LLVM Tutorial

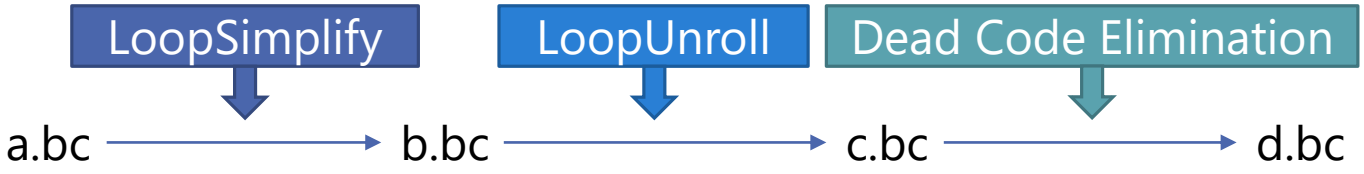
IR Optimization (Part 2)

2019. 04. 10

IR Optimization

- LLVM enables modular optimizations through the LLVM pass framework

- Each **LLVM pass** performs code analysis and transformations on LLVM IR
  - Example



IR Code Transformation

- Add or delete instructions in existing functions
- Define new functions
- Import external functions

- If source code is changed, return `true` at `runOnXXX`

```
bool Hello::runOnModule(Module &M) override {


  return false;
}
```

LLVM IRBuilder

- Help to create instructions and insert them into a b asic block

- Include "`llvm/IR/IRBuilder.h`"

- Simple Declaration
  - `IRBuilder<> Builder(I)`
    - Insert instructions before I (Instruction*)
  - `IRBuilder<> Builder(BB)`
    - Insert instructions at the end of BB (BasicBlock*)

LLVM IRBuilder

- Available member functions
  - http://llvm.org/doxygen/classllvm_1_1IRBuilder.html
  - Memory Instructions

```
LoadInst *   CreateLoad (Type *Ty, Value *Ptr, const Twine &Name="")
```

- `Type *Ty`: Pointer type
- `Value *Ptr`: Pointer to load

```
StoreInst *   CreateStore (Value *Val, Value *Ptr, bool isVolatile=false)
```

- `Value *Val`: Value to store
- `Value *Ptr`: Pointer to store the value

LLVM IRBuilder

- Available member functions
  - Arithmetic Instructions

```
Value *   CreateAdd (Value *LHS, Value *RHS, const Twine &Name="",
                     bool HasNUW=false, bool HasNSW=false)

Value *   CreateSub (Value *LHS, Value *RHS, const Twine &Name="",
                     bool HasNUW=false, bool HasNSW=false)

Value *   CreateMul (Value *LHS, Value *RHS, const Twine &Name="",
                     bool HasNUW=false, bool HasNSW=false)
```

* NUW = No Unsigned Wrap, NSW = No Unsinged Wrap

LLVM IRBuilder

- Available member functions
  - Other Instructions

CallInst *	**CreateCall** (**Value** *Callee, **ArrayRef**< **Value** *> Args=**None**, **const Twine** &**Name**="", **MDNode** *FPMathTag=nullptr)

ReturnInst *	**CreateRetVoid** () Create a 'ret void' instruction. More...
ReturnInst *	**CreateRet** (**Value** *V) Create a 'ret <val>' instruction. More...

LLVM IRBuilder

- Example 1
  - Insert a function call for every basic block

```
std::vector<Value*> args(0);
for (BasicBlock &BB : F) {
  IRBuilder<> Builder(BB->getTerminator());
  CallInst *newCallInst = Builder.CreateCall(MarkBBEnd, args, "");
}
```

LLVM IR Types

- LLVM manages IR types as class instances

- How to get Type instances
  - Use static member functions of class Type

static IntegerType *	getInt1Ty (LLVMContext &C)
static IntegerType *	getInt8Ty (LLVMContext &C)
static IntegerType *	getInt16Ty (LLVMContext &C)
static IntegerType *	getInt32Ty (LLVMContext &C)
static IntegerType *	getInt64Ty (LLVMContext &C)
static IntegerType *	getInt128Ty (LLVMContext &C)

static Type *	getFloatTy (LLVMContext &C)
static Type *	getDoubleTy (LLVMContext &C)

LLVM IR Types

- Type class member functions
  - Check a type: `isVoidTy`, `isHalfTy`, `isFloatTy`, …

bool	**isHalfTy** () **const** Return true if this is 'half', a 16-bit IEEE fp type. More...
bool	**isFloatTy** () **const** Return true if this is 'float', a 32-bit IEEE fp type. More...
bool	**isDoubleTy** () **const** Return true if this is 'double', a 64-bit IEEE fp type. More...

