vms Software

Porting OpenVMS to x86_64
Update

September 27, 2016

vms Software

Porting OpenVMS to x86_64
Update

This information contains forward looking statements and is provided solely for your convenience. While the information herein is based on our current best estimates, such information is subject to change without notice.

Update Topics

- “The Plan” has not changed
- Code Generation
- Re-Architecting the Early Boot Path
- Dump Kernel
- Boot Manager Graphical Interface
- Running in Two Processor Modes
- Memory Management
- Paravirtualized Drivers
- Software Interrupt Services (SWIS)

Porting Play Book (The Plan)

Chapter 1 – Executable Images

- **Definition**: Register Mapping, Calling Standard extensions
- **Creation:** Compilers, Assembler
- **Action:** LIBRARIAN, LINKER, INSTALL, Image Activator
- **Analysis:** SDA, DEBUG/XDELTA, ANALYZE IMAGE, ANALYZE OBJECT

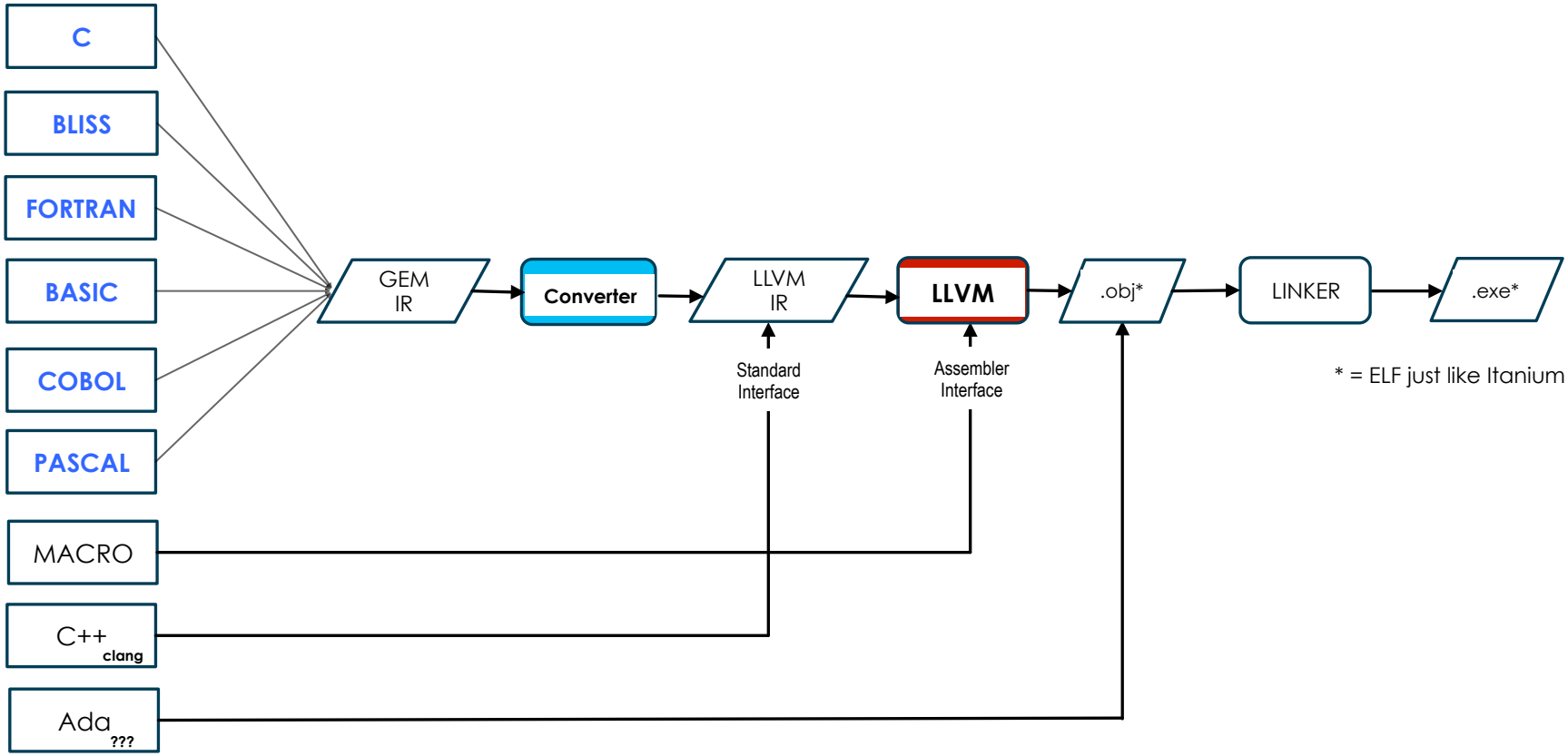
Chapter 2 – Architecture-Specific Needs (a.k.a. “The 5%”)

