A UNIFICATION OF SOFTWARE AND HARDWARE; A NEW TOOL FOR HUMAN THOUGHT

Glen B. Haydon
WISC Technologies, Inc.
La Honda, CA 94020

The following discussion briefly develops a philosophical basis with which to unify the hardware and software tools of a computer development system. The result is an improved match between software and hardware.

The nature of the human mind and thought processes are not understood. However, there appears to be a mismatch between human thought and the rapidly growing use of computers as tools to help men think. Software engineers and hardware engineers seem to be working in different directions. If we could unify the software and hardware of computers along new lines, we might find a better tool to aid us in our intellectual endeavours. Perhaps a unification of software and hardware would provide a better model to simulate part of the activities of the human brain.

Origins of Language

The development of speech and natural languages produced a tool for the development of human thought. In an interesting paper by James Cooke Brown and William Greenhood entitled "PATERNITY, JOKES AND SONG: A POSSIBLE EVOLUTIONARY SCENARIO FOR THE ORIGINS OF MIND AND LANGUAGE " , (*Cultural Futures Research*, Vol VIII, No.2, Winter 1983/84), a new perspective to the development of natural languages is presented. The paper is a long one and carefully argued with many references.

The origins begin with the development of speech as a tool for communication. Along with communication has come the internal activity of the mind, thinking. In the development of language, the burden of disambiguation grows geometrically with every increase in sentence length. The development of grammar attempts to accomplish the disambiguation.

A Logical Language - LOGLAN

In his FORWARD to *LOGLAN 1: A LOGICAL LANGUAGE* , 3rd Ed. (The Loglan Institute, Inc. 1975, 1701 Northeast 75th Street, Gainesville, FL 32601) James Cook Brown begins:

"At the beginning of the Christmas Holidays, 1955, I sat down before a bright fire to commence what I hoped would be a short paper on the possibility of testing the social psychological implications of the Sapir-Whorf hypothesis [relating lanugage to thought]. I meant to proceed by showing that the construction of a tiny model language, with a grammar borrowed from the rules of modern logic, taught to subjects of different nationalities, in a laboratory setting, under conditions of control, would permit a decisive test. I have been writing appendices for that paper ever since. ... "

And now, over thirty years later, the appendices continue to develop. The language became known as LOGLAN. It was described in the literature, in the June 1960 issue of *Scientific American*. Books and publications have continued over the years. Within the past 5 years the language has been refined with a completely unambiguous machine parsable grammar. Currently, a number of minor revisions to the language are being summarized and a new publication should be forthcoming before long.

History of Computing

Several years ago, Hans Nieuwenhuyzen called my attention to two books. The first was *A HISTORY OF COMPUTING IN THE TWENTIETH CENTURY*, (N. Metropolis, J. Howlet and Gian-Carlo Rota, Editors, 1980 Academic Press.) Computers have changed with time. Originally, von Neumann thought of the computer as a number cruncher. Perhaps it was Turing who showed that computers can be symbol-manipulating machines. The hardware design of computers started from these perspectives. Early programming languages dealt with methods trying to use the newly developed hardware to solve real problems.

The second book was *HISTORY OF PROGRAMMING LANGUAGES*, (Richard L Wexelblat, Editor, 1981, Academic Press). The history traces the development of many languages to bridge the gap between real problems and the tools provided with computer hardware. The computer language, FORTRAN was developed as a numerical scientific number cruncher and continues to this day as a major programming language for scientific computation. Other languages which immediately followed were also number crunchers. These were batch processing languages. On-line languages were devised nearly a decade later.

Business applications with number storage and crunching came later. The introduction of string and list processing followed. It was always a problem to make the newer application requirements fit on hardware designed for number crunching. At best, the fit has not been optimal.

Thus the problems addressed with computer hardware expanded from number crunching to assisting in other areas of human thinking and problem solving. As software engineers developed languages, the importance of a divide and conquer approach became apparent. Structured programming became the tool of software engineers. Libraries of program modules were developed. However, the hardware techniques of number crunching do not lend themselves to efficient execution of structured programs requiring sequences of subroutine calls to a variety of modules.

Progress in Hardware Design

In conjunction with the developing languages, the hardware engineers made great strides to support the computational applications addressed by the early languages. The hardware design has been oriented to improving the speed of execution of sequential operations.

In hardware development there has been a trade off between the speed and semantic content of the operations and the physical limitations of the speed of memory access. The increased complexity of instructions increased semantic content of each operation, but with many operations taking many machine cycles. Other techniques have been developed to increase the speed of memory access.

