LLVM Tutorial

Back-end Compilation

2019. 04. 11

Back-end Compilation

- Translate IR into **machine code** code

- Overall Process

$$\begin{tikzcd}\text{IR} \arrow[r] & \text{SelectionDAG} \arrow[r] & \text{MachineInst} \arrow[r] & \text{MCInst}\end{tikzcd}$$

  - Convert IR to SelectionDAG
  - Legalize SelectionDAG
  - Select instructions from SelectionDAG
  - Schedule instructions in SelectionDAG
  - Allocate registers
  - Insert prologue and epilogue code
  - Emit machine code

SelectionDAG

- DAG (Directed Acyclic Graph)
  - Node: Operation (SDNode)
  - Edge: Dependence
    - Data, Order, Scheduling

Optimize DAG by combining SDNode nodes

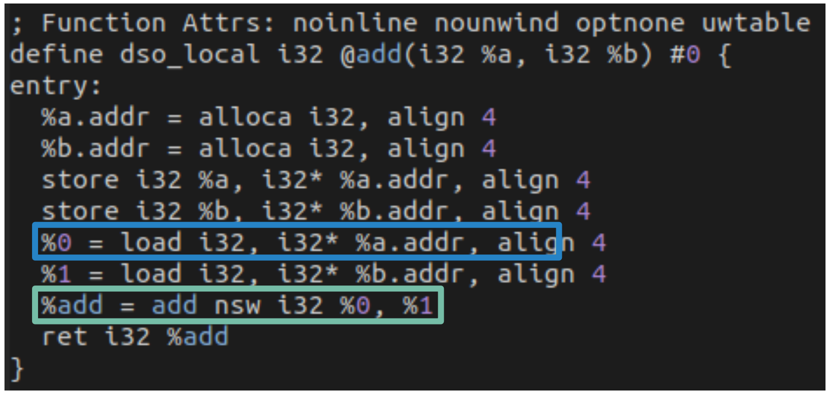
- `llc`: Static Assembler
  - To see SelectionDAG before optimization,

```
$ llc -fast-isel=false -view-dag-combine1-dags test.ll
$ dot -Tpdf xxx.dot -o xxx.pdf
```

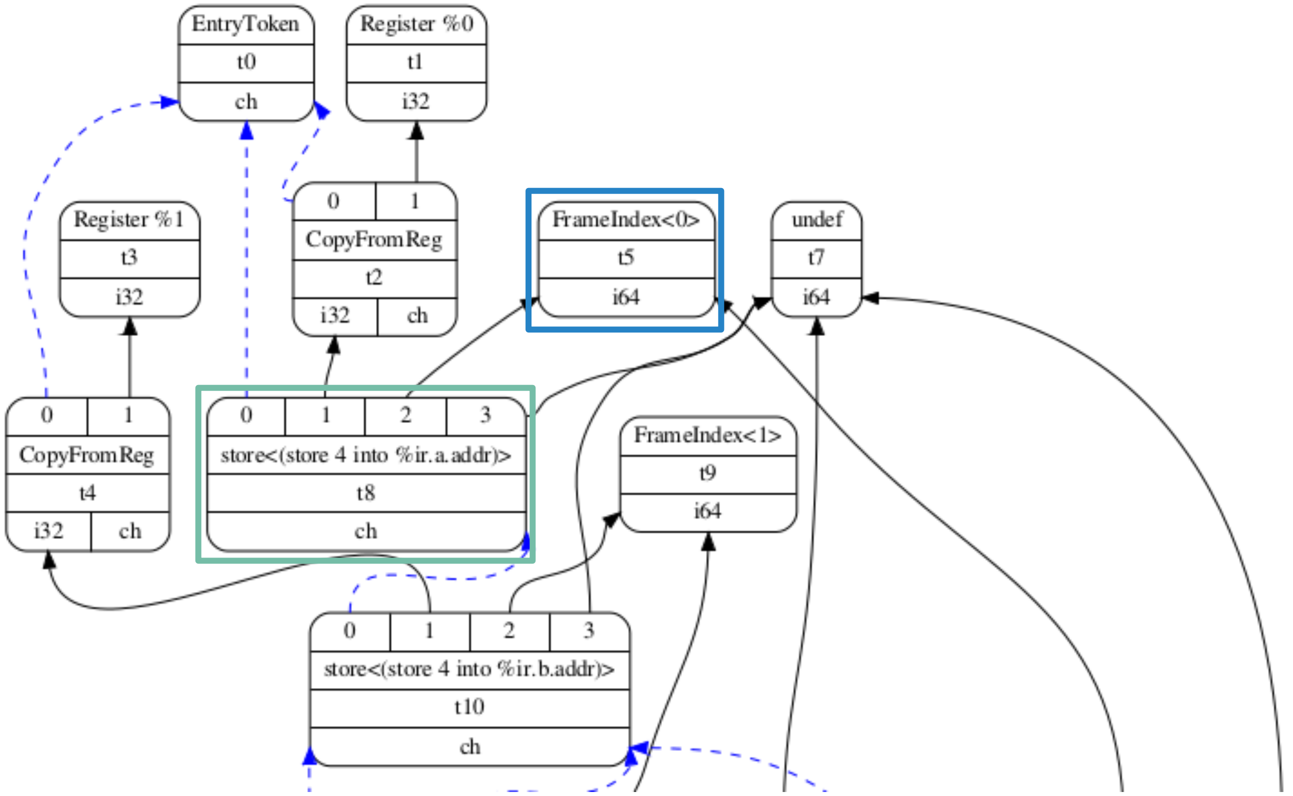
  - Note) Fast instruction selection must be disabled!

SelectionDAG

- Example (add.ll)

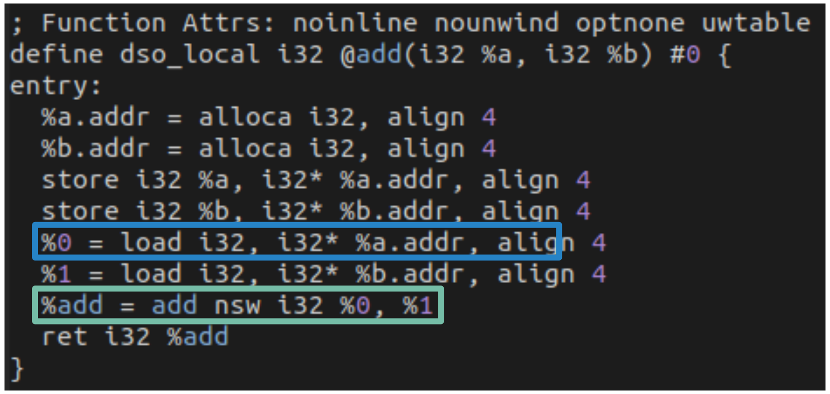


```
; Function Attrs: noinline nounwind optnone uwtable
define dso_local i32 @add(i32 %a, i32 %b) #0 {
entry:
  %a.addr = alloca i32, align 4
  %b.addr = alloca i32, align 4
  store i32 %a, i32* %a.addr, align 4
  store i32 %b, i32* %b.addr, align 4
  %0 = load i32, i32* %a.addr, align 4
  %1 = load i32, i32* %b.addr, align 4
  %add = add nsw i32 %0, %1
  ret i32 %add
}
```

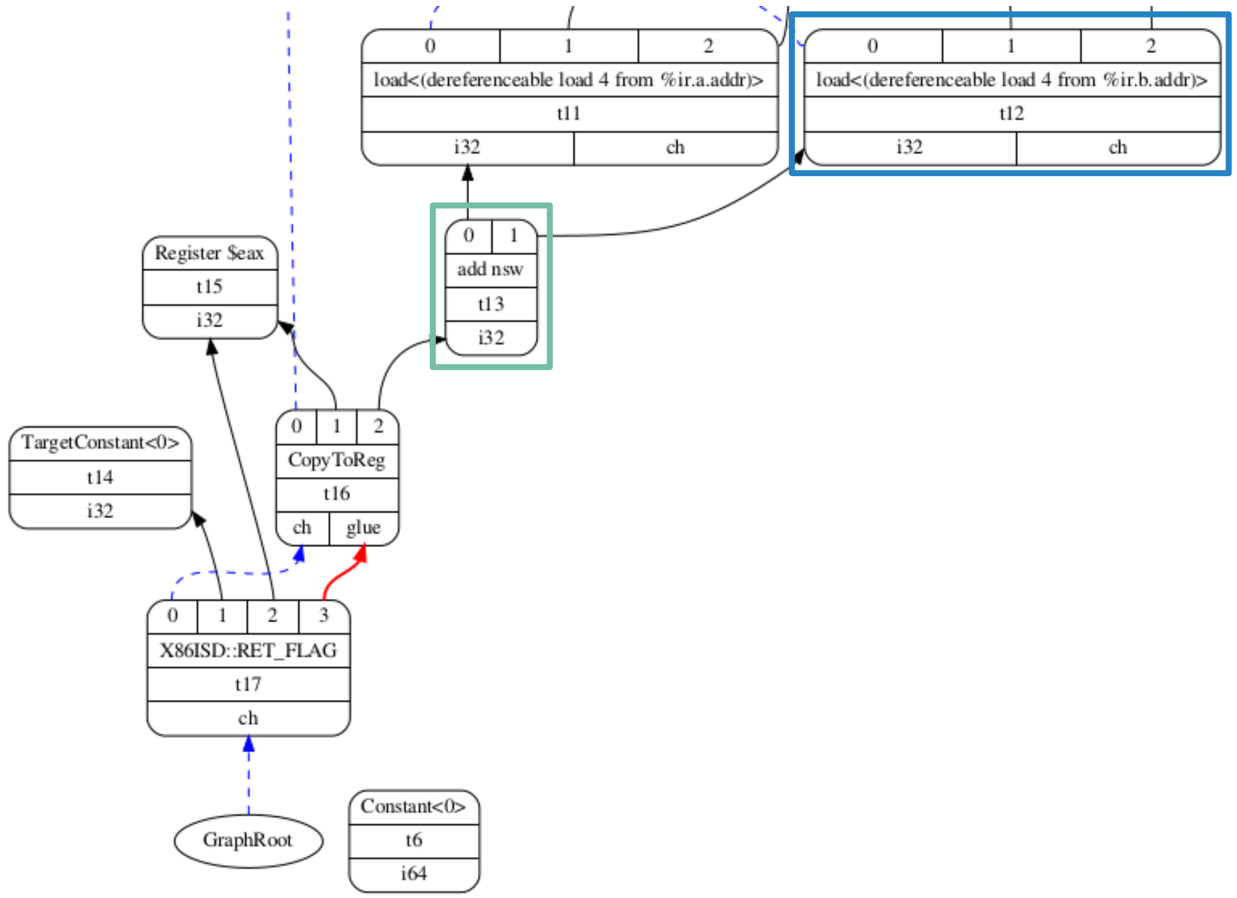


SelectionDAG

- Example (add.ll)



```
; Function Attrs: noinline nounwind optnone uwtable
define dso_local i32 @add(i32 %a, i32 %b) #0 {
entry:
  %a.addr = alloca i32, align 4
  %b.addr = alloca i32, align 4
  store i32 %a, i32* %a.addr, align 4
  store i32 %b, i32* %b.addr, align 4
  %0 = load i32, i32* %a.addr, align 4
  %1 = load i32, i32* %b.addr, align 4
  %add = add nsw i32 %0, %1
  ret i32 %add
}
```



load<(dereferenceable load 4 from %ir.a.addr)> t11 i32 ch
load<(dereferenceable load 4 from %ir.b.addr)> t12 i32 ch
add nsw t13 i32
Register $eax t15 i32
TargetConstant<0> t14 i32
CopyToReg t16 ch glue
X86ISD::RET_FLAG t17 ch
GraphRoot
Constant<0> t6 i64

