RETROSPECTIVE:

What Have We Learned from the PDP-11 — What We Have Learned from VAX and Alpha

Gorden Bell

Senior Researcher
Microsoft Corp., Bay Area Research Center
San Francisco, CA

W.D. Strecker

Sr. VP and Chief Technology Officer
Digital Equipment Corporation,
Maynard, MA

Introduction

The PDP-11, VAX-11 (usually just VAX), and Alpha have been the strategic computer hardware architectures of Digital Equipment Corporation (DEC) from the early 1970's to the present. Although it would be a stretch to consider them variants of a single computer architecture, there are enough common properties in the architectures themselves and in the major software systems supporting the architectures to consider them members of an architecture family.

Our paper "Computer Structures: What Have We Learned from the PDP-11" [1] was written at the time the VAX architecture was being developed, and the learning reported in that paper would strongly influence the design of the VAX architecture.

In this retrospective, we will review how the PDP-11 learning influenced VAX. Next we will discuss what we learned from the VAX architecture. Finally we will discuss the design of the Alpha architecture and how its design and entry into the market resulted not only from VAX learning, but also from environmental factors inside DEC.

PDP-11

The PDP-11 is a CISC architecture with a 16-bit virtual address. The first PDP-11 implementation – the PDP-11/20 – was introduced in 1969. In [2] there is a detailed discussion of the goals and constraints for the design of the PDP-11. Deliberately oversimplifying, these include: (1) provide the ability to build processors with a wide range of performance and function, (2) provide efficient (8-bit) byte processing, (3) provide a flexible, compiler friendly programming model, and (4) provide a flexible I/O structure.

The PDP-11 architecture is a general register design (8 16-bit registers) with the program counter and stack pointer located in the general registers. An elegant set of register-based memory addressing modes combined with the general register structure to produce an architecture that can be programmed as a stack machine, a general register machine, or a memory-to-memory machine. Memory is addressable to the (8-bit) byte and the conditional branch mechanism is based on condition codes. I/O is handled by providing I/O device registers with memory addresses: the registers can then be manipulated by ordinary instructions.

The PDP-11 was a major commercial success, providing the majority of DEC's growth, revenues and profits from the early 1970's to the early 1980's. Also, the PDP-11 significantly influenced computer architecture with its elegant addressing modes and its I/O structure.

The PDP-11 architecture proved to have two real limitations. The first was the 16-bit virtual address space. This will be discussed in the next section. The second was the instruction set and the instruction set encoding. The original PDP-11 had operations to move, add, subtract, compare, and conditional branch on 8- and 16-bit integers. These operations together with the addressing modes were encoded in such a way to effectively exhaust the code space of the PDP-11 instruction format.

This situation made it impossible to compatibly extend the PDP-11 with any consistency or efficiency. The addressing modes could not reasonably be extended or redefined to support a greater than 16-bit virtual address. It was impossible to efficiently add additional instructions in a manner architecturally consistent with the basic instruction set. When certain additional instructions and other capabilities were needed to meet market requirements (e.g. extended integer arithmetic, floating point, and memory management) they were added as implementation specific options and often weren't compatible across implementations. The result of all this was that the PDP-11 was not compiler friendly (given the state of DEC and industry compiler technology in the 1970's). Most PDP-11 language processors were

either (1) interpreters or (2) compilers that compiled to an intermediate form that was interpreted at run time.

Far surpassing the lack of architectural control and consistency in PDP-11 hardware implementations was the state of PDP-11 software. Depending on how one counts, there were about 4 operating system families with about 10 named variants. These operating systems supported an arbitrary variety of sometimes incompatible language processor, data management, and transaction processing software. It was understood that this situation would be completely unmanageable and could not be afforded for the future.

