CHAIR FOR COMPILER CONSTRUCTION

cfaed CENTER FOR ADVANCING ELECTRONICS DRESDEN

The structure of LLVM backends

Dr. Norman A. Rink

Technische Universität Dresden, Germany

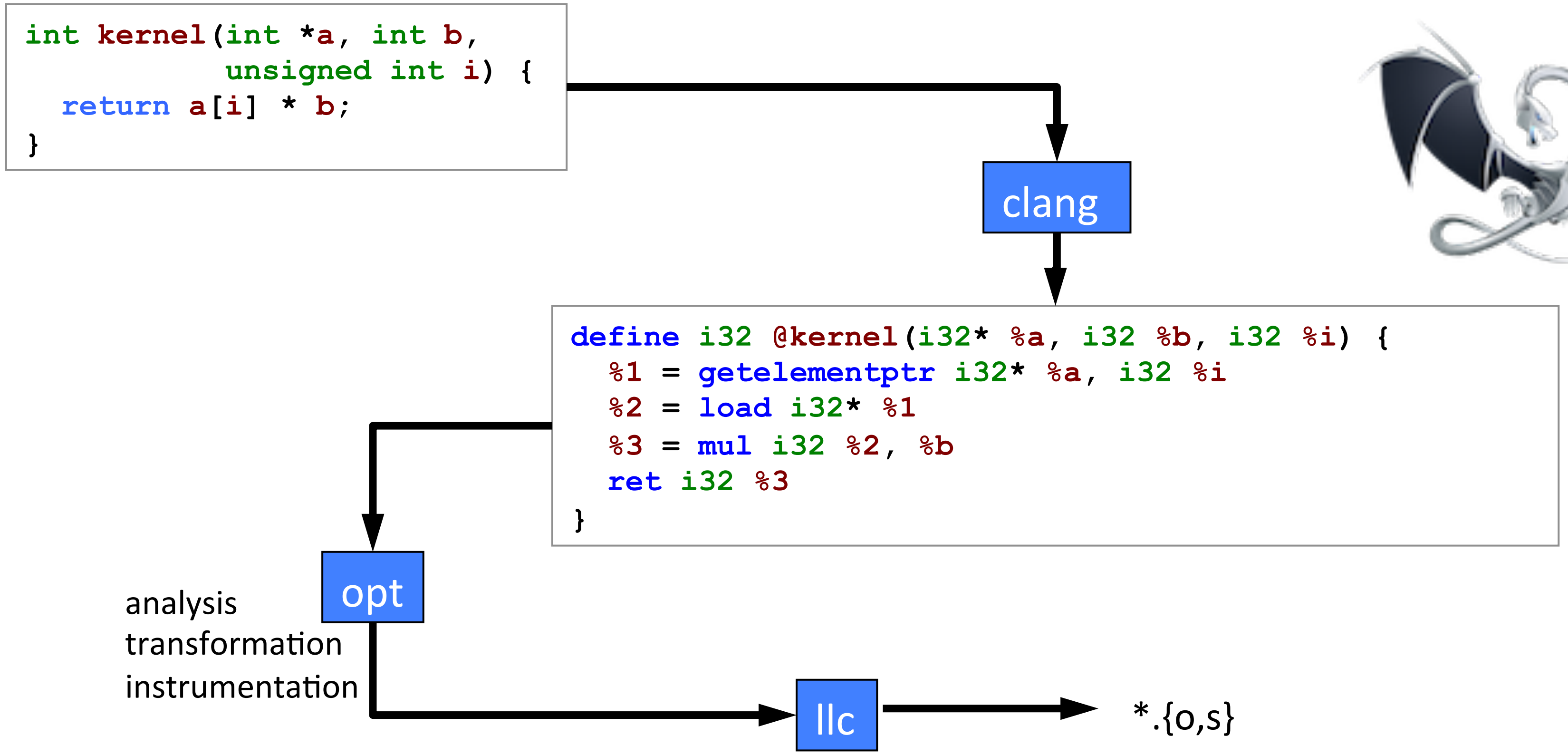
norman.rink@tu-dresden.de

Bloomberg Clang/LLVM Sprint Weekend

6-7 February 2016

London, England

Introduction



```
int kernel(int *a, int b,
           unsigned int i) {
  return a[i] * b;
