The Anitra Project

How simple can a basic computer be designed?

by Eirik Bakke

Goal

Although hypothetical minimalist computers has long been an interesting study in Computer Science, there exists very few actual hardware implementations of such computers, and the goal of this investigation was to design one from scratch using the standard integrated logic circuits of digital electronics.

The Investigation

I started my investigation by defining the exact requirements for my computer. Then, as I was aiming to keep its design as simple as possible, I tried to logically deduce what components would unquestionably be required parts of it. Using only components presumed to be essential, I could put together a structure that should be capable of fetching and executing instructions in memory. From this, I went on by designing the necessary control logic and defining the circuits in detail on paper.

Conclusions

Given the initial requirement to restrict its design to standard circuits, it seems possible to devise a computer of ultimate simplicity by proving the need for each hardware component logically. However, the argument used in the investigation is not watertight, and it does not cover in detail the computer's control circuits.

The computer components found to be essential included data retaining units for holding addresses and values during memory write cycles, a data retaining unit resembling a program counter for holding additional addresses independent of the former ones, and an arithmetic adder for combining two values from memory into one. It is interesting to note that familiar components such as a program counter and an arithmetic adder are needed even in a minimalist computer. However, efficient simplifications may be done by limiting branching functionality and the memory area where instructions can reside.

The Anitra Computer

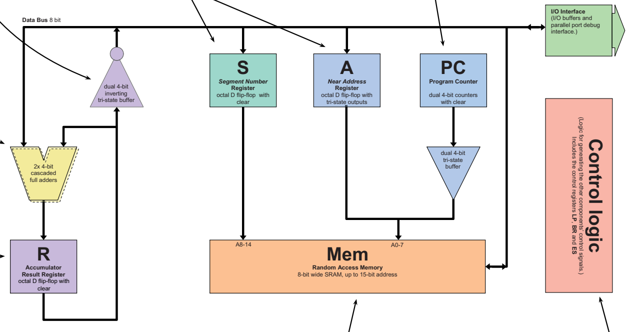
Inverting buffer: Feeds the memory with data from R only during write cycles, and enhances the functionality of the executed instructions by inverting the result before returning it.

Segment Number Register ("S") and Near Address Register ("A"): Together stores a full two-byte address originating from memory to be used during write cycles and data fetching read cycles.

Program Counter ("PC"): Because Anitra must look somewhere in memory to fetch the address of someplace else, there must be two ways to generate addresses rather than a single one. PC generates a near address that, together with S in its cleared state, will provide an additional full address to be used during instruction fetching read cycles. Buffers controls whether PC or A currently feeds data to their common part of the address bus.

I/O interface: Provides a way for external components to communicate with Anitra's software, and for uploading software to memory. Technically speaking not an essential part of a computer, since it can be removed at any time during execution without altering the computations being performed.

Arithmetic adder: Provides a convenient way to combine two values from memory into one, a fundamental operation for all computers. It is configured so that new values coming from memory will be added to the current value of R before saved in the same register.


Data Bus 8 bit
dual 4-bit inverting tri-state buffer
S
Segment Number Register
octal D flip-flop with clear
A
Near Address Register
octal D flip-flop with tri-state outputs
PC
Program Counter
dual 4-bit counters with clear
I/O Interface
(I/O buffers and parallel port debug interface.)
2x 4-bit cascaded full adders
dual 4-bit tri-state buffer
R
Accumulator Result Register
octal D flip-flop with clear
A8-14
A0-7
Mem
Random Access Memory
8-bit wide SRAM, up to 15-bit address
Control logic
(Logic for generating the other components' control signals.)
Includes the control registers LP, BR and ES

Accumulator Result Register ("R"): Stores values or sums of values loaded from memory, and returns them when they are to be written back in memory.

Memory ("Mem"): Stores instructions and data for Anitra. A fundamental component in any computer, although passive in itself.

Control logic: Generates the control signals that make all the other components perform their actions in the appropriate order, so that Anitra can successfully fetch and execute instructions.

"A computer is a machine that, given enough time and memory, can perform any computational operation on a set of data. A minimalist computer is a computer that satisfies this requirement with only a minimum level of architectural complexity."