From the PDP-11 to VAX

The Bell and Strecker paper [1] has often been quoted because of this statement:

"There is only one mistake that can be made in a computer design that is difficult to recover from – not providing enough address bits for memory addressing and memory management. The PDP-11 followed the unbroken tradition of nearly every known computer. Of course, there is a fundamental rule of computer (and perhaps other) designs that helps to alleviate this problem: any well-designed machine can be evolved through at least one major change. It is extremely embarrassing that the PDP-11 had to be evolved with memory management only two years after the paper was written outlining the goal of providing increased address space. All predecessor DEC designs have suffered the same problem and only the PDP-10 evolved over a ten year period before a change was made to increase its address space."

By 1975, the PDP-11's 16-bit virtual address had become a real limit for applications. (Various approaches had been used in the PDP-11 implementations to extend the physical address space to beyond 2^{16} bytes, but they did not solve the application problem.)

Moore's Law [3] predicts that DRAM chip capacity increases 4 times every 3 years. Thus, if memory chip prices are constant, and if users pay a constant amount for computers, then the number of address bits needed to address a constant price memory will grow by one address bit every 18 months.

If a 16-bit address was reaching its limit in 1975, then one could determine the likely lifetime of any address size expansion. We defined the PDP-11's successor — VAX — to have a virtual address of 32 bits. Thus we concluded that the VAX architecture – based upon the above model that the only fundamental limitation on architecture lifetime is addressing – should comfortably last about 24 years: until 1999. It would turn out that the limit on the VAX architecture lifetime wasn't the size of the virtual address.

VAX

The VAX is a CISC architecture with a 32-bit virtual address. The first VAX implementation – the VAX-11/780 – was introduced in 1978 [4]. The design of VAX was started in 1975. The overarching goal was to produce a 'compatible' extension of the PDP-11 that would solve the virtual address space limitation of the PDP-11 (the name VAX-11 is derived from 'Virtual Address eXtension of the PDP-11). The principal constraint on the design of VAX was that – despite the doubled virtual address size – VAX code would be no bigger than equivalent PDP-11 code.

A strong goal was to eliminate the chaos of the PDP-11 software. This was approached in three ways. (1) VAX was to have a single strategic operating system — VMS — with real-time, time-sharing, and transaction processing capabilities. (2) To make the VAX compiler friendly (again in the context of DEC's mid-1970's compiler technology), an extreme focus was placed on instruction set completeness and regularity. (3) The VAX software environment was to be based on the model that any software can 'call' any other software. To strongly motivate software developers to follow this model, VAX defined a number of 'software' data types (i.e. subroutine stack frames, queues, variable length bit fields, character strings, and decimal strings) and provided instruction support for these data types. 'Software' is used in the sense that most applications would see little performance degradation if the data types were implemented in software.

Given the code size constraint and the limitations discussed above on extending the PDP-11 instruction set, the VAX instruction format is not a superset of the PDP-11 instruction format. Instead a new instruction format was designed for VAX and formal PDP-11 compatibility was provided by a tightly integrated PDP-11 compatibility mode that allowed execution of PDP-11 instructions in the VAX virtual address space.

The VAX architecture has the same general structure as the PDP-11 – general registers (extended to 16 32-bit registers), with the program counter and stack pointer located in the register set, a rich set of register-based addressing modes (extended to include scaled indexing and an efficient encoding of literals). VAX also has the same data types, condition codes, and byte addressing as the PDP-11.

As introduced above, VAX extended the PDP-11 by defining new data types – queues, variable length bit fields, subroutine stack frames, character strings, and decimal strings – and a complete set of instructions to operate on these data types.

To achieve the code size constraint, VAX defined an extremely space efficient method of encoding instructions. Instructions are provided in multiple forms with implicit and explicit operands. For example, the 32-bit integer ADD instruction is provided in the following forms:

```
increment   A       ; add 1 to A
add         A, B    ; add A to B
add         A, B, C ; add A to B and
                      store the sum in C
```

Every explicit instruction operand (A, B, and C in the previous example) is specified in a general way using any of the VAX addressing modes. This leads to the VAX instruction format of

[*opcode, operand-1-specifier, ..., operand-n-specifier*]

where *n* is the number of explicit operands. VAX instructions were defined with 0-6 explicit operands.