}
```

```
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

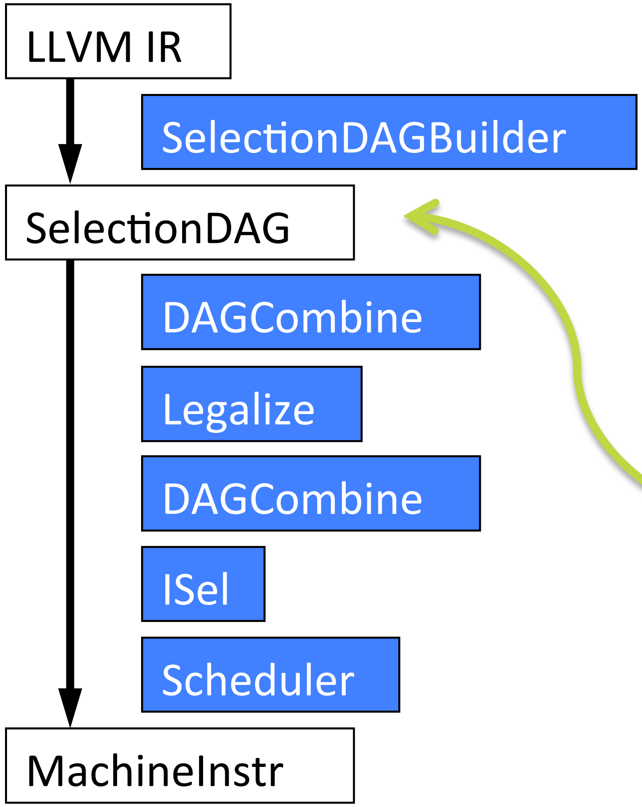
Instruction selection

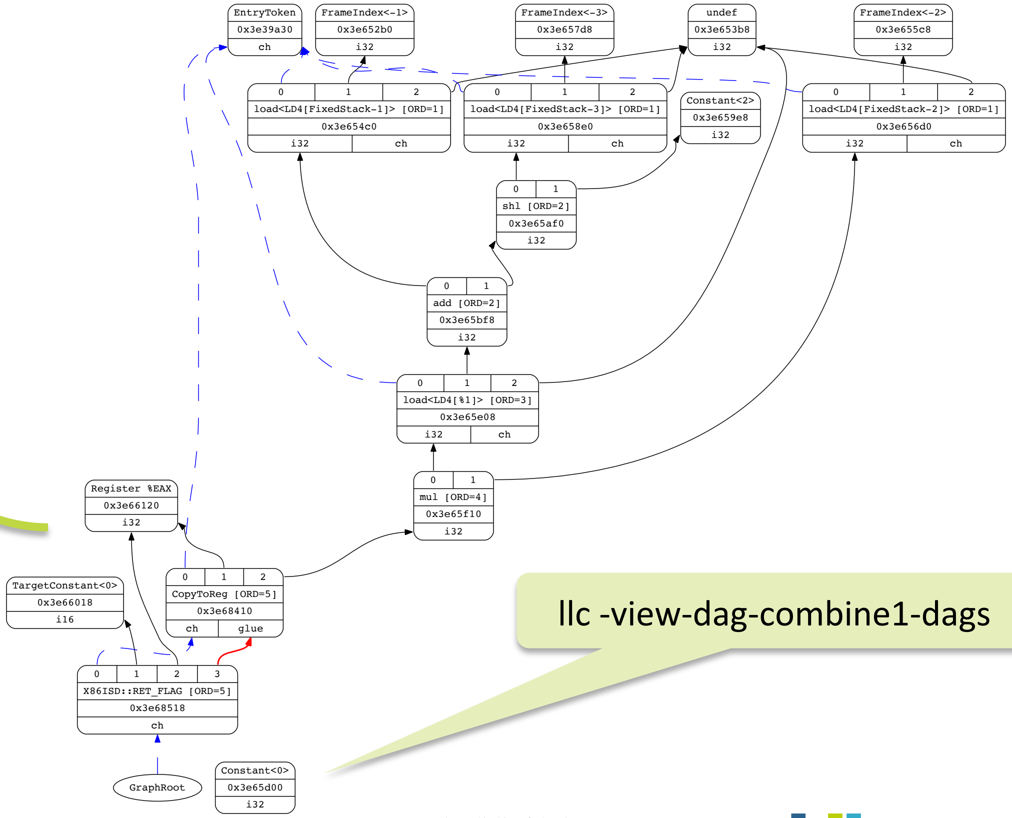
```
define i32 @kernel(i32* %a, i32 %b, i32 %i) {
  %1 = getelementptr i32* %a, i32 %i
  %2 = load i32* %1
  %3 = mul i32 %2, %b
  ret i32 %3
}
```

CAVEAT:

Code samples and graphs in this talk not from the most recent version of LLVM.