- Booting
- Interrupts, Exceptions
- Memory Management: protection types, access modes, address space, etc.
- Atomic Instructions
- Floating Point
- Special needs for code in assembler (e.g. VAX QUEUE instruction emulation)

Chapter 3 – Compiling and Linking Everything Else (a.k.a. “The 95%”)

- Large task but mostly mechanical
- Flush out any remaining ‘inter-routine linkage’ problems

Future VMS Compiler Strategy



- **Continue with current GEM-based frontends**
- Use open source LLVM for backend code generation
- ***Create internal representation (IR) converter***
- LLVM targets x86, ARM, PowerPC, MIPS, SPARC, and more

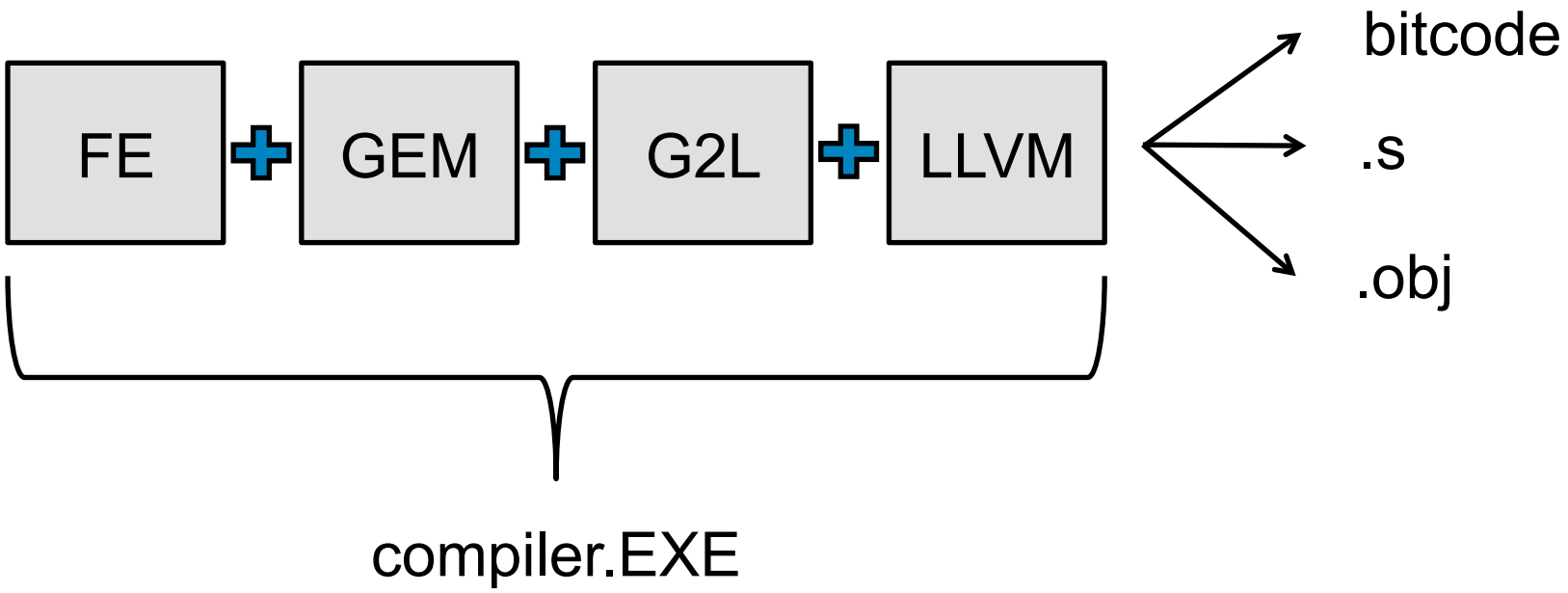
Code Generation

- GEM-to-LLVM (G2L) Converter
  - Started with C frontend
  - Focus on language constructs (and ignore builtins)
  - Converter continues to grow and improve
- C Compiler
  - DEC C Test Suite: 4000+ of 4200 compilation tests pass
  - High percentage of run-time tests pass
  - Many failures due to things not yet implemented
  - For runtime tests
    - On VMS: LLVM outputs bitcode file
    - On linux: link with clang and run
  - Started compiling OS modules
  - Continuing to automate tests – execution and analysis

Code Generation, continued

- As G2L matured, added building BLISS compiler
  - Adding BLISS requirements not already exposed by C
  - Generalizing some C-centric assumptions in G2L
  - Streamlining G2L's parsing of GEM IR
  - Accommodating LLVM being "stricter" than GEM
- BLISS Compiler: Compiling simple program
- MACRO Compiler
  - LLVM integration complete
  - XMACRO uses different LLVM interface than C & BLISS
  - Refining internal structure to best fit x86 needs
  - Initial x86 code generation in progress
- LLVM: many development and debugging options