VAX was a huge commercial success. VAX provided the majority of DEC's growth, revenues and profits from the early 1980's to the early 1990's. The success of VAX was clearly inseparable from the VMS operating system. VMS was a true 32-bit virtual memory operating system that performed well from its first release. VMS embedded DECnet such that it was transparent to applications whether any file or other I/O operation was local or over the network. On top of VMS was a highly integrated (and network-transparent) software set including multiple language compilers, a database (RDB), transaction processing (ACMS), and an 'integrated office' environment (ALL-IN-1). VMS invented and first implemented the now-pervasive concept of clusters [5,6].

When it was clear that the market was responding to VAX and VMS, DEC moved to VAX and VMS as its sole computer system strategy [7]. All other DEC systems were put in niche roles or moved to what was essentially a maintenance mode.

It is notable that the VAX architecture remained essentially unchanged over the last 20 years. The only material addition was the early adding of 2 floating-point data types. The only other material change was to define permissible subsets [8] of the architecture. (The MicroVAX architecture was such a subset.) Processors implementing VAX subsets generate sufficient state on encountering an instruction not in the subset to enable transparent, efficient software interpretation of the missing instruction.

A retrospective on VAX must reflect a strong sense of time. The VAX embodiment of the goals and constraints of PDP-11 compatibility, the code size constraint, instruction set completeness and regularity, and hardware support for 'software' data types was absolutely key to moving from the PDP-11 (and other DEC architectures) to the hugely successful VAX and VMS business. However, once that success was achieved, the VAX architecture carried a complexity burden that would make it particularly vulnerable to the RISC concept.

From VAX to Alpha

In 1980, Patterson [9] discussed the modern RISC architecture concept. By the mid-1980's, there was a general consensus in DEC that for a given amount of CPU logic in a given technology, a RISC processor could achieve (at least) twice the performance of a CISC processor. There was no consensus, however, on what to do about this.

There were two rational strategic responses to the RISC challenge:

1. Since the RISC advantage would 'only' be a factor of 2, DEC could 'tough it out' until the limits of a 32-bit address space would force a new architecture (predicted, as discussed above, to be about 1999). To 'tough it out' would be (1) to focus on the most aggressive possible microprocessor implementations of VAX, (2) to use multiprocessing and clustering to achieve performance, and (3) to accept limited success in some market segments where (1) and (2) would be marginal (e.g. UNIX workstations). Effectively, this was the type of strategy successfully employed by IBM for the '360' architecture and Intel for the 'x86' architecture – the other two important CISC architectures with a large customer base.

2. Since CISC was going to 'lose' by at least a factor of 2 to RISC, it was essential to embrace RISC ASAP, define a new or use an already defined RISC architecture, and get products to the market in a timely manner. Especially important, it was necessary to get DEC's then strategic software system – VMS – ported to the RISC architecture (or perhaps, even better, made processor independent, and sold industry-wide). Effectively, this was the type of strategy successfully employed by Sun in moving from the '68000' to SPARC.

As we discussed in the last section, DEC's VAX business was a huge success, and it was very profitable. At this time, DEC's senior leadership was operating under the philosophy best captured by the two phrases 'if some strategy is good, less strategy is better,' and 'if some internal competition is good, more internal competition is better.'

As a result, exploiting the considerable profits of the VAX business, an overwhelming array of internal projects in processor technology, VAX and RISC architectures, and operating systems were launched. During the second half of the 1980's, major projects were undertaken in various combinations of 3 different ECL gate array technologies, high performance multichip packaging, advanced

custom CMOS, 3 internally developed and one externally developed (MIPS) 32-bit RISC architectures, a 64-bit RISC architecture (Alpha), multiple system and I/O busses, a new UNIX operating system, and two new proprietary operating systems! Knowing that all these projects could not possibly be successful, DEC's product development organization was locked in internecine warfare.

