The von Neumann Architecture of Computer Systems

H. Norton Riley

Computer Science Department
California State Polytechnic University
Pomona, California

September, 1987

Any discussion of computer architectures, of how computers and computer systems are organized, designed, and implemented, inevitably makes reference to the "von Neumann architecture" as a basis for comparison. And of course this is so, since virtually every electronic computer ever built has been rooted in this architecture. The name applied to it comes from John von Neumann, who as author of two papers in 1945 [Goldstine and von Neumann 1963, von Neumann 1981] and coauthor of a third paper in 1946 [Burks, et al. 1963] was the first to spell out the requirements for a general purpose electronic computer. The 1946 paper, written with Arthur W. Burks and Hermann H. Goldstine, was titled "Preliminary Discussion of the Logical Design of an Electronic Computing Instrument," and the ideas in it were to have a profound impact on the subsequent development of such machines.

Von Neumann's design led eventually to the construction of the EDVAC computer in 1952. However, the first computer of this type to be actually constructed and operated was the Manchester Mark I, designed and built at Manchester University in England [Siewiorek, et al. 1982]. It ran its first program in 1948, executing it out of its 96 word memory. It executed an instruction in 1.2 milliseconds, which must have seemed phenomenal at the time. Using today's popular "MIPS" terminology (millions of instructions per second), it would be rated at .00083 MIPS. By contrast, some current supercomputers are rated at in excess of 1000 MIPS. And yet, these computers, such as the Cray systems and the Control Data Cyber 200 models, are still tied to the von Neumann architecture to a large extent.

Over the years, a number of computers have been claimed to be "non-von Neumann," and many have been at least partially so. More and more emphasis is being put on the necessity for breaking away from this traditional architecture in order to achieve more usable and more productive systems. The expectations for the fifth generation systems seem to require that substantially new architectures be evolved, and that both hardware and software be freed from the limitations of the von Neumann architecture [Sharp 1985].

We all know what the von Neumann architecture is, of course. At least we have strong intuitive feelings about it because this is what we have always used. This is "the way computers work." But to really comprehend what choices there are for computer designers, to appreciate what new choices must be found, it is necessary to have a more definitive understanding of what the von Neumann architecture is and is not and what its implications are.

Von Neumann begins his "Preliminary Discussion" with a broad description of the general-purpose computing machine containing four main "organs." These are identified as relating to arithmetic, memory, control, and connection with the human operator. In other words, the arithmetic logic unit, the control unit, the memory, and the input-output devices that we see in the classical model of what a computer "looks like."

To von Neumann, the key to building a general purpose device was in its ability to store not only its data and the intermediate results of computation, but also to store the instructions, or orders, that brought about the computation. In a special purpose machine the computational procedure could be part of the hardware. In a general purpose one the instructions must be as changeable as the numbers they acted upon. Therefore, why not encode the instructions into numeric form and store instructions and data in the same memory? This

frequently is viewed as the principal contribution provided by von Neumann's insight into the nature of what a computer should be.

He then defined the control organ as that which would automatically execute the coded instructions stored in memory. Interestingly he says that the orders and data can reside in the same memory "if the machine can in some fashion distinguish a number from an order" [Burks, et al., p. 35].

And yet, there is no distinction between the two in memory. The control counter (what we now usually call the program counter) contains the address of the next instruction, and that word is fetched to be executed. Whatever the control unit "believes" to be an order or to be data is treated as such. One ramification of this is that the instructions can operate upon other instructions, producing a self-modifying program. This has not been considered good form for many years, because of the implications for program debugging and the desire for reentrant code in some situations. It is possible that new developments in artificial intelligence may bring fresh attention to the possibilites afforded by this characteristic [Bishop 1986].

