OpenVMS x86-64 Cross-Compiler Release Notes
Revision E9.2-1_XGF4
2-Jun-2022

1 Changes For The V9.2 Release

This release includes refreshed BLISS and Macro-32 compilers; and LINKER and ANALYZE tools.

2 All Code Must Be Recompiled

During this early phase of OpenVMS testing, the compilers are undergoing extensive changes for features like exception handling, floating point support, and calling standard changes. These new features often involved changes to both the compilers and operating system.

All code must be recompiled with this cross-tools kit to use on the matching OpenVMS OS release. The use of older object files or executable files might produce incorrect behavior. There is no "upward-compatibility" at this time. We will provide our traditional "upward-compatibility" promise as we approach the V9.2 release.

3 VSI BLISS

The x86-64 cross-tools kit includes the VSI BLISS-32 and BLISS-64 cross-compilers hosted on OpenVMS IA64 that generate code for OpenVMS x86.

The commands to invoke the BLISS cross-compilers are BLISS/X32 and BLISS/X64.

These cross-compilers behave very much like the native IA64 compilers in terms of command line options and language features. In addition, some lexical functions were added along with changes to how LINKAGES and GLOBAL REGISTERs are implemented for communication with Macro-32 code.

The compiler versions numbers are:

```
$ bliss/x32/version
BLISS-32X T1.12-129-50W5U
$ bliss/x64/version
BLISS-32X T1.12-129-50W5U
```

3.1 New Lexical Functions

The lexical function %BLISS now access BLISS32X and BLISS64X as the compiler name parameter.

- The lexical function %BLISS(BLISS32X) will return 1 when compiled with the BLISS-32 x86 compiler; otherwise it will return 0.

- The lexical function %BLISS(BLISS64X) will return 1 when

compiled with the BLISS-64 x86 compiler; otherwise it will return 0.

The lexical function %TARGET now accepts X86_64 and X86 as a parameter. The function will return 1 when compiled with either BLISS-32 for x86-64 or BLISS-64 for x86-64. %TARGET will also return 0 for names it does not recognize so a misspelling like "X64_64" will just return 0 and not be flagged as an error.

3.2 LINKAGEs And GLOBAL REGISTERs

Unlike other BLISS target architectures, BLISS registers on x86 are mapped to memory locations managed by the operating system. (See the section for Macro-32 for more information). BLISS will automatically convert uses of Alpha registers R0-R30 into memory loads and stores to backing memory locations. The behavior should be identical to that on OpenVMS Alpha or OpenVMS Itanium. However, since these registers are now memory operations, BLISS linkages written as a performance enhancement might now result in slower code since the default calling conventions will pass the first six arguments in x86-64 hardware registers and the remaining arguments on the stack.

3.3 /ALPHA_REGISTER_MAPPING

The /ALPHA_REGISTER_MAPPING DCL qualifier from BLISS for Itanium is available on the x86 compilers but is hardcoded on. It cannot be turned off. All register references correspond directly to the Alpha R0-R30 register set to be compatible with the Macro-32 compiler's behavior.

3.4 Bugs Fixed Since The T1.12-125 Release

- Ensure that the count before a PLIT stays immediately before the data and does not have alignment holes inserted.
- Handle a GLOBAL BIND to a location "outside" of the variable. For example,
```
 OWN V : VECTOR[5,LONG];
 GLOBAL BIND GB = V[-10];
```
- Insert compiler-generated filler to ensure data layout of complex PLITs and UPLITs matches the alignment and layout from Itanium systems.

3.5 Bugs Fixed Since The T1.12-123 Release

- The BLISS builtins that emulate VAX floating now actually use VAX floating format.

3.6 Known Issues

- More Than One MODULE Per Source File

BLISS on prior OpenVMS systems allow multiple MODULE/ELUDOMs in a single source file. Such files will get an LLVM assertion or ACCVIO. We will attempt to resolve this in a future release, but we suggest that you break up the single source file into multiple source files with a single MODULE per file.

4 VSI C

The x86-64 cross-tools kit includes the VSI C x86-64 cross-compiler hosted on OpenVMS IA64 that generates code for OpenVMS x86.

The command to invoke the C cross-compiler is XCC.

The cross-compiler behaves very much like the native IA64 compiler in terms of command line options and language features.

