indicates what happens for a subroutine call.

The most important things to decode for next state function 1 are:

- Control what register to transfer to ALU input register A
- Determine if this register A value must be stored in memory, or memory must be read
- Determine address register (or second register operand) and connect it to the address bus
- Load ALU input register B from memory, from an immediate or from a register
- Determine the shift amount if ALU-B loaded from register (not shown in diagram)
- Handle 32-bit instruction

In more detail, this is what the decode function must do for next state function 1:

- control what register to transfer to ALU register A
    - Is determined by the 3 "R" bits.
    - For decimal add (DADD), 6 must be added before storing in ALU A register
- determine if this register A value must be stored in memory, or memory
    - Must be stored if MMM = 010 and bit14=0 (ST instruction, zero page or memory)
- determine address register (or second register operand) and connect it to the address bus
    - Determined by bit 5 and 6 of the instruction
    - For register operands, bit 0 determines low or high register
    - When bit15=0, do not select any address register (zero page)
    - For SUB instructions, register port 2 must complement its output
- Load ALU register B from memory, from an immediate or from a register (through shifter)
    - Is determined by bit 14 and 15 of the instruction:

	15	14	Load B from
Zero page	**0**	**0**	**Memory**
Immediate	**0**	**1**	**Immediate**
Memory	**1**	**0**	**Memory**
Register	**1**	**1**	**Shifter**

- determine the shift amount if ALU-B loaded from register (not shown in diagram)
    - To be defined later
- Handle 32-bit instruction
    - If "32bit"flag was set, reset it. If "32bit"flag was not set:
    - If L-bit was set, not in immediate mode, and "32bit" flag is not set, then do not increment PC and set "32bit" flag.

For next state function 2, the important control signals are:

- Determine the ALU operation
- Determine ALU carry input, arithmetic mode, decimal mode
- Determine in which register the result must be stored
- For a LD PC instruction, the incremented PC must be stored in DH
- Determine if the result must be stored (it might be a conditional instruction or compare)
- Determine the new value of the T flag
- Handle 32 bit instruction

Details for these control signals will follow later.

--- end ---