# Compiling LLVM IR

Hosein Ghahramanzadeh

# Clang Compiler

Setting up:

- Download binaries
  - LLVM Download Page
  - Download the latest version
  - Install and add to PATH
- Build from source
  - Getting the source code using SVN
    - How to build
  - Alternative mirror github repository
    - How to clone and build

# Compiling Hello World

- Check if clang is installed and available.
  - ```
    E:\LLVM>clang
    clang.exe: error: no input files
    ```
  - Clang is installed.
  - Otherwise try reinstalling or check that it is added to PATH.
    ```

# Compiling Hello World

- Create main.ll with code:
  - ```
    @msg = internal constant [13 x i8] c"Hello World!\00"
    declare i32 @puts(i8*)
    define i32 @main() {
        call i32 @puts(i8* getelementptr inbounds ([13 x i8], [13 x i8]*
    @msg, i32 0, i32 0))
        ret i32 0
    }
    ```
    ```

# Compiling Hello World

- Compile with:
  - ```
    E:\LLVM\Test1>clang main.ll
    warning: overriding the module target triple with
    x86_64-pc-windows-msvc19.0.24210 [-Woverride-module]
    1 warning generated.
    ```
  - The warning indicates that no explicit target is specified. The target is set to current machine's architecture.
  - Creates binary, on windows machines a.exe.
  - Also does the linking.
    ```

# Compiling Hello World

- Options
  - `-S` Compile to assembly code
  - `-o` Specify the output file
  - `-O[0, 1, 2, 3]` Specify the optimization level
  - `-mllvm` Specify the llvm flags
    - --x86-asm-syntax=[intel, att] Specify the output x86 assembly syntax, Intel or AT&T
- Example
  - ```
    E:\LLVM\Test1>clang main.ll -S -o main.asm -O3 -mllvm
    --x86-asm-syntax=intel
    warning: overriding the module target triple with
    x86_64-pc-windows-msvc19.0.24210 [-Woverride-module]
    1 warning generated.
    ```
    ```

# Cross Compilation

- So far we have not been specifying compilation target.
- Not specifying compilation target causes, compilation to current architecture, hence the warning.
- You can specify the target to current machine, or other architectures you desire to compiler to.
- This is not in the scope of this presentation, you can find `-target` options in the link here and here.

# So what was all this for?

- Write a CG that generates LLVM IR code.
- Run Compile LLVM IR to assembly. (Or directly compile to executable)
- Assemble the assembly code to object file.
- Link object file to get binary.

# Assembling and Linking

- You can use clang for assembling you assembly code.
  - `clang main.asm -c -o main.obj` Create object file from assembly code.
- After getting object files of all the compilation units, these object file should be linked to achieve final binary.
- Linkage can also be done using clang.
  - clang main.obj -o main.exe Link main.obj

# Why does linking matter?

- ```
  @msg = internal constant [13 x i8] c"Hello World!\00"
  declare i32 @puts(i8*)
  define i32 @main() {
      call i32 @puts(i8* getelementptr inbounds ([13 x i8], [13 x i8]* @msg,
  i32 0, i32 0))
      ret i32 0
  }
  ```
- Where is the definition of puts?
  - In other already compiler binary.
  - Linker links this declaration to the appropriate binary.
- Multiple compilation units
  - Declaration of function may not be present in single compilation unit.
  - Link the compilation units together.
  ```

# Multiple compilation units?

- main.ll

```
declare i32 @function()
define i32 @main() {
    call i32 @function()
    ret i32 0
}
```

- function.ll

```
@msg = internal constant [13 x i8] c"Hello World!\00"
declare i32 @puts(i8*)
define i32 @function() {
    call i32 @puts(i8* getelementptr inbounds ([13 x i8], [13 x i8]* @msg,
i32 0, i32 0))
    ret i32 0
}
```

# Multiple compilation units?

- Link the two together:
  - `clang main.obj function.obj -o main.exe`
  - Note that compiling .ll files and assembling can be done separately.
  - Doing it all together.
    - clang main.ll function.ll -o main.exe
- It is not necessary to do the assembling and linking using clang.