The compiler version number is:

```
$ xcc/version
VSI C X7.4-547 (GEM 50V6F) for X86 on OpenVMS IA64 V8.4-2L1
```

Note that the cross compiler will not interfere with the normal CC installed on the system. You can invoke either CC or XCC without having to perform any intervening setup unless you also use the DECC$SYSTEM_INCLUDE, DECC$USER_INCLUDE or DECC$TEXT_LIBRARY logicals. Both compilers use those logicals in the same way, so if you use them to locate header files that might not be suited to both IA64 and x86-64, you may need to redefine them.

The cross compiler uses the logical X86$LIBRARY to find DECC$RTLDEF.TLB and DECC$SHR.EXE. In addition, any header files normally found in SYS$LIBRARY and SYS$STARLET_C.TLB must also be present in X86$LIBRARY.

This cross compiler behaves very much like the native IA64 compiler in terms of command line options, language features, etc. The primary differences are in the pragma linkage support and the builtin functions. Also, the macros specifying IA64 architecture are not predefined, instead __x86_64 and __x86_64__ are predefined (which is the same practice in Clang and gcc compilers on x86-64).

4.1 Pragma Linkage

For x86-64, only general-purpose registers R0 through R30 are allowed. These registers are not mapped to any x86-64 hardware registers but rather to legacy pseudo registers that are used to support interfaces with Macro-32 code.

4.2 Using Itanium Builtins

The philosophy for the builtin functions is that any existing uses of IA64 builtins should continue to work under x86-64 where possible, but that the compiler will issue diagnostics where it cannot support an IA64 builtin on x86-64.

The builtins.h header file contains a section conditionalized for __x86_64 with all the x86-specific builtins. This section also includes macro definitions for all the registers that can be specified to the __getReg/__setReg/__getIndReg/__setIntReg builtins.

4.3 Bugs Fixed Since The X7.4-528 Release

- Outbound calls to routines using pragma linkage that specify "r16" and "r17" as parameter locations should be treated to mean "standard location for this target"

4.4 Bugs Fixed Since The X7.4-493 Release

- VAX floating support has been enabled to match the Itanium behavior
- Modify the "align(page)" attribute to aligned to 2**13 bytes on x86-64

4.5 Known Issues

- long double

  The long double data type is not yet fully supported. Known issues include compile-time initialization of global/static variables (including structures/unions with long double data types) and calls to math intrinsic functions.

- varargs.h vs stdarg.h

  Due to how the AMD64 ABI Calling Standard is defined, varargs.h is awkward to support. Most Linux platforms don't even support it at all. We have tried to retain as much as possible, but strongly suggest that you convert to using <stdarg.h> instead. It will require source modifications but they will work on Alpha and Itanium so you can keep common code going forward.

5 VSI Fortran

The x86-64 cross-tools kit includes the VSI Fortran cross-compiler hosted on OpenVMS IA64 that generates code for OpenVMS x86.

The command to invoke the Fortran cross-compiler is FORTRAN

The cross-compiler behave very much like the native IA64 compiler in terms of command line options and language features.

The compiler version number is:

```
$ fortran/version
VSI Fortran X8.4-104966 (GEM 50V4V) for X86 systems
```

The cross compiler uses the same CLD as the installed Fortran compiler. Using the cross compiler will prevent you from running an installed native Fortran compiler. You have to deassign the F90$MAIN and F90$MSG logical names to regain access to the native compiler.

This version of Fortran adds support for the ENTRY statement and fixes several compiler crashes when using the /DEBUG qualifier.

As mentioned above, the cross-tools for V9.1-A does not include a new Fortran compiler with VAX floating support. Last minute testing uncovered issues with complex numbers and the RTL that could not be resolved in time for the release. We will fix these issues as soon as possible and provide a new version upon request.

5.1 Bugs Fixed Since The X8.4-104965 Release

- Fix bugs with multi-ENTRY point routines interaction with statement functions and certain format statements.
- Ignore the /SEPARATE_COMPILATION qualifier. The version of LLVM used by the cross-compilers does not easily support this feature. We will revisit this in a future update.

5.2 Known Issues

- Some of the run-time overflow checking has not been implemented yet
- REAL*16 is not fully supported at this time

6 VSI Macro-32 (XMACRO)

The x86-64 cross-tools kit includes the VSI Macro-32 (XMACRO) cross-compiler hosted on OpenVMS IA64 that generates code for OpenVMS x86.

The command to invoke the Macro-32 cross-compiler is MACRO

The cross-compiler behave very much like the native IA64 compiler in terms of command line options and language features.

