{sample: true}

About the Author

I was born in Nairobi, Kenya the son of missionaries. I spent most of my childhood in Kenya and Tanzania until I graduated from Rift Valley Academy and returned to the US for college. I graduated from Geneva College in 1978 with a double major in Physics and Computer science. I was very fortunate to get a job with IBM in Endicott N.Y. where I worked as a Computer Engineer testing the design of mid-range mainframes based on the 370 Architecture. A large part of my job involved writing software which verified that the hardware worked according to specifications.

The intended audience for this publication are people who understand both C++ and Assembler language programming. Familiarity with the Architectures of various CPUs from Intel, AMD, IBM, ARM, Sun, etc. also helps. But I also desire to use this document as an educational tool for people studying Computer Science at a College level, as well as people who are self taught. I will do my best to explain new ideas as they come up in future chapters. For now if I use unfamiliar terms, please be patient, I'll explain them later.

I was involved in a project which used the 801 RISC processor developed by the research division of IBM. The system presented two architectures to the user, the RISC 801 and the IBM 370. After a long time in development the system was finally announced and sold as the IBM 9370. While it could have run in "native mode" using a Unix like operating system, that capability was never announced or sold. The 9370 did not do well in the marketplace, but eventually it was used as the platform for the AS/400 system which was very successful. See this Wikipedia article: IBM 9370.

I was responsible to write the specification and design the software that would run self-diagnosing test cases to test the 801 Architecture. These were designed to set up initial conditions, execute a few instructions, predict the results and then compare actual results to those predicted. I called the software AVIS (Architecture Verification for Iliad Systems). The code name "Iliad" was used before it was changed to "Fort Knox". (See link above). The software was a micro Operating System written in 801 Assembly language. My job also involved testing the I/O system and my responsibility was to test the various telecommunication protocols. I had a bad habit of overselling my abilities and over-promising results and deadlines. I guess I was "Scotty" (from Star Trek) in reverse.

In 1986 I resigned from IBM and went to Seminary to become a pastor. Some saw it as a mid-life crisis, but I considered it a new calling. I will never regret that decision, but I will say it increased, rather than decreased my stress level. Yet overall it was incredibly rewarding to be able to help other people spiritually as they went through times of crisis. I was a bi-vocational pastor and helped to support myself through free-lance programming, substitute teaching and working in the mental health field. At the time of this writing, I am often substitute teaching in Special Education classes. I work with students with Downs Syndrome, Autism, and various kinds of emotional needs because of abuse. I also enjoy volunteering at the City Mission, doing Bible studies with men in the drug and alcohol rehabilitation program.

But I never lost my passion for trying to learn about computer architectures. I can't remember when I started writing ideas in notebooks for my own RISC architecture. What started out as "doodling", morphed into a hobby, then became something I actually wanted to publish. I was studying the C++ programming language and mentally "compiling" various operators, control structures, and data structures into the X-86 Instruction Set Architecture which I was also learning at the time. I will go into the details later, but I basically wanted to come up with a simple RISC architecture which would be a closer fit to the C++ language. I named it RISC++. I want to make it clear from the outset that **any** High Level Language can be compiled to the RISC++ machine code. C++ is merely the model used to define the instructions and data types. I will use examples of C++ code and show the resulting RISC++ Assembler language that would be created to support those structures.

I have been writing and re-writing RISC++ for over 20 years, all the time studying advances in "real" architectures and trying not to be obsolete before I publish it. Thomas Edison is quoted as saying that all his "failed" experiments were not failures at all. He just found 2000 ways to **NOT** make a light bulb, before he found the one that worked. I do not claim to have found the "perfect light bulb". But I do think this document itself, can be an effective educational tool to learn about the history of computer architectures from the 1980's to the present. It also shows how various High Level Languages can be

compiled into simpler, more efficient machine language code than in current architectures.

In my semi-retirement I continue to work as a part-time substitute teacher, who loves gardening, hiking, but most of all playing with my grandchildren. I am a Star Trek fan and I like to binge-watch all the series on Netflix. My favorite is "Enterprise" and my second favorite is "Deep Space Nine", but I like them all. I enjoy board games, but I'm not as much of a fanatic as my wife. For music, I enjoy the "oldies but goodies", like Simon and Garfunkle, Kenny Rogers, Johnny Cash, John Denver, Peter Paul, and Mary, but also Scott Joplin and Classical. But my favorite music of all is the old Hymns. Basically I'm a pastor/geek who is still stuck in the sixties.

At the time of the publication of this book, there is no hardware which implements the RISC++ Architecture. I claim the ideas in this book as my personal intellectual property. I ask that the copyright be honored and that I be given credit for any of the ideas that are original to me. Part of me hopes that some of these ideas may be patented and even make me some money. But at the present time I rely on the copyright protection of my ideas. I will also make every effort to credit the works of others, and I ask that I be credited with my ideas. I want this document to be used for educational purposes and I want the ideas to be as wide spread as possible. In the end, it all comes down to trust.

I hope to stimulate discussion and get ideas for improving the design. Many of the ideas have come through my own study using internet resources. I will do my best to provide internet links or printed material where people can study these ideas on their own. I use Wikipedia extensively, but I caution that these articles cannot necessarily be considered authoritative, but only as a starting point for further study. After reading this book and following the links, may I suggest that you make a donation to Wikipedia for providing such a rich set of relevant online information.

I would welcome comments and suggestions from experts in various fields. I would greatly appreciate input from people who design compilers for various languages such as C++. I would also welcome comments from hardware designers, from operating system experts, and from people who design critical performance software. I am not an expert in any of those fields, but I have spent a great deal of time studying various computer architectures both at the ISA level and the implementations in hardware. I have built on top of what others have done and tried to streamline and pick the best features.

I value the input of others, and I want these ideas discussed and evaluated. Please be kind. This is my baby.