  - Get the point type of a type

PointerType *	**getPointerTo** (**unsigned** AddrSpace=0) **const** Return a pointer to the current type. More...

How to Create Function

- Multiple ways to create a function
  - `getOrInsertFunction` in `Module` class
    - Signature

FunctionCallee getOrInsertFunction (StringRef Name, Type *RetTy, ArgsTy... Args)

Same as above, but without the attributes. More...

    - Usage

```
FunctionCallee addFun = M.getOrInsertFunction(
  "add",
  Type::getInt64Ty(Context),
  Type::getInt64Ty(Context),
  Type::getInt64Ty(Context)
);
```

Return Type

Argument Types

How to Create Function

- Multiple ways to create a function
  - `getOrInsertFunction` in `Module` class
    - Signature

```
FunctionCallee  getOrInsertFunction (StringRef Name, FunctionType *T)
```

    - Usage

```
std::vector<Type*> formals(2);
formals[0] = Type::getInt64Ty(Context);
formals[1] = Type::getInt64Ty(Context);

FunctionType *addFunType = FunctionType::get(
    Type::getInt64Ty(Context), formals, false);
FunctionCallee addFun = M.getOrInsertFunction(
  "add", addFunType);
```

Argument Types

Return Type

How to Create Function

- Multiple ways to create a function
  - `Create` In `Function` class
    - Signature

static **Function** * **Create** (**FunctionType** *Ty, **LinkageTypes Linkage**, **const Twine** &N, **Module** &M)

Creates a new function and attaches it to a module. More...

    - Usage

```
std::vector<Type*> formals(2);
formals[0] = Type::getInt64Ty(Context);
formals[1] = Type::getInt64Ty(Context);

FunctionType *addFunType = FunctionType::get(
    Type::getInt64Ty(Context), formals, false)
Function *addFun = Function::Create(
  addFunType, GlobalValue::InternalLinkage,
  "add", &M
);
```

How to Create BasicBlock

- `Create` in `BasicBlock` class
  - Signature

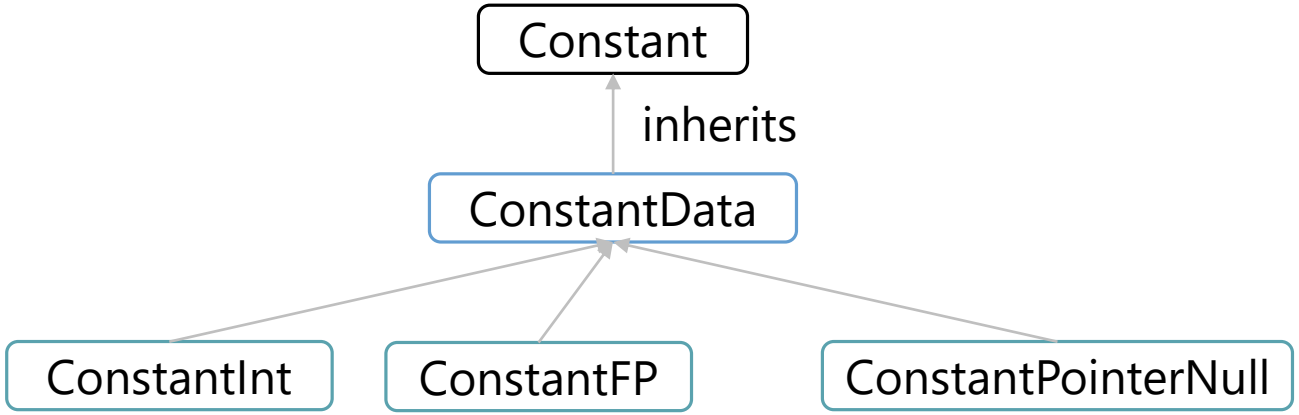
```
static BasicBlock *   Create (LLVMContext &Context, const Twine &Name="",
                              Function *Parent=nullptr, BasicBlock *InsertBefore=nullptr)
```

