The structure of LLVM backends

Dr. Norman A. Rink
Technische Universität Dresden, Germany
norman.rink@tu-dresden.de

Bloomberg Clang/LLVM Sprint Weekend
6-7 February 2016
London, England
```c
int kernel(int *a, int b, unsigned int i) {
    return a[i] * b;
}
```

```assembly
define i32 @kernel(i32* %a, i32 %b, i32 %i) {
    %1 = getelementptr i32* %a, i32 %i
    %2 = load i32* %1
    %3 = mul i32 %2, %b
    ret i32 %3
}
```
Passes

**IRPasses**
- target specific
- e.g. transformation of intrinsics, expanding of atomic operations

**InstSelector**

**MachinePasses**

**ExpandISelPseudos**

**PreRegAlloc**
- may want to run a scheduler here

**RegAlloc**
- use one of LLVM’s predefined register allocators

**PostRegAlloc**

**PrologEpilogInserter**
- predefined pass
- requires hooks to manage frame and stack pointers

**ExpandPostRAPseudos**

**PreSched2**
- e.g. anything that aides subsequent scheduling

**PostRAScheduler**
- can use predefined scheduler here

**PreEmitPass**
- e.g. VLIW packing
Instruction selection

define i32 @kernel(i32* %a, i32 %b, i32 %i) {
    %1 = getelementptr i32* %a, i32 %i
    %2 = load i32* %1
    %3 = mul i32 %2, %b
    ret i32 %3
}

CAVEAT:
Code samples and graphs in this talk not from the most recent version of LLVM.
Instruction selection – part I

LLVM IR

SelectionDAGBuilder

SelectionDAG

DAGCombine

Legalize

DAGCombine

ISel

Scheduler

MachineInstr

Ilc -view-dag-combine1-dags
Instruction selection – part II

LLVM IR

SelectionDAGBuilder

SelectionDAG

DAGCombine

Legalize

DAGCombine

ISel

Scheduler

MachineInstr

llc -view-sched-dags
SelectionDAG pattern matching

(set $dst (load (add $base (shl $index, $scale))))

MOV32rm $dst, ($base, $index, $scale)
SelectionDAG patterns and TableGen

class X86Inst<\text{bits}<8\text{>}> opcod, Format f, \text{dag} outs, \text{dag} ins, \text{string} AsmStr, \text{list}<\text{dag}> pattern, 
\text{InstrItinClass itin}> : \text{Instruction} 
  \begin{align*}
    \text{bits}<8\text{>} & \text{ Opcode = opcod; } \\
    \text{Format Form = f; } \\
    \text{dag} \text{ OutOperandList = outs;} \\
    \text{dag} \text{ InOperandList = ins;} \\
    \text{string AsmString = AsmStr;} \\
    \text{let Pattern = pattern;} \\
    \text{let Itinerary = itin;}
  \end{align*}

\text{def MOV32rr : X86Inst<0x89, MRMDestReg, (outs GR32:$dst), (ins GR32:$src),}\n  \text{"mov\{l\}t{$src, $dst|$dst, $src}"}, [], \text{IIC_MOV}>;

\text{def MOV32rm : X86Inst<0x8B, MRMSrcMem, (outs GR32:$dst), (ins i32mem:$src),}\n  \text{"mov\{l\}t{$src, $dst|$dst, $src}"},
  [(\text{set GR32:$dst, (load addr:$src)})], \text{IIC_MOV_MEM}>;

\text{def ADD32rr : X86Inst<0xC1, MRMDestReg, (outs GR32:$dst), (ins GR32:$src),}\n  \text{"add\{l\}t{$src, $dst|$dst, $src}"},
  [(\text{set GR32:$dst, (add GR32:$dst, GR32:$src)})], \text{IIC_ADD_REG}>;
Instruction selection – part II (again)

LLVM IR

SelectionDAGBuilder

SelectionDAG

DAGCombine

Legalize

DAGCombine

ISel

Scheduler

MachineInstr

scheduler input for kernel:

llc -view-sched-dags

GraphRoot

Instruction selection – part II (again)
Instruction selection – part III

Scheduling-Units Graph for sunit-dag.kernel:

SU(0): CopyToReg [ORD=5] [ID=0] 
RETL [ORD=5] [ID=0] 
0x3db97d0

SU(1): IMUL32rm<Mem:LD4[FixedStack-2]> [ORD=4] [ID=1] 
0x3db98f0

SU(2): MOV32rm<Mem:LD4[%1]> [ORD=3] [ID=2] 
0x3db9a10

SU(3): MOV32rm<Mem:LD4[FixedStack-3](align=8)> [ORD=1] [ID=3] 
0x3db9b30

SU(4): MOV32rm<Mem:LD4[FixedStack-1](align=16)> [ORD=1] [ID=4] 
0x3db9c50

SU(5): CopyToReg [ORD=5] [ID=0] 
RETL [ORD=5] [ID=0] 
0x3db97d0

Scheduling-Units Graph for sunit-dag.kernel:

Ilc-view-sunit-dags
Instruction selection – part IV

Frame Objects:
- fi#-3: size=4, align=8, fixed, at location [SP+12]
- fi#-2: size=4, align=4, fixed, at location [SP+8]
- fi#-1: size=4, align=16, fixed, at location [SP+4]

BB#0: derived from LLVM BB %0
- %vreg0<def> = MOV32rm <fi#-3>
- %vreg1<def> = MOV32rm <fi#-1>
- %vreg2<def> = MOV32rm %vreg1<kil1>, 4, %vreg0<kil1>
- %vreg3<def> = IMUL32rm %vreg2<kil1>, <fi#-2>
- %EAX<def> = COPY %vreg3<kil1>
- RETL %EAX

llc -debug
Passes – again

IRPasses
- target specific
- e.g. transformation of intrinsics, expanding of atomic operations

InstSelector

MachinePasses

ExpandISelPseudos

PreRegAlloc
- may want to run a scheduler here

RegAlloc
- use one of LLVM’s predefined register allocators

PostRegAlloc

PrologEpilogInserter
- predefined pass
- requires hooks to manage frame and stack pointers

ExpandPostRAPseudos

PreSched2
- e.g. anything that aides subsequent scheduling

PostRAScheduler
- can use predefined scheduler here

PreEmitPass
- e.g. VLIW packing
What have I not covered? (selection)

- Talk was mostly about how LLVM does instruction selection.
- Have not looked at any of LLVM’s built-in passes:
  - register allocators
  - various schedulers
  - There is always room for improving things …
  - … or hook in your own favorite MachinePass.

- Have not looked at code emission:
  - neither assembly …
  - … nor object code
  - Relocating symbols can be a target-specific issue here.

- Have not delved into TableGen …
  - … because it is poorly documented.

- TableGen was viewed as part of the instruction selection phase.
  - It can do more (cf. Clang).
  - The SelectionDAG is not everyone’s favourite.
  - Nothing keeps you from hooking in your favourite algorithm for instruction selection.

CAVEAT:
Opinions not based on the most recent version of LLVM.
Pointers

- “Writing an LLVM backend”
  - http://llvm.org/docs/WritingAnLLVMBackend.html

- “Building an LLVM backend”

- “Tutorial: Creating an LLVM Backend for the Cpu0 Architecture”

Get inspired:
Always start your LLVM project by looking for code that achieves a similar task.
The structure of LLVM backends

Dr. Norman A. Rink
Technische Universität Dresden, Germany
norman.rink@tu-dresden.de

Thank you.