VMS Software

News from VMS Software Inc. (VSI)

OpenVMS Compiler Update
April 27, 2021

John Reagan
Terry R. Holmes

Agenda

- General Update
- Compilers
- Q & A

General Update

General Update

- Release of V8.4-2L3
    - Minor Upgrade. Most customers have this right to upgrade by contacting VSI support.
    - Rollup of 20 Patch Kits in a single installable package.
    - Standard Support thru 2028 (compared to V8.4-2L1 with standard support thru 2024.)

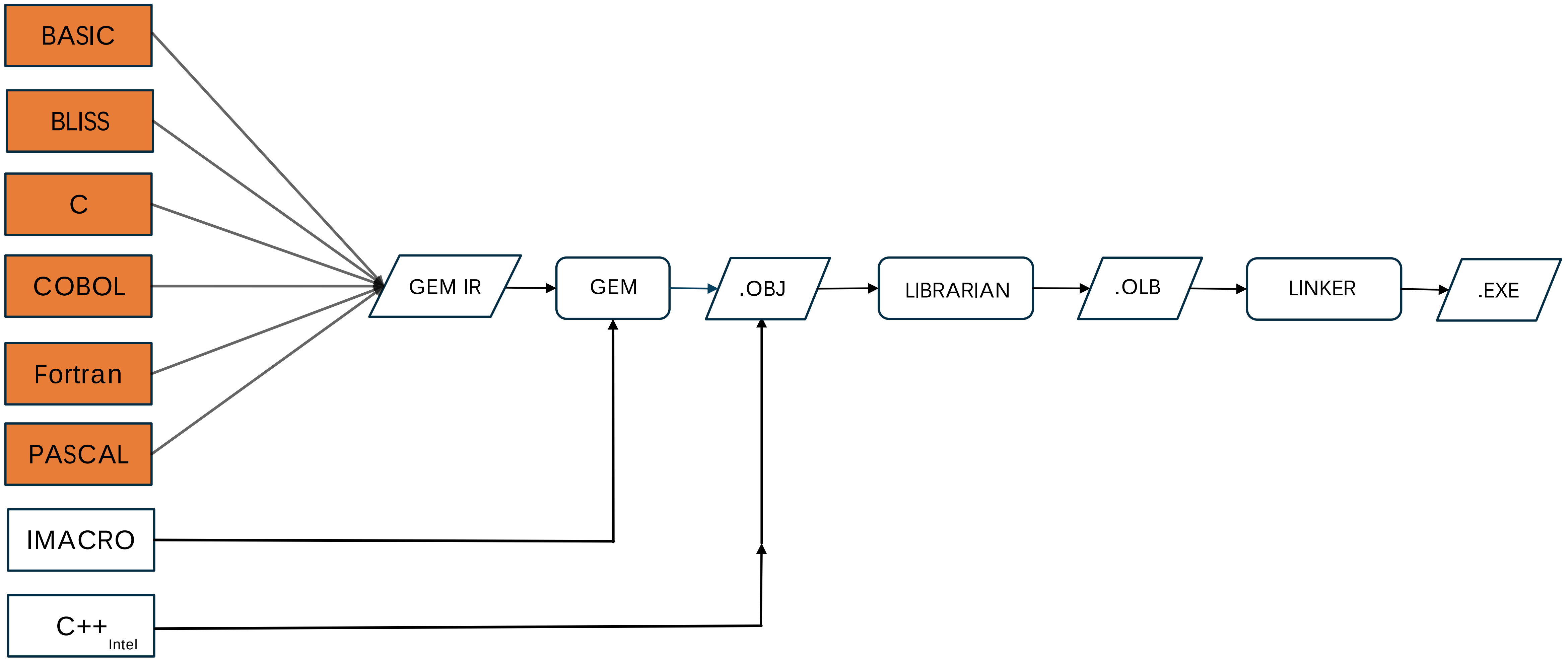
- Added another person to our European Professional Services team based out of our Copenhagen, Denmark offices.

