LLVM: Code Generation

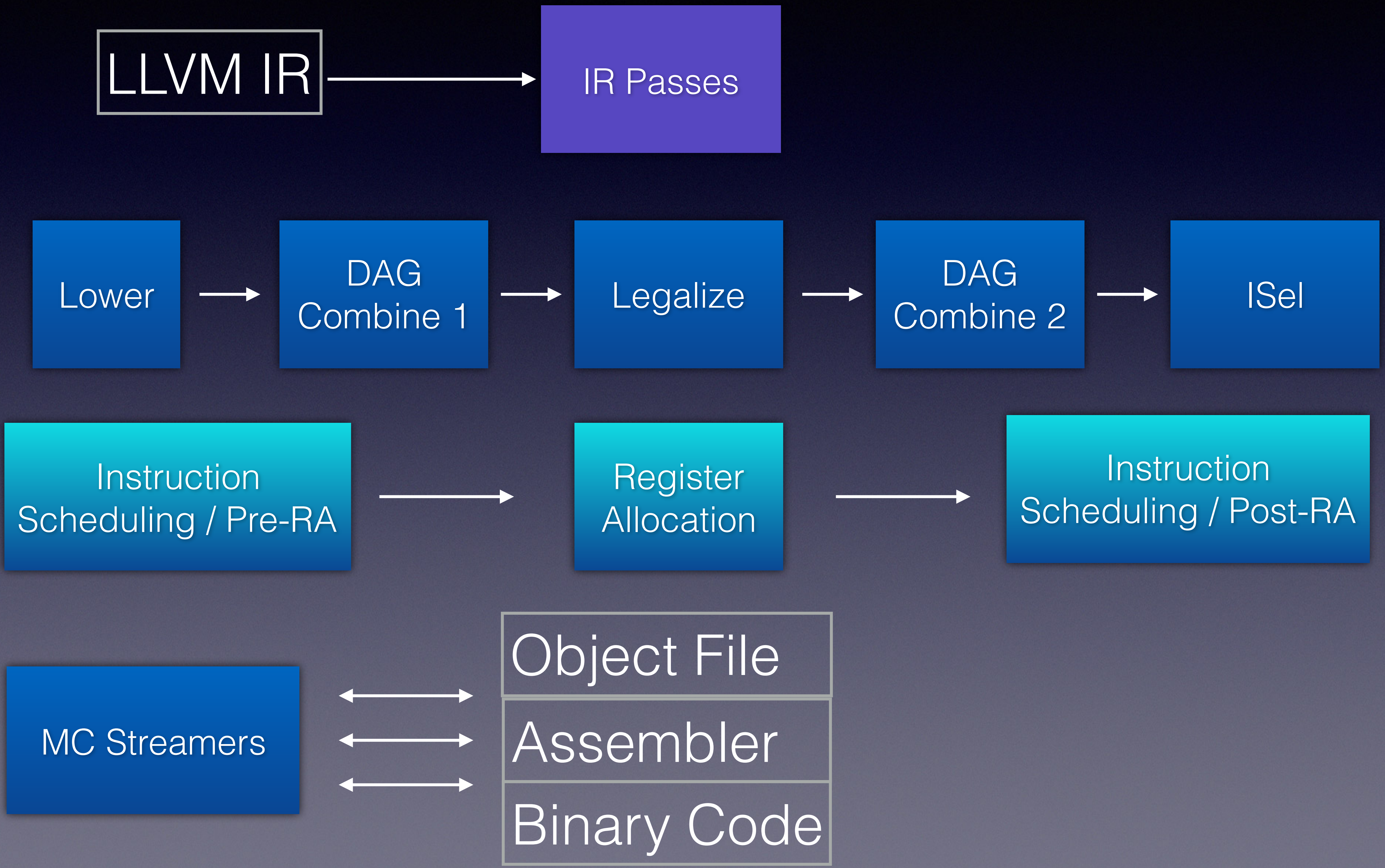
吴钊
IBM

Legal Disclaimer

Agenda

- Instruction Selection
- Instruction Scheduling
- Register Allocation
- Machine Code Emission
- Answer Pre-collected Questions
- Q & A