Instruction selection – part I



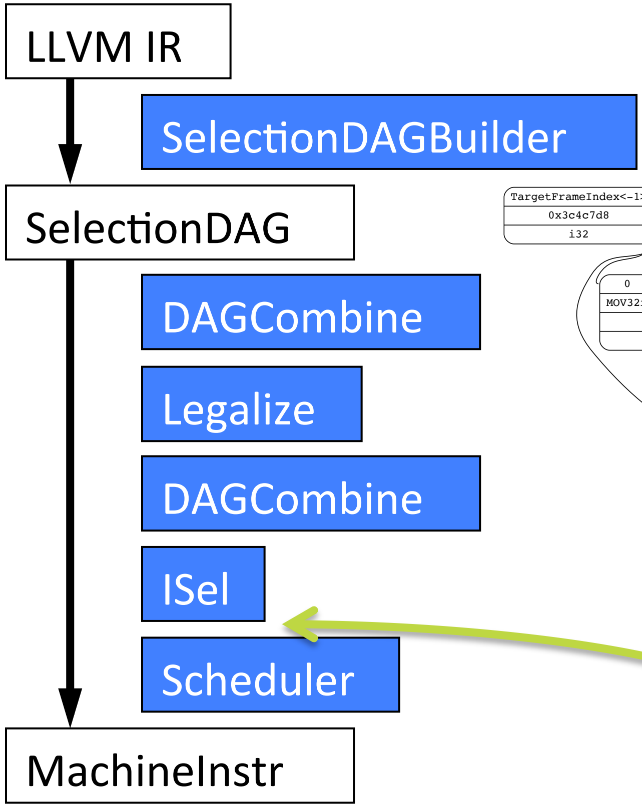


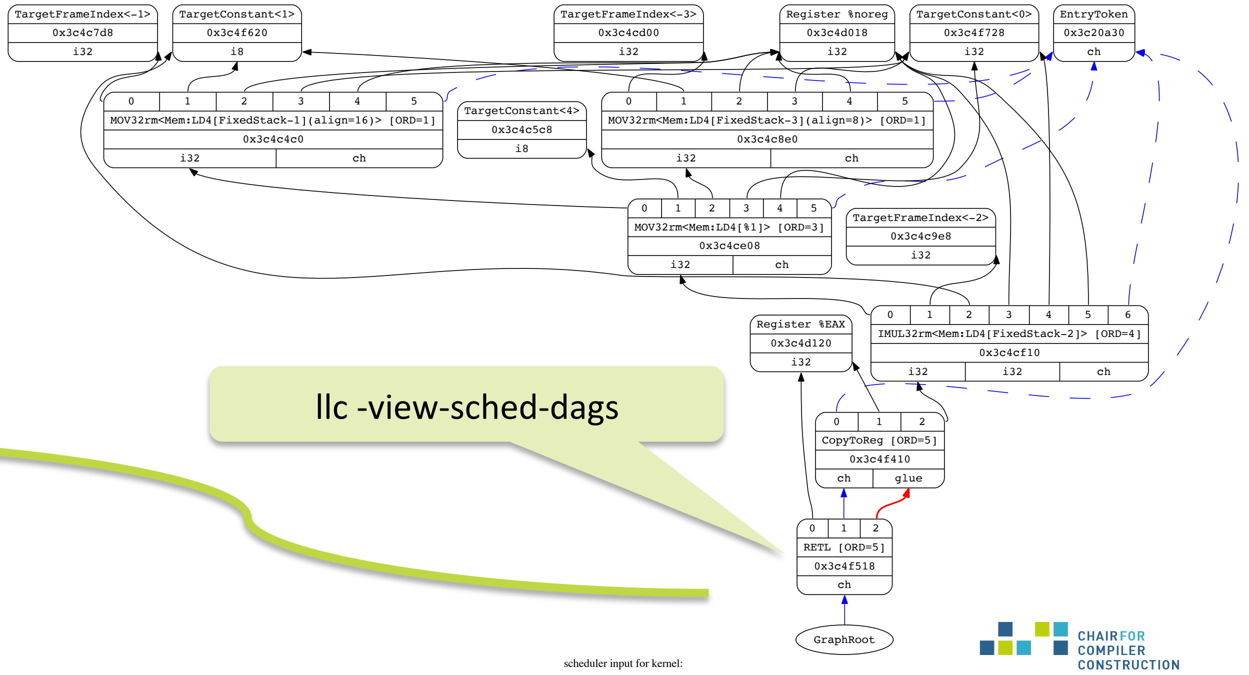
EntryToken 0x3e39a30 ch
FrameIndex<-1> 0x3e652b0 i32
FrameIndex<-3> 0x3e657d8 i32
undef 0x3e653b8 i32
FrameIndex<-2> 0x3e655c8 i32
load<LD4[FixedStack-1]> [ORD=1] 0x3e654c0 i32 ch
load<LD4[FixedStack-3]> [ORD=1] 0x3e658e0 i32 ch
Constant<2> 0x3e659e8 i32
load<LD4[FixedStack-2]> [ORD=1] 0x3e656d0 i32 ch
shl [ORD=2] 0x3e65af0 i32
add [ORD=2] 0x3e65bf8 i32
load<LD4[%1]> [ORD=3] 0x3e65e08 i32 ch
mul [ORD=4] 0x3e65f10 i32
Register %EAX 0x3e66120 i32
TargetConstant<0> 0x3e66018 i16
CopyToReg [ORD=5] 0x3e68410 ch glue
X86ISD::RET_FLAG [ORD=5] 0x3e68518 ch
GraphRoot
Constant<0> 0x3e65d00 i32

dag-combine1 input for kernel:

llc -view-dag-combine1-dags

