"compiler-rt" runtime libraries

The compiler-rt project consists of:

- **builtins** - a simple library that provides an implementation of the low-level target-specific hooks required by code generation and other runtime components. For example, when compiling for a 32-bit target, converting a double to a 64-bit unsigned integer is compiling into a runtime call to the "__fixunsdfdi" function. The builtins library provides optimized implementations of this and other low-level routines, either in target-independent C form, or as a heavily-optimized assembly.

  builtins provides full support for the libgcc interfaces on supported targets and high performance hand tuned implementations of commonly used functions like __floatundidf in assembly that are dramatically faster than the libgcc implementations. It should be very easy to bring builtins to support a new target by adding the new routines needed by that target.

- **sanitizer runtimes** - runtime libraries that are required to run the code with sanitizer instrumentation. This includes runtimes for:
  - AddressSanitizer
  - ThreadSanitizer
  - UndefinedBehaviorSanitizer
  - MemorySanitizer
  - LeakSanitizer
  - DataFlowSanitizer
- **profile** - library which is used to collect coverage information.
- **BlocksRuntime** - a target-independent implementation of Apple "Blocks" runtime interfaces.

Clients

Currently compiler-rt is primarily used by the Clang and LLVM projects as the implementation for the runtime compiler support libraries. For more information on using compiler-rt with Clang, please see the Clang Getting Started page.

Platform Support

builtins is known to work on the following platforms:

- Machine Architectures: i386, X86-64, SPARC64, ARM, PowerPC, PowerPC 64.
- OS: DragonFlyBSD, FreeBSD, NetBSD, OpenBSD, Linux, Darwin.

Most sanitizer runtimes are supported only on Linux x86-64. See tool-specific pages in Clang docs for more details.

Source Structure

A short explanation of the directory structure of compiler-rt:

For testing it is possible to build a generic library and an optimized library. The optimized library is formed by overlaying the optimized versions onto the generic library. Of course, some architectures have additional functions, so the optimized library may have functions not found in the generic version.

- include/ contains headers that can be included in user programs (for example, users may directly call certain function from sanitizer runtimes).
- lib/ contains libraries implementations.
- lib/builtins is a generic portable implementation of **builtins** routines.
- lib/builtins/(arch) has optimized versions of some routines for the supported architectures.
- test/ contains test suites for compiler-rt runtimes.

Get it and get involved!

Generally, you need to build LLVM/Clang in order to build compiler-rt. You can build it either together with llvm and clang, or separately.

To build it together, simply add compiler-rt to the -DLLVM_ENABLE_RUNTIMES= option to cmake.

To build it separately, first build LLVM separately to get llvm-config binary, and then run:

- cd llvm-project
- mkdir build-compiler-rt
- cd build-compiler-rt
- cmake ../compiler-rt -DLLVM_CONFIG_PATH=/path/to/llvm-config
- make

Tests for sanitizer runtimes are ported to llvm-lit and are run by **make check-all** command in LLVM/Clang/compiler-rt build tree.

compiler-rt libraries are installed to the system with **make install** command in either LLVM/Clang/compiler-rt or standalone compiler-rt build tree.

If you have questions, please ask on the Discourse forums under the Runtime category. Commits to compiler-rt are automatically sent to the llvm-commits mailing list.