Pipeline


LLVM IR
IR Passes
Lower
DAG Combine 1
Legalize
DAG Combine 2
ISel
Instruction Scheduling / Pre-RA
Register Allocation
Instruction Scheduling / Post-RA
MC Streamers
Object File
Assembler
Binary Code

Example

```
// sum.c
int sum (int x, int y)
{
  return x+y;
}
```

LLVM IR

- clang -S -emit-llvm sum.c

```
; ModuleID = 'sum.c'
source_filename = "sum.c"
target datalayout = "e-m:o-i64:64-f80:128-n8:16:32:64-S128"
target triple = "x86_64-apple-macosx10.12.0"

define i32 @sum(i32 %x, i32 %y) {
entry:
  %x.addr = alloca i32, align 4
  %y.addr = alloca i32, align 4
  store i32 %x, i32* %x.addr, align 4
  store i32 %y, i32* %y.addr, align 4
  %0 = load i32, i32* %x.addr, align 4
  %1 = load i32, i32* %y.addr, align 4
  %add = add nsw i32 %0, %1
  ret i32 %add
}
```

Simple? NO.

LLVM IR

```
opt -mem2reg sum.ll | llvm-dis
```

```
define i32 @sum(i32 %x, i32 %y) {
entry:
  %add = add nsw i32 %x, %y
  ret i32 %add
}
```

Selection DAG

- Data Structure: Directed Acyclic Graph
- Lower than LLVM IR / First strictly backend IR
- Visit every basic block
- Nodes: instruction / operand
- Edges: use-def relationship, enforce order

Selection DAG Construction & Lowering

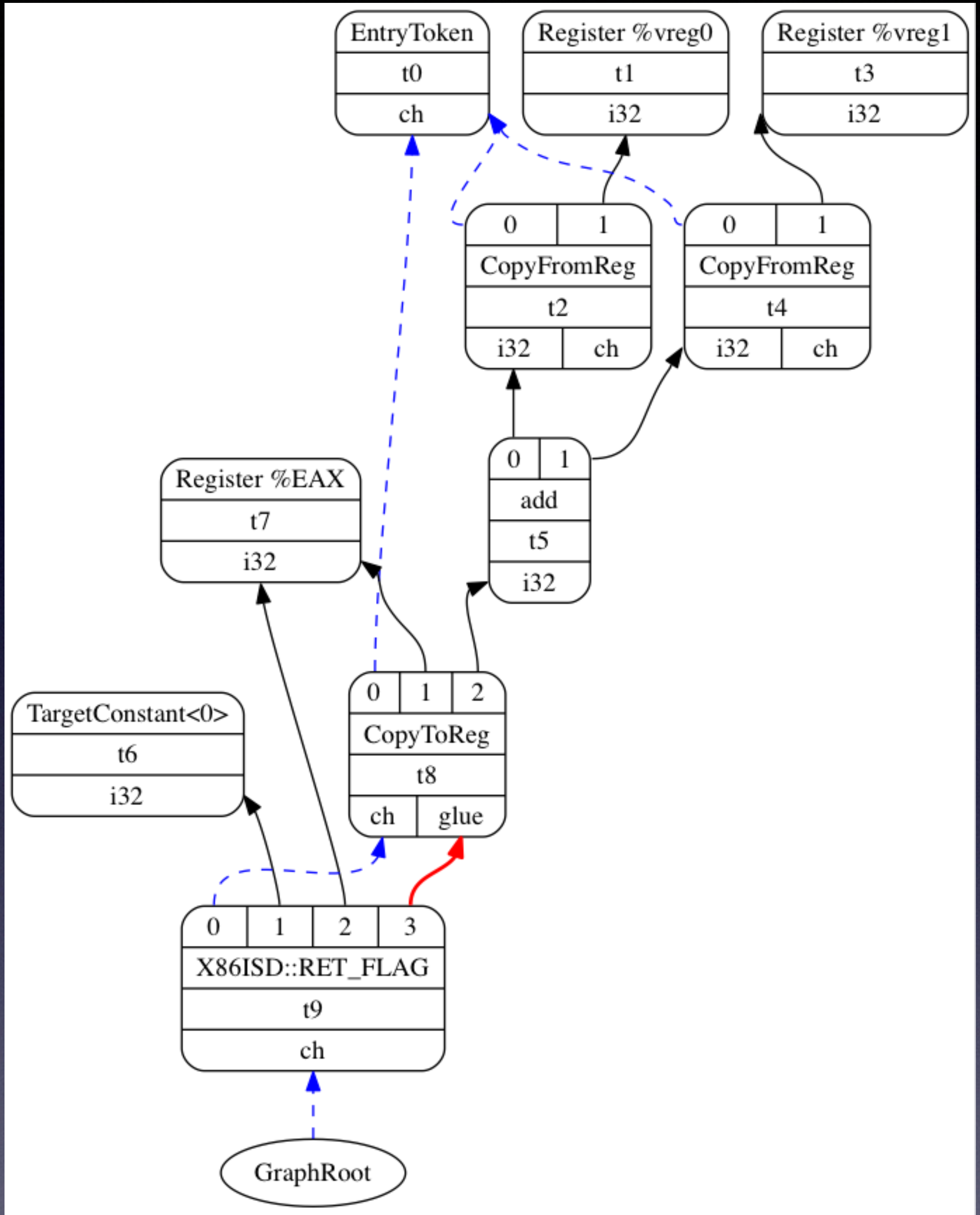
```
llc -view-dag-combine1-dags sum.ll
```

