LLVM Bitcode

Current LLVM Backend

Cmm
-> Textual IR
-> Optimizer
-> Compiler
-> Mangler
-> Assember
-> Linker

Why Bitcode

- Textual IR (can) change with every LLVM release
- Bitcode IR is supposed to be readable up to and including the next major version

The textual format is not backwards compatible. We don’t change it too often, but there are no specific promises.
The bitcode format produced by a X.Y release will be readable by all following X.Z releases and the (X+1).0 release.
— LLVM Developer Policy

Bitcode format

- Bit based format
- Primitives: fixed width ints, var width ints, 6-bit chars
- Basic structure:
  - Header
  - Blocks
  - Records

github.com/angerman/data-bitcode

LLVM Bitcode

- Additionally signed var width ints
- Defines additional Blocks and Records
- Can encode LLVM Modules

github.com/angerman/data-bitcode-llvm

EDSL

```
helloWorld = mod "helloWorld" [
  def_ "main" ([i32, ptr i8ptr] --> i32) $ \[ argc, argv ] -> do
      block "entry" $ do
        strPtr  <- gep (global "str" (cStr "hello world\n")) [int32 0, int32 0]
        ccall (fun "printf" (vararg $ [i8ptr] --> i32)) [strPtr]
        ret $ int32 0
  ]
```

github.com/angerman/data-bitcode-edsl

LLVM Bitcode plugin

- Based on the LlvmGen in GHC
- EDSL → LLVM Module → Bitcode

```
main = putStrLn "Hello World"
```

```
$ ghc HelloWorld.hs -fplugin Data.BitCode.Plugin
$ ./HelloWorld
Hello World
```

github.com/angerman/data-bitcode-edsl

Open issues

- Performance is terrible
- No complete Cmm coverage yet
- Missing Metadata support
- No function level VST support

Thanks!

Questions?