Compilers

OpenVMS Alpha and Itanium History

- Common code generator named GEM with a target independent IR and symbol table
- Macro-32 compiler to allow OS to use existing VAX “assembly”
  - Macro has target instruction knowledge and uses a different interface into GEM
- Itanium switched to ELF/DWARF
  - Linker, Debug, Traceback, Librarian, Analyze, etc underwent substantial work
- Itanium switched to Intel/EDG-based C++ compiler (C++03)
- Remaining Itanium compilers (BASIC, BLISS, C, COBOL, Fortran, Pascal) still use the GEM backend

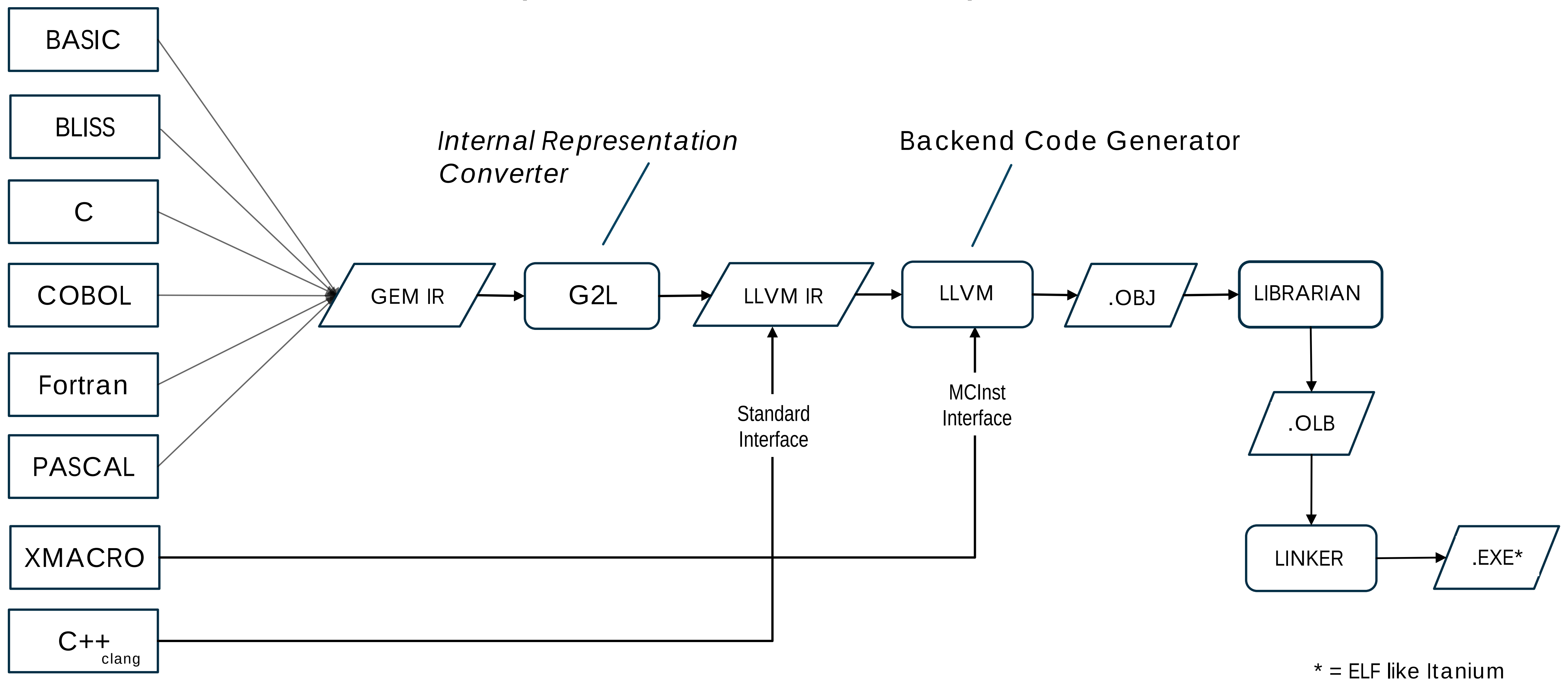
OpenVMS Itanium Compilers



OpenVMS x86-64

- GEM is stale and doesn’t know x86-64
- Instead of throwing people at GEM, we picked LLVM to get modern code base and not have to chase all possible chip features
- Still need to provide “recompile and go” for customers
- Leverage existing frontends which generate GEM IR/symtab
- Create a GEM IR to LLVM IR converter (G2L)
- Leverage clang as our C++ offering to provide up-to-date standards compliance
- Macro compiler (xmacro) needs knowledge of target architecture and is a large piece of work

OpenVMS x86-64 Compilers



* = ELF like Itanium

LLVM Meets Itanium C++

- So which LLVM for the cross-compilers?
  - Itanium C++ is only C++03 compliant
  - LLVM (even back in 2015) uses C++11 syntax
  - LLVM 3.4.2 is the last version that could be compiled with C++03
  - Clang 3.4.2 couldn’t get through the Itanium C++ compiler

GEM Meets LLVM

- Map GEM IR nodes (~275) to LLVM IR nodes
  - Many GEM nodes have nice simple mappings, but some result in interesting IR sequences or clever converter abstractions (strings and packed decimal for example)
  - Some concepts like uplevel-references, multi-entrypoint routines, and static data initialization are very different with LLVM
- Converter currently about 50K lines of C++ (source & headers & comments)
- long double and VAX floating support is TBD
- Will continue to use GEM support routines for command line, file I/O, etc.