Line Meaning:

- Black: data flow dependence
- Blue: non-data flow dependence
- Red: MUST be glued

DAG State:

- Target-Independent node: add …
- Target-Specific node: X86ISD:RET_FLAG…



dag-combine1 input for sum:entry

Selection DAG Construction & Lowering

- Related LLVM Code
- Selection DAG : include/llvm/CodeGen/SelectionDAG.h
- Node(SDNode): include/llvm/CodeGen/SelectionDAGNodes.h
- Node Result Type: include/llvm/CodeGen/ValueTypes.h
- Target-Independent Node: include/llvm/CodeGen/ISDOpcodes.h
- Target-Specific Node: lib/Target/<Target>/<Target>ISelLowering.h
- Helper Class: SelctionDAGBuilder: lib/CodeGen/SelectionDAG/SelectionDAGBuilder.h
- Processdure: For example x86

Clang -> addPassesToEmitFile ->addPassesToGenerateCode- >X86PassConfig::addInstSelector->createX86ISelDag->**X86DAGToDAGISel** -> **SelectionDAGISel** ->runOnFunction->SelectAllBasicBlocks ->SelectBasicBlock->SelectionDAGBuilder::visit->visitXXX…

- Let me show your code…(If I have time…)

Selection DAG Combine

- Optimize on DAG
  - (add x, 0) -> (x)
- Make DAG closer to target
- Run twice
  - named combine-1 and combine-2
  - before and after DAG type-legalization
  - why twice?
- view it using llc -view-legalize-types-dags sum.ll

Selection DAG Combine

- Related Code
- DAGCombiner: lib/CodeGen/SelectionDAG/DAGCombiner.cpp
- Target-Specific Combiner: llvm/lib/Target/<Target>/<Target>ISelLowering.cpp
- Processdure:
- CodeGenAndEmitDAG -> Combine->…

Selection DAG Legalization

- Make the illegal be legal.
  - i64 on 32 bits backend
  - i147 integer
  - vector type operation
- has two type procedures: legalize normal type & legalize vector type
- has one operation legalization:
  - for example SDIV on x86 -> SDIVREM
- view it using llc -view-dag-combine2-dags sum.ll