By the end of the 1980's DEC had essentially lost control of its system strategy. It wasn't explainable or affordable, and remarkably still hadn't done everything necessary to successfully implement either of the two strategic alternatives discussed above. It wasn't until 1992/1993 that DEC changed its senior leadership and regained control of its system strategy.

Alpha

Alpha is a RISC architecture with a 64-bit virtual address [10]. The first Alpha implementation – the 21064 single chip microprocessor – was introduced in early 1992. Computer systems using the 21064 were introduced at the end of 1992.

The goals of the Alpha architecture design were high performance, longevity, support for running the VMS and UNIX operating systems, and support for existing VMS and UNIX applications. The goals of high performance and longevity were met by a RISC approach with extreme attention to details that might interfere with high-speed implementations, a 64-bit virtual address, and PALcode (to be discussed later). The goals of VMS/VAX and ULTRIX/MIPS (DEC's UNIX offering was called ULTRIX and ran on the MIPS architecture) application support were met with data type and addressing compatibility with VAX and (little-endian) MIPS, PALcode, and binary translation (discussed later).

Compared to VAX, the design of Alpha can be considered 'classic' RISC. There are 32 64-bit general-purpose registers and 32 64-bit floating point registers. All instructions are 32 bits in length. The programming model is load/store: the only memory operations are load from memory to register and store to memory from register. All other operations are between registers. The only data types are integer and floating point with VAX compatibility. The principal integer data type is 64 bits, and there is very limited instruction support for smaller integers. Memory is addressable to the byte but there are strong size and alignment constraints on memory accesses. There are no condition codes: conditional branches are based on testing the state of a register.

In addition to 'classic' RISC techniques, Alpha has some novel approaches for enabling high-speed implementations. For example, there is a very flexible approach to specifying and handling arithmetic exceptions. A conditional move instruction eliminates branches in certain instruction sequences. Certain instructions contain hints about branch targets and data prefetching.

PALcode (Privileged Architecture Library) provides a means of implementing the privileged architecture seen by an operating systems. Privileged architecture includes context switching, interrupts, exceptions, and memory management. In Alpha, PALcode is implemented with ordinary instructions running in physical memory, with interrupts off, and access to all machine state. The PALcode is tailored to the needs of each operating system (e.g. VMS, UNIX, and Windows NT).

Rather than hardware compatibility modes, binary translation is used to run VMS/VAX-based and ULTRIX/MIPS-based applications on Alpha. The binary translator takes, say, VMS/VAX-based executable code and compiles it to the extent possible to VMS/Alpha-based executable code. A runtime interpreter paired with an incremental compiler handles the portion of the code that cannot be initially compiled. During runtime interpretation, enough additional information and context is gathered to significantly extend the scope and optimization of the initial compilation.

Binary translation was very successful in executing applications from VAX and MIPS on Alpha. Recently it has been used successfully to execute Windows NT/x86 applications on Windows NT/Alpha.

Around Alpha a unified system strategy was developed. The strategy consisted of (1) an aggressive long-term road map for Alpha microprocessors, (2) a family of systems from workstations to large-scale multiprocessor systems using the Alpha microprocessor. (3) PCI bus-based I/O for all systems, and support by three operating systems: VMS (evolved from 32-bit to 64-bit support), UNIX (called DIGITAL UNIX: a new pure 64-bit operating system based on OSF technology) and Windows NT (provided by Microsoft). All other DEC systems were put in niche roles or moved to what was essentially a maintenance mode.

This strategy was well executed by DEC. The various Alpha microprocessors maintained a significant performance lead over competitive RISC and CISC microprocessors. The transition of VMS/VAX to VMS/Alpha and ULTRIX/MIPS to DIGITAL UNIX/Alpha went smoothly. In application areas – particularly databases - where 64-bit addressing could be exploited, Alpha performance and 64-bit DIGITAL UNIX functionality set the competitive benchmark.