The compiler version number is:

```
$ macro/flag=compiler_version tt:
XMAC X6.0-111 (GEM 50F9M)
.end
```

Yes, we know that if you don't type ".end" but rather type a control-Z, you'll get a compiler error and traceback. A proper /VERSION qualifier will be added.

6.1 Bugs Fixed Since The X6.0-109 Release

- Sign-extended underlying register after a BBSS, BBSC, BBCS, or BBCC that might have changed the signbit
- Fix handling of auto-increment operands which also are used as target operands. The auto-increment was deferred until the end of the instruction but should have been ignored. For example,
  ```
  MOVL (R7)+, R7
  ```
- CMPB and CMPW of the most negative number would set incorrect condition codes and subsequent conditional branches would be

wrong. This is due to the subtle difference between the VAX CMPL instruction and the x86 cmp instruction.

6.2 Bugs Fixed Since The X6.0-107 Release

- The code for the BBxx and BBxxI instructions was not byte-granular and the interlocked BBSSI/BBCCI did not provide the atomic interlocked access.

6.3 Known Issues

- Most of the IA64_ builtins from OpenVMS IA64 are not supported on OpenVMS x86-64. The builtins tend to be very architecture specific and have no counterparts on x86-64.

- The EVAX_EXTWH, EVAX_EXTLH, EVAX_EXTQH, EVAX_INSWH, EVAX_INSLH, and EVAX_INSQH builtins are not supported since we didn't find any uses in the OS code. If they are required, let us know.

- The EVAX_INSBL, EVAX_INSBH, EVAX_INSWL, EVAX_INSLL, and EVAX_INSQL are not supported with a non-literal as the 2nd operand. If they are required, let us know.

- Due to differences in architecture and calling standards, code that JSBs to a .CALL_ENTRY might have to be modified if the code accesses the argument list. Since this would have not been legal on the VAX, the behavior is poorly defined at best. The solution is to create a CALLG-style argblock, copy the arguments, and use the EVAX_CALLG_64 or CALLG instruction to transfer control to the .CALL_ENTRY target.

- You will see a message

  %XMAC-I-CONCODEXP, built-in used does not set condition codes; earlier instruction used instead

  if you use the PROBER or PROBEW VAX instructions. On OpenVMS x86, these are implemented via macros. The current macro expansion triggers these false messages from the compiler. The underlying macro expansion is correct. We'll remove these messages in a future release.

- The VAX floating and VAX packed decimal instructions are not available. On OpenVMS Alpha and OpenVMS Itanium via a set of macros and some emulation routines. Those routines are not available yet. While the macros in STARLET.MLB might expand, there may be undefined symbols at link-time or undefined behavior.

7 VSI Pascal

The x86-64 cross-tools kit includes the VSI Pascal cross-compiler hosted on OpenVMS IA64 that generates code for OpenVMS x86.

The command to invoke the Pascal cross-compiler is PASCAL

The cross-compiler behave very much like the native IA64 compiler in

terms of command line options and language features.

The compiler version number is:

```
$ pascal/version
VSI Pascal x86-64 X6.3-136 (50VCS) on OpenVMS I64 V8.4-2L1
```

The cross compiler uses the same CLD as the installed Pascal compiler. Using the cross compiler will prevent you from running an installed native Pascal compiler. You have to deassign the PASCAL, PASCALER1, and PASCALER2 logical names to regain access to the native compiler.

7.1 Bugs Fixed Since The X6.3-133 Release

- A new DCL /USAGE=64BIT_TO_DESCR option has been added to allow a well-defined P2 address (00000000.8xxxxxxx) to be fetched from the DSC$A_POINTER field of a 32-bit descriptor. This allows P2-allocated varibles to be passed to conformant array parameters.

- Fix a bug with SET OF CHAR constructors used characters with values greater than 127.

- Fix a bug that prevented TO BEGIN DO, TO END DO, and [INITIALIZE] from working.

7.2 Bugs Fixed Since The X6.3-132 Release

- VAX floating support has been enabled to match the Itanium behavior

7.3 Known Issues

- Some of the run-time overflow checking has not been implemented yet

- Some of the run-time error messages produce a bogus NONAME message in addition to the appropriate error. For example,
  ```
  $ run dka100:[pvs56]err06t
   ERROR...6.4.5-15 (ERR06T)
  %NONAME-W-NOMSG, Message number 00000000
  %PAS-F-SUBASGVAL, subrange assignment value is out of range
  ```

- QUADRUPLE is not fully supported at this time

- Using bound procedure values (PROCEDURE parameters that rely on uplevel references) will result in link-time error with a pair of missing RTL routines. These routines will be implemented soon.

