To: Distribution
Date: 28 May 1985

From: Dave Cutler
Dileep Bhandarkar
Wayne Cardoza
Dave Orbits
Rich Witek

Subject: Architecture Review

A strategic effort has begun within the company to define a new architecture for the 1990s that will complement our current VAX/VMS offering.

To this end, a small group has been chartered by engineering management to define and draft an architectural standard for this new family of machines. The intent is to run the architectural process as we did for the VAX, and to solicit and encourage various people to contribute. The architecture standard will be widely reviewed.

Below is the initial set of goals and constraints the architectural group has set for the new architecture. We would like you to review these goals and provide feedback in the form of a written response.

It is important to undertake this effort, and have the proper goalset.

In the future, you will also be asked to review the architecture document. We welcome your input and comments.

Assumptions

The following major assumptions have been made for this architectural effort:

1. There will be one, and only one, architecture across all implementations.

2. The architecture is not restricted to be a RISC architecture. It is a non-VAX architecture that will be an upward extension of the VAX/VMS family.

3. The aspects of VAX that make it hard to build fast machines quickly and easily will be changed.

4. This is a "from scratch" effort. The VAX System Reference Manual will be used as the starting point.

5. This is a strategic effort within the company. There is a high-level of commitment to implement this architecture and provide the necessary resources.

6. An implementation may use microcode. (That is, there no constraint against the use of microcode.)

7. This design work is similar to the the VAX architectural effort; we are not going to produce anything stopgap.

8. An architectural document will be produced as soon as possible. It will be reviewed by a larger group and then by the corporation.

9. It is expected that it will take around three months to define the architecture.

10. It is assumed that an operating system environment will be constructed that has a compatible file system, network, and user interface with VMS. It is also assumed that the system can be clustered with VMS.

11. It is assumed that ULTRIX will be ported to this architecture.

Goals

The major goals are:

1. To minimize the software investment over time.

2. To allow VMS layered products to be moved to the new architecture.

3. To allow faster, more cost effective machines to be built.

4. To allow shorter development cycles.

5. To allow for parallelism in instruction execution.

6. To allow for Symmetric Multiprocessing (SMP).

7. To have a "pipelineable" orientation.

8. To be "subsettable."

9. To be the new corporate architecture for 1990s.

10. To fix anticipated deficiencies and limitations in VAX, e.g., limited physical and virtual address size.

11. To support the Digital I/O interconnect strategy.

12. To provide floating point accuracy greater than or equal to VAX.

13. To ensure that high-level language programs produce the same results as they would on a VAX.

Non-Goals

Non-goals are:

1. To include VAX-compatibility mode.
2. To support Unibus/Qbus/Massbus peripherals.
3. To translate VAX macrocode transparently and efficiently.

Constraints

The architecture is constrained by the following:

1. It must support VAX-compatible data types.
2. It must support VAX-compatible memory addressing (byte addressability).
3. It must be compatible with the way VAX handles interlocked I/O queues.
4. It must be able to execute the same executable image on every implementation.
5. It must provide at least twice the performance/cost of a particular VAX implementation, using the same technology.

Posted: Tue 28-May-1985 15:03 PDST
To: @ART

+---------------+
! d i g i t a l ! I n t e r o f f i c e M e m o r a n d u m
+---------------+

To: List

Date: 30 May 1985
From: Richard Grove
Dept: Technical Languages
DTN : 381-2082 Loc.: ZKO2-3/N30
Enet: Orphan::Grove
File: RBG066 Rev.: 1

Subj: DECwest Trip Report: Technical Summary

CAUTION

The NONVAX program discussed here is a major strategic undertaking; all information about the program is COMPANY CONFIDENTIAL.

Please limit the distribution of this report.

Many of the topics mentioned here are still under discussion and all will be subject to a wide-ranging formal corporate review. This report is a snapshot taken in mid-May.

1 GENERAL

This memo summarizes my visit to DECwest Engineering May 13-16. The purpose of my visit was to meet with Dave Cutler and Don MacLaren and to begin work on software architecture issues for the NONVAX family of high-performance machines. The meeting was very successful. There is a substantial consensus on technical issues and approaches. We outlined the software architecture documents that need to be produced and we started work on some of the technical content. We agreed on an informal process for getting more people involved in the technical work.

The new architecture project has no official name or codeword, so I have used the name NONVAX in this report.