SelectionDAG Legalization

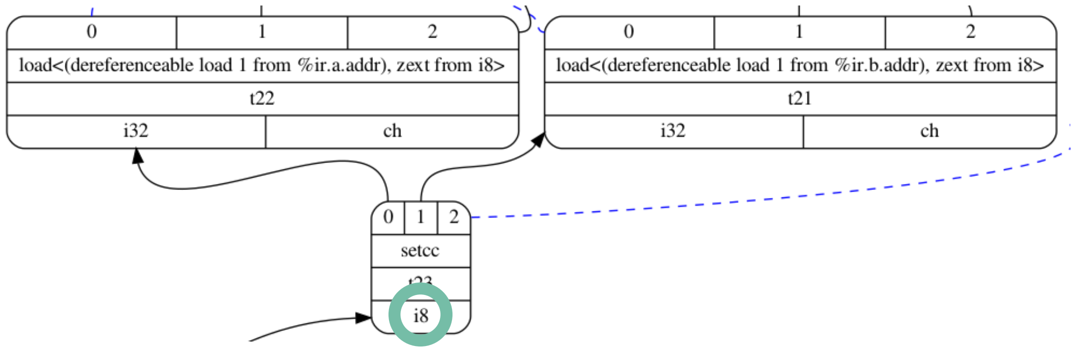
- Legalize SelectionDAG for a target
  - Type legalization
    - Promote smaller data types to larger data types
    - Truncate larger data types into smaller data types
    - Ex) If the target only supports the i64 type, change the i32 type to the i64 type.
  - Instruction legalization

- To see SelectionDAG before and after legalization,

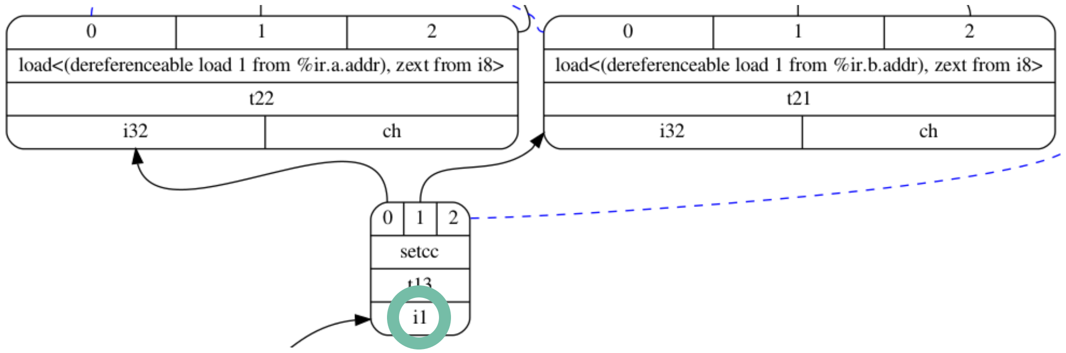
```
$ llc -fast-isel=false –view-legalize-types-dags test.ll
$ llc -fast-isel=false –view-legalize-dags test.ll
$ llc -fast-isel=false –view-dag-combine2-dags test.ll
```

SelectionDAG Legalization

- Example 1
  - Before type legalization

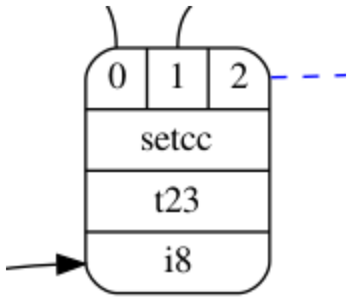

0
1
2
load<(dereferenceable load 1 from %ir.a.addr), zext from i8>
t22
i32
ch
0
1
2
load<(dereferenceable load 1 from %ir.b.addr), zext from i8>
t21
i32
ch
0
1
2
setcc
t23
i8

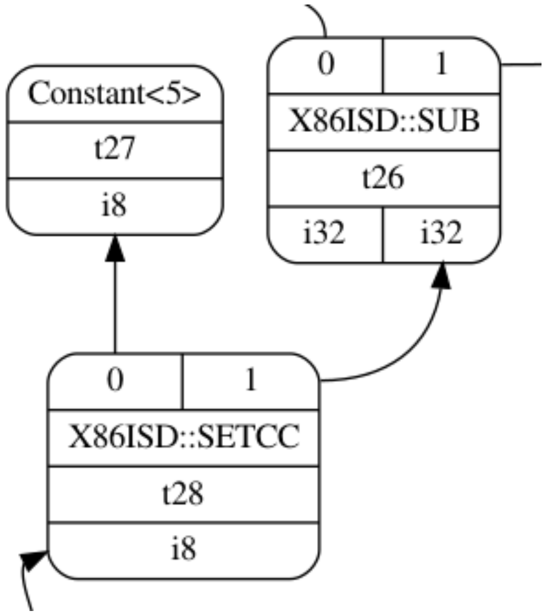
  - After type legalization


0
1
2
load<(dereferenceable load 1 from %ir.a.addr), zext from i8>
t22
i32
ch
0
1
2
load<(dereferenceable load 1 from %ir.b.addr), zext from i8>
t21
i32
ch
0
1
2
setcc
t13
i1

SelectionDAG Legalization

- Example 2
  - Before legalization
  - After legalization





Instruction Selection

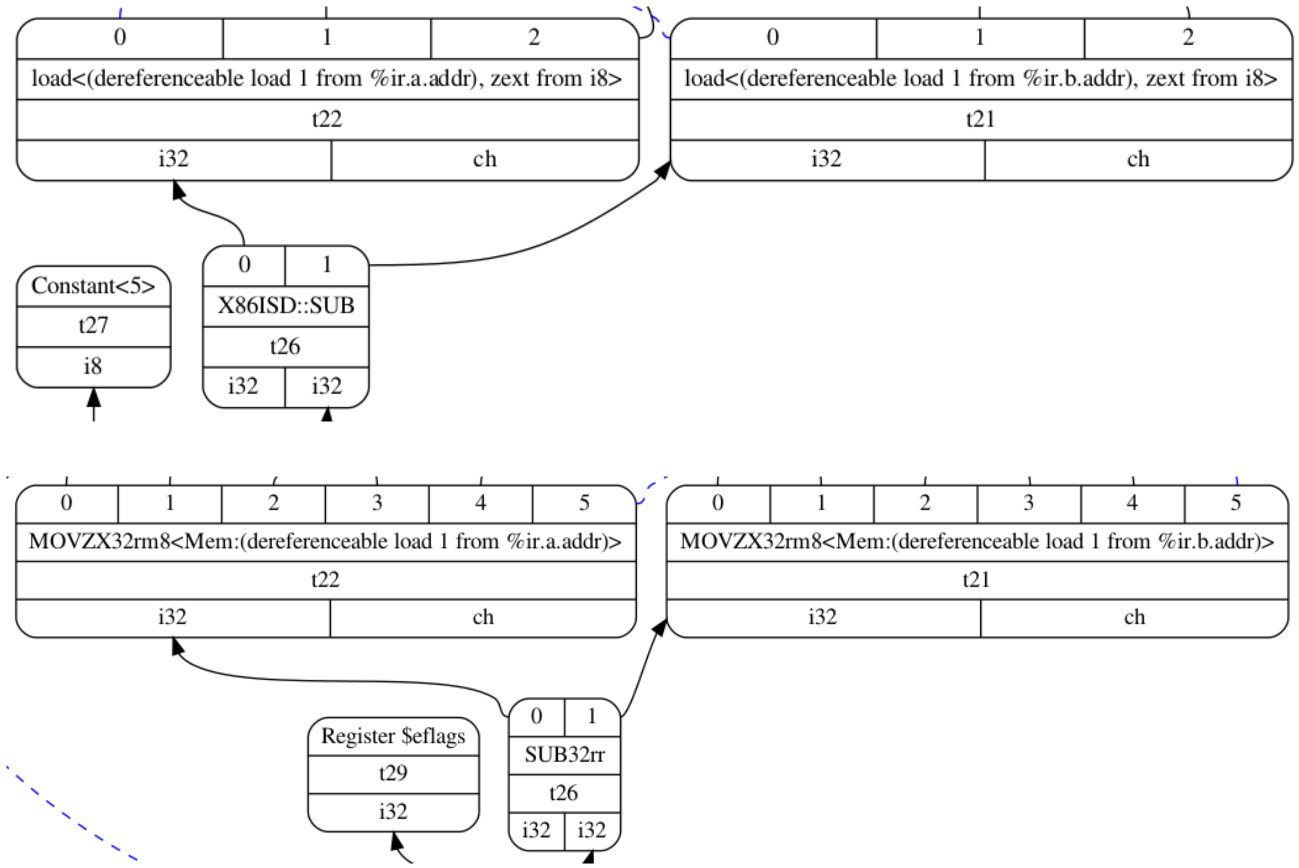
- Map target-independent instructions to target-specific i nstructions
  - `llvm/lib/Target/XXX/XXXInstrYYY.td`
    - ex) llvm/lib/Target/X86/X86InstrArithmetic.td
    - Contain the definitions of instructions of a target

- To see SelectionDAG after instruction selection,

```
$ llc -fast-isel=false –view-sched-dags test.ll
```

Instruction Selection

- Example



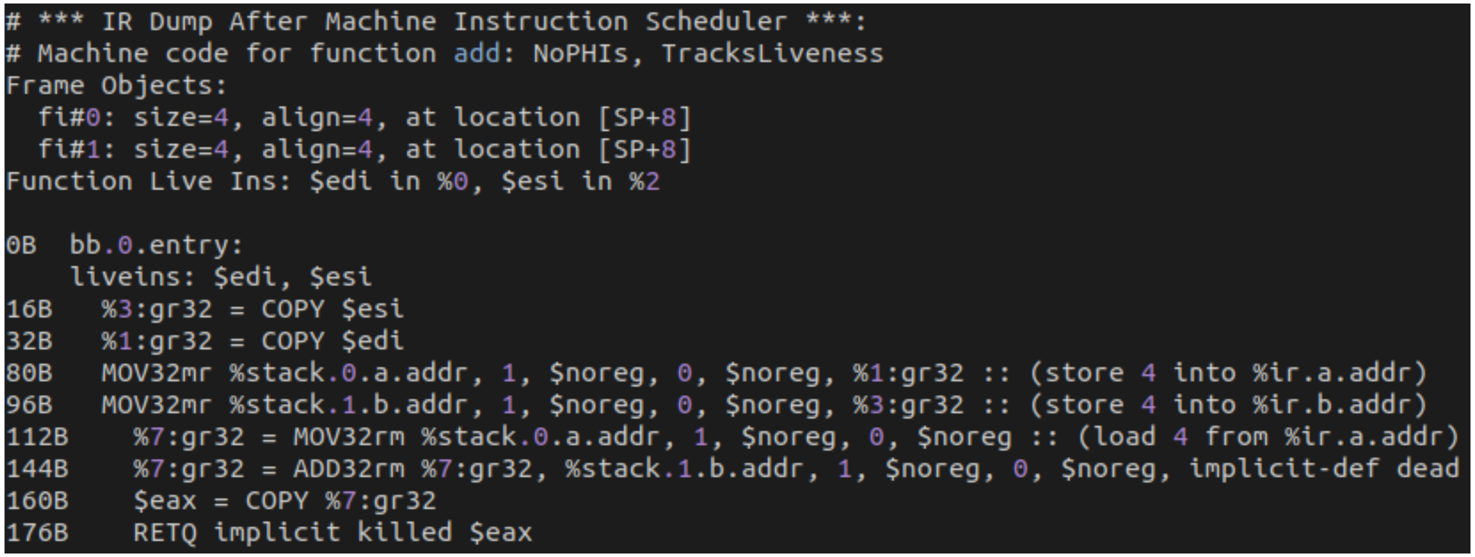
0
1
2
load<(dereferenceable load 1 from %ir.a.addr), zext from i8>
t22
i32
ch
0
1
2
load<(dereferenceable load 1 from %ir.b.addr), zext from i8>
t21
i32
ch
Constant<5>
t27
i8
0
1
X86ISD::SUB
t26
i32
i32
0
1
2
3
4
5
MOVZX32rm8<Mem:(dereferenceable load 1 from %ir.a.addr)>
t22
i32
ch
0
1
2
3
4
5
MOVZX32rm8<Mem:(dereferenceable load 1 from %ir.b.addr)>
t21
i32
ch
Register $eflags
t29
i32
0
1
SUB32rr
t26
i32
i32