Von Neumann devoted most of his "Preliminary Discussion" to the design of the arithmetic unit. The details of this are the least interesting part of the paper from the standpoint of the organization of his computer, and its influence on future developments. The capabilities of the arithmetic unit were limited to the performance of some arbitrary subset of the possible arithmetic operations. He observes that "the inner economy of the arithmetic unit is determined by a compromise between the desire for speed of operation...and the desire for simplicity, or cheapness, of the machine" [Burks, et al., p. 36]. What is interesting, and important, is that this issue continued to dominate design decisions for many years. It is less true now that hardware costs have become a substantially less critical concern.

The concepts put forth by von Neumann were, for their time, quite remarkable--so much so that they provided the foundations for all of the early computers developed, and for the most part are still with us today. Why then do we require something different? What is it about this architecture that we find constraining and counterproductive? Why must there be new, different, "non-von Neumann" machines?

Myers [1982] defines four properties that characterize the von Neumann architecture, all of which he feels are antithetical to today's needs. One of these was discussed above, that is the fact that instructions and data are distinguished only implicitly through usage. As he points out, the higher level languages currently used for programming make a clear distinction between the instructions and the data and have no provision for executing data or using instructions as data.

A second property is that the memory is a single memory, sequentially addressed. A third, which is really a consequence of the previous property, is that the memory is one-dimensional. Again, these are in conflict with our programming languages. Most of the resulting program, therefore, is generated to provide for the mapping of multidimensional data onto the one dimensioned memory and to contend with the placement of all of the data into the same memory.

Finally, the fourth property is that the meaning of the data is not stored with it. In other words, it is not possible to tell by looking at a set of bits whether that set of bits represents an integer, a floating point number or a character string. In a higher level language, we associate such a meaning with the data, and expect a generic operation to take on a meaning determined by the meaning of its operands.

In characterizing the difficulties presented by these four properties, their inconsistencies with higher level languages are emphasized. And yet, these higher level languages were, for the most part, designed to be utilized for the purpose of programming the existing von Neumann style computers. In a very real sense, the entire realm of software has been developed under the umbrella of this architecture and may be aptly referred to as von Neumann software. Thus, the hardware and software have served to perpetuate each other according to the underlying von Neumann model.

One facet of this is the fundamental view of memory as a "word at a time" kind of device. A word is transferred from memory to the CPU or from the CPU to memory. All of the data, the names (locations) of the data, the operations to be performed on the data, must travel between memory and CPU a word at a time. Backus [1978] calls this the "von Neumann bottleneck." As he points out, this bottleneck is not only a physical limitation, but has served also as an "intellectual bottleneck" in limiting the way we think about computation and how to program it.

Obviously, the computers we use today are not simply larger, faster EDVACs. Numerous improvements have been made through the introduction of, for example: index registers and general purpose registers; floating point data representation; indirect addressing; hardware interrupts to redirect program execution upon detection of certain events; input and output in parallel with CPU execution; virtual memory; and the use of multiple processors [Myers 1982]. It is significant that each of these made some improvement over the original model, some of them quite significant improvements, but none fundamentally changed the architecture. In other words, the four properties discussed above still hold, and the von Neumann bottleneck still exists with or without these improvements. It is also significant to note that all of these improvements were implemented before 1960!

Some other improvements can be noted that really have no effect on the architecture at all. For example, cache memory provides a way of making block transfers (still a word at a time) into a smaller, faster memory device. It is an improvement in the implementation, not in the architecture. A similar view holds for the use of multiple memory modules, with parallel transfers from more than one module. All this does is effectively increase the word length, while still functioning in a word at a time mode [Stone 1987]. Again, many such improvements were originally implemented during the 1950s and 1960s.

So what would characterize a "non-von Neumann" machine? Some examples will illustrate.

One is what Myers calls "self-identifying data," or what McKeeman [1967] calls "typed storage." In the von Neumann computer, the instructions themselves must determine whether a set of bits is operated upon as an integer, real, character, or other data type. With typed storage, each operand carries with it in memory some bits to identify its type. Then the computer needs only one ADD operation, for example, (which is all we see in a higher level language), and the hardware determines whether to perform an integer add, floating point add, double precision, complex, or whatever it might be. More expensive hardware, to be sure, but greatly simplified (and shorter) programs. McKeeman first proposed such "language directed" design in 1961. Some computers have taken steps in this direction of high-level language architecture, becoming "slightly" non-von Neumann.