Instruction selection – part II

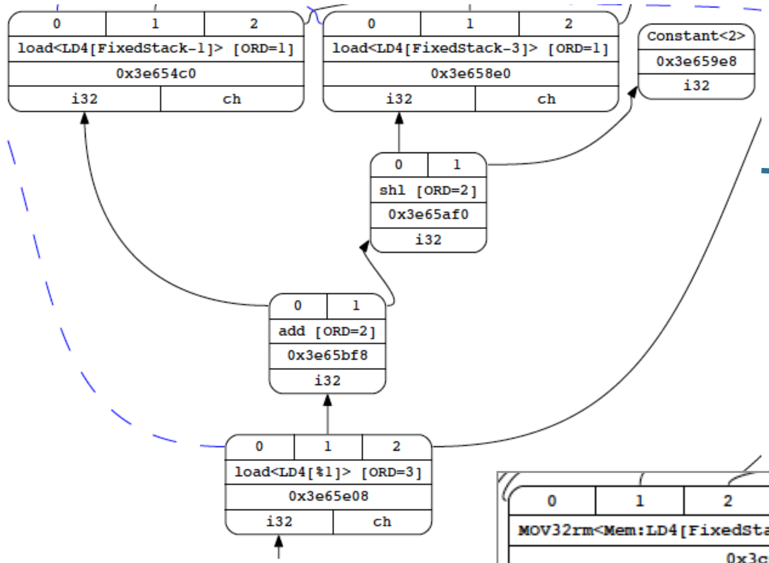




TargetFrameIndex<-1> 0x3c4c7d8 i32
TargetConstant<1> 0x3c4f620 i8
TargetFrameIndex<-3> 0x3c4cd00 i32
Register %noreg 0x3c4d018 i32
TargetConstant<0> 0x3c4f728 i32
EntryToken 0x3c20a30 ch
MOV32rm<Mem:LD4[FixedStack-1](align=16)> [ORD=1] 0x3c4c4c0 i32 ch
TargetConstant<4> 0x3c4c5c8 i8
MOV32rm<Mem:LD4[FixedStack-3](align=8)> [ORD=1] 0x3c4c8e0 i32 ch
MOV32rm<Mem:LD4[%1]> [ORD=3] 0x3c4ce08 i32 ch
TargetFrameIndex<-2> 0x3c4c9e8 i32
Register %EAX 0x3c4d120 i32
IMUL32rm<Mem:LD4[FixedStack-2]> [ORD=4] 0x3c4cf10 i32 i32 ch
CopyToReg [ORD=5] 0x3c4f410 ch glue
RETL [ORD=5] 0x3c4f518 ch
GraphRoot
scheduler input for kernel:

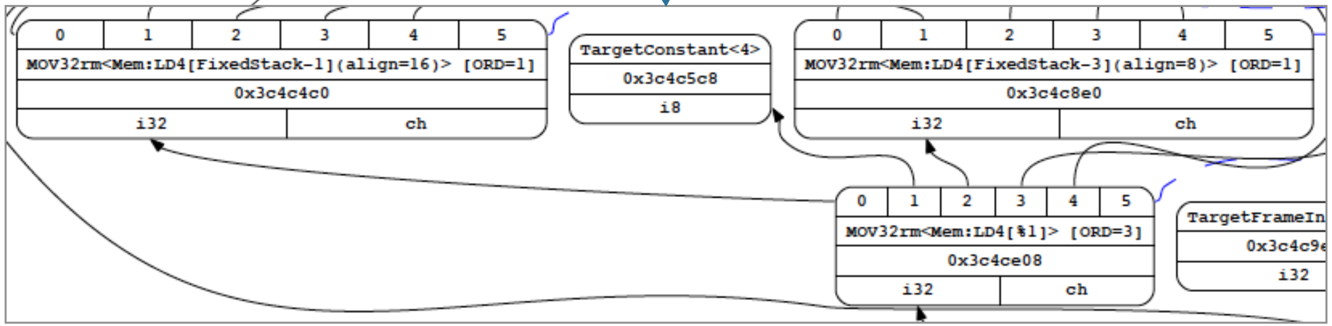
llc -view-sched-dags

