# GSOC 2022 final report

This report summarizes

## Basic Info

- **Name:** Sunho Kim
- **Email:** ksunhokim123@gmail.com
- **Github Username:** sunho
- **Organization:** LLVM
- **Project Title:** Write JITLink support for new format/architecture

## About LLVM

LLVM is an open source framework for building compiler toolchains. It is a powerhouse behind various modern languages such as swift, rust and julia which use LLVM for generating efficient machine code. There are two major approaches in the compiler world: AOT (ahead of time) and JIT (just in time). In AOT compilation, the compiler transforms the entire source code to machine code before the runtime. Whereas in JIT compilation, the compiler runs "just in time" during the runtime, transforming some part of source code to machine code on demand. Since AOT compilation and JIT compilation differ in many characteristics, LLVM supports separate APIs and toolchains dedicated for JIT compilers

## Project description

JITLink is LLVM's new JIT linker designed to support a variety of new features, which includes full static initializer, thread local storage, and small code model, that were not possible in RuntimeDyld, the old JIT linker API. JITLink's generic linker algorithm needs to be specialized to support the target object format (COFF, ELF, MachO), and architecture (arm, arm64, i386, x86-64). This project aims to implement the JITLink specialization for ELF/aarch64 and COFF/x86-64.

## Approach and workflow

Described in detail in fowlling slides: Road Map, Final Progress.

## Goals

- Write JITLink backend for ELF/AARCH64 (arm64 gnu linux)
    - Implement ELF/AARCH64 relocations in JITLink.
    - Implement orc runtime features for arm64 linux including TLS variable registration.
    - Battle-test the implementations by enabling JITLink in julia language and check it fixed JIT issues in arm64 linux.
- Write JITLink backend for COFF/X86_64 (x86_64 msvc windows)
    - Write a generic COFFLinkGraphBuilder that handle PE/COFF object files.
    - Extend JITLink/ORC infrastructure to deal with PE/COFF specific linking rules.
    - Implement COFF/X86_64 relocations in JITLink.
    - Write a foundation for COFF orc runtime with a purpose to support SEH execption table, atexit handler, and static initializer registration.
    - Battle-test the implementations by trying to statically link full VC runtime libraries and microsoft STL library.
    - Battle-test the implementations by enabling JITLink in clang-repl and check it enabled execution of complicated MSVC compliant code using STL, win32 api, and third-party static library out-of-shelf

## Landed patches

### ELF/AARCH64

- D128601 [ORC][ORC_RT][AArch64] Implement TLS descriptor in ELFNixPlatform.
- D126286 [JITLink][ELF/AARCH64] Generic aarch64 patch fixups

---

- PR45859 Enable JITLink on aarch64 linux. (Julia)

---

- D126628 [JITLink][AARCH64][NFC] Create isLoadStoreImm12 function by splitting getPageOffset12Shift

- D126287 [JITLink][ELF/AARCH64] Implement R_AARCH64_ADR_PREL_PG_HI21 and R_AARCH64_ADD_ABS_LO12_NC
- D126630 [JITLink][ELF/AARCH64] Implement R_AARCH64_LDST*_ABS_LO12_NC relocation types
- D127057 [JITLink][ELF/AARCH64] Implement R_AARCH64_ADR_GOT_PAGE and R_AARCH64_LD64_GOT_LO12_NC
- D127058 [JITLink][ELF/AARCH64] Implement R_AARCH64_PREL32 and R_AARCH64_PREL64
- D126387 [JITLink][AARCH64] Fix overflow range of Page21
- D127059 [JITLink][ELF/AARCH64] Implement R_AARCH64_JUMP26
- D127061 [JITLink][ELF/AARCH64] Implement Procedure Linkage Table
- D127062 [JITLink] Remove CodeAlignmentFactor and DataAlignmentFactor validation
- D127063 [JITLink][ELF/AARCH64] Implement eh frame handling
- D127558 [JITLink][AArch64] Unify table managers of ELF and MachO.
- D127559 [JITLink][AArch64] Lift fixup functions from aarch64.cpp to aarch64.h. (NFC)
- D127584 [JITLink][AArch64] Implement MoveWide16 generic edge.
- D127585 [JITLink][ELF][AArch64] Implement R_AARCH64_MOVW_UABS_G*_NC.
- D127715 [JITLink][ELF] Log enum name of unsupported relocation type.
- D127940 [JITLink][AArch64][NFC] Suppress unused variable error.
- D127060 [ORC] Add initial support for aarch64 in ELFNixPlatform

## COFF/X86_64

- D128968 [JITLink][COFF] Initial COFF support.
- D130456 [ORC][COFF] Introduce COFFVCRuntimeBootstrapper.
- D130479 [ORC_RT][COFF] Initial platform support for COFF/x86_64.
- D131833 [ORC][COFF] Introduce DLLImportDefinitionGenerator.

---

- D129720 [JITLink][COFF] Don't dead strip COMDAT associative symbol.
- D129721 [JITLink][COFF] Handle out-of-order COMDAT second symbol.
- D129754 [JITLink][COFF] Implement IMAGE_SYM_CLASS_LABEL.
- D129764 [ORC][COFF] Properly set weak flag to COMDAT symbols.
- D129936 [JITLink][COFF][x86_64] Reimplement ADDR32NB/REL32.
- D129937 [JITLink][COFF] Handle duplicate external symbols.
- D129939 [JITLink][COFF] Implement IMAGE_WEAK_EXTERN_SEARCH_NOLIBRARY/LIBRARY.