Another approach aims at avoiding the von Neumann bottleneck by the use of programs that operate on structures or conceptual units rather than on words. Functions are defined without naming any data, then these functions are combined to produce a program. Such a functional approach began with LISP (1961), but had to be forced into a conventional hardware-software environment. New functional programming architectures may be developed from the ground up [Backus 1978, Eisenbach 1987].

A third proposal aims at replacing the notion of defining computation in terms of a sequence of discrete operations [Sharp 1985]. This model, deeply rooted in the von Neumann tradition, sees a program in terms of an orderly execution of instructions as set forth by the program. The programmer defines the order in which operations will take place, and the program counter follows this order as the control executes the instructions. This "control flow" approach would be replaced by a "data flow" model in which the operations are executed in an order resulting only from the interdependencies of the data. This is a newer idea, dating only from the early 1970s.

Our appetite for computing power is insatiable. Standing on the threshold of the fifth generation, we clearly expect more from future computers than just more speed. We have come this far on the basis of enhancements

to the existing architectures and their implementation. The use of smaller, cheaper, and faster components is coupled with greater and greater parallelism. The most difficult task connected with adopting new architectures is that it is hard to think about them using our von Neumann oriented minds.

Two final points should be made. The first is a reiteration of what was said earlier: that it is no less to von Neumann's credit that we now find his design inadequate. The persistence of this architecture may be taken as ample testimony to that. Second, it is no less to von Neumann's credit that such credit must fairly be shared with some other pioneers. In particular, Arthur Burks, J. Presper Eckert, Hermann Goldstine, and John Mauchly, along with numerous others, contributed to the creation of the first general purpose, stored program, electronic digital computer. Von Neumann received the principal credit because he took the time to document the ideas, to elaborate the concepts, to instruct the rest of the world about them. He also received credit because the substance of his reputation gave the greater weight to his words. That seems fair enough.

References

Backus, J. 1978. Can programming be liberated from the von Neumann style? A functional style and its algebra of programs. *Communications of the ACM 21*, 8, (August), 613-641.

Bishop, P. 1986. *Fifth Generation Computers*. Ellis Horwood Limited, Chichester, England.

Burks, A. W., Goldstine, H. H., and von Neumann, J. 1963. Preliminary discussion of the logical design of an electronic computing instrument. In Taub, A. H., editor, *John von Neumann Collected Works*, The Macmillan Co., New York, Volume V, 34-79.

Eisenbach, S. 1987. *Functional Programming: Languages, Tools, and Architectures*. Ellis Horwood Limited, Chichester, England.

Goldstine, H. H. and von Neumann, J. On the principles of large scale computing machines. In Taub, A. H., editor, *John von Neumann Collected Works*, The Macmillan Co., New York, Volume V, 1-32.

McKeeman, W. M. 1967. Language directed computer design. In *Proceedings of the 1967 Fall Joint Computer Conference.* Thompson Publications, Washington, 413-417.

Myers, G. J. 1982. *Advances in Computer Architecture.* John Wiley & Sons, New York.

Sharp, J. A. 1985. *Data Flow Computing.* Ellis Horwood Limited, Chichester, England.

Siewiorek, D. P., Bell, G. C., and Newell, A. 1982. *Computer Structures: Principles and Examples.* McGraw-Hill Book Company, New York.

Stone, H. S. 1987. *High-Performance Computer Architecture.* Addison-Wesley Publishing Company, Reading, Massachusetts.

Von Neumann, J. 1981. First draft of a report on the EDVAC." In Stern, N. *From ENIAC to Univac: An Appraisal of the Eckert-Mauchly Computers.* Digital Press, Bedford, Massachusetts.