Instruction Scheduling

- Assign the order of execution of instructions from the DAG
  - SelectionDAG becomes **a list of MachineInstr**
  - SelectionDAG nodes are destroyed

Instruction Scheduling

- Example



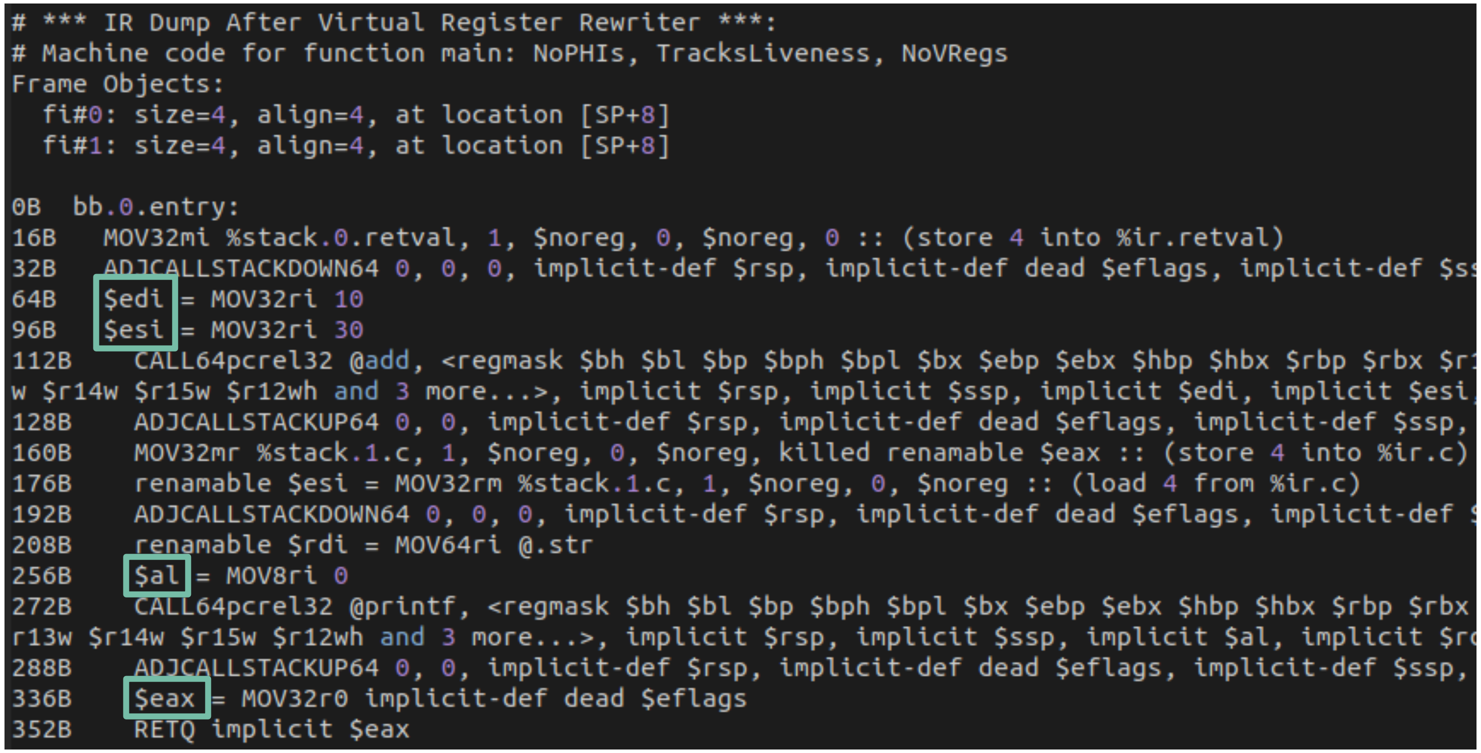
```
# *** IR Dump After Machine Instruction Scheduler ***:
# Machine code for function add: NoPHIs, TracksLiveness
Frame Objects:
  fi#0: size=4, align=4, at location [SP+8]
  fi#1: size=4, align=4, at location [SP+8]
Function Live Ins: $edi in %0, $esi in %2

0B   bb.0.entry:
     liveins: $edi, $esi
16B    %3:gr32 = COPY $esi
32B    %1:gr32 = COPY $edi
80B    MOV32mr %stack.0.a.addr, 1, $noreg, 0, $noreg, %1:gr32 :: (store 4 into %ir.a.addr)
96B    MOV32mr %stack.1.b.addr, 1, $noreg, 0, $noreg, %3:gr32 :: (store 4 into %ir.b.addr)
112B     %7:gr32 = MOV32rm %stack.0.a.addr, 1, $noreg, 0, $noreg :: (load 4 from %ir.a.addr)
144B     %7:gr32 = ADD32rm %7:gr32, %stack.1.b.addr, 1, $noreg, 0, $noreg, implicit-def dead
160B     $eax = COPY %7:gr32
176B     RETQ implicit killed $eax
```

Register Allocation

- Assign physical registers to virtual registers
  - Virtual registers can be infinite, but the physical registers of a mac hine are limited
  - llvm/lib/Target/XXX/XXXRegisterInfo.td
    - Ex) llvm/lib/Target/X86/X86RegisterInfo.td
    - Contain the definitions of registers of a target

Register Allocation

- Example



```
# *** IR Dump After Virtual Register Rewriter ***:
# Machine code for function main: NoPHIs, TracksLiveness, NoVRegs
Frame Objects:
  fi#0: size=4, align=4, at location [SP+8]
  fi#1: size=4, align=4, at location [SP+8]

0B  bb.0.entry:
16B   MOV32mi %stack.0.retval, 1, $noreg, 0, $noreg, 0 :: (store 4 into %ir.retval)
32B   ADJCALLSTACKDOWN64 0, 0, 0, implicit-def $rsp, implicit-def dead $eflags, implicit-def $ss
64B   $edi = MOV32ri 10
96B   $esi = MOV32ri 30
112B    CALL64pcrel32 @add, <regmask $bh $bl $bp $bph $bpl $bx $ebp $ebx $hbp $hbx $rbp $rbx $r
w $r14w $r15w $r12wh and 3 more...>, implicit $rsp, implicit $ssp, implicit $edi, implicit $esi,
128B    ADJCALLSTACKUP64 0, 0, implicit-def $rsp, implicit-def dead $eflags, implicit-def $ssp,
160B    MOV32mr %stack.1.c, 1, $noreg, 0, $noreg, killed renamable $eax :: (store 4 into %ir.c)
176B    renamable $esi = MOV32rm %stack.1.c, 1, $noreg, 0, $noreg :: (load 4 from %ir.c)
192B    ADJCALLSTACKDOWN64 0, 0, 0, implicit-def $rsp, implicit-def dead $eflags, implicit-def $
208B    renamable $rdi = MOV64ri @.str
256B    $al = MOV8ri 0
272B    CALL64pcrel32 @printf, <regmask $bh $bl $bp $bph $bpl $bx $ebp $ebx $hbp $hbx $rbp $rbx
r13w $r14w $r15w $r12wh and 3 more...>, implicit $rsp, implicit $ssp, implicit $al, implicit $r
288B    ADJCALLSTACKUP64 0, 0, implicit-def $rsp, implicit-def dead $eflags, implicit-def $ssp,
336B    $eax = MOV32r0 implicit-def dead $eflags
352B    RETQ implicit $eax
```

Code Emission

- Lower the code
  - from code generator abstractions (`MachinInstr`)
  - to machine code layer abstractions (`MCInst`)

- Emit assembly code (*.s)

Machine Code Optimization

How to Optimize Machine Code

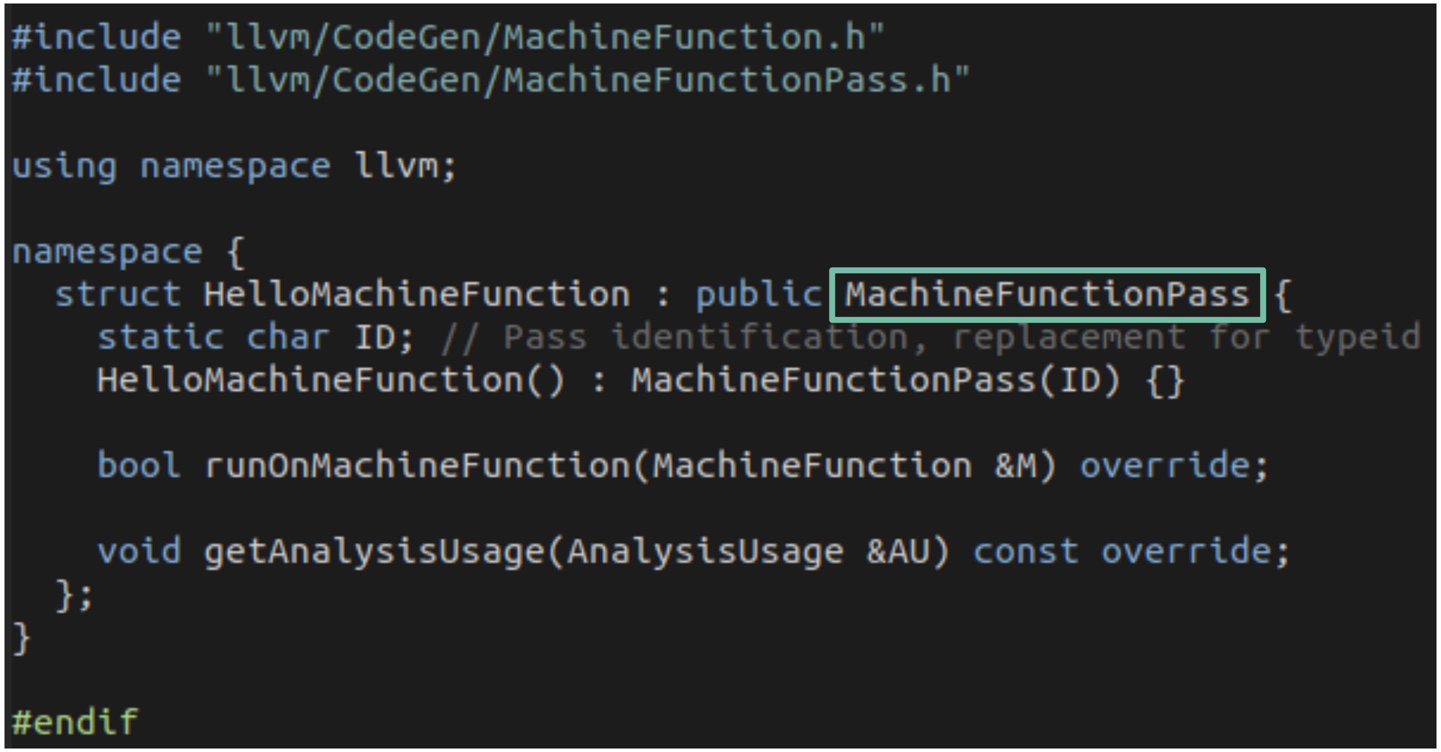
- Almost silimar to IR optimization
- LLVM Passes that inhertis **`MachineFunctionPass`**
  - Implement optimization code in `runOnMachineFunction`