The hardware and software products that are being designed can be briefly described as "a non-VAX architecture that will be an upward extension of the VAX/VMS family". The initial product will be oriented toward the scientific/technical computational market. The operating system will have a compatible file system, network, and user interface with VMS. NONVAX systems will also be able to be clustered with VAX/VMS systems.

GENERAL

The system should provide higher-level languages that are compatible with VAX. Key languages for the FCS product are FORTRAN, and perhaps C or Pascal. BLISS will be needed for porting VMS layered products and a new systems implementation language will be used for writing most of the NONVAX operating system. We expect that there will be strong demand for the other VMS languages, and that all will be ported soon after FCS.

In general, Dave expects that DECwest will produce:

1. A NONVAX hardware emulator and the first NONVAX product
2. Operating system
3. File system and RMS
4. DECnet
5. Linker and some utilities
6. System implementation language, most likely based on the ELN Pascal language and compiler

There was agreement that SpitBrook would produce:

1. FORTRAN compiler
2. BLISS compiler to be used in porting various VMS layered products to NONVAX. While the operating system will not use BLISS, it will be an important implementation language for many products.
3. RTL
4. DEBUG
5. Additional compilers and layered products for future releases

2 SCHEDULE

The project has no schedule yet, but the following dates and timescales are being discussed:

- On the order of 3 years to FCS, 2.5 years to Field Test
- July 1985 - publish NONVAX architecture SRM, first draft
- Summer 1985 - review/revise/approve SRM
- Fall 1985 - begin detailed design of operating system. This effort is expected to proceed like the original VMS design. The first 6 months will be devoted to design of the operating system and documenting that design in a Working Design Document.
- March 1986 - First version of Implementation Language cross-compiler available to operating system group.

3 HARDWARE ARCHITECTURE

The NONVAX hardware architecture team was meeting at DECwest and Don MacLaren and I spent one day talking with them about various language issues. The hardware architecture team consists of:

- Dave Cutler
- Dileep Bhandarkar (VAX architecture, LTN)
- Wayne Cardoza (VMS)
- Dave Orbits (SAFE, MRO)
- Rich Witek (Chips, HLO)

The hardware architecture team is making good progress. They expect to produce a complete draft of the SRM by late June or early July. The draft SRM will be widely distributed and reviewed within DEC, just as the original VAX SRM.

Attachment 2 is the architecture team's statement of goals and constraints.

3.1 Architecture Characteristics

This section enumerates a number of key characteristics of the hardware architecture that is being defined and comments on some of the software implications.

- NONVAX has a simple instruction set architecture that is intended to facilitate high-speed implementations and pipelining.

  Each instruction is 32 bits long and there are a small number of instruction formats.

- NONVAX supports VAX datatypes, memory layouts, and byte addressability. These are the key attributes required in order to produce high-fidelity VAX-compatible compilers for NONVAX. The floating point format is the same scrambled VAX representation, and bytes and bits within a longword are numbered and addressed like a VAX.

- NONVAX has 64 registers, each is 64 bits wide. The machine is a Load/Store machine. That is, only Load and Store instructions can reference memory operands. All arithmetic and logical operations are performed on values contained in the registers. In short, it looks a good deal like a CDC 6600.

  The current NONVAX design is strongly influenced by the SAFE design that has been done in Marlboro. The NONVAX is much more like SAFE than RISC machines such as TITAN or HR32.

- The major departure from VAX datatypes is that NONVAX addresses are 48 bits. The standard semiconductor metrics (i.e. one address bit every 2 years) indicate that VAX will run out of physical and virtual address bits sometime in the 1990's. The architecture team is convinced that NONVAX must provide more address bits.

  The 48-bit address space is "flat". That is there are no visible segments, and address arithmetic propagates carries through 48 bits (across page and segment boundaries).

  The design of the page tables and the operating system is intended to support very large sparse address spaces at a very low cost in physical memory for segment and page tables. For example, it is deemed reasonable to allocate 32Mb for a task stack and to isolate that stack by a 32Mb guard region of unallocated memory.

  Larger addresses will be the most painful problem in providing VAX-compatible higher level languages. Data structures that contain pointers will have to allow a larger field, and so the structure layouts cannot be identical between VAX and NONVAX. In languages that have real pointer types (e.g. Ada, Pascal, PL/I), most of this can be hidden by the compiler. FORTRAN programmers that manipulate pointers using the %LOC function and 32-bit integers won't be so fortunate.