SelectionDAG pattern matching



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

```
MOV32rm $dst, ($base, $index, $scale)
```



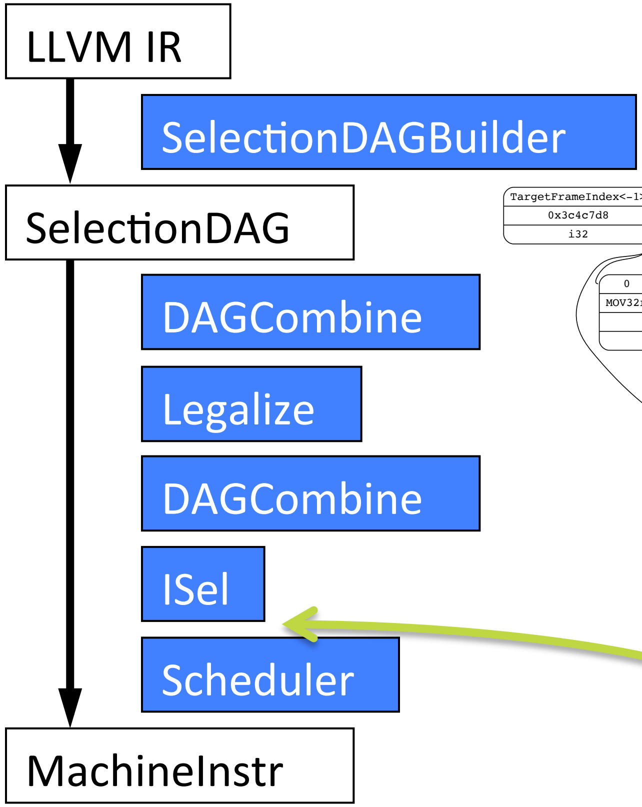
SelectionDAG patterns and TableGen

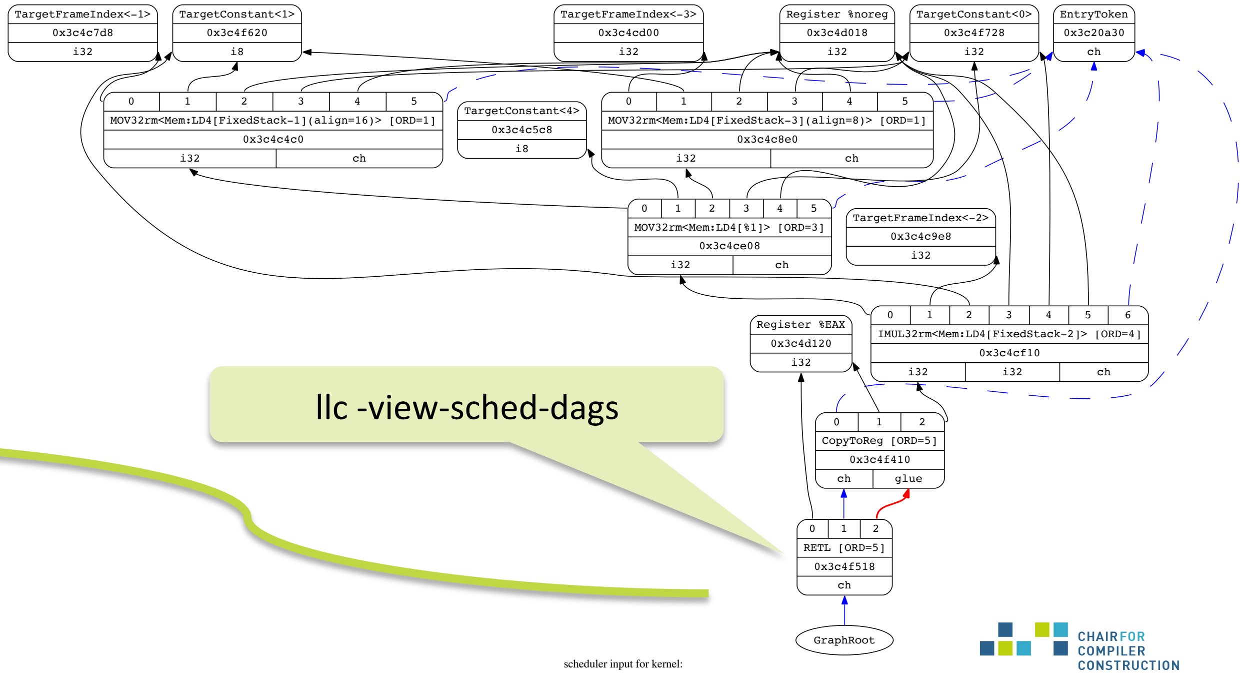
```
class X86Inst<bits<8> opcod, Format f, dag outs, dag ins, string AsmStr, list<dag> pattern,
              InstrItinClass itin> : Instruction {
          bits<8> Opcode = opcod;
          Format Form = f;
          dag OutOperandList = outs;
          dag InOperandList = ins;
          string AsmString = AsmStr;
          let Pattern = pattern;
          let Itinerary = itin;
}
def MOV32rr : X86Inst<0x89, MRMDestReg, (outs GR32:$dst), (ins GR32:$src),
                      "mov{l}\t{$src, $dst|$dst, $src}", [], IIC_MOV>;