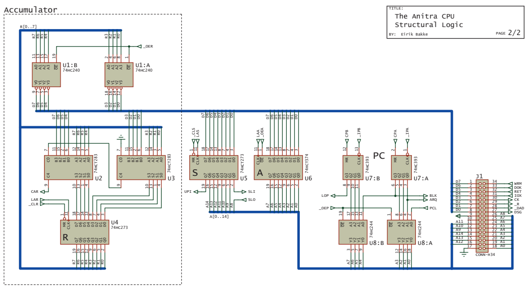
Anitra's two universal instructions

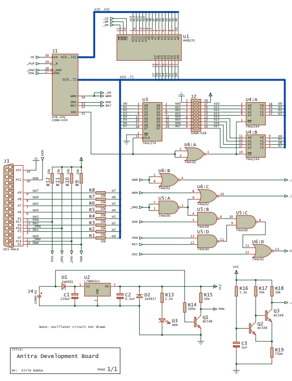
A computer works by mechanically executing a sequence of so-called instructions in memory. An instruction is a signal for the computer to carry out a single low-level computational operation. Such operations may read and alter values in memory, or affect the order instructions are executed. By combining such instructions in the appropriate way, more advanced operations may be synthesised. The Anitra computer has a predefined area in memory where instructions may reside, subdivided into 16 blocks of 8 instructions each. These will execute over and over again in an eternal loop. While modern computers have hundreds of different instructions available for the programmer to use, Anitra has only two:

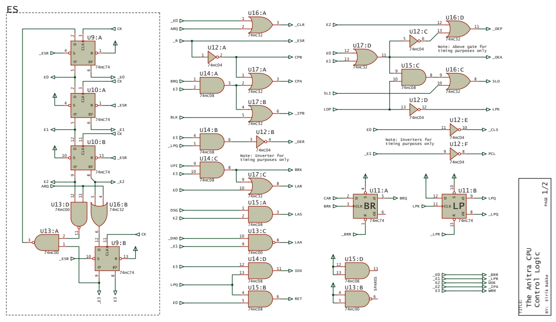
"mov S,Q"	Read the value in memory at address S, and save an inverted copy at address Q.
"add S,Q"	Read the values in memory at addresses S and Q, add them together, and save the result inverted at address Q. If the result is too large to fit into Q, skip the rest of the instructions in the current block. (In the latter case, discard the result's most significant bit and save the rest. If the current instruction is the last one in its block, skip all the instructions in the following block instead.)

An address is a value identifying a single place in memory. In Anitra, an 8-bit computer, each such place may hold a value made up of 8 binary digits. A value is said to be inverted if each of its digits are complemented (1s becomes 0s and vice versa)

Although Anitra's two different instructions may seem fairly primitive, and although the area of memory where instructions are executed can only hold 128 such instructions, there is still in fact no limitation on what software can be written using them. Because the instructions can access any part of the memory, and because some instructions may be used to reprogram other ones, there are plenty of ways to extend Anitra's functionality to a more practical level. Together, Anitra's two instructions are truly universal.




Note: oscillator circuit not drawn
TITLE: Anitra Development Board
BY: Eirik Bakke
PAGE 1/1


ES
Note: Above gate for timing purposes only
Note: Inverter for timing purposes only
Note: Inverters for timing purposes only
BR
LP
TITLE: The Anitra CPU Control Logic
BY: Eirik Bakke
PAGE 1/2

Theory of Operation

(Summary)
WARNING: Geeks only!

Anitra is built around a 32-kilobyte SRAM memory chip with an 8-bit data bus. Full 15-bit addresses for the memory are made from a 7-bit segment number and an 8-bit near address. The two data latches S and A may both retrieve data from the memory through the data bus. The segment number is always prepared in latch S. The latter may contain data from the memory, or it may be reset to zero. The near address can come from either latch A or the counter PC, depending on which of the two currently has its tri-state outputs enabled. PC's first and last 4 bits may be increased or reset individually. PC's least significant bit is controlled directly by the control logic. A simple accumulator system is connected to the data bus. The data latch R may be latched, which will add the current data from memory to its existing value through an arithmetic adder, or it may be reset to zero. The result may be directed back to memory during write cycles through an inverting tri-state buffer. To execute instructions, the control logic outputs a 7-step sequence of control signals, one for each clock cycle. A set of D flip-flops, called ES, is configured to keep track of the current execution state. Each instruction is 4 bytes long, consisting of two two-byte full addresses. The most significant bit of the segment number of the last address is used to distinguish between two different instruction types.

The execution sequence starts by resetting latches R and S, and selecting PC. PC's least significant bit is set to 0. This will generate the address of the current instruction's first byte, which the memory will now load and place on the data bus. In the next step, this byte is latched to A while PC is still selected. PC's least significant bit is then set to 1. The memory will load the second instruction byte, which is next latched to S while A is selected. PC is increased. The address that is now passed to memory is no longer the address of the instruction itself, but the pointer that was specified as its first argument. The memory will load the value at this address, which is next latched to the accumulator. Since latch R has been reset and initially contains zero, nothing will be added to the value before it is latched this time. This sequence is then repeated to fetch the next two bytes of the instruction and load the value pointed to by its second argument. At this point, the loaded value may or may not be latched to the accumulator to be added to the value of the previous argument, depending on the current instruction type. If it is latched, and if the arithmetic addition operation overflows and returns a carry signal, the instruction will branch and have PC skip to the instruction in the beginning of the next instruction block. In any case, the inverse of the resulting accumulator value is finally written to memory at the same address as the last instruction argument.