- The page size will be much larger than VAX, probably at least 16Kb. There is also talk of having the architecture specify a minimum (16Kb) and maximum (512Kb) page size rather than a single fixed page size. In order to port images between different implementations, the software would have to make the smallest unit of virtual allocation and protection be the maximum page size (512Kb).

  The larger page size is desirable because it will provide more untranslated address bits to be used by direct-map caches, and there will be less overhead for managing pages in the operating system. The larger page size is undesirable because the allocation granularity is larger and there will be more breakage when allocating small chunks that have different protection attributes.

- The NONVAX hardware does not support unaligned data access. That is, INTEGER*4 quantities must be aligned on a longword boundary, REAL*8 on a quadword boundary, etc. The NONVAX operating system will handle alignment faults and quietly fix up non-aligned references, much as MicroVAX emulates string and decimal instructions. However, this emulation will incur a substantial performance penalty; on the order of 100 times slower than a properly aligned reference. Because the performance hit is so large, we expect that compilers will provide an optional "natural boundary" alignment when laying

out records and data structures. The compilers will also provide the VAX "pack to the nearest byte" layout. Both VAX and NONVAX compilers should provide both layouts, and should provide a reporting option to identify misaligned items.

- NONVAX is a "base register" machine. Since all instructions are 32 bits long and addresses are 48 bits long, you can't include a full address in a single instruction. Thus unlike the PDP-10, PDP-11, and VAX you must use base-displacement addressing to reference memory operands. As another consequence, the calling conventions define the format of a linkage section and when you call a procedure you must provide a pointer to the called procedure's linkage section as an argument.

  Load/store instructions have a signed 14-bit displacement field, so the displacement range is +-8K bytes from the base address. Jump-class instructions (JSR, conditional and unconditional branches) have a 20-bit displacement in longword units which provides +-2Mb displacements. Compilers won't need Jump/Branch resolution since 2Mb of code in a single object module seems like plenty (at least for the first release).

- There are Load/Store instructions to load bytes, words, longwords, and quadwords. For short quantities, you can sign-extend or zero-extend as you load the value into a 64-bit register. All integer and logical operations operate on 64-bit register values. There are convert instructions to convert a 64-bit value to a byte/word/longword and check for overflow.

- The instruction set is much simpler than the VAX. This means that the following classes of VAX instructions are NOT provided:

  - There are no string instructions.

  - There are no packed decimal instructions.

  - There are no variable bitfield instructions. Field extraction and insertion are performed by loading a quadword into a register and using SHIFT, ROTATE, AND, and OR instructions. Accessing a bit within a packed bit array will involve separating the bit index into a byte offset and bit within byte, fetching the byte, and then shifting and masking to isolate the bit.

  - In the current proposal, the only floating type with full hardware support is G-floating. There are CVTFG and CVTGF instructions so you can do an F-floating VAX operation by converting both F operands to G, performing a G operation specifying chopping, and converting the result back to F to get the correctly rounded VAX F-float

result. Note that you must perform these converts after EVERY F operation in order to get the same results that VAX does. If you keep temps in G format, you get floating results like the C language where all intermediate calculations are done in double precision.

There has been an active debate about providing more support for F-floating, a full set of F-floating operations. It does not seem desirable to have a machine where single precision is slower than double precision.

- There is no hardware support for D-floating or H-floating.

  There is concern about whether customers will accept a NONVAX that does not efficiently support D-floating; there was a lot of resistance to the G-only MicroVAX I.

  Initial estimates suggest that NONVAX software can probably perform H-floating operations in times that are similar to the VAX warm-microcode approach on machines like VENUS and Nautilus.

We expect that the software will provide a CISRTL (Complex Instruction Set RTL) of highly tuned low-overhead subroutines to perform operations equivalent to many of the missing VAX instructions (strings, decimal, D-float, H-float, etc). The architecture team expects that these "micro code in macro code" routines should provide performance on string operations that is comparable to what VAX microcode delivers.

o NONVAX does not have exception enables (e.g. integer overflow) in the Processor Status word. Rather, enables and rounding mode are encoded directly into the instructions. Thus there are two integer add instructions: ADD (no overflow signaled) and ADDV (add and check for overflow), and similarly for other instructions.

  All arithmetic exceptions are faults; the PC is backed up to the faulting instruction and no result is stored.

o NONVAX does not have condition codes. The compare instructions produce a boolean result (0 or 1) in a general register, and you use a branch on low bit instruction to actually branch on the result of the comparison. This improves branching performance because the compiler can schedule the comparison so that the boolean result is available when the branch is executed. There is also a set of (integer) compare against 0 and jump instructions.

o NONVAX provides Execute-only protection on pages. Thus code generators cannot put literals in the code section.

