# Department of Computing

**CS 354: Compiler Construction**

**Class:** BSCS-8AB

# Lab [12] LLVM and Code Generation

**Date:** 17th Dec, 2021

# Time: Friday (10:00 – 1:00 & 02:00 - 5:00)

# Instructor: Dr Syed Atif Ali Shah

**Lab Instructor: M . Danyal Sadiq**

# Lab [12]: LLVM and Code Generation

**Introduction**

Low Level Virtual Machine – LLVM is a modern module based compiler infrastructure which is written in C++ (mainly) and freely available at ([www.llvm.org](http://www.llvm.org)). It was started in 2000 at University of Illinois at Urbana–Champaign by Chris Lattner and Vikram Adve. LLVM is an unlimited register machine. Register values can only be set once and are typed (meaning they have a specific type).

**Objectives**

1. Understand LLVM compilation/installation procedure and LLVM-IR.

**Tools/Software Requirement**

1. LLVM: An open source framework for compiler construction ([www.llvm.org](http://www.llvm.org))

**Description**

From its beginning in December 2000, LLVM was designed as a set of reusable libraries with well-defined interfaces [1]. At the time, open source programming language implementations were designed as special-purpose tools which usually had monolithic executables. For example, it was very difficult to reuse the parser from a static compiler (e.g., GCC) for doing static analysis or refactoring. While scripting languages often provided a way to embed their runtime and interpreter into larger applications, this runtime was a single monolithic lump of code that was included or excluded. There was no way to reuse pieces, and very little sharing across language implementation projects.

Over the last ten years, LLVM has substantially altered this landscape. LLVM is now used as a common infrastructure to implement a broad variety of statically and runtime compiled languages (e.g., the family of languages supported by GCC, Java, .NET, Python, Ruby, Scheme, Haskell, D, as well as countless lesser known languages). It has also replaced a broad variety of special purpose compilers, such as the runtime specialization engine in Apple's OpenGL stack and the image processing library in Adobe's After Effects product. Finally LLVM has also been used to create a broad variety of new products, perhaps the best known of which is the OpenCL GPU programming language and runtime.

More details at: <https://llvm.org/>

**Lab Tasks**

1. Download LLVM 5.0.1 from <http://releases.llvm.org/download.html#5.0.1> (or later)
   1. Extract it using the following command

tar –xvJf llvm-5.0.1.src.tar.xz

* 1. Verify that the C Front-end (Clang) is installed on your VM. Otherwise install it.
  2. Configure the LLVM 5.0.1 for compilation. What commands did you use?

(Hint: You can take help from ./configure --help)

* 1. Compile LLVM 5.0.1 and Install it. What commands were used?
  2. In order to use LLVM, do you need to modify your PATH environment variable? If yes, then how did you accomplish this task?

1. Create a C source file named **test1.c** with the following code:

|  |
| --- |
| #include <stdio.h>  int main( )  {  printf ("Hi, my LLVM + CLANG!\n");  } |

* 1. Compile it using Clang (**clang -o test1 test1.c**). This will produce native code, then run it (**./test1**). To generate LLVM IR, in assembly form, run **clang test1.c -S -emit-llvm** which will produce **test1.ll**.
  2. Now run the same command but with optimization flag (**-O3**) and compare the output. Compile other input programs, with and with-out optimization, for example:

|  |
| --- |
| int main ()  {  int x=0;  return x++;  } |

**Deliverables**

You are required to upload a **PDF/Word document** with your answers with screenshots, using the link created on LMS.

**References:**

[1] Chris Lattner and Vikram Adve: "LLVM: A Compilation Framework for Lifelong Program Analysis & Transformation". Proc. 2004 International Symposium on Code Generation and Optimization (CGO'04), Mar 2004.