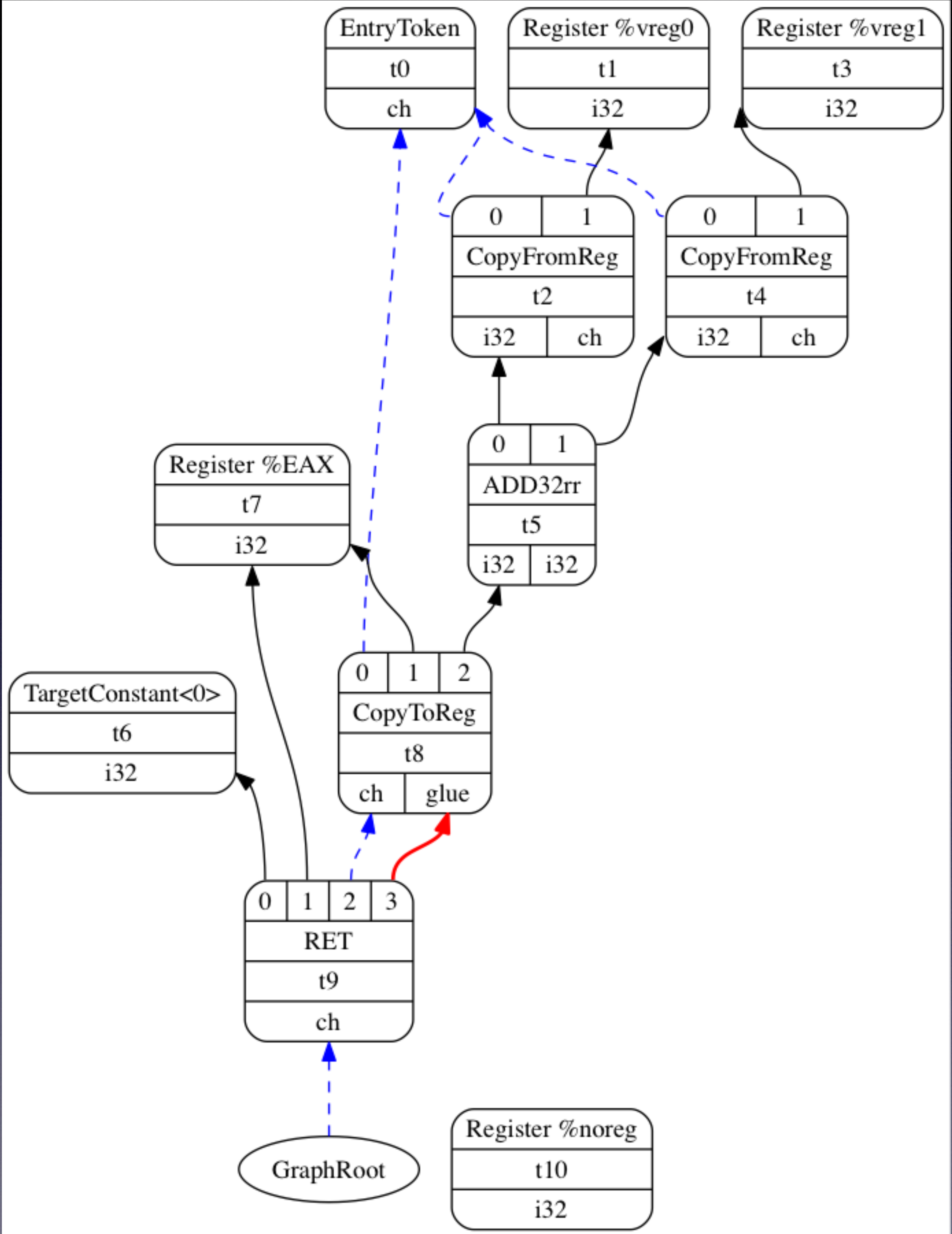
Selection DAG Legalization

- Related LLVM Code
- Common Type Legalization: lib/CodeGen/SelectionDAG/LegalizeTypes.h
- Operation Legalization: lib/CodeGen/SelectionDAG/LegalizeDAG.cpp
- Target-Specific: lib/Target/<Target_Name>/<Target>ISelLowering.cpp
- Processdure:

CodeGenAndEmitDAG -> LegalizeTypes / LegalizeVectors / Legalize ->…

Instruction Selection

- Turn target-independent to target-specific node
- Based on pattern match
- Pattern-Match is generated by Table-Gen
    - def RET : PseudoI<(outs), (ins i32imm:$adj, variable_ops), [(X86retflag timm:$adj)]>;
    - X86retflag -> def X86retflag : SDNode<"X86ISD::RET_FLAG", SDTX86Ret, [SDNPHasChain, SDNPOptInGlue, SDNPVariadic]>;
- view it using llc -view-sched-dags sum.ll



scheduler input for sum:entry

Instruction Selection

- Related LLVM Code
- Target DAGToDAGISel: lib/Target/<Target>/<Target>ISelDAGToDAG.cpp
- Target Code by Table-Gen: lib/Target/<Target>/<Target>.td
  - which will be generated .inc file (<Target>GenDAGISel.inc) in the build dir.
  - Pattern-Match process is somehow complex, need Table-Gen knowledge background. I have no time to explain, wish have chance to do it. If you are interested, contact me offline, I would like to write one post for you later.
- Processdure:

