Where we are

Time	Details	Type
8:30 - 8:35	Introduction, welcome and objectives	Presentation
8:35 - 8:55	An overview of MLIR and LLVM	Presentation
8:55 - 9:20	The xDSL framework	Presentation
9:20 - 9:25	Introduction to the hands-on activity	Presentation
9:25 - 10:00	Logging into ARCHER2 and hands-on practical activity	Hands-on
10:00 - 10:30	Morning break	
10:30 - 10:35	Welcome back and overview of second part	Presentation
10:35 - 11:45	Hands-on practical activity	Hands-on
11:45 - 11:55	Wash up from practical activities, highlighting key take-away points	Presentation
11:55 - 12:00	Conclusions & next steps to continue working with the technologies	Presentation

Wrap up of exercise one

- Hopefully you have managed to run exercise one by now and have an executable that runs on ARCHER2

1. The decorator *@python_compile in our python_compiler* module drives translation of the Python code into our *tiny py* dialect
   - This is the entry point into MLIR
2. We run the *tiny-py-to-standard* transformation to lower this into the standard dialects
3. This IR is passed to mlir-opt which converts it into LLVM IR
4. LLVM IR is compiled by clang into an executable

Single Static Assignment (SSA) form

Retrieves the address and pointer of our string

Calls the printf function with our string as an argument

```
"builtin.module"() ({
  "func.func"() ({
    %0 = "llvm.mlir.addressof"() {"global_name" = @str0} : () -> !llvm.ptr<!llvm.array<13 x i8>>
    %1 = "llvm.getelementptr"(%0) {"rawConstantIndices" = array<i32: 0, 0>} : (!llvm.ptr<!llvm.array<13 x i8>>) -> !llvm.ptr<i8>
    "func.call"(%1) {"callee" = @printf} : (!llvm.ptr<i8>) -> ()
    "func.return"() : () -> ()
  }) {"sym_name" = "main", "function_type" = () -> (), "sym_visibility" = "public"} : () -> ()
  "llvm.mlir.global"() ({
  }) {"global_type" = !llvm.array<13 x i8>, "sym_name" = "str0", "linkage" = #llvm.linkage<"internal">, "addr_space" = 0 : i32,
"constant", "value" = "Hello world!\n", "unnamed_addr" = 0 : i64} : () -> ()
  "func.func"() ({
  }) {"sym_name" = "printf", "function_type" = (!llvm.ptr<i8>) -> (), "sym_visibility" = "private"} : () -> ()
}) : () -> ()
```

Signature definition of printf function

The string that we will print