  - Usage

```
BasicBlock *entry = BasicBlock::Create(Context, "entry", addFun);
```

How to Create Constant

- Is-A relationship of Constant classes



How to Create Constant

- get in ConstantInt class
  - Signature

```
static Constant *   get (Type *Ty, uint64_t V, bool isSigned=false)
```

- get in ConstantFP class
  - Signature

```
static Constant *   get (Type *Ty, double V)
```

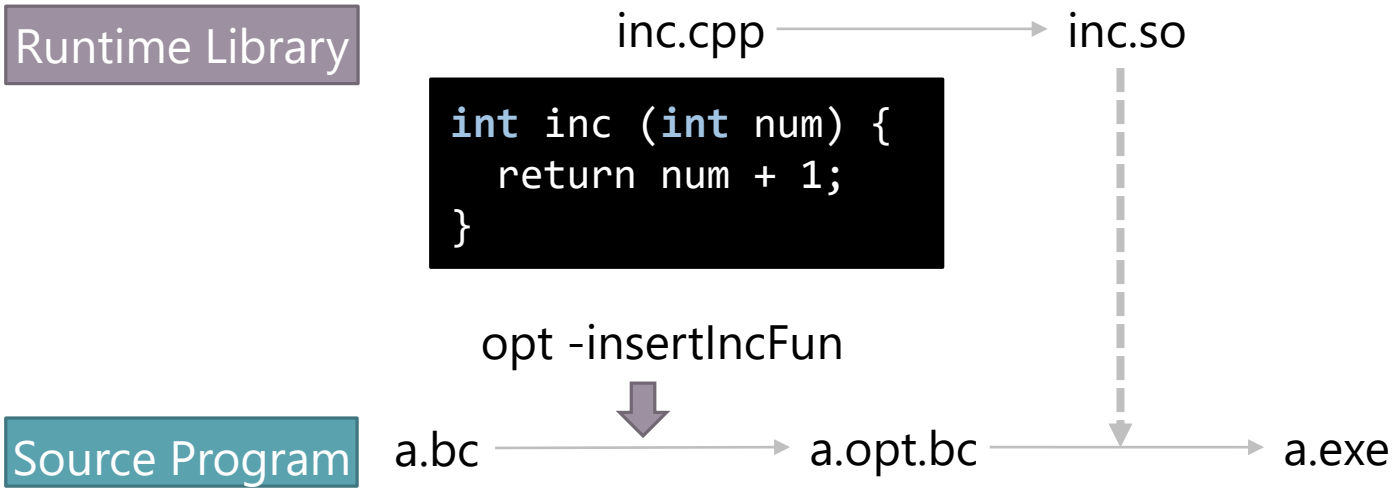
Practice 1: Insert Inc Function

- Goal
  - Learn how to create a function and instructions

- Steps
  1) Implement InsertIncFunction that inherits ModulePass
     - Create a function named "Inc"
       - `int inc (int n) { return n + 1; }`
     - Create a basic block for the Inc function
  2) Fill the Inc function with instructions
  3) Run the pass with `opt` on a sample program

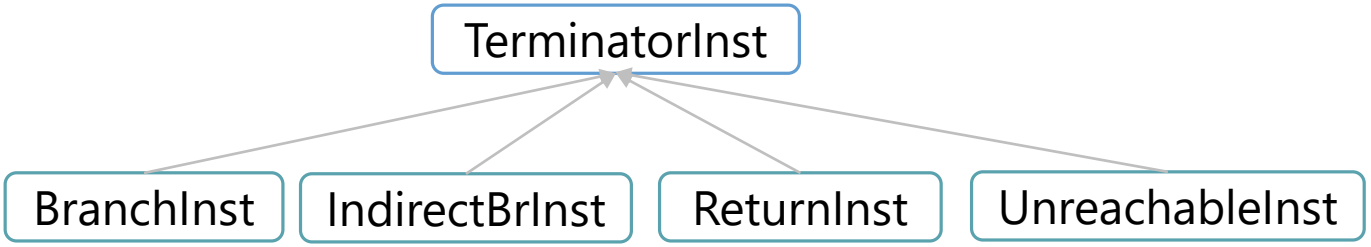
How to Instrument Runtime Function Call

- It is time-consuming to implement functions in LLVM IR
- Implement functions in C or C++, then **link them as a library!**