CodeGenAndEmitDAG->DoInstructionSelection->Select->SelectCode(Generated by Table-Gen)->SelectCodeCommon->…

View DAG State

LLC Option	Brief Description
-view-dag-combine1-dags	Before DAG combine 1
-view-legalize-types-dags	Before legalize type
-view-dag-combine-lt-dags	After legalize type 2 before DAG combine
-view-legalize-dags	Before legalization
-view-dag-combine2-dags	Before DAG combine 2
-view-isel-dags	Before instruction selection
-view-sched-dags	After ISel before scheduling

Instruction Scheduling

- CPU can’t execute DAG
  - DAG -> linear list (SSA form)->MachineInstr
- Another IR: MachineInstr + MachineBB + Machine Function
- Run twice: before Register Allocation(RA) and after RA.
  - Pre-RA: work on SDNode
  - Post-RA: work on MachineInstr
- Target can have own hook scheduler
  - such as X86Scheduler.td (Table-Gen file)

Instruction Scheduling

- Related Code
- Scheduler: Source / RegPressure / Hybrid / View / ILP
    - lib/CodeGen/SelectionDAG/ScheduleDAGRRList.cpp
- InstrEmitter: lib/CodeGen/SelectionDAG/InstrEmitter.h
- Procedure:

Pre-RA: CodeGenAndEmitDAG->createScheduler->Run->EmitScheduler->InstrEmitter::EmitCode->…

Post-RA: Register Allocation Ctor(such as RAGreedy)->initializeMachineSchedulerPass->MachineScheduler::runOnFunction->createMachineScheduler->scheduleRegions->scheule->….

Machine Instructions

- very close to target
- LLVM IR -> DAG -> Machine Instructions
- view the information using llc -print-machineinstrs sum.ll
  - can specify pass name to -print-machineinstrs

Machine Instructions

```
# After Instruction Selection:
# Machine code for function sum: Properties: <SSA, tracking liveness,
HasVRegs>
Function Live Ins: %EDI in %vreg0, %ESI in %vreg1

BB#0: derived from LLVM BB %entry
    Live Ins: %EDI %ESI
        %vreg1<def> = COPY %ESI; GR32:%vreg1
        %vreg0<def> = COPY %EDI; GR32:%vreg0
        %vreg2<def,tied1> = ADD32rr %vreg0<tied0>, %vreg1, %EFLAGS<imp-
def,dead>; GR32:%vreg2,%vreg0,%vreg1
        %EAX<def> = COPY %vreg2; GR32:%vreg2
        RET 0, %EAX

# End machine code for function sum.
```

Register Allocation

- Unlimited virtual registers -> limited physical registers
  - Compiler Textbook: K-Coloring graph problem
  - Fact: Spill very often and should be the first
- LLVM Register Allocator
  - Basic / Greedy (default since 3.0) / Fast / PBQP
- view it: llc -march=sparc(or omit it) -debug-only=regalloc sum.ll

Register Allocation

```
selectOrSplit GR64_NOSP:%vreg4 [16r,80r:0)  0@16r w=4.353448e-03
AllocationOrder(GR64) = [ %RAX %RCX %RDX %RSI %RDI %R8 %R9 %R10 %R11 %RBX
%R14 %R15 %R12 %R13 %RBP ]
AllocationOrder(GR64_NOSP) = [ %RAX %RCX %RDX %RSI %RDI %R8 %R9 %R10 %R11
%RBX %R14 %R15 %R12 %R13 %RBP ]
hints: %RSI
assigning %vreg4 to %RSI: SIL [16r,80r:0)  0@16r

selectOrSplit GR64_with_sub_8bit:%vreg3 [32r,80r:0)  0@32r w=inf
AllocationOrder(GR64_with_sub_8bit) = [ %RAX %RCX %RDX %RSI %RDI %R8 %R9 %R10
%R11 %RBX %R14 %R15 %R12 %R13 %RBP ]
hints: %RDI
assigning %vreg3 to %RDI: DIL [32r,80r:0)  0@32r

selectOrSplit GR32:%vreg2 [80r,96r:0)  0@80r w=inf
AllocationOrder(GR32) = [ %EAX %ECX %EDX %ESI %EDI %R8D %R9D %R10D %R11D %EBX
%EBP %R14D %R15D %R12D %R13D ]
hints: %EAX
assigning %vreg2 to %EAX: AH [80r,96r:0)  0@80r AL [80r,96r:0)  0@80r
```