- D129941 [JITLink][COFF] Implement IMAGE_COMDAT_SELECT_LARGEST partially.
- D129944 [JITLink][COFF] Consider lib/dll files in llvm-jitlink.
- D129945 [JITLink][COFF] Don't dead strip seh frame of exported function.
- D129952 [ORC][COFF] Handle COFF import files of static archive.
- D130175 [JITLink][COFF] Implement dllimport stubs.
- D130178 [JITLink][COFF][x86_64] Implement ADDR64 relocation.
- D130276 [JITLink][COFF] Implement include/alternatename linker directive.
- D130450 [JITLink] Relax zero-fill edge assertions.
- D130452 [JITLink][COFF][x86_64] Implement remaining IMAGE_REL_AMD64_REL32_*.
- D130454 [JITLink][COFF] Handle COMDAT symbol with offset.
- D130898 [IntelJITEvents] Add missing include.
- D130275 [JITLink][COFF][x86_64] Implement SECTION/SECREL relocation.
- D130451 [JITLink][COFF][x86_64] Stub SECREL relocation to external symbol.
- D132524 [JITLink][COFF] Use DLLImportDefinitionGenerator for creating PLT stubs.
- D132525 [ORC][ORC_RT][COFF] Support dynamic VC runtime.
- D132780 [ORC][ORC_RT][COFF] Remove public bootstrap method.
- D132781 [ORC][LLJIT] Move orc platform support to public orc namespace.

### clang-repl

- D128037 [ORC][LLJIT] Define atexit symbol in GenericLLVMIRPlatformSupport.
- D127991 [clang-repl] Remove memory leak of ASTContext/TargetMachine.
- D129175 [ORC] Fix weak hidden symbols failure on PPC with runtimedyld
- D129242 [clang-repl] Add host exception support check utility flag.
- D128589 [clang-repl] Support destructors of global objects.
- D130788 [clang-repl] Disable building when LLVM_STATIC_LINK_CXX_STDLIB is ON.

## Communication and Work Management

- We had a discord group chat where I could chat freely with all of the mentors. Our mentors responded to all of my questions very promptly (within few hours to 2 days) with a great length (sometimes hundreds of

words) of insightful answers. It never ceases to suprise me how thougtful and on-track are the discussion from my mentors.
- Code review of my pathces was all done by the same mentors in this GSOC group. It was very thorough and prompt. When we required a non-trivial high-level decision, we chatted in the real-time discord group chat about a variety of solutions seeking for the best engineering trade-offs.
- In general, I was free to do my own desired amount of work in each weak as long as I was aware of the immediate weakly goals. However, my mentors quickly gave me a direction for the next tasks each time I finished a certain goal so that I was not lost.
- I joined a weekly zoom meeting with other GSOC contributors in compiler-research group in Princton University where we shared our progresses in the last week with occasional presentations. I had done two presentations in this group which helped me to organize the goals of my project.

## Future scope

### TODOs

- Enabling JITLink in clang-repl for x86_64 msvc windows. I believe clang-repl and ORC JIT stack can evolve together as clang and llvm IR/MC stack has evolved together. Espeically, clang-repl test cases are executed for real in almost all buildbots where JIT is supported. Enabling full COFF backend in clang-repl would allow us to discover shortcomes in the new JITLink support promptly as well as pushing msvc c++ support in clang-repl to another level. I had a proof-of-concept integration with clang-repl on local branch which worked pretty perfectly, but haven't polished it and submit a patch to in-tree.

### Future (long term) goals

- Better support for COFF JITLink backend without ORC runtime case. Currently, COFF JITLink backend can't even run hello world c program without orc runtime. In order for JITLink to be a drop-in replacment of runtimedyld, it should be possible to execute at least case like hello world by its own. Espeically, a lot of JIT users (namely those who develop machine language recompiler or reoptimizer) don't need all complicated runtime features such as SEH execptions.
- Better documentation and examples on how to use new JIT infrastructures such as JITLink and ORC runtime. There are bunch of documentations on ORC and JITLink but they are somewhat outdated.

- Enabling JITLink in more platforms in clang-repl with more complicated c++ test cases. They would serve as nice real-world in-tree integration test cases for new JIT infrastructure.
- Better validation of dllexported symbols. Currently, there is no check if symbols are dllexported when creating jump thunks or dllimport GOT entry. This causes incorrect creation of jump thunk to data symbol when the codegen assumed the library data symbol to be inside static library without any warning for example.
- ORC out-of-process executor support in windows. ORC supports remotely transferring JIT'd code to executor process which can even be inside other computer in network. All groundworks in JIT linker stack is done for windows. We only need to implement a windows remote ExecutionProcessControl class using windows RFC API.

## My learnings

I can't list every lesson I leanred through this project because they are so many. This project has been the most worthwhile and challening software engineering project I have done. Especially, the code I have written for ORC runtime support for COFF/X86_64 has been one of the proudest pieces of work in my life since I started coding 10 years ago.

- I genuinely learned the importance of communication in software engineering. I was surprised how much of time can be saved early on by discussing a high-level design before going on to code and experiment the idea.
- I witnessed how setting short-term goals step by step can boost my productivity.
- I now completly grasp what's going on inside linker and object files. I'm pretty confident that I can build a linker for any object format from scratch if I'm given enough information and time.
- I learned how to work in an enormously large scale codebase like that of LLVM. It's still a bit of annoyance when it occasionally recompiles whole repository which takes about 2 to 3 hours, but I can bear with it.

## Note of thanks

Thanks you so much for all the oppertunities and supports Lang, Vassil and Stefan!! It truly has been a wonderful and life-changing months for me.