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

def ADD32rr : X86Inst<0xC1, MRMDestReg, (outs GR32:$dst), (ins GR32:$src),
                      "add{l}\t{$src, $dst|$dst, $src}”,
                      [(set GR32:$dst, (add GR32:$dst, GR32:$src))], IIC_ADD_REG>;
```

Instruction selection – part II (again)

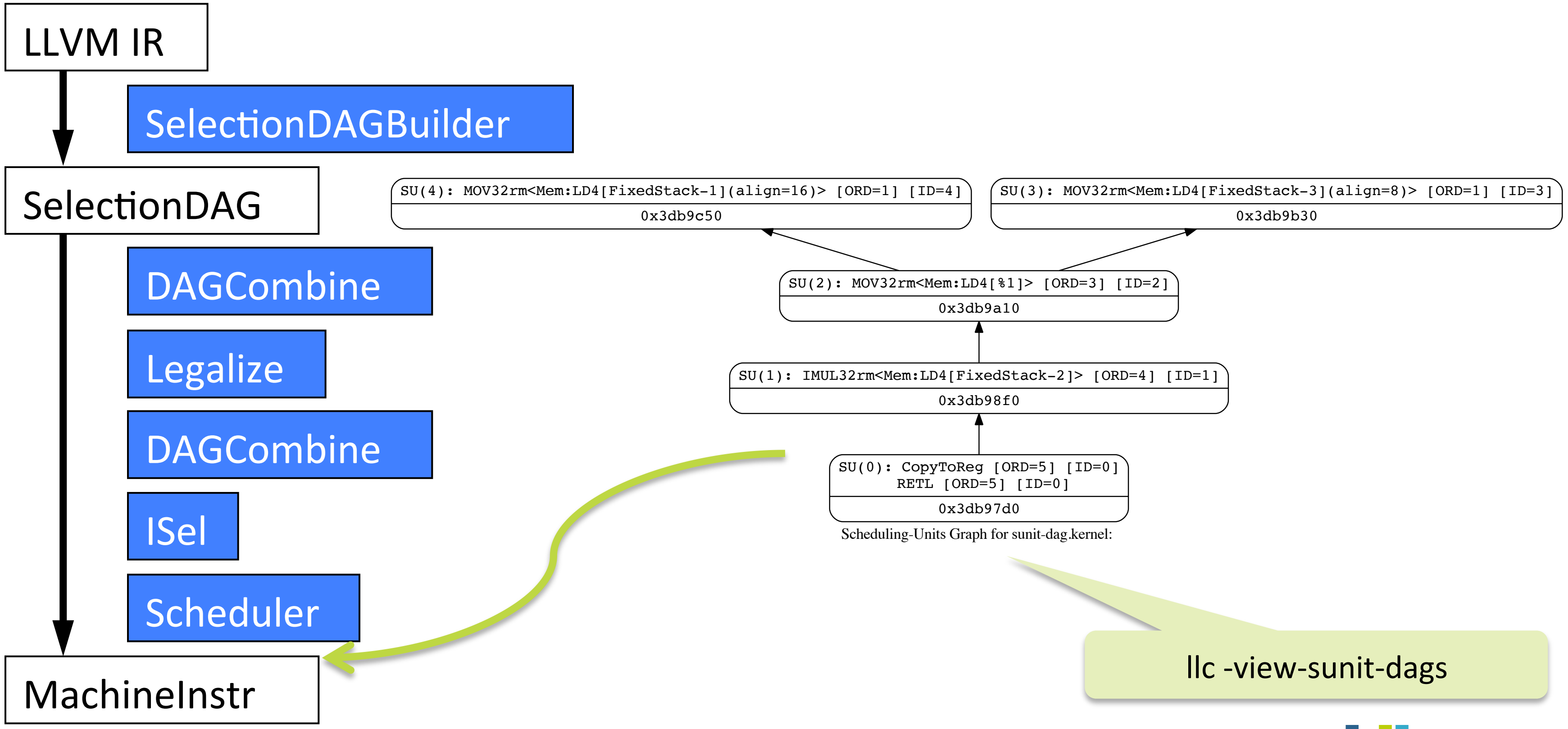




TargetFrameIndex<-1> 0x3c4c7d8 i32
TargetConstant<1> 0x3c4f620 i8
TargetFrameIndex<-3> 0x3c4cd00 i32
Register %noreg 0x3c4d018 i32
TargetConstant<0> 0x3c4f728 i32
EntryToken 0x3c20a30 ch
MOV32rm<Mem:LD4[FixedStack-1](align=16)> [ORD=1] 0x3c4c4c0 i32 ch
TargetConstant<4> 0x3c4c5c8 i8
MOV32rm<Mem:LD4[FixedStack-3](align=8)> [ORD=1] 0x3c4c8e0 i32 ch
MOV32rm<Mem:LD4[%1]> [ORD=3] 0x3c4ce08 i32 ch
TargetFrameIndex<-2> 0x3c4c9e8 i32