How To Change Control Flow

- Change the *terminator* instruction
  = the last instruction of a basic block

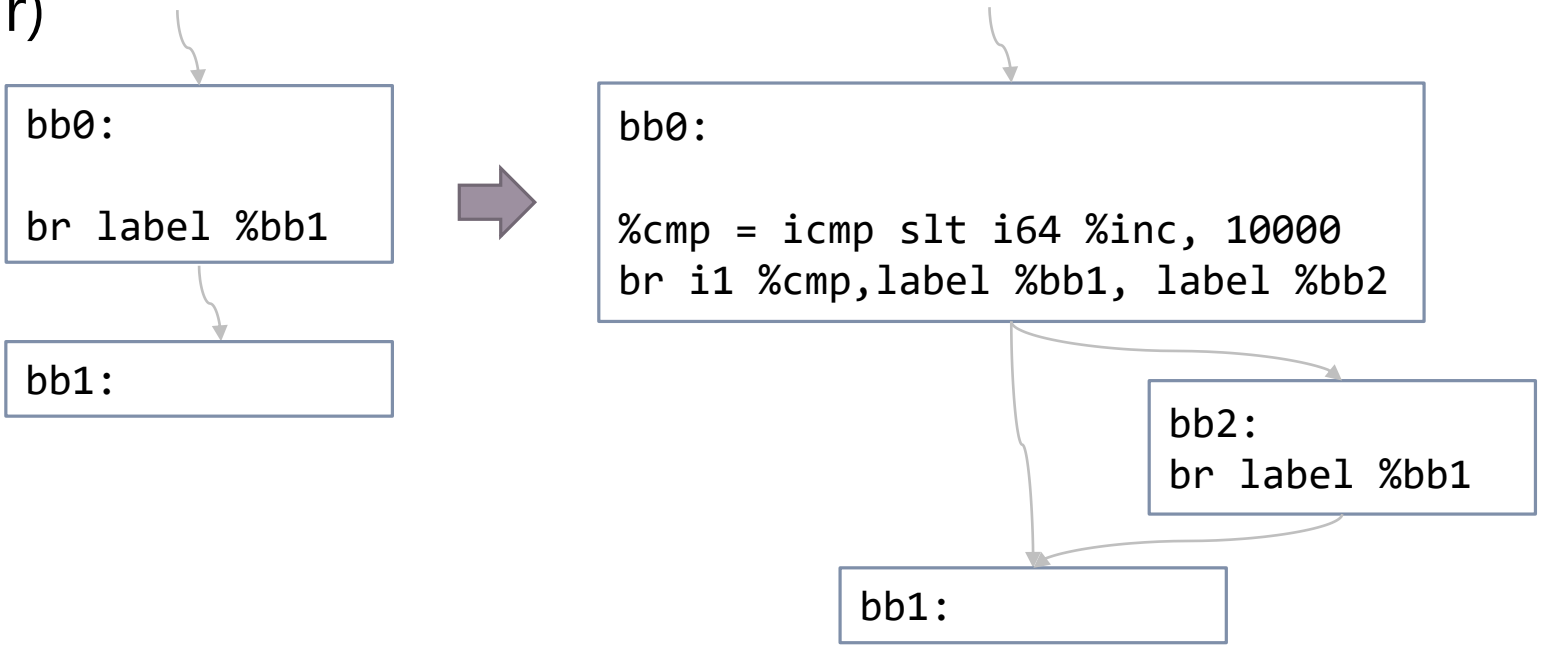


- Conditional
- Unconditional

- A basic block **always** need a terminator!