OpenVMS Meets x86-64

- Linker updated for x86-specific relocations, sections, and global offset table (GOT)
- Analyze updated for x86 instructions, better DWARF dumping, more ELF content
- Traceback/Debug updated for additional DWARF tags
- Using AMD64 ABI with minor additions for argcount
- Memory model doesn’t match the small/medium/large models found on Linux
- Code moved to 64-bit address space by default but static data/stack remain in 32-bit address space
- x86-64 instructions need address data (GOT) at a fixed-offset from the code

Macro-32 Meets x86-64

- Macro compiler on Alpha and Itanium present the 32 64-bit wide Alpha registers
- Itanium Macro uses a clever register mapping to move them around but still keeps them in hardware registers
- Trying to squeeze 32 registers into the 16 x86-64 registers doesn't work
- Alpha Registers are now memory locations that are managed by the OS
- They look and feel just like Alpha registers so almost all Macro code moves from Alpha or Itanium to x86-64 with no changes
- Companion changes to BLISS/C for explicit linkages were needed as well

LLVM Meets OpenVMS

- A handful of OpenVMS additions to LLVM
  - Mixed pointer size linker relocations
  - Ensure static data and routines are always accessed through the GOT
  - Complex LTCE linker relocations
  - ".note" section generation for module name, compilation date/time, etc
  - ABI additions for argument count and arg-info in RAX register
  - Additional DWARF language tags and listing line numbers
  - Additional EH unwind descriptors for VAX pseudo registers and condition handlers
  - Hooks for machine code listings (currently using ANALYZE/OBJ/DISA)
  - Didn't use the LLVM 3.4.2 optimizer (confusing when stepping through code; needs more G2L work; and couldn't get the Itanium C++ compiler to compile it)

ABI and OpenVMS-isms

- Will use the industry standard AMD64 ABI with a few upward compatible extensions for things like argument count and to assist in performing VMS-style argument list homing
- Will use libc++ and some of libc++ ABI
- Continue to use OpenVMS' debugger and linker
- Continue to use parts of GEM for command line, source file management, etc

Clang Meets OpenVMS

- Need to add/modify some behaviors like dual-sized pointers; interface with GEM for command line processing, include file processing, error message generation, listing file generation, additional pragmas, reading headers from .TLB files, etc.
- The exact list of additions is TBD
- Headers will need some “#ifdef __clang” for differences
- What to do for pointer-size, sizeof(long), or size_t?

Source code changes

- va_list format is different on x86-64. You cannot “copy” with an assignment like you can on Alpha and Itanium. You have to use va_copy() macro which has been added to the C compiler. Other compilers that have argument list functions recreate the VAX-like argument list as needed.
- Macro code that calls routines that preserve R0/R1 needs a source change
- We have seen several instances of code doing 64-bit writes into 32-bit variables. GEM often inserted alignment holes and masked such bad code. LLVM does not add alignment holes and such writes often overwrote other variables. Only seen in complex BLISS or C code dealing with different sized pointer variables.

Native Bootstrapping

- Added limited OpenVMS-isms to LLVM/clang 10.0.1 on Linux to make 10.0.1v
  - argcount and CRTL name prefixing were the two major items
- Use 10.0.1v to compile it all again on Linux
- Move all the objects to OpenVMS x86 and carefully link to get a native clang and LLVM libraries to use with the other frontends and G2L
- And the same with the libcxx/libcxxabi/compiler-rt libraries

Optimizing

- Cross-compilers produce non-optimizing code
- Enabling optimization for clang involves just building all of the components
- Enabling optimization for other compilers is a multi-step process
  - Just building the optimizer will provide some optimization
  - G2L needs to provide additional LLVM metadata for pointer alias optimization
  - Initial native compilers will also produce unoptimized (or limited optimized) code
- Since LLVM has a different set of optimizations there is a slight change that it can expose bugs in the code that have survived for a long time

Current Status

- All cross-compilers are in place other the BASIC
- BASIC is targeted for an update later this year
- Clang is running on OpenVMS x86 and can compile a handful of both C and C++ modules. It does not have a DCL command line, awareness of .TLB files, or any additional OpenVMS-ism
- Traceback mostly works but Debugger still in development
- OpenVMS x86 V9.1 will still use cross-compilers but native compilers will appear in stages. Initially non-optimized but optimization will be enabled in subsequent V9.1 releases

Q & A

Thank You

To learn more please contact us:

vmssoftware.com
info@vmssoftware.com
+1.978.451.0110