Register Allocation

- Related Code
- Register Allocator (Greedy): lib/CodeGen/RegAllocGreedy.cpp
- Procedure:
- addPassesToEmitFile->addPassesToGenerateCode->PassConfig->addMachinePasses()->createRegAllocPass->createTargetRegisterAllocator->createGreedyRegisterAllocator->RAGreedy ctor->intialize many passes
- RegisterCoalescing could be hooked by <Target>InstrInfo

Post-RA

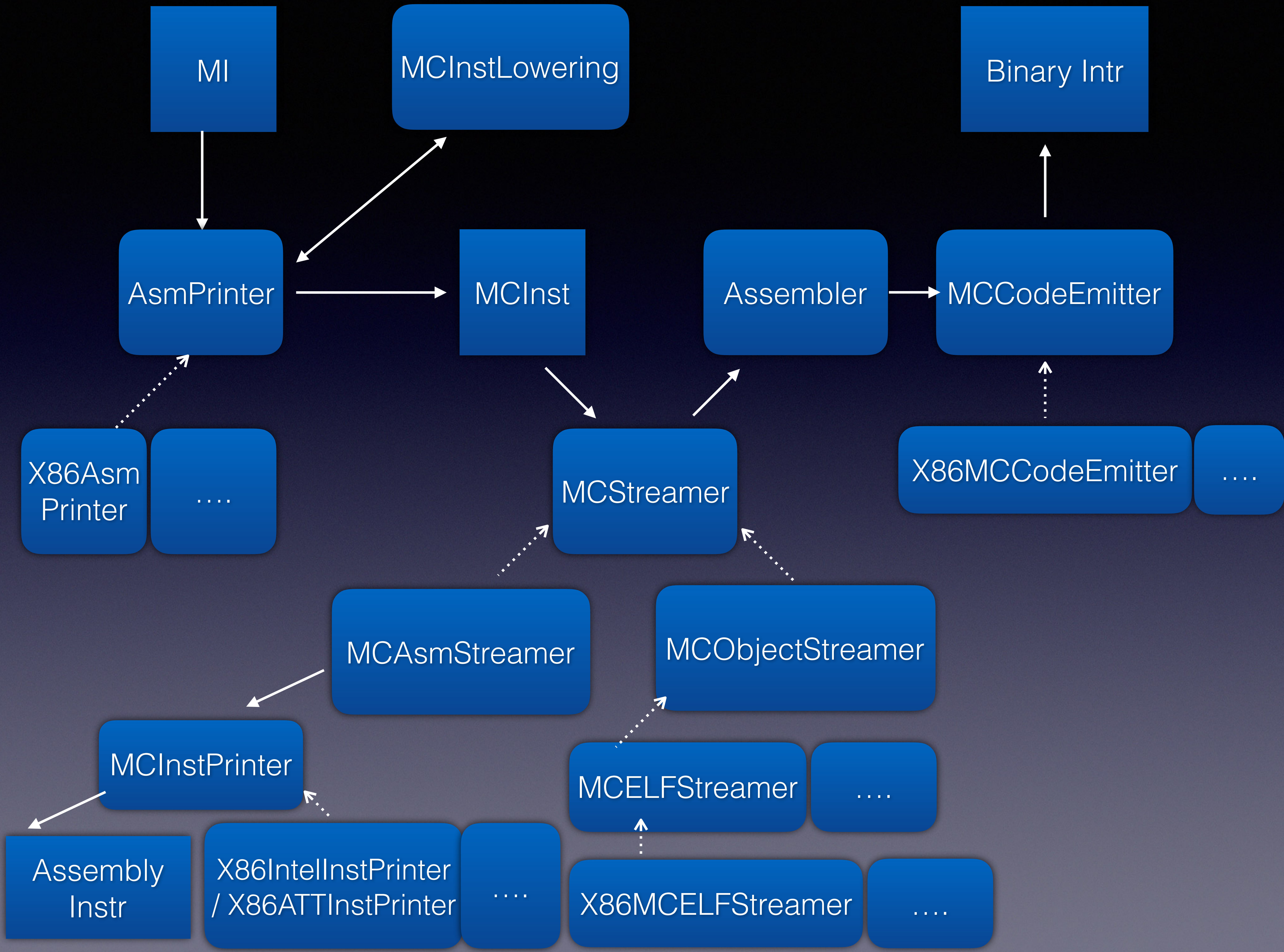
- Prologue and epilogue
  - Prologue sets up the stack frame and callee-saved registers during the beginning of a function
  - Epilogue cleans up the stack frame prior to function return
  - lib/Target/<Target>/<Target>FrameLowering.cpp
  - <Target>FrameLowering::emitPrologue and emitEpilogue