- Dynamically loaded at run-time
  - llc -load PASS_LIBRARY_PATH -PASS_NAME
  - example

```
$ llc -stop-after=isel test.ll –o test.mir
$ llc –load MyPass.so -run-pass=machine-scheduler a.mir -o
a_sched.mir
$ llc -start-after=isel a_sched.mir -o a.s
```

Skeleton Code

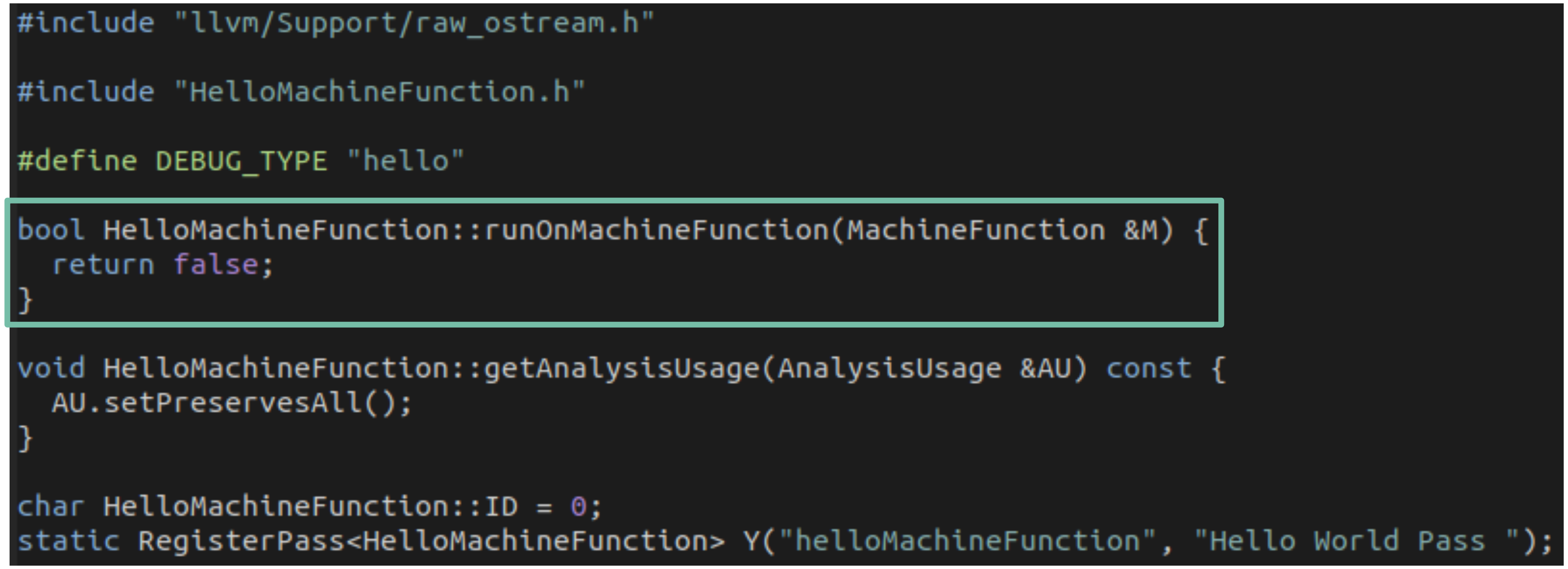
- Header File (HelloMachineFunction.h)



```
#include "llvm/CodeGen/MachineFunction.h"
#include "llvm/CodeGen/MachineFunctionPass.h"

using namespace llvm;

namespace {
  struct HelloMachineFunction : public MachineFunctionPass {
    static char ID; // Pass identification, replacement for typeid
    HelloMachineFunction() : MachineFunctionPass(ID) {}

    bool runOnMachineFunction(MachineFunction &M) override;

    void getAnalysisUsage(AnalysisUsage &AU) const override;
  };
}

#endif
```

Skeleton Code

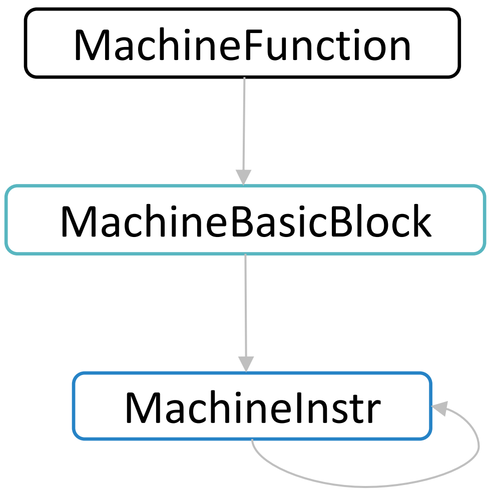
- Source File (HelloMachineFunction.cpp)



```cpp
#include "llvm/Support/raw_ostream.h"

#include "HelloMachineFunction.h"

#define DEBUG_TYPE "hello"

bool HelloMachineFunction::runOnMachineFunction(MachineFunction &M) {
  return false;
}

void HelloMachineFunction::getAnalysisUsage(AnalysisUsage &AU) const {
  AU.setPreservesAll();
}

char HelloMachineFunction::ID = 0;
static RegisterPass<HelloMachineFunction> Y("helloMachineFunction", "Hello World Pass ");
```

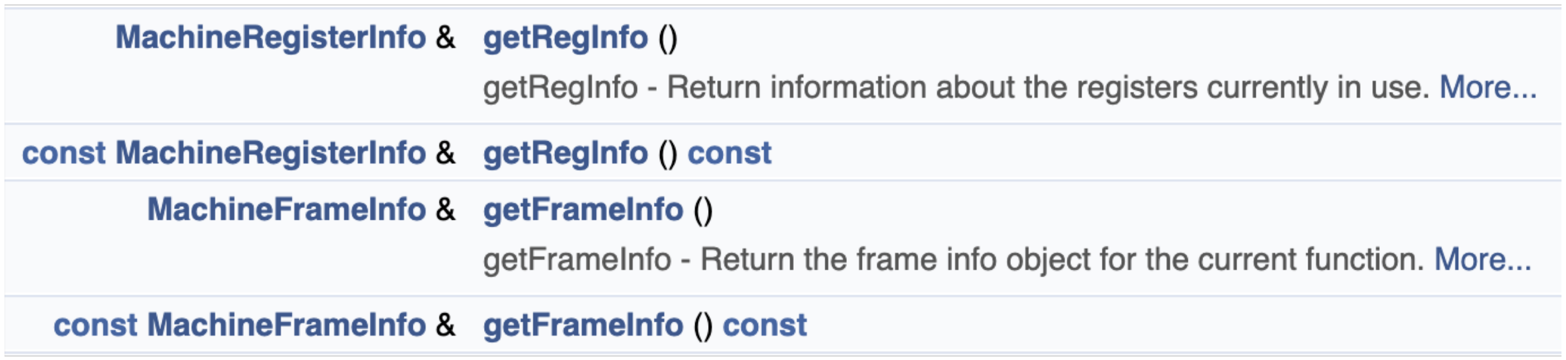
LLVM Machine Code Classes

- Has-a relationship of Machine Code classes



MachineFunction Class

- Provide lower-level information than (IR) Function
  - For example,



MachineRegisterInfo &	getRegInfo () getRegInfo - Return information about the registers currently in use. More...
const MachineRegisterInfo &	getRegInfo () const
MachineFrameInfo &	getFrameInfo () getFrameInfo - Return the frame info object for the current function. More...
const MachineFrameInfo &	getFrameInfo () const

Example

- Print all machine instructions

```
void SimplePrinter::runOnMachineFunction(MachineFunction &MF) {
  for (MachineBasicBlock &MBB : MF)
    for (MachineInstruction &MI : MBB)
      out() << MI;
}
```

How to Add a New Target

How to Add a New Target

- All targets are under
  - `llvm/lib/Target`
  - Available targets: X86, ARM, RISCV, Mips, AMDGPU, …
- First step to add a new target
  - Create a folder for the target at `llvm/lib/Target`

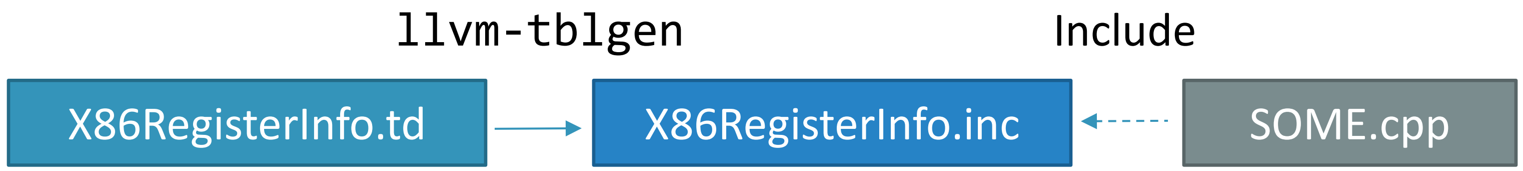
How to Add a New Target

- (Minimal) Steps
  - Define registers and register classes
  - Define calling convention
  - Define the instruction set
  - Implement target lowering
  - Select an instruction
  - Implement frame lowering
  - Register a target

LLVM TableGen

- Facilitate describing target description
  - Register definition
  - Register aliases
  - Register classes
  - Instruction definition
  - Calling convention

- Interpret *.td into *.inc (c++)



LLVM TableGen

- Generates various **target-specific data structures and functions** in *.inc files

- Important to implement a back-end compiler
  - Understand *.td files
    - Ex) TOY.td, TOYRegisterInfo.td, TOYCallingConv.td, TOYInstrFormats.td, TOYInstrInfo.td, TOYOperators.td
  - Know what *.inc files provide
    - Ex) TOYGenSubtargetInfo.inc, TOYGenRegisterInfo.inc, TOYGenInstrInfo.inc, TOYGenAsmWriter.inc, TOYGenDAGISel.inc, TOYGenCallingConv.inc, TOYGenMCCodeEmitter.inc