In an alternate approach to increasing hardware speed, hardware designers have tried to reduce the number of operations with each instruction, each of which would then require

only a single processing cycle. Many registers are used rather than slower machine memory to further increase speed.

In the course of these hardware engineering efforts, little attention has been given to efficient subroutine calls.

Progress in Software Design

Software designs have taken other directions. Compilers were developed to translate the newer languages to the machine language of the hardware. Modern language optimizing compilers have many different ways of handling subroutine calls. Not infrequently, when speed is required, the subroutine is simply duplicated in line. Though longer, such machine code will run faster.

Compilation is essentially a batch process. Often multiple passes through the source code are required. Batch processes are slow. A program needs to be completely recompiled to test it. It used to be that such batch programs took overnight to run. Compilers have been designed to run ever faster, but they still require minutes to process. Program development is inhibited by the slow turn-around of batch processing.

With structured programming, it would be desirable to have an instantaneous turn-around on tests of new procedures as they are written. A software development system should also have instantaneous turn-around on tests of connected structures in building the final program. The conventional development systems requiring a compile, load and go for each test is not conducive to good software development.

The Hardware-Software Mismatch

The sequential methods of hardware design are mismatched with structured programming. Sequential methods are also a mismatch with the thought processes of the software developer. The process is almost a random jumping of ideas in the process of thinking. Structured programming seems to be better matched with the thinking process. As such it provides a tool for simulation and study of thought processes. For example: What are the differences between left and right hemisphere processes?

Computer software is divided into smaller and smaller procedures. The process is similar to the divide and conquer process of problem solving. As programs are written, regardless of the language used, they tend to follow a process of natural thought. A translator is required to take a programming language following thought processes and structured programming, and produce machine code which can be run inefficiently on hardware designed to run sequentially.

Unification of Hardware and Software

A rethinking of the hardware design is necessary to better match the direction of software development. Rather than sequential efficiency, what is needed is subroutine call efficiency. It would be ideal if subroutine calls could come for free. This is one of the results of the ideas presented in Phil Koopman Jr's invited paper at this conference. Some of those ideas are summarized here.

Stack oriented Machines

Samelson and Bauer described an ALGOL translator using multiple stacks. (See *A HISTORY OF COMPUTING IN THE TWENTIETH CENTURY* referred to above.) Though a US patent was issued on a full wiring diagram, no hardware was built. At the time, they turned to implementing their ideas in software. Prior to the recent work of Phil Koopman Jr, hardware designers of general purpose processors have not adopted the stack concepts in developing hardware better suited to structured programming.

It is time to adopt the proposals of Samelson and Bauer. An efficient multiple hardware stack machine will contribute to a functional unification of hardware and software. Such a hardware design provides for subroutine calls with no cost in processor time. It contrasts dramatically with the time penalty for subroutine calls.

Writable Control Store

Machine operations should have the semantic content optimized according to the specific requirements of new applications in the software development process. This can be done by using software control of hardware components with writable control store machines. The process divides the hardware components into smaller pieces and allows the software engineer to assemble their functions into optimal operations according to the application requirements.

In the history of computers, writable control structures have been used. Bit slice technology with writable instructions are available but have not been widely exploited.

A Unified Design

A rethinking of hardware design, has led to a writable instruction set computer (WISC) interfaced with multiple dedicate hardware stacks as proposed by Samelson and Bauer.

The first results of such a rethinking of hardware design were presented and discussed at the 1986 Rochester Forth Conference by Phil Koopman Jr and Glen B. Haydon. The design was available then as a wire-wrapped kit. The design is now available on a pair of printed circuit boards.

Also at the 1986 Rochester Forth Conference, Phil Koopman Jr demonstrated the operation of his initial design of an enhanced system. During the past year the design has undergone several itterations. At this, the 1987 Rochester Forth Conference, Phil Koopman Jr is presenting an invited paper in which he details his concepts of the problems and implementation of a hardware design to solve the problems.

Conclusions

I have endeavored to review some of the more philosophical ideas leading to a better match between the computer tools available and the human thought processes. The result has been a unification of structured programming of software engineering with the necessary hardware to run such software efficiently.

To me, one of the greatest potential powers of modern computers is the ability to simulate problems. Perhaps the unification of software and hardware will provide an improved tool to better understand man's way of thinking and problem solving.