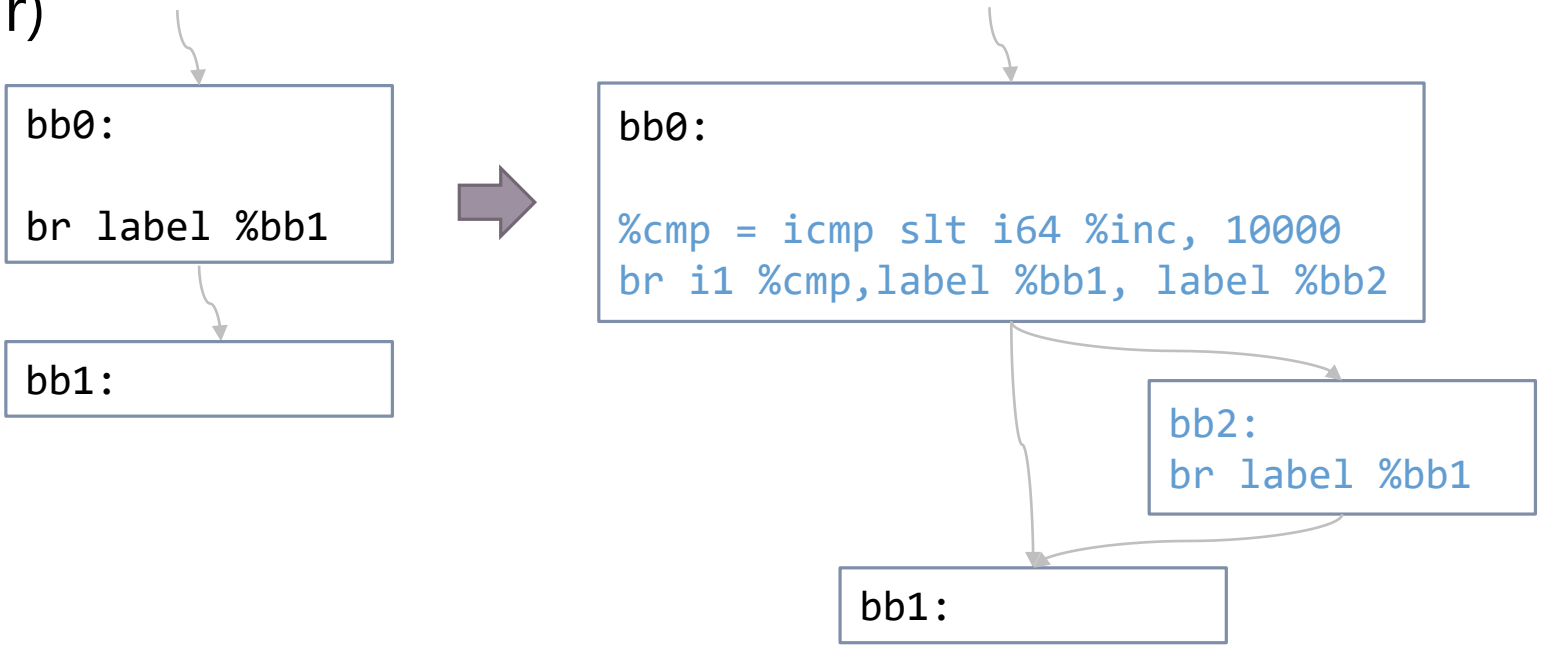
How To Change Control Flow

- Unconditional branch → Conditional branch (Detour)