- Uplevel GOTOs will generate a run-time error by mistake.

- The cross-compiler will accept PEN files created by the VSI Pascal compiler for Itanium systems with an informational messages. Normally, the compiler will not accept PEN files from other platforms, but the cross-compiler allows this.

8 VSI COBOL

The x86-64 cross-tools kit includes the VSI COBOL cross-compiler hosted on OpenVMS IA64 that generates code for OpenVMS x86.

The command to invoke the COBOL cross-compiler is COBOL

The cross-compiler behave very much like the native IA64 compiler in terms of command line options and language features.

The compiler version number is:

```
$ cobol/version
VSI COBOL x86-64 X3.1-0013 (50V8U) on OpenVMS IA64 V8.4-2L1
```

The cross compiler uses the same CLD as the installed COBOL compiler. Using the cross compiler will prevent you from running an installed native COBOL compiler. You have to deassign the COBOL and COBOL$MSG logical names to regain access to the native compiler.

8.1 Bugs Fixed Since The X3.1-0012 Release

- VAX floating support has been enabled

8.2 Known Issues

- The /NATIONALITY=JAPAN qualifier may cause an internal compiler error
- VSI has only conducted some limited testing ourselves so there are other errors not yet enumerated. Your help is appreciated.

9 X86-64 Assembler

This kit includes the LLVM tool named "llvm-mc". This provides a native x86-64 assembler that is highly compatible with the gnu "gas" assembler.

It is actived as a "foreign command" in DCL and a symbol "llvm_mc" is created by the setup script.

The compiler version number is:

```
$ llvm_mc -version
LLVM (http://llvm.org/):
  LLVM version 3.4.2
  DEBUG build with assertions.
  Built May  9 2018 (14:34:01).
  Default target: x86_64-pc-linux-gnu
  Host CPU: (unknown)

  Registered Targets:
    x86    - 32-bit X86: Pentium-Pro and above
    x86-64 - 64-bit X86: EM64T and AMD64
```

A sample command is:

llvm_mc -filetype=obj -o=objectfilename.obj sourcefilename.s

Specify "--help" for additional options.

9.1 Known Issues

- Source Files Must Be STREAM_LF

  llvm-mc will ACCVIO if the assembly source file is not STREAM_LF format.

10 Known Issues For All Compilers

1. The cross-compilers ignore the /OPTIMIZE qualifier and currently generate non-optimized code. Future compilers (other than the Macro-32 compiler) will provide the complete set of LLVM optimizations. The Macro-32 compiler does provide some optimization at present but additional code quality improvments will appear in future releases.

2. The cross-compilers ignore the /MACHINE_CODE qualifier. You can use the ANALYZE/OBJECT/DISASSEMBLE command to see the generated code. You can also use the undocumented /SWITCH=ASSEMBLY to get an assembly code output file with the suffix ".S" instead of an ".OBJ" file. The assembly code file also contains static data declarations and initializations.

3. Debug support is not fully implemented and the compiler may generate an assertion when using /DEBUG.

4. VAX floating support has been enabled for all compilers other than Fortran. The Fortran compiler and RTL have additional work remaining. We will provide an updated Fortran compiler when available.

5. /DEBUG support is not yet complete. The compilers may generate an assertion when compile code with /DEBUG. In that case, remove the qualifier and enter a bug report with a reproducer.

6. Quadruple precision floating point (long double, REAL*16, QUADRUPLE, etc.) is currently not supported. The upcoming native compilers will introduce that support.

11 VSI Linker, ANALYZE/OBJECT, And ANALYZE/IMAGE

The x86-64 cross-tools kit includes the cross-linker hosted on OpenVMS IA64 that generates images for OpenVMS x86. The kit also contains an Itanium-hosted ANALYZE that works on both Itanium and x86 objects and images.

The command to invoke the cross-linker is LINK. The command to invoke the ANALYZE tool is ANALYZE.

The linker version number is "I02-89" and be found in a link map file.

The analyze version number is "I01-85" and can be found in the analyze output.

- Using Both /TRACEBACK and /DEBUG Using both /TRACEBACK and /DEBUG incorrect omits all traceback information from the resulting image. This bug will be fixed in the next release.