Unfortunately, the strategy was late. By 1992/1993 *de facto* standards had been set by competitors for RISC and (32-bit) UNIX. Despite the

simplicity of the strategy, and the technical excellence, DEC would struggle to get product volumes adequate for a profitable systems business.

However, a new opportunity is emerging for Alpha. As the UNIX and the Windows NT market moves from 32 to 64 bits, Alpha is the only mature, high performance 64-bit RISC architecture with pure 64-bit UNIX and Windows NT support. A complete retrospective on Alpha awaits the industry 32- to 64-bit transition.

Summary

The PDP-11, VAX-11, and Alpha can be considered members of an architecture family starting in the 1960's and extending to the present.

The PDP-11 was a huge commercial success for DEC. The PDP-11 was the *de facto* standard 16-bit minicomputer in the 1970's. The basic PDP-11's design was extremely elegant and it significantly influenced future computer architecture. However, the PDP-11's 16-bit virtual address space and the inability to efficiently and consistently extend the architecture, led to its successor – VAX – being designed only 6 years after its introduction.

The VAX was similarly a huge commercial success for DEC. VAX and its closely related software system – VMS – became the *de facto* standard for 32-bit virtual memory networked computing in the 1980's. However, VAX, driven by its initial design goals and constraints, was a complex architecture, and was particularly challenged (internally and externally) by the RISC concept that competitively emerged in the mid-1980's.

DEC's internal situation in the second half of the 1980's made it impossible to achieve a timely, rational response to the RISC challenge. By the time the Alpha strategy emerged in 1992/1993, DEC had lost momentum in the market and other vendors had established *de facto* standards in RISC and UNIX. This situation would impact the commercial success of Alpha despite its superior technical attributes. However, the Alpha story awaits completion of the industry transition from 32 to 64 bits, starting – as we predicted in 1975 – in about 1999.

References

[1] G. Bell W. D. and Strecker, "Computer Structures: What Have We Learned from the PDP-11," *The 3rd Annual Symposium on Computer Architecture Conference Proceedings*, pp. 1-14, 1976.

[2] C. G. Bell, R. Cady, H. McFarland, B. Delagi, J. O'Laughlin, R. Noonan and W. Wulf, "A New Architecture for Mini-Computers - The DEC PDP-11," *Proceedings of the Sprint Joint Computer Conference*, pp. 657-675, AFIPS Press, 1970.

[3] R. R. Schaller, "Moore's Law: Past, Present, and Future," *IEEE Spectrum*, pp.52-59, June 1997.

[4] W. D. Strecker, "VAX-11/780: A Virtual Address Extension to the DEC PDP-11 Family," *Proceedings of the National Computer Conference*, pp. 967-980, AFIPS Press, 1978.

[5] W. D. Strecker, "Clustering VAX Superminicomputers Into Large Multiprocessor Systems," *Electronics*, pp. 143-146, October 20, 1983.

[6] N. Kronenberg, H. Levy, W. Strecker, "VAX Clusters: A Closely Coupled Distributed System," *ACM Transactions on Computer Systems*, May 1986.

[7] C. G. Bell, "Toward a History of (Personal) Workstations," *ACM Conference on the History of Personal Workstations*, January 9, 1986, published in Goldberg, A., A History of Personal Workstations, Addison-Wesley, Reading, MA, 1988.

[8] T. E. Leonard, *VAX Architecture Reference Manual*, DEC Books, Burlington, MA, 1987.

[9] D. A. Patterson and D. R. Ditzel, "The Case for the Reduced Instruction Set Computer," pp. 25-33, *Computer Architecture News*, October 1980.

[10] R. L. Sites, *Alpha Architecture Reference Manual*, DEC Press, Burlington, MA, 1992.