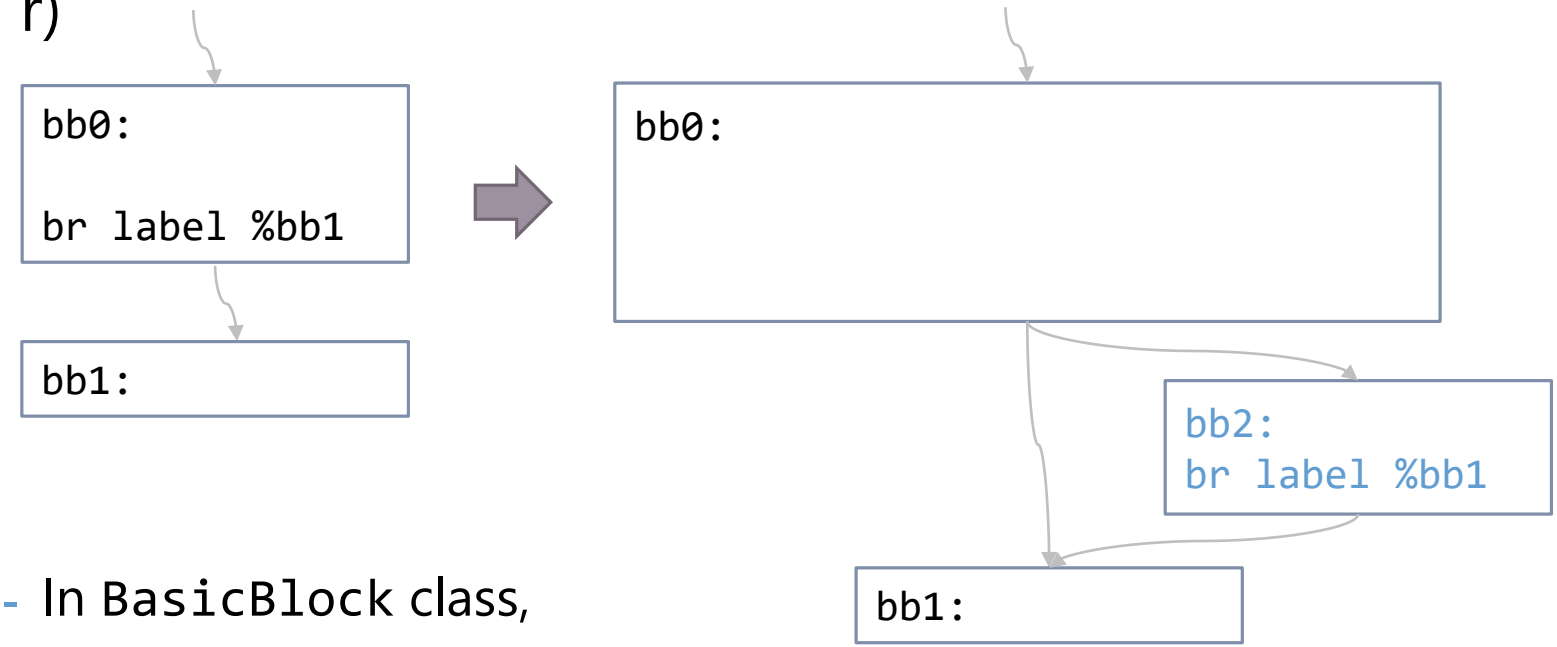
How To Change Control Flow

- Unconditional branch → Conditional branch (Detour)



How To Change Control Flow

- Unconditional branch → Conditional branch (Detour)