Toy Architecture

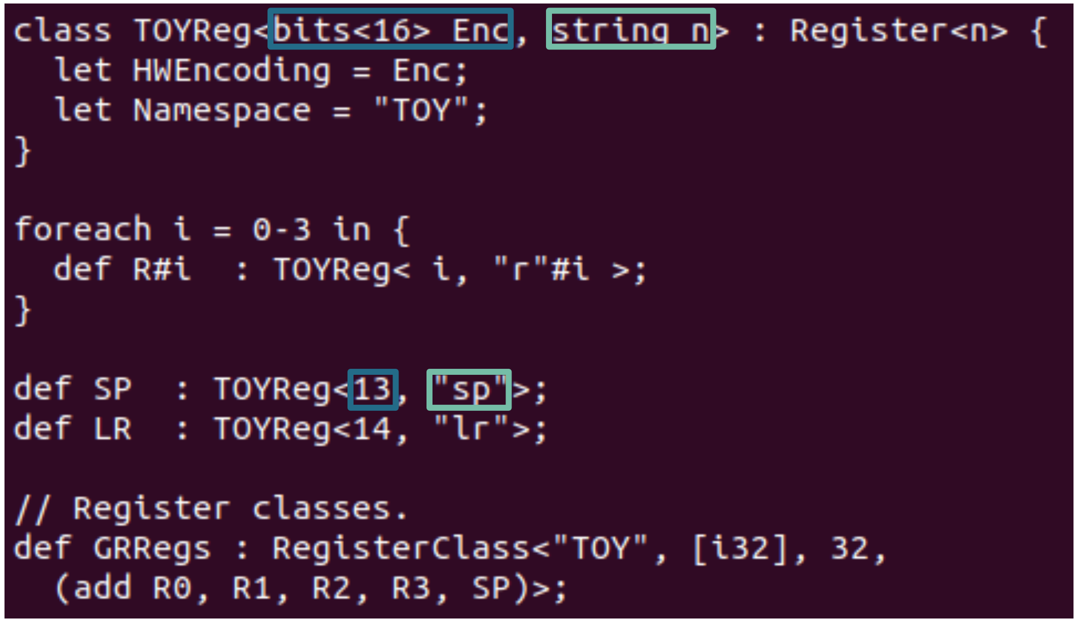
- A simple RISC-type architecture

- Registers
  - General registers: r0-r3
  - Stack Pointer: sp
  - Link register: lr

- Calling Convention
  - Arguments: r0-r1
  - Return value: r0

Registers and Register Classes

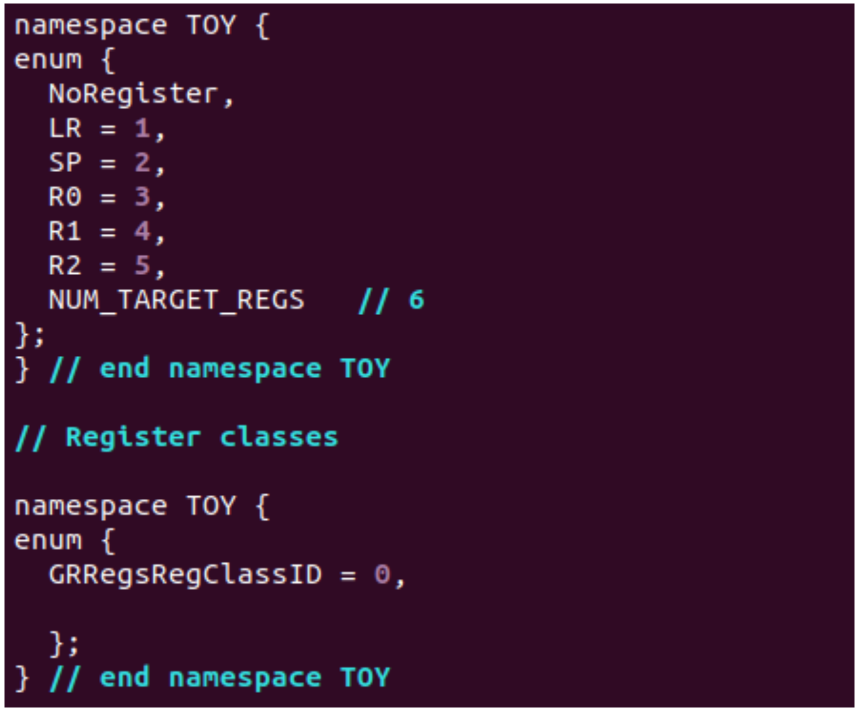
- How to define registers and register classes
  - `llvm/lib/Target/TOY/TOYRegisterInfo.td`



```
class TOYReg<bits<16> Enc, string n> : Register<n> {
  let HWEncoding = Enc;
  let Namespace = "TOY";
}

foreach i = 0-3 in {
  def R#i  : TOYReg< i, "r"#i >;
}

def SP  : TOYReg<13, "sp">;
def LR  : TOYReg<14, "lr">;

// Register classes.
def GRRegs : RegisterClass<"TOY", [i32], 32,
  (add R0, R1, R2, R3, SP)>;
```

Registers and Register Classes

- After TableGen
  - `llvm-objects/lib/Target/TOY/TOYGenRegisterInfo.inc`



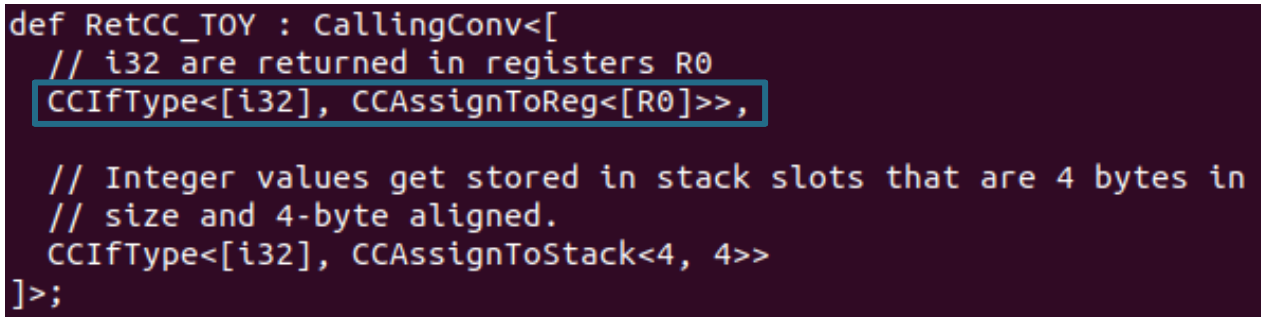
```
namespace TOY {
enum {
  NoRegister,
  LR = 1,
  SP = 2,
  R0 = 3,
  R1 = 4,
  R2 = 5,
  NUM_TARGET_REGS   // 6
};
} // end namespace TOY

// Register classes

namespace TOY {
enum {
  GRRegsRegClassID = 0,

  };
} // end namespace TOY
```

Calling Convention

- When calling a method,
  - How parameters are passed
    - Put a1, …, ak in registers r1, …, rk
    - Put ak, …, an into stack
  - Which registers the called function must preserve for the caller (Callee-saved registers)
    - Caller assumes that these registers will contain the same values be fore and after the call
  - Which registers the caller must preserve for the callee (Caller -saved registers)
    - Callee function uses caller-saved registers without saving them

Calling Convention

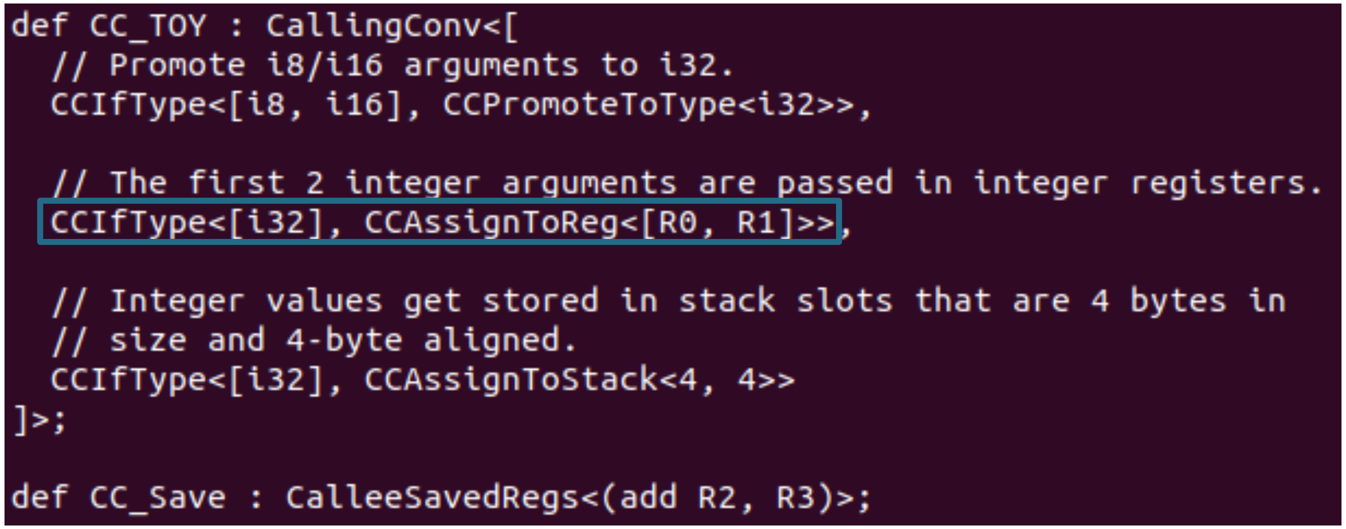
- How to define calling convention
  - `llvm/lib/Target/TOY/TOYCallingConvention.td`
  - Return Value Calling Convention



```
def RetCC_TOY : CallingConv<[
  // i32 are returned in registers R0
  CCIfType<[i32], CCAssignToReg<[R0]>>,

  // Integer values get stored in stack slots that are 4 bytes in
  // size and 4-byte aligned.
  CCIfType<[i32], CCAssignToStack<4, 4>>
]>;
```

Calling Convention

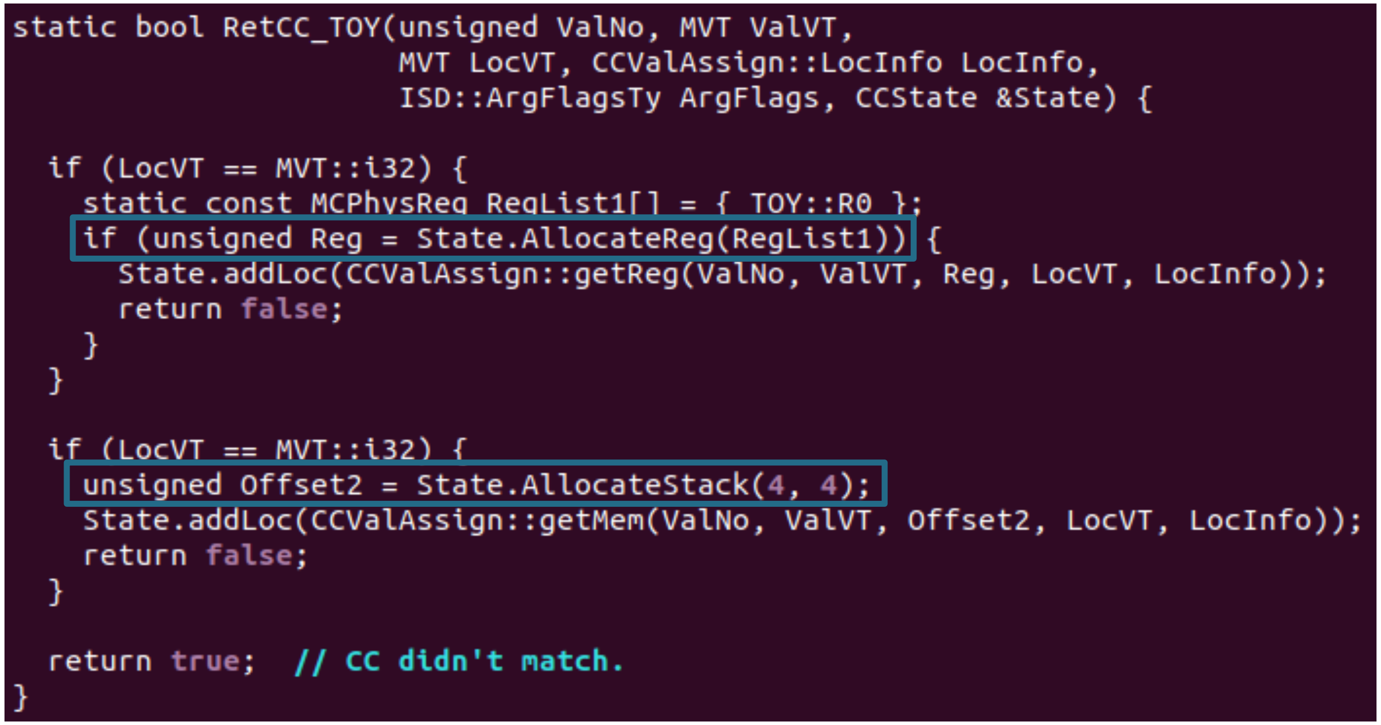
- How to define calling convention
  - `llvm/lib/Target/TOY/TOYCallingConvention.td`
  - Argument Calling Convention



```
def CC_TOY : CallingConv<[
  // Promote i8/i16 arguments to i32.
  CCIfType<[i8, i16], CCPromoteToType<i32>>,

  // The first 2 integer arguments are passed in integer registers.
  CCIfType<[i32], CCAssignToReg<[R0, R1]>>,

  // Integer values get stored in stack slots that are 4 bytes in
  // size and 4-byte aligned.
  CCIfType<[i32], CCAssignToStack<4, 4>>
]>;

def CC_Save : CalleeSavedRegs<(add R2, R3)>;
```