Inside a Compiler



Re-Architecting the Early Boot Path

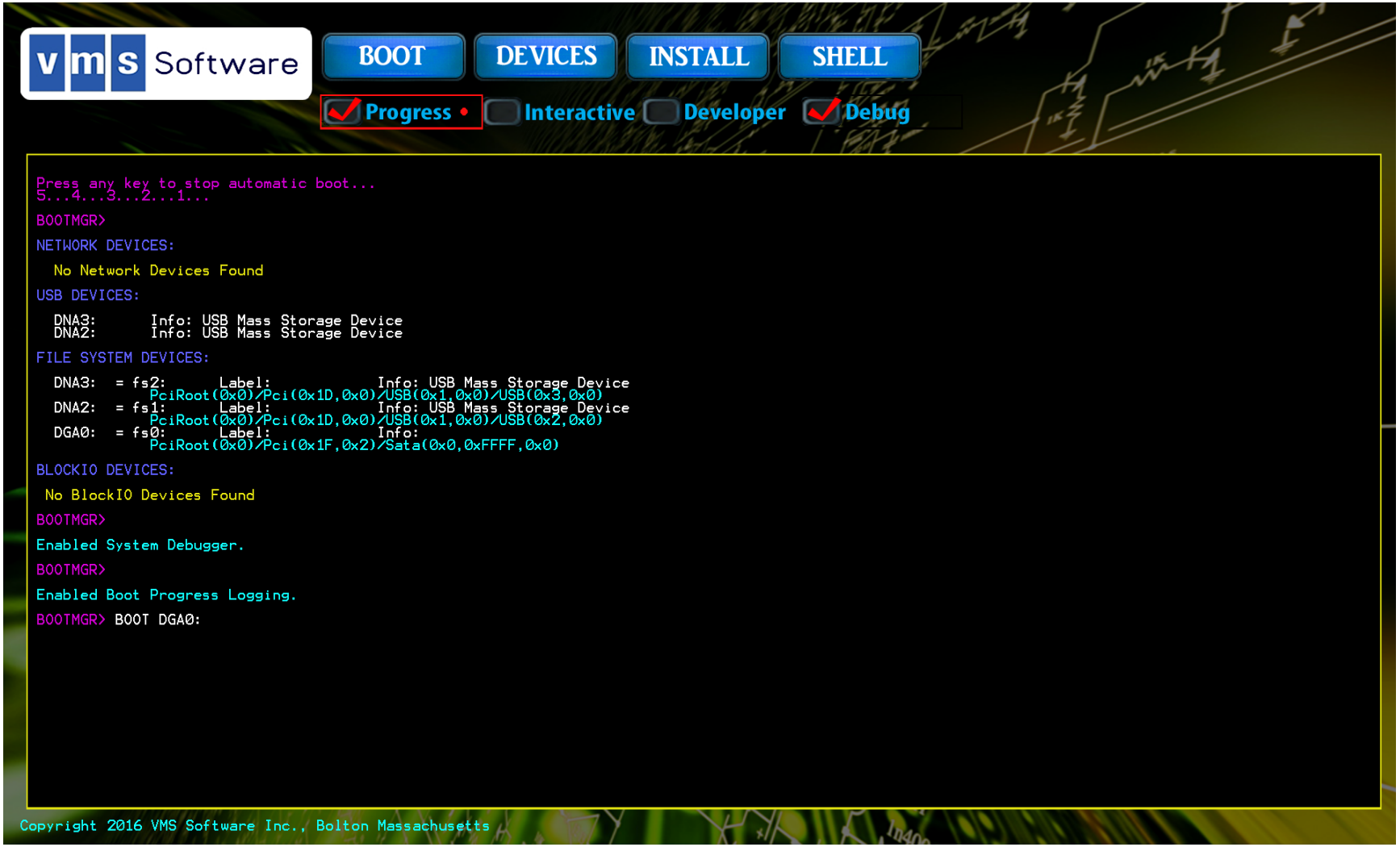
- Itanium: VMS_LOADER.EFI / IPB / SYSBOOT
- x86: VMS_BOOTMGR.EFI / ~~XPB~~ / SYSBOOT
- Goals:
  - Always boot from Memory Disk (SYS$MD.DSK)
  - Eliminate the need for boot drivers
  - Never touch the "primitive file system" again
- Results: Mission Accomplished! (well, 95% as of today)
- Memory Disk file allows easy integrity checking and signing
- Never write another xxBTDRIVER
- On-disk structure of the system disk is no longer a factor prior to loading the full file system
- Local booting – UEFI has its own BlockIO protocol
- But without boot drivers, how do you write crash dumps?

Dump Kernel

- A second, minimalist OS instance is loaded into memory during normal boot - but not booted
- Its memory allocation has a special tag
- Boot Manager leaves SYS$MD.DSK in memory
- As the system goes down BUGCHECK
  - Gathers the necessary data and stores it
  - Calls the Boot Manager to boot the Dump Kernel
- Boots as far as a specialized SYSINIT (plus a little)
- Retrieves the data, writes the dump file using the run-time driver, and initiates a shutdown

Boot Manager Graphical Interface

An early Snapshot, much more to come



Running in Two Processor Modes

- **Review:**
  - x86 has four modes (rings) 0, 1, 2, 3 but they do not provide the strict hierarchy of memory access protection expected by VMS
  - Example: Cannot allow kernel write and prevent exec write
  - VMS will run in two modes: kernel (0) and user (3)
  - Supervisor and Executive modes managed in software
  - With a small (we think) amount of change we can test two-mode operation on Itanium now.
- ***2-Mode Prototyping on Itanium Completed***
- **Results:**
  - PROBE emulation does PTE lookup
  - Did not boot and run complete system
  - Did get far enough to verify transitions to/from Supervisor mode with correct access and mode information
- **Proved the methodology to be sound**
- **Implementation details will differ, but not greatly, on x86**

Memory Management

Review:

- Page protection requires page tables per mode, therefore four levels of page tables
- No PROBE instruction; look it up in page tables
- Available page sizes – 4KB, 2MB, 1GB

