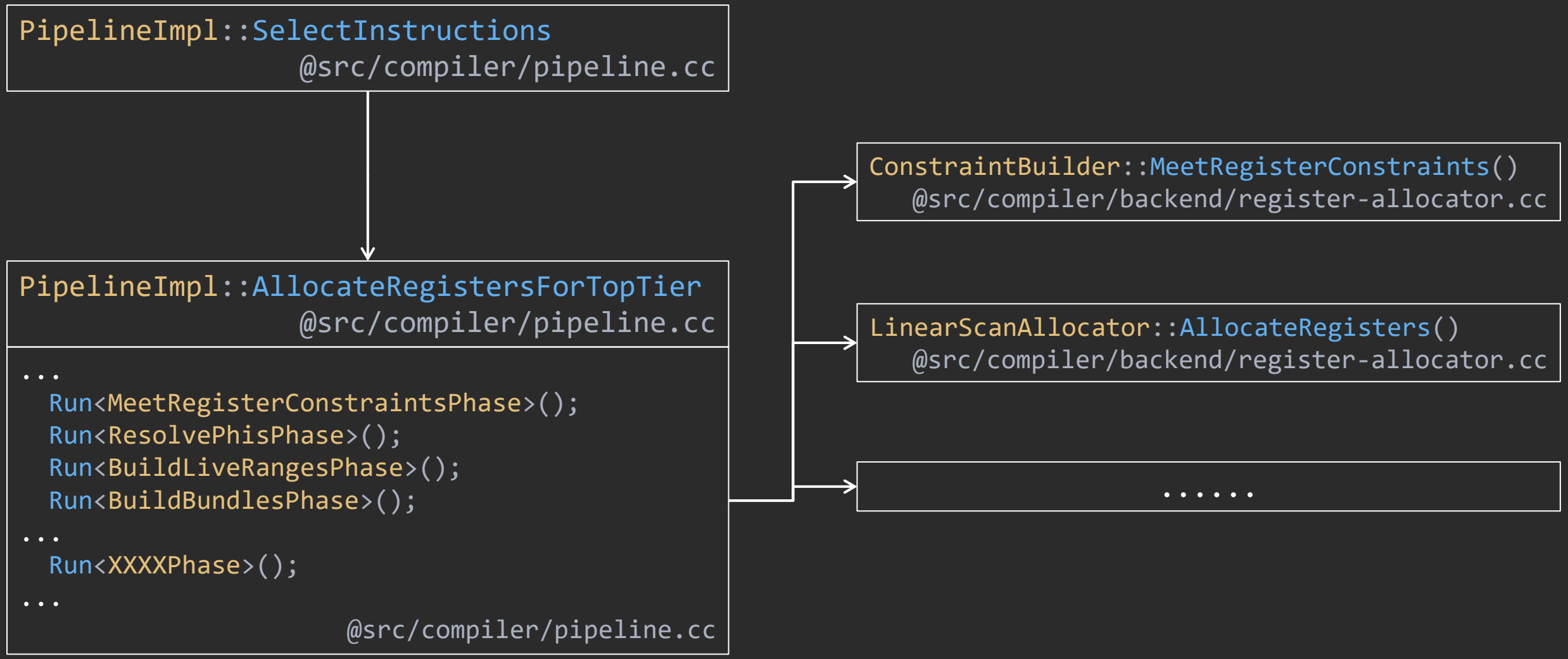
PLCT Lab

Register Allocator in V8

Liqiang TAO
taolq@outlook.com

10/29/20

Overview


PipelineImpl::SelectInstructions
@src/compiler/pipeline.cc
PipelineImpl::AllocateRegistersForTopTier
@src/compiler/pipeline.cc
...
Run<MeetRegisterConstraintsPhase>();
Run<ResolvePhisPhase>();
Run<BuildLiveRangesPhase>();
Run<BuildBundlesPhase>();
...
Run<XXXXPhase>();
...
@src/compiler/pipeline.cc
ConstraintBuilder::MeetRegisterConstraints()
@src/compiler/backend/register-allocator.cc
LinearScanAllocator::AllocateRegisters()
@src/compiler/backend/register-allocator.cc
.....

Overview

```
bool PipelineImpl::SelectInstructions(
    Linkage* linkage)
                                @src/compiler/pipeline.cc
```

```
void PipelineImpl::AllocateRegistersForTopTier(
    const RegisterConfiguration* config,
    CallDescriptor* call_descriptor,
    bool run_verifier)
                                @src/compiler/pipeline.cc
```

```
void PipelineImpl::AllocateRegistersForMidTier(
    const RegisterConfiguration* config,
    CallDescriptor* call_descriptor,
    bool run_verifier);
                                @src/compiler/pipeline.cc
```

AllocateRegistersForMidTier

```
@src/compiler/pipeline.cc
...
if (FLAG_turboprop_mid_tier_reg_alloc) {
  AllocateRegistersForMidTier(config, call_descriptor, run_verifier);
} else {
  AllocateRegistersForTopTier(config, call_descriptor, run_verifier);
}
...
```

```
@src/flags/flag-definitions.h
...
DEFINE_BOOL(turboprop_mid_tier_reg_alloc, false,
            "enable experimental mid-tier register allocator")
...
```

AllocateRegistersForTopTier

```
@src/compiler/pipeline.cc
...
  Run<MeetRegisterConstraintsPhase>();
  Run<ResolvePhisPhase>();
  Run<BuildLiveRangesPhase>();
  Run<BuildBundlesPhase>();

  TraceSequence(info(), data, "before register allocation");
...

  Run<AllocateGeneralRegistersPhase<LinearScanAllocator>>();

  if (data->sequence()->HasFPVirtualRegisters()) {
    Run<AllocateFPRegistersPhase<LinearScanAllocator>>();
  }

...
  Run<DecideSpillingModePhase>();
  Run<AssignSpillSlotsPhase>();
  Run<CommitAssignmentPhase>();
...
```

AllocateRegistersForTopTier

```
@src/compiler/pipeline.cc
...
  Run<AllocateGeneralRegistersPhase<LinearScanAllocator>>();
...

template <typename RegAllocator>
struct AllocateGeneralRegistersPhase {
  DECL_PIPELINE_PHASE_CONSTANTS(AllocateGeneralRegisters)

  