Calling Convention

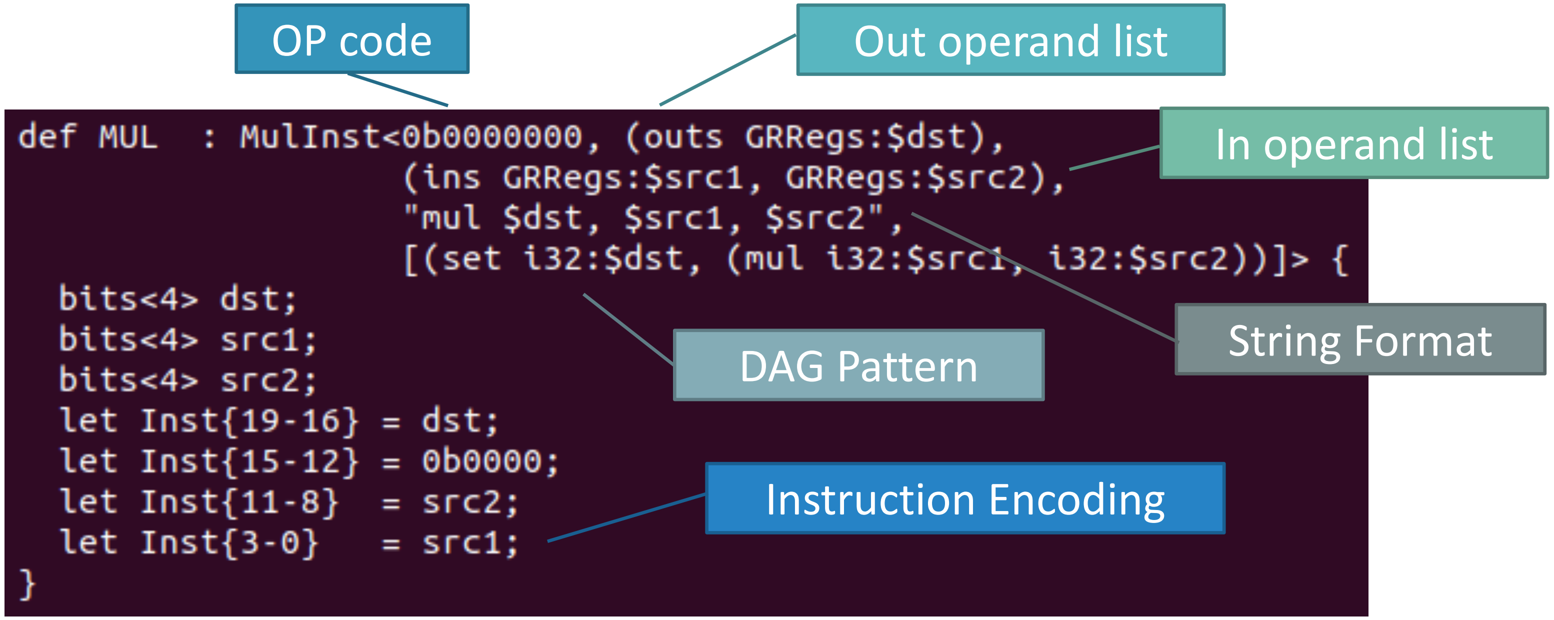
- After TableGen
  - `llvm-objects/lib/Target/TOY/TOYGenCallingConv.inc`



```
static bool RetCC_TOY(unsigned ValNo, MVT ValVT,
                      MVT LocVT, CCValAssign::LocInfo LocInfo,
                      ISD::ArgFlagsTy ArgFlags, CCState &State) {

  if (LocVT == MVT::i32) {
    static const MCPhysReg RegList1[] = { TOY::R0 };
    if (unsigned Reg = State.AllocateReg(RegList1)) {
      State.addLoc(CCValAssign::getReg(ValNo, ValVT, Reg, LocVT, LocInfo));
      return false;
    }
  }

  if (LocVT == MVT::i32) {
    unsigned Offset2 = State.AllocateStack(4, 4);
    State.addLoc(CCValAssign::getMem(ValNo, ValVT, Offset2, LocVT, LocInfo));
    return false;
  }

  return true;  // CC didn't match.
}
```

Instruction Set

- How to define instructions
  - `llvm/lib/Target/TOY/TOYInstrInfo.td`



```
def MUL  : MulInst<0b0000000, (outs GRRegs:$dst),
                   (ins GRRegs:$src1, GRRegs:$src2),
                   "mul $dst, $src1, $src2",
                   [(set i32:$dst, (mul i32:$src1, i32:$src2))]> {
  bits<4> dst;
  bits<4> src1;
  bits<4> src2;
  let Inst{19-16} = dst;
  let Inst{15-12} = 0b0000;
  let Inst{11-8}  = src2;
  let Inst{3-0}   = src1;
}
```

Target Lowering

- SelectionDAG Legalization
  - Replace or remove operations and data types that are not supported natively in a SelectionDAG

- How to implement target lowering
  - Inherit `TargetLowering` class
    - `SDValue LowerFormalArguments (...)`
    - `SDValue LowerCall (...)`
    - `SDValue LowerReturn (...)`
    - `...`

Instruction Selection

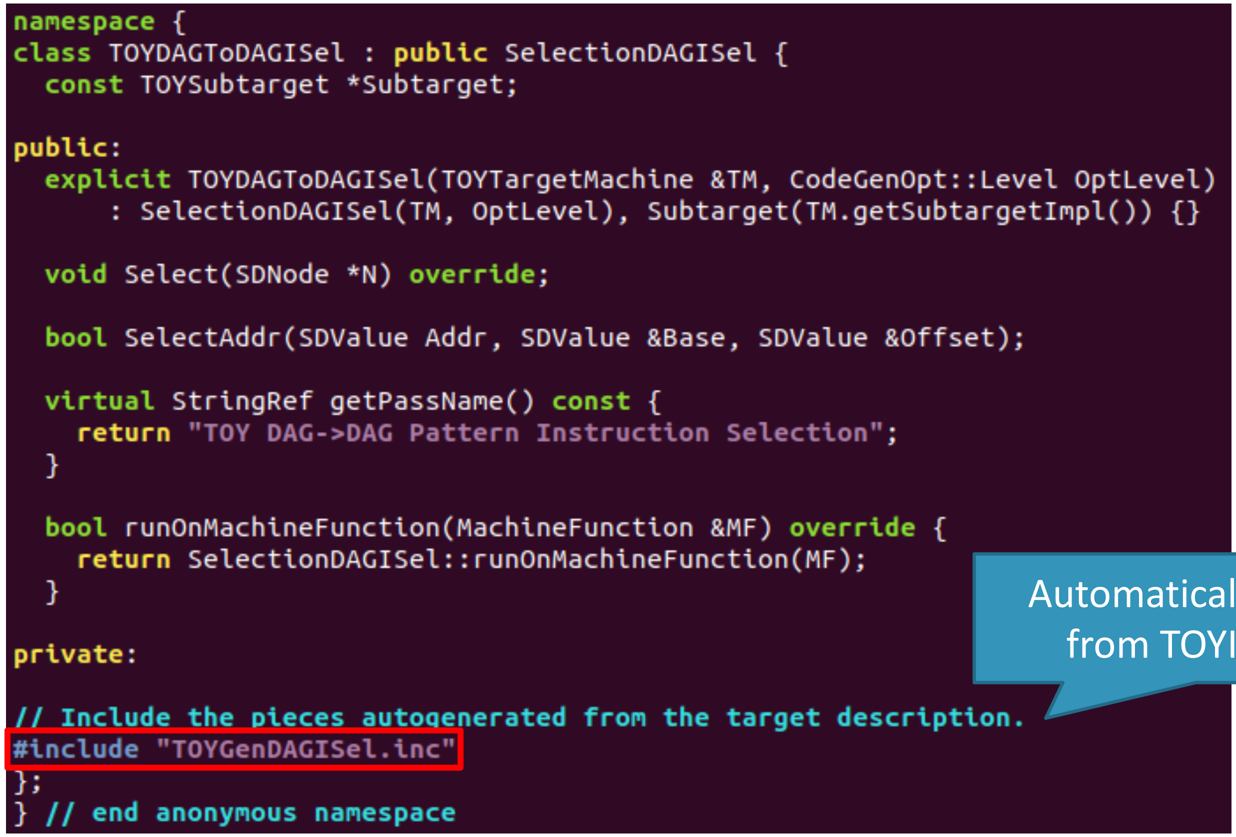
- Lower to taget-specific instructions

- How to implement instruction selection
  - Inherit `SelectionDAGISel` class
    - `void Select(SDNode *N)`
      - Select instruction for a SDNode

    - `bool SelectAddr(SDValue Addr, SDValue &Base, SDValue &Offset)`
      - Calculate the base and offset of the address for load and store operations

Instruction Selection

- How to implement instruction selection
  - `llvm/lib/Target/TOY/TOYISelDAGtoDAG.cpp`

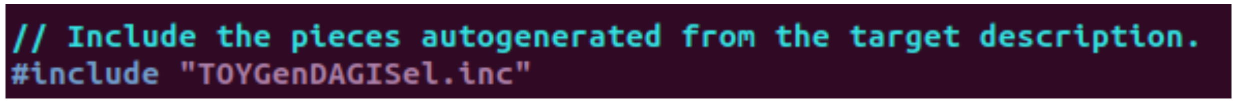


```
namespace {
class TOYDAGToDAGISel : public SelectionDAGISel {
  const TOYSubtarget *Subtarget;

public:
  explicit TOYDAGToDAGISel(TOYTargetMachine &TM, CodeGenOpt::Level OptLevel)
      : SelectionDAGISel(TM, OptLevel), Subtarget(TM.getSubtargetImpl()) {}

  void Select(SDNode *N) override;

  bool SelectAddr(SDValue Addr, SDValue &Base, SDValue &Offset);

  virtual StringRef getPassName() const {
    return "TOY DAG->DAG Pattern Instruction Selection";
  }

  bool runOnMachineFunction(MachineFunction &MF) override {
    return SelectionDAGISel::runOnMachineFunction(MF);
  }

private:

// Include the pieces autogenerated from the target description.
#include "TOYGenDAGISel.inc"
};
} // end anonymous namespace
```