- NONVAX provides a BPT instruction, but there is no T-bit. In order to execute an instruction that has been replaced by a breakpoint, the debugger will have to decode and interpret the instruction. The simple fixed-format instruction set should make this a reasonable approach.

- The hardware is designed to support symmetric multi-processing and the operating system will support multiple threads of execution within a single process (address space). The Run-Time Library will have to be fully reentrant; AST-reentrancy is not sufficient. All compilers should (at least optionally) generate fully reentrant code and calling sequences.

4 SOFTWARE ARCHITECTURE

Don MacLaren and I enumerated a number of specifications that will define the software architecture. These cover familiar topics from VAX such as data types, calling conventions, run-time environment structure, object language, debug symbol table, etc.

We expect that the software architecture process for NONVAX will be much like the VAXS/VAXL process. Initial proposals for new designs will be prepared by small working groups and reviewed by a larger group representing many languages and products. Stable specifications will be updated by an ECO process. We plan to collect all specifications for the software architecture in a multi-volume notebook set. An outline for the notebook set is included as Attachment 1 of this trip report.

We worked on calling conventions in detail and came up with Rev 0 of the calling conventions. We will be presenting this proposal to groups in DECwest and SpitBrook and then writing up a first draft.

We discussed goals and constraints for VAX compatibility. Our general goal is that user programs written in higher-level languages such as Pascal or FORTRAN should run without change on the NONVAX and produce similar results. In some cases it may be necessary to make changes in programs. For example, a FORTRAN program that manipulates addresses in INTEGER*4 variables won't work on a machine with 48-bit pointers. It should be possible to modify such programs so that the modified source can be compiled to work correctly on both machines.

NONVAX compilers will use the same set of default types that VAX compilers do. That is, in FORTRAN or Pascal, "integer" will mean a 32-bit signed longword, "real" will mean 32-bit F-floating, and "double" will mean 64-bit floating (G-floating or D-floating depending on the compiler's /G_FLOATING switch). Since INTEGER*8 will be an important type on NONVAX, we expect that we will also have to provide INTEGER*8 in VAX compilers.

We expect that VAX and NONVAX will coexist (in clusters and networks) for a relatively long time, so it is very important to do a good job on compatibility issues.

5 CALLING STANDARD

The NONVAX calling standard must preserve a number of key attributes of the VAX calling standard, including:

- Mixed language programming.
- Use of procedure calls as the primary interface to all services and subsystems.
- NONVAX must preserve the "programmer's view" of procedure calls as seen on VAX from the perspective of a higher level language.
- NONVAX will provide by value, by reference, and by descriptor mechanisms for argument transmission and the caller will have the same degree of control as on VAX.
- NONVAX will provide a condition handling mechanism that is functionally equivalent to VAX.

The NONVAX calling standard will add a number of new attributes:

- There will be greater emphasis on performance of procedure calls.
- NONVAX will provide "lighweight" procedures whose performance is similar to JSB linkage on VAX. Lightweight procedures will be invoked using the standard calling sequence, so this is purely an optimization performed by the compiler based on the complexity and requirements of the called procedure.
- The standard NONVAX procedure call will pass the first 4 arguments in registers; procedures with 4 or fewer arguments won't use an argument list in memory.
- All descriptor formats must be modified to accomodate 48-bit addresses. Some improvements in descriptor design are planned. For example, it seems desirable to provide only non-contiguous array descriptors; this will improve communication between FORTRAN and Pascal or PL/I.
- All parametric procedures will be passed as bound procedure values rather than as entry point addresses.
- The NONVAX calling standard must make effective use of the much larger number of registers. Registers R0-R15 will be scratch registers and need not be preserved. Interprocedural

analysis (in a mixed language environment) will tailor linkages to reduce call overhead.

I will be distributing a draft of the NONVAX calling standard for review and comment.

6 DEVELOPING THE SOFTWARE ARCHITECTURE

Don MacLaren will be visiting SpitBrook June 11-14 to continue work on the software architecture. I will serve as host and will organize a number of technical meetings. The primary topics for discussion are:

1. NONVAX software architecture

   A general discussion of the overall software architecture, the specifications that need to be written, and architectural process.