New: We will expose 1GB pages to applications even if VMS does not use them internally

Memory Management, continued

SYSBOOT:

- Initial debugging: compile/link with Windows Visual Studio (just like the Boot Manager)
- Some code has been converted from BLISS to C to get an early start
- Memory bit map constructed based on memory descriptors passed from Boot Manager
- PFN database constructed from bitmap
- Level 1 page tables created
- Successful access through PTEs
- Evaluating address space layout options

Paravirtualization

- Using the VIRTIO API for storage driver
- VIRTIO implemented by many hypervisors (KVM, XEN, ….)
- Using linux VIRTIO header files
- Using existing VMS driver GSPDRIVER (Generic Storage Port driver) as a model
- Completed the SCSI/fibre channel IOGEN configuration code
- Started implementation of driver routines
- Evaluating other paravirtualized drivers such as network and console

Software Interrupt Services (SWIS)

- **Review:**
  - SWIS is conceptually architecture independent but code is specific for each platform and it performs the same logical functions
- **Recently Completed:**
  - Design for accessing per-CPU data
  - Basic design for mode changes (system services and interrupts)
- **In progress:**
  - Detailed design for system service calling
  - Detailed design for interrupt and exception handling

Other

- Started secondary CPUs
- Working on getting debug tools from Intel
- Coming soon
  - Start work on LINKER in a few weeks
  - Have LLVM generate .obj rather than .bc
  - Plan for complete rework of USB support
  - Enhancing the build for x86
  - Definitions and guidelines for conditionalization
  - Getting serious about Calling Standard updates

For more information, please contact us at:

RnD@vmssoftware.com

VMS Software, Inc. • 580 Main Street • Bolton MA 01740 • +1 978 451 0110

OpenVMS Rolling Roadmap

2017

H1 2017 - Independent Kits

- JAVA 1.8
- VSI TCP/IP 10.5
- 64b Availability Manager
- CRTL: C99 features, header updates
- Digital Signing for ISVs
- OpenSSL 1.1.x

H2 2017 - OpenVMS V8.x

HPE Integrity System Support

- HPE Kittson-based servers
- 16Gb Fibre Channel for rx2800

VMS Advanced File System

- Improved Performance over ODS-2/5
- Current (32b) Size Limitations

VSI TCP/IP 10.x

Enhanced Password Management

H2 2016 Announcements

- C++ compiler availability
- Support for Pure Storage, MSA2040, ESL G3, LTO-7, MSL6480

2018

OpenVMS V9.0

Itanium & x86-64

- Increased Network Performance
- Stronger Password Encryption
- VMS Advanced File System: Support for Disks > 2TB

Itanium

Additional servers & I/O devices, depending on customer feedback

x86-64 Early Adopters

- Cross-built on Itanium
- OpenVMS as a virtual machine guest
- Selected HPE Servers (Intel and AMD)
- Graphical Boot Manager

2019

OpenVMS V9.x

Itanium & x86-64

- Security
  - Secure Boot
  - Encrypted Crash Dumps
  - Volume (Data-at-Rest) Encryption
- Enhanced Application Isolation and Management

x86-64 General Release

- Native Build
- More virtual machine guest options
- OpenJDK
- Dynamic Binary Translator for Alpha and Itanium Images
- Same compilers supported as on Itanium with standards updates for
  - C
  - C++
  - FORTRAN

Additional releases may occur. The order in which various improvements are added to these releases will be determined by readiness, hardware availability, and customer feedback. Depending on timing, Independent Kit candidates may be introduced in a release.

2-Mode Prototype, a few details

- Use 2 RID bits to indicate mode (priority level) for which the TB entry is valid
- Change PROBE emulation to do a PTE lookup
- Use C version of emulated VAX queue instructions
- Modify TB purge
- Check/Set the RID bits when changing modes
- TB miss handler checks RID in order to get correct mode for new entry