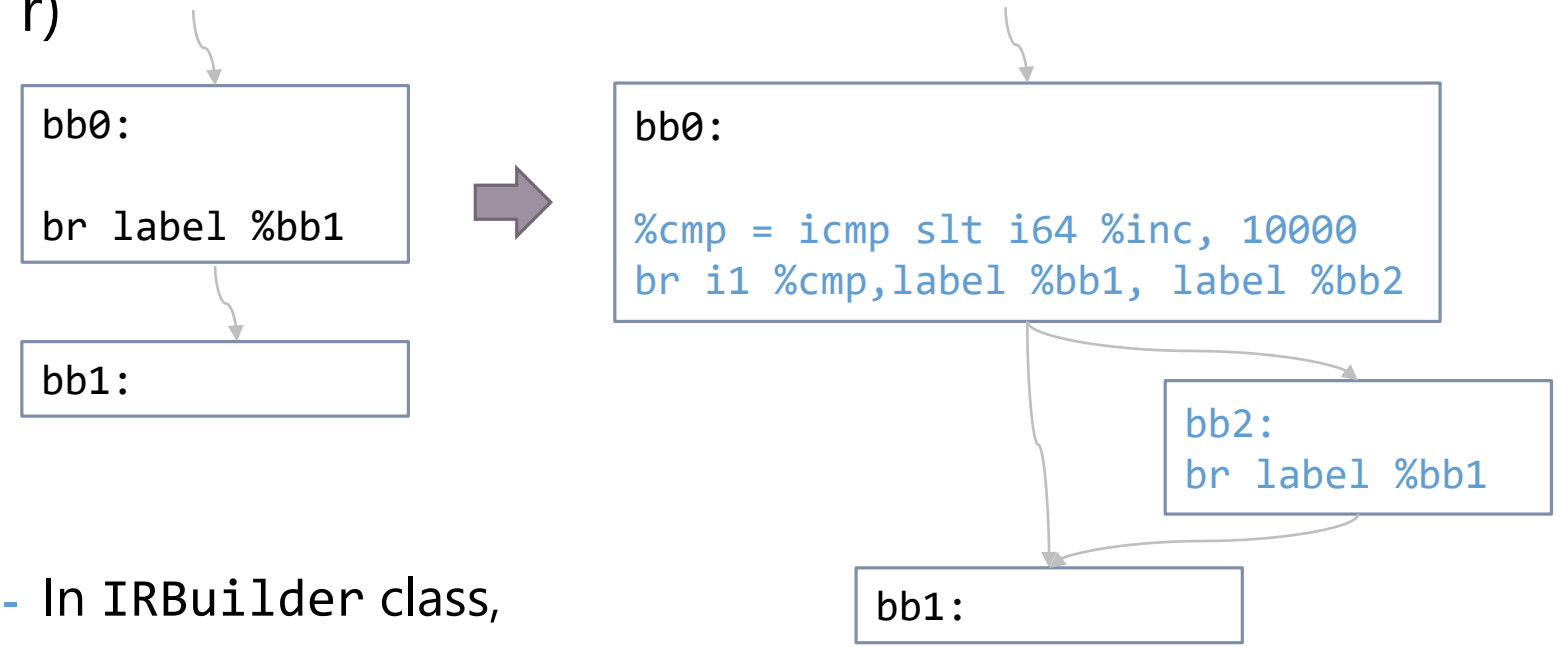
- In `BasicBlock` class,

Instruction *	**getTerminator** ()

BasicBlock *	**splitBasicBlock** (**Instruction** *I, **const Twine** &BBName="")

How To Change Control Flow

- Unconditional branch → Conditional branch (Detour)



- In `IRBuilder` class,

```
Value *   CreateICmpSLT (Value *LHS, Value *RHS, const Twine &Name="")
```

```
BranchInst *   CreateCondBr (Value *Cond, BasicBlock *True, BasicBlock *False,
```

Practice 2: printInt to printEven

- Goal
  - Learn how to change control flow

tutorial/optimization/control.c

```c
#include <stdio.h>

void printInt(int i) {
  printf("%d\n", i);
}

int main () {
  printInt(1);
  printInt(2);
  printInt(3);
  printInt(4);

  return 0;
}
```

- Steps
  1) Implement a FunctionPass
     - Get the function pointer of `printInt`
       - **Tip:** `M.getFunction("printInt");`
     - Change printInt to print only even numbers
       - 'srem' and 'icmp' instructions
  2) Run the pass with opt on control.c

How to Instrument Runtime Function Call

- Steps

1) Write a runtime function in C/C++
   - Use `extern "C"` for c++ functions due to name mangling

2) In Pass code, create a function with `getOrInsertFunction`
   - The function name and type in LLVM IR must match!
   - No need to fill the function (like a extern function)

3) Link the runtime function

How to Instrument Runtime Function Call

- Example
  - LoadTracerRuntime.cpp

```
extern "C"
void traceLoadInstr(void *addr, InstID instID) {
  // Do something
}
```

  - LoadTracerPass.cpp

```
traceLoadInstr = M.getOrInsertFunction(
    "traceLoadInstr",
    Type::getVoidTy(Context),
    Type::getInt64Ty(Context),  // Address
    Type::getInt64Ty(Context)); // Instruction ID
```