Post-RA

- Abstract Frame Indexes Elimination
    - converting each frame index to a real stack offset for all machine instructions that contain stack references
    - lib/Target/<Target>/<Target>RegisterInfo.cpp
    - <Target>RegisterInfo::eliminateFrameIndex

MC Inst & Code Emission

- MC Instruction **lightweight** representation of one single instruction
  - carry less information compared by MI.
  - defined by backend or disassembler from binary
- LLVM can convert MC Instruction directly into machine code


MI
MCInstLowering
Binary Intr
AsmPrinter
MCInst
Assembler
MCCodeEmitter
X86Asm Printer
….
MCStreamer
X86MCCodeEmitter
….
MCAsmStreamer
MCObjectStreamer
MCInstPrinter
MCELFStreamer
….
Assembly Instr
X86IntelInstPrinter / X86ATTInstPrinter
….
X86MCELFStreamer
….

MCInst & Code Emission

- Related Code
- MC Framework: include/llvm/MC
- Procedure:

LLVMTargetMachine::addPassesToEmitFile->createAsmPrinter->runOnMachineFunction->EmitFunctionBody->EmitInstruction->…

Pre-Collected Questions

- Q: 能否以类和函数相互调用的过程大概讲一下LLVM执行的流程，网上大部分是关于LLVM各个模块相互关系，对于各个模块的对应代码及其相互调用函数不清楚，现在我对llvm理解的状态是知道大概每个过程，但对过程执行缺乏理解
- A:

Pre-Collected Questions

- Q: 如何阅读LLVM源码，相对其他代码，阅读LLVM源码的效率太低
- A:

Pre-Collected Questions

- Q: 推荐一些LLVM小白阅读材料，LLVM官网上的需要基础，函数基本只有一两句话的解释，如何使用这些函数不太理解。
- A:

Pre-Collected Questions

- Q: AST和IR遍历(pass), IR 遍历，有些概念不理解比如use-def chain和def-use chain，还有这些遍历一般用于哪些情况下。
- A:
- use-def (UD) chain : every use knows its single definition / if use one constant definition -> Constant Propagation
  - a =1; c = a + b; // c uses a
- def-use (DU) chain: every definition knows its uses / if definition without no use / DCE.
  - a = x / y; // a doesn't have use.

Pre-Collected Questions

- Q: DAG的数据结构及其在后端代码生成各个过程中DAG结构的变化
- A:

Pre-Collected Questions

- Q: 自定义InstrInfo在llvm中如何被调用使用

- A: 增加你的指令在<Target>InstrInfo.td, 通常会被<Target>.td包含。LLVM将会生产<Target>GenInstrInfo.inc文件(Pattern-Match，SelectCode)和<Target>ASMWriter.inc(<Target>ASMWriter::printInstruction)

- 参考范例: PPCInstrInfo.td : rotl

- 推荐Pattern-Match文章：https://github.com/draperlaboratory/fracture/wiki/How-TableGen's-DAGISel-Backend-Works

Q & A?

Thanks!
See you afternoon!
Enjoy C++ next!