Register %EAX 0x3c4d120 i32
IMUL32rm<Mem:LD4[FixedStack-2]> [ORD=4] 0x3c4cf10 i32 i32 ch
CopyToReg [ORD=5] 0x3c4f410 ch glue
RETL [ORD=5] 0x3c4f518 ch
GraphRoot
scheduler input for kernel:

```
llc -view-sched-dags
```

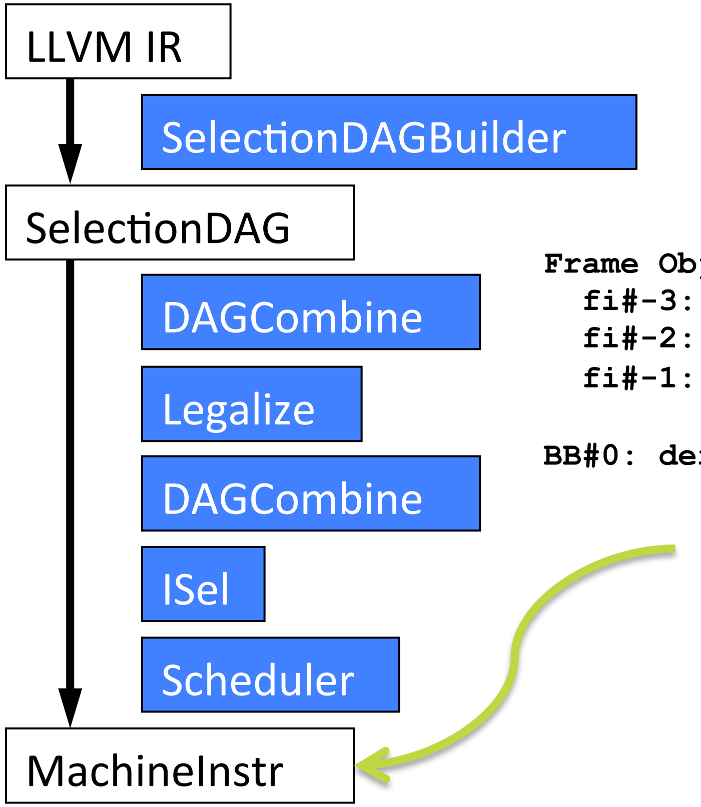
Instruction selection – part III


LLVM IR
SelectionDAGBuilder
SelectionDAG
DAGCombine
Legalize
DAGCombine
ISel
Scheduler
MachineInstr
SU(4): MOV32rm<Mem:LD4[FixedStack-1](align=16)> [ORD=1] [ID=4]
0x3db9c50
SU(3): MOV32rm<Mem:LD4[FixedStack-3](align=8)> [ORD=1] [ID=3]
0x3db9b30
SU(2): MOV32rm<Mem:LD4[%1]> [ORD=3] [ID=2]
0x3db9a10
SU(1): IMUL32rm<Mem:LD4[FixedStack-2]> [ORD=4] [ID=1]
0x3db98f0
SU(0): CopyToReg [ORD=5] [ID=0]
RETL [ORD=5] [ID=0]
0x3db97d0

Scheduling-Units Graph for sunit-dag.kernel:

```
llc -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<kill>, 4, %vreg0<kill>
          %vreg3<def> = IMUL32rm %vreg2<kill>, <fi#-2>
          %EAX<def> = COPY %vreg3<kill>
          RETL %EAX
```

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"
  - http://llvm.org/devmtg/2014-10/Slides/Cormack-BuildingAnLLVMBackend.pdf
- "Tutorial: Creating an LLVM Backend for the Cpu0 Architecture"
  - http://jonathan2251.github.io/lbd/llvmstructure.html

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

CHAIR FOR COMPILER CONSTRUCTION

cfaed CENTER FOR ADVANCING ELECTRONICS DRESDEN

The structure of LLVM backends

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

Thank you.

TECHNISCHE UNIVERSITÄT DRESDEN

DRESDEN concept

DFG

WR WISSENSCHAFTSRAT