2. NONVAX calling standard

   Discussion and review of the first draft of the NONVAX calling standard.

3. System Implementation Language (SIL) requirements

   As noted above, current thinking is that the NONVAX systems implementation language will be based on the ELN Pascal compiler and language. The operating system group has a goal of writing "almost all" of the operating system in a higher-level language. They estimate that there will be 10K to 20K of very low level kernel code written in Macro, and all the rest in the SIL.

   All of the RTL, including the math library, will be written in the SIL. The SIL must provide facilities for exploiting all features of the hardware architecture.

   The SIL effort has a number of constraints, with schedule being the most pressing. With less than a year to develop an initial version, we can't plan to design and implement the ultimate SIL.

   The goal of this session is to have an open but disciplined brainstorming session on the requirements for a new systems implementation language. All participants must agree to avoid language chauvinism and religious fanaticism and focus on requirements.

   We assume there will be two primary implementation languages: BLISS-64 and a new NONVAX implementation language derived from ELN Pascal. The focus of this discussion will be requirements for the new language; we may also discuss BLISS

extensions to support VAX-NONVAX portability.

4. DEBUG and PCA

   Symbol table requirements and design for DEBUG and PCA.

5. RTL and multitasking

   A goal of the NONVAX software architecture is to provide support for multiple threads of execution within a single process (address space). This will provide facilities like tasking in Ada and VAXELN.

[end of RBG066.RNO]

SOFTWARE ARCHICTECTURE SPECIFICATIONS RELATED TO LANGUAGES

Don MacLaren -- 20-May-1985

This is an initial outline of specifications needed in this area with commentary on the purpose of some of them. It's based on notes from a discussion with Rich Grove, but I have added some additional material.

We have divided the specs into three "books", with the first book being the one of current interest. There is overlap between the material included here and what one finds in operating system documentation.

NOTE: This is a preliminary outline of a set of working documents. Nothing is approved; everything is subject to change.

1 THE EXECUTION ENVIRONMENT

This book, together with the hardware architecture book, describes the runtime environment in which compiled programs execute -- excluding operating system characteristics that have little effect on compiled code. Because the instruction set contains no high-level instructions such as the VAX call, some of the material here would fall in the VAX hardware manual.

1.1 Data Types

This section describes all data types recognized in the hardware architecture. Recognition doesn't imply comprehensive support. However it does mean that treatment of the data type will be covered in the calling standard and that the debugger will understand the type.

Alignment and structure mapping are covered.

1.2 Stack Structure And Register Conventions

Covers stack frames, LP, SP, FP, exception handling and unwinding.

1.3 The Calling Standard

1. Linkage conventions including linkage section related to calling.
2. Argument lists: registers vs. memory, optional, variable-length, etc.

3. Standard and Variant Argument Passing Conventions By Data Type.

4. By-value arguments -- a true thorn bush for those that don't fit in a register.

5. Descriptors.

6. Returning Function Values.

1.4 Tasking

1.5 AST's

In particular, an explanation of the best ways to make code AST reentrant or non-AST-interruptible.

1.6 Status Codes And Error Messages

This to include a more contemporary fao capability.

1.7 CIS Specifications

Specifications for standard complex instruction sequences. This covers sequences for things like MOVC, but it is not restricted to analouges of the VAX instructions.

How such sequences fit into the system must also be explained.

1.8 Miscellaneous Issues

1. One-time initilization in programs.

2. Same thing in tasks?

3. Effect of hardware exception treatment when reflected to the level of a typical language.

1.9 Compatibility

Significant differences from VAX/VMS and how to deal with them.

2 COMPILER INTERFACES

This book describes various substantial structures that are the compilers' interfaces to other system tools. The title was chosen in desperation.

The architectural specifications in this book are open; they will be made available to users.

1. The Object Language.

2. Debug Symbol Table.

3. CDD interface.

4. Librarian interface.

5. Diagnostics Output Records. This is what the language-sensitive editors read to help a user correct his source program.

6. Definition Modules. One of these modules contains definitions of data structures and procedure entries. The form is acceptable to all the compilers.

7. Information Output Records. These give additional information about a compilation, its usage of symbols, etc.

Items 6 and 7 are new, at least relative to our VMS environment. Their inclusion is aimed at supporting improved programming environments.

3 INTERNAL COMPILER ARCHITECTURE

This book describes the common structure of compilers that use the common back end. It is the basis for the development of the initial set of product compilers.

This is not likely to be a public document.