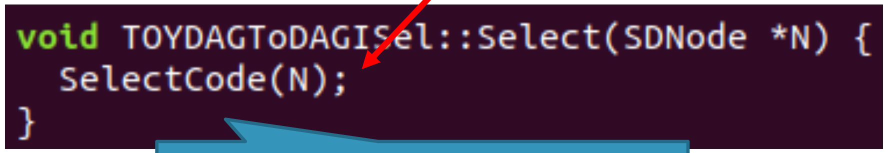
Automatically generated from TOYInstInfo.td

Instruction Selection

- How to implement instruction selection
  - `llvm/lib/Target/TOY/TOYISelDAGtoDAG.cpp`
  - `void Select(SDNode *N)`



```
// Include the pieces autogenerated from the target description.
#include "TOYGenDAGISel.inc"
```

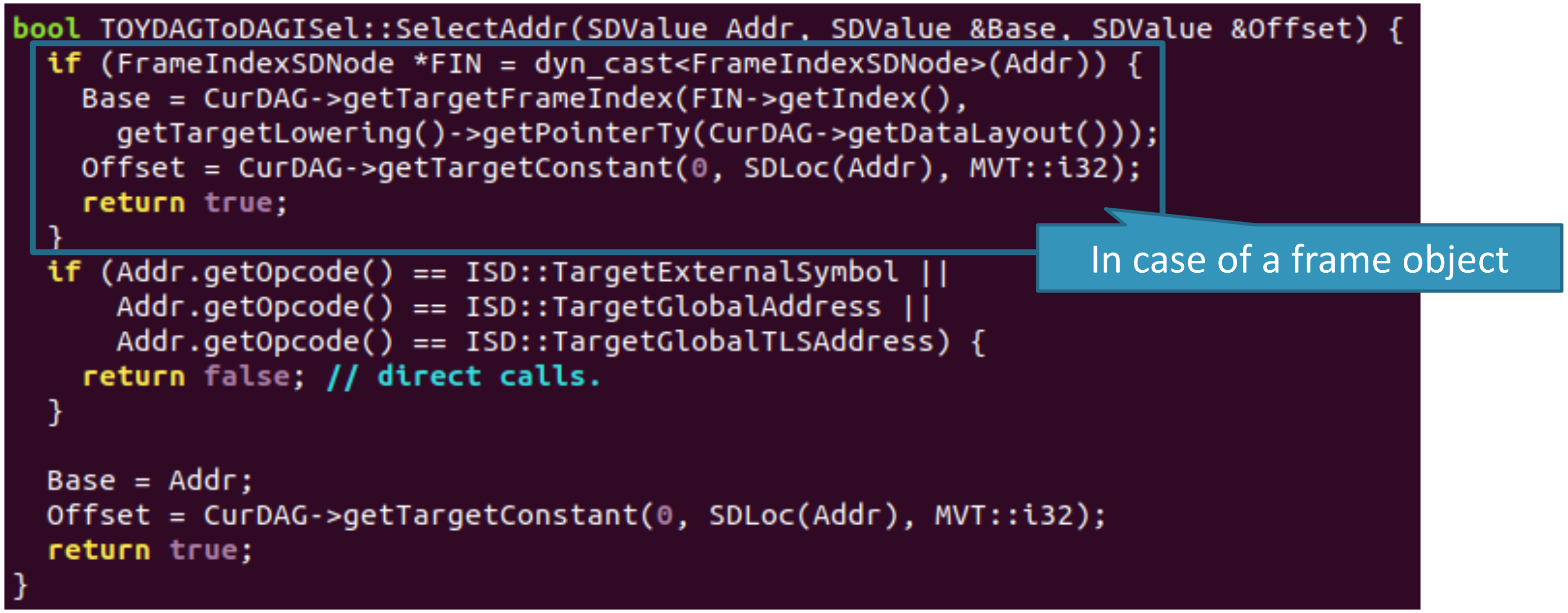


```
void TOYDAGToDAGISel::Select(SDNode *N) {
  SelectCode(N);
}
```

Automatically generated from TOYInstInfo.td

Instruction Selection

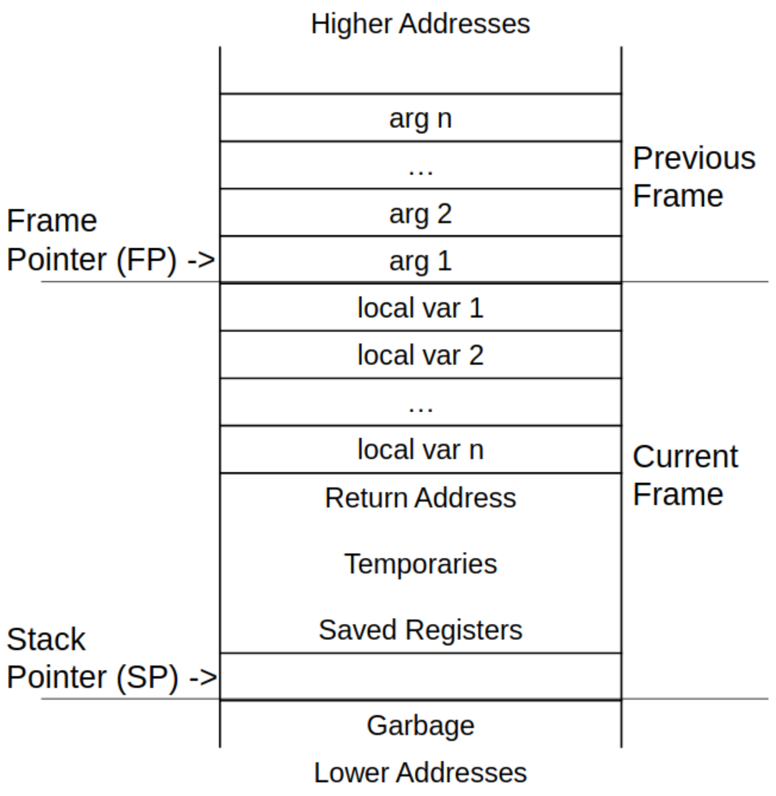
- How to implement instruction selection
  - `llvm/lib/Target/TOY/TOYISelDAGtoDAG.cpp`
  - `bool SelectAddr(SDValue Addr, SDValue &Base, SDValue &Offset)`



```
bool TOYDAGToDAGISel::SelectAddr(SDValue Addr, SDValue &Base, SDValue &Offset) {
  if (FrameIndexSDNode *FIN = dyn_cast<FrameIndexSDNode>(Addr)) {
    Base = CurDAG->getTargetFrameIndex(FIN->getIndex(),
      getTargetLowering()->getPointerTy(CurDAG->getDataLayout()));
    Offset = CurDAG->getTargetConstant(0, SDLoc(Addr), MVT::i32);
    return true;
  }
  if (Addr.getOpcode() == ISD::TargetExternalSymbol ||
      Addr.getOpcode() == ISD::TargetGlobalAddress ||
      Addr.getOpcode() == ISD::TargetGlobalTLSAddress) {
    return false; // direct calls.
  }

  Base = Addr;
  Offset = CurDAG->getTargetConstant(0, SDLoc(Addr), MVT::i32);
  return true;
}
```

Frame Lowering

- (Stack) Frame lowering
  - Emit the function **prologue**
    - Move the stack pointer (SP) to allocate a new stack frame
  - Emit the function **epilogue**
    - Move the stack pointer (SP) to destroy the stack frame

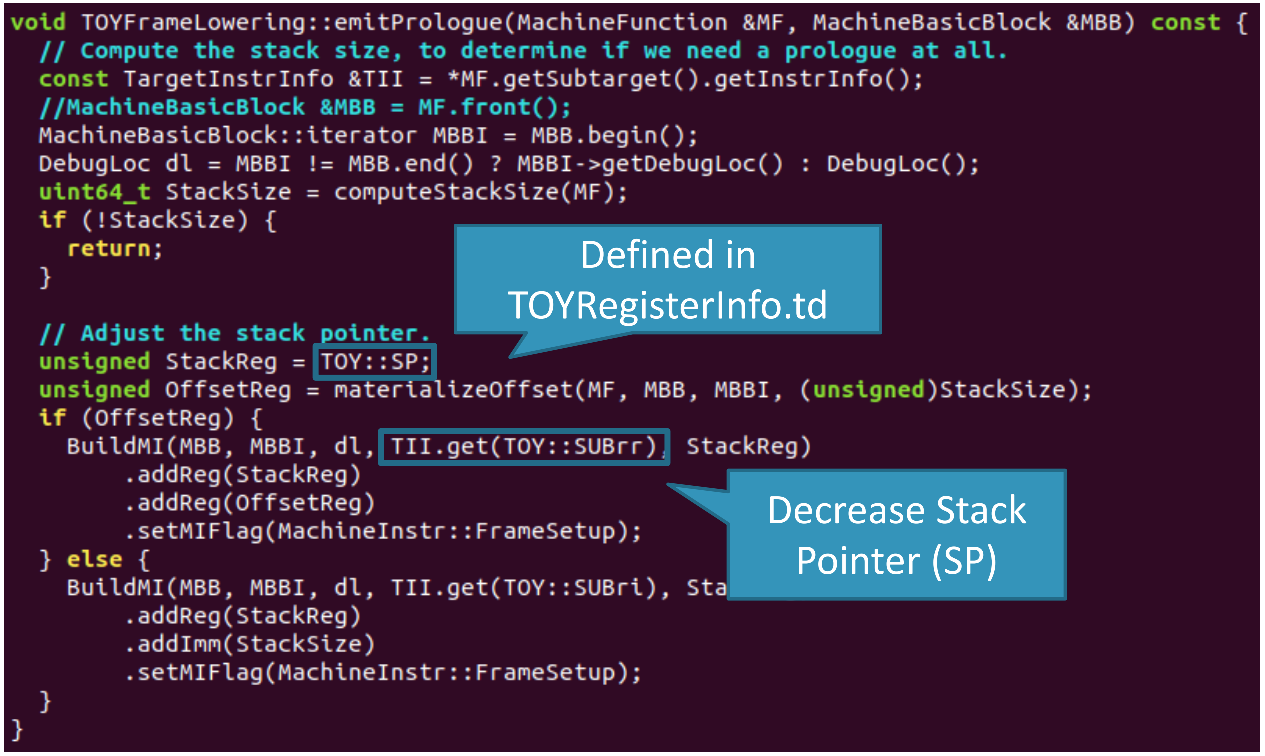


Frame Lowering

- How to Implement Frame Lowering
  - Inherit `TargetFrameLowering` class
    - `void emitPrologue(MachineFunction &MF, MachineBasicBlock &MBB)`
      - Emit the function prologue
    - `void emitEpilogue(MachineFunction &MF, MachineBasicBlock &MBB)`
      - Emit the function epilogue