- New Informational Messages in Linker I02-82

  When the cross-linker encounters writeable code sections, with PSECT attributes set to WRT and EXE, it now prints the following informational message:

  ```
  %ILINK-I-MULPSC, conflicting attributes for section <PSECT name>
          conflicting attribute(s): EXE,WRT
          module: <module name>
          file: <obj-or-olb-filename>
  ```

  When the cross-linker finds unwind data in a module, but no section with the PSECT attribute set to EXE, it prints the following informational message:

  ```
  %ILINK-I-BADUNWSTRCT, one or more unwind related sections are
  missing or corrupted
          section: .eh_frame, there is no non-empty EXE section
          module: <module name>
          file: <obj-or-olb-filename>
  ```

  These messages are seen mainly with Macro-32 and BLISS source modules. It is recommended to make all code sections non-writeable. It is recommended to have code in sections with the PSECT attribute set to EXE

- Starting with version I02-82, the linker now includes additional traceback and debug information in the image file. This additional information will result in slightly larger image files. The information is not read by the image activator so it will not result in slower image activation. This new information is only used by the newly supported traceback and a future release of the debugger.

- The x86-64 cross-linker and cross-analyzer accept the same qualifiers and options as the native IA64 linker and ANALYZE. The linker qualifier /SEGMENT_ATTRIBUTE=SHORT= is ignored because there is no short segment on x86-64.

- The x86-64 cross-linker uses the X86$LIBRARY logical for default library searches.

- Some parts of ELF object and image files are processor-specific, and so will be different on this new processor architecture. Certain flags, the ELF relocation types, and ANALYZE's disassembly output are different.

- There is no GP or short data segment in x86-64. Instead, code segments will have an accompanying global offset table (GOT) segment. These are marked with the FIXED OFFSET attribute in Linker maps and ANALYZE output.

- There are no function descriptors on x86-64, so the way ANALYZE displays transfer vectors and symbol vectors is different.

- Each x86-64 symbol vector entry contains two addresses, compared to the single address in IA64 symbol vector entries.

- The default page size for x86-64 is 0x2000 (4Kb), compared to 0x10000 (8Kb) for IA64.

- Code will be placed in 64-bit-addressible P2 space by default. You can override this by using the /SEGMENT_ATTRIBUTE=CODE=P0 linker qualifier.

- With code now in 64-bit P2 space, you will encounter DIR32NOT32BITS linker errors if you attempt to initialize static data with the address of a routine. For example, creating the arguments to a call to $LKWSET accepts a vector of 2 32-bit addresses. For 64-bit addresss, you should use $LKWSET_64.

- Non-code PSECTs marked with the EXE attribute by linker options files will also result in that PSECT being loaded into 64-bit address space. OpenVMS x86 requires that all static data remain in 32-bit address space. You should remove any non-code PSECT EXE attributes from linker options files.

12 Programming Changes For OpenVMS X86-64

12.1 Code In 64-bit Address Space

On all prior releases of OpenVMS, user code resides in the 32-bit P0 address space. Since the stack and 32-bit heap memory also reside in the P0 address space, a large executable could restrict the amount of stack or heap.

On OpenVMS Itanium, it is possible to place code into P2 space by using the LINK qualifier /SEGMENT=CODE=P2 (C++ programs must be compiled with /POINTER_SIZE=64 in order to use this feature).

For OpenVMS x86, we have changed the default to place all executable code into 64-bit P2 space. You can restore the old behavior with /SEGMENT=CODE=P0.

The LINKER creates small stub routines in 32-bit P0 space to allow the address of a routine to be stored in a 32-bit variable.

In most cases, the move to 64-bit address space is invisible to a program. However, there are two places where you might notice.

- The PC field in the exception signal array is only 32-bits wide. Condition handlers would need to check the 64-bit signal array for the correct value.

- Code that attempts to use the $LKWSET system service to "lock" code into the working set can encounter a LINKER DIRNOT32BITS error trying to store the 64-bit code address will not fit into a 32-bit data structure passed to $LKWSET (the address of the 32-bit stub routine would not give the intended behavior). Programs should have already been using the LIB$LOCK_IMAGE routine that was provided beginning with OpenVMS Itanium.

12.2 OpenVMS X86-64 Calling Standard

OpenVMS x86 is using the AMD64 ABI calling standard (the same one used by Linux 64-bit systems) with a small set of upward compatible extensions. The OpenVMS Calling Standard document has been updated to include the x86 information and lists the OpenVMS specific extensions.

12.3 LIB$WAIT And Other Floating Routines

With VAX floating now enabled in the compilers, the system routines like LIB$WAIT, CVT$CONVERT_FLOAT, LIB$CVT_DX_DX, and others now work correctly.