How to Instrument Runtime Function Call

- Example
  - LoadTracerPass.cpp

```
if(LoadInst *Load = dyn_cast<LoadInst>(&I)) {
  // Some Code
  actuals.resize(2);
  actuals[0] = CastedAddr;
  actuals[1] = ConstantInt::get(Type::getInt64Ty(Context), instID);
  CallInst::Create(traceLoadInstr, actuals, "", Load);
}
```

How to Instrument Runtime Function Call

- Example
  - Compilation Process
    - Compile the runtime code

```
$ clang++ -c –fpic LoadTracerRuntime.cpp -o LoadTracerRuntime.o
$ clang++ -shared -o LoadTracerRuntime.so LoadTracerRuntime.o
```

    - Run the pass that inserts runtime function calls

```
$ opt –load LoadTracer.so -traceload test.bc –o test.opt.bc
```

    - Link the runtime code

```
$ clang++ test.opt.bc -o test.exe -lLoadTracerRuntime
```

Practice 3: Dynamic CallCount

- Goal
  - Learn how to insert runtime function calls

- Steps
  1) Implement a DynCallCount pass that inherits `ModulePass`
     - Insert countCall() before every CallInst instructions
     - Insert printResult() before every Ret instructions in the function `main`
       - Code at `tutorial/optimization/runtime/callCount.c`
  2) Compile and run the pass
  3) Compile the runtime library
     - Type '`make`' at `tutorial/optimization/runtime`
  4) Link the runtime library

LLVM IR Metadata

- Contain information about an instruction or a function

Named Metadata

```
!llvm.module.flags = !{!0}
!llvm.ident = !{!1}

!0 = !{i32 1, !"wchar_size", i32 4}
!1 = !{!"clang version 8.0.0 (git@git.corelab.or.kr:corelab/clang.git 7973f6c2602b1e37f00a710ffa0c798a3f321e58) (git@git.corelab.or.kr:corelab/llvm.git c55bcb2f96806a3d9e5718497cede4665b27c8a4)"}
```

(Unnamed) Metadata

LLVM IR Metadata

- `Instruction` class
  - To manage Metadata

bool **hasMetadata** () **const**

Return true if this instruction has any metadata attached to it. More...

MDNode * **getMetadata** (**StringRef** Kind) **const**

Get the metadata of given kind attached to this **Instruction**. More...

void **setMetadata** (**StringRef** Kind, **MDNode** ***Node**)

  - Example

```
%tmp5 = load i32, i32* %arrayidx, align 4, !tbaa !3
```

```
!3 = !{!5, !5, i64 0}
```

LLVM IR Metadata

- `Instruction` class
  - Usage
    - Integer metadata

```
Constant* IdV = ConstantInt::get(Type::getInt64Ty(Context), Id);
Metadata* IdM = (Metadata*) ConstantAsMetadata::get(IdV);
vector<Metadata*> MDs = { IdM };
MDNode* Node = MDNode::get(Context, MDs);
I.setMetadata("tutorial", Node);
```

    - String metadata

```
MDString *MDStr = MDString::get(getLLVMContext(), Name);
MDNode* Node = MDNode::get(Context, MDStr);
I.setMetadata("tutorial", Node);
```

LLVM IR Metadata

- `In Module` class
  - To manage Metadata

NamedMDNode *	**getNamedMetadata** (**const Twine** &**Name**) **const** Return the first **NamedMDNode** in the module with the specified name. More...
NamedMDNode *	**getOrInsertNamedMetadata** (**StringRef Name**) Return the named **MDNode** in the module with the specified name. More...
void	**eraseNamedMetadata** (**NamedMDNode** *NMD) Remove the given **NamedMDNode** from this module and delete it. More...

  - Example

```
!llvm.module.flags = !{!0}
!llvm.ident = !{!1}
```

Practice 4: Instruction Namer

- Goal
  - Learn how to insert metadata to n LLVM IR

- Result
  1) Implement a `InstNamer` pass that inherits `ModulePass`
     - Assign an unique integer to each instruction
     - Insert metadata that contains the unique identifier to each instruc tion
  2) Compile and run the `InstNamer` pass