Frame Lowering

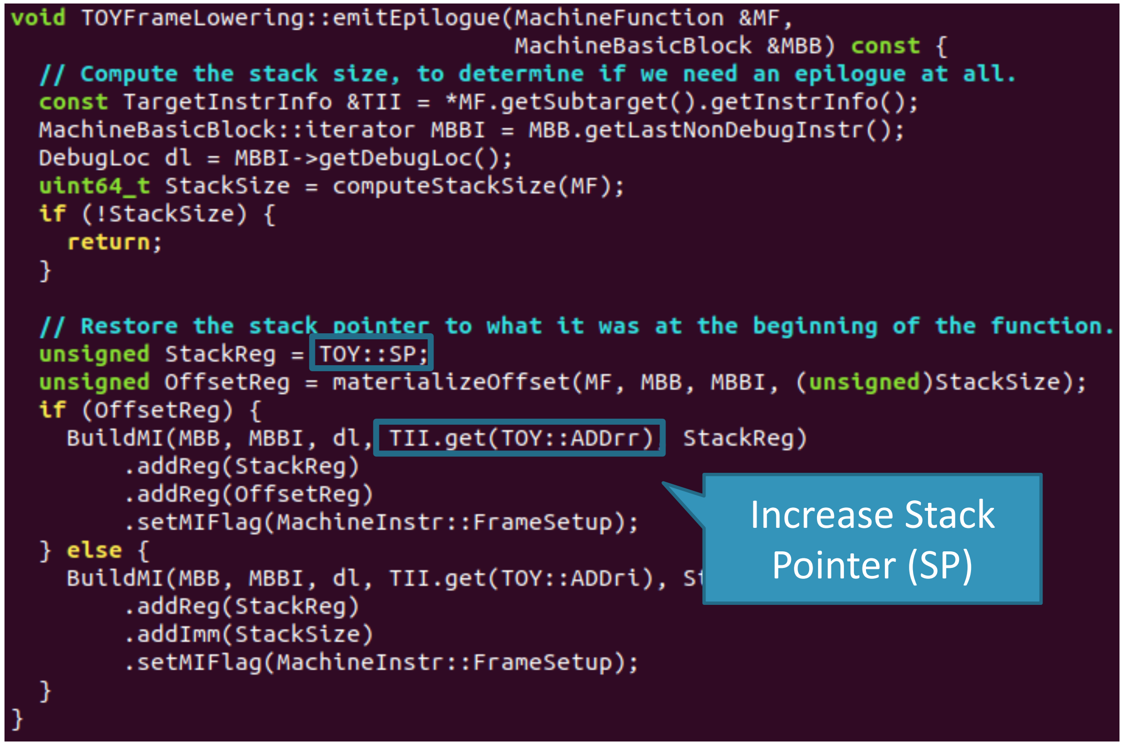
- How to Implement Frame Lowering
  - `llvm/lib/Target/TOY/TOYFrameLowering.cpp`
  - Function Prologue



```
void TOYFrameLowering::emitPrologue(MachineFunction &MF, MachineBasicBlock &MBB) const {
  // Compute the stack size, to determine if we need a prologue at all.
  const TargetInstrInfo &TII = *MF.getSubtarget().getInstrInfo();
  //MachineBasicBlock &MBB = MF.front();
  MachineBasicBlock::iterator MBBI = MBB.begin();
  DebugLoc dl = MBBI != MBB.end() ? MBBI->getDebugLoc() : DebugLoc();
  uint64_t StackSize = computeStackSize(MF);
  if (!StackSize) {
    return;
  }

  // Adjust the stack pointer.
  unsigned StackReg = TOY::SP;
  unsigned OffsetReg = materializeOffset(MF, MBB, MBBI, (unsigned)StackSize);
  if (OffsetReg) {
    BuildMI(MBB, MBBI, dl, TII.get(TOY::SUBrr), StackReg)
        .addReg(StackReg)
        .addReg(OffsetReg)
        .setMIFlag(MachineInstr::FrameSetup);
  } else {
    BuildMI(MBB, MBBI, dl, TII.get(TOY::SUBri), Sta
        .addReg(StackReg)
        .addImm(StackSize)
        .setMIFlag(MachineInstr::FrameSetup);
  }
}
```

Frame Lowering

- How to Implement Frame Lowering
  - `llvm/lib/Target/TOY/TOYFrameLowering.cpp`
  - Function Epilogue

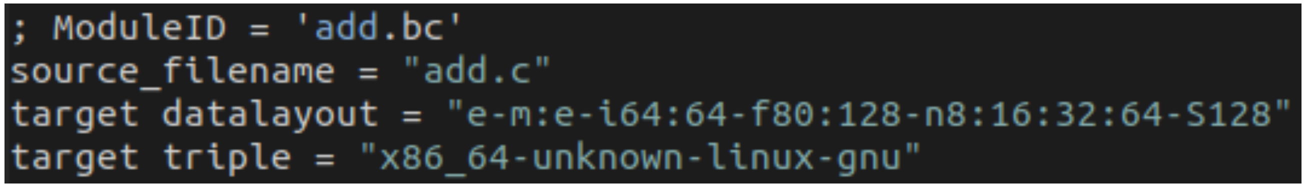


```
void TOYFrameLowering::emitEpilogue(MachineFunction &MF,
                                    MachineBasicBlock &MBB) const {
  // Compute the stack size, to determine if we need an epilogue at all.
  const TargetInstrInfo &TII = *MF.getSubtarget().getInstrInfo();
  MachineBasicBlock::iterator MBBI = MBB.getLastNonDebugInstr();
  DebugLoc dl = MBBI->getDebugLoc();
  uint64_t StackSize = computeStackSize(MF);
  if (!StackSize) {
    return;
  }

  // Restore the stack pointer to what it was at the beginning of the function.
  unsigned StackReg = TOY::SP;
  unsigned OffsetReg = materializeOffset(MF, MBB, MBBI, (unsigned)StackSize);
  if (OffsetReg) {
    BuildMI(MBB, MBBI, dl, TII.get(TOY::ADDrr)  StackReg)
        .addReg(StackReg)
        .addReg(OffsetReg)
        .setMIFlag(MachineInstr::FrameSetup);
  } else {
    BuildMI(MBB, MBBI, dl, TII.get(TOY::ADDri), St
        .addReg(StackReg)
        .addImm(StackSize)
        .setMIFlag(MachineInstr::FrameSetup);
  }
}
```

TargetMachine Class

- Describe the target machine
  - Triple: <arch><sub>-<vendor>-<sys>-<abi>
    - Ex) armv7a-none-linux-eabi
  - Data Layout
    - Ex)"E-p:32:32-f128:128:128"
      - E: Big Endianness
      - p: pointer, f: floating point, i:integer
      - Numbers: size, ABI alignment, and preferred alignment

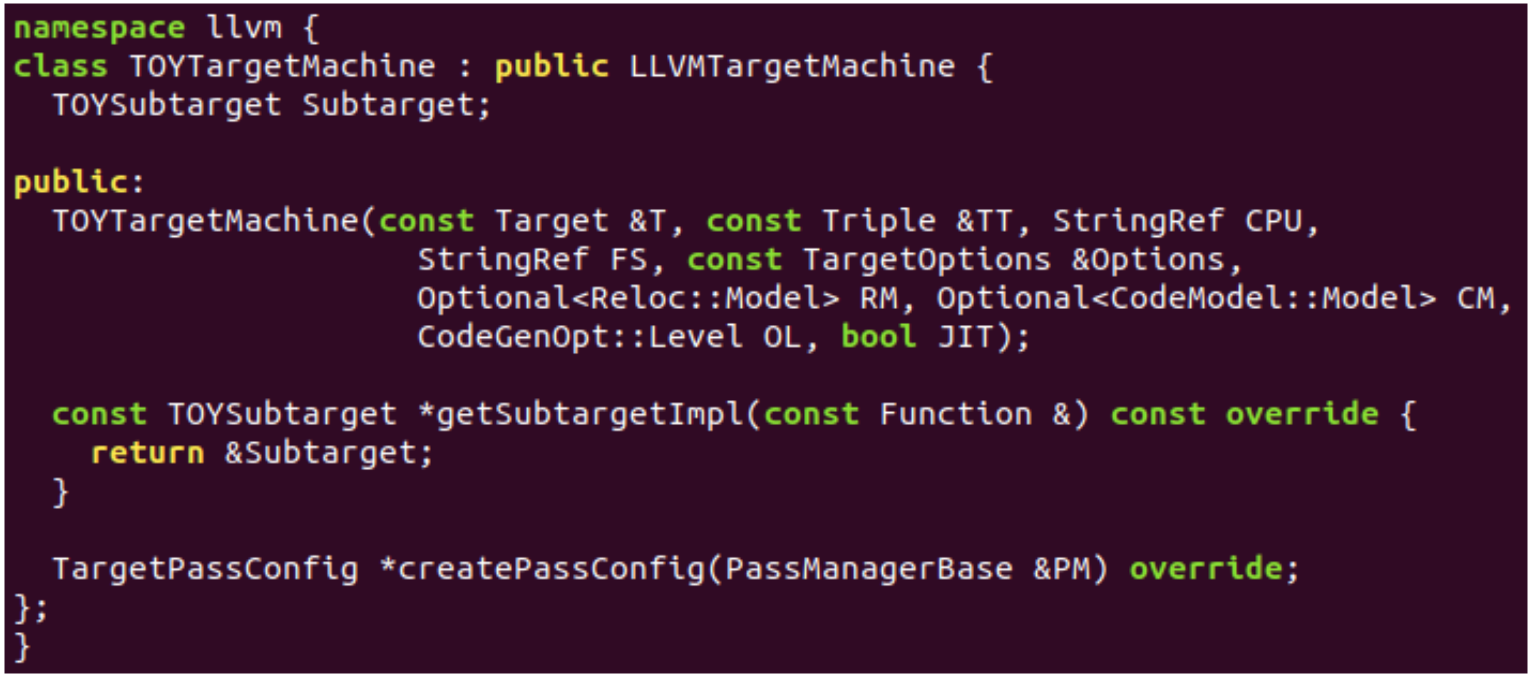
  - At the top of a *.ll file,



```
; ModuleID = 'add.bc'
source_filename = "add.c"
target datalayout = "e-m:e-i64:64-f80:128-n8:16:32:64-S128"
target triple = "x86_64-unknown-linux-gnu"
```

TargetMachine Class

- `llvm/lib/Target/TOYTargetMachine.h`



```
namespace llvm {
class TOYTargetMachine : public LLVMTargetMachine {
  TOYSubtarget Subtarget;

public:
  TOYTargetMachine(const Target &T, const Triple &TT, StringRef CPU,
                   StringRef FS, const TargetOptions &Options,
                   Optional<Reloc::Model> RM, Optional<CodeModel::Model> CM,
                   CodeGenOpt::Level OL, bool JIT);

  const TOYSubtarget *getSubtargetImpl(const Function &) const override {
    return &Subtarget;
  }

  TargetPassConfig *createPassConfig(PassManagerBase &PM) override;
};
}
```

More things to do…

- Implement InstPrinter
  - `llvm/lib/Target/TOY/InstPrinter/InstPrinter.h`
  - `llvm/lib/Target/TOY/InstPrinter/InstPrinter.cpp`
- Implement AsmPrinter
  - `llvm/lib/Target/TOY/AsmPrinter.cpp`
- Register a target
- Support a subtarget
  - `llvm/lib/Target/TOY.td`
  - `llvm/lib/Target/TOYSubtarget.h`
  - `llvm/lib/Target/TOYSubtarget.cpp`

Backup Slides

TableGen

- Register Class
  - class Register<string n> {
  -   string Namespace = "";
  -   string AsmName = n;
  -   string Name = n;
  -   int SpillSize = 0;
  -   int SpillAlignment = 0;
  -   list<Register> Aliases = [];
  -   list<Register> SubRegs = [];
  -   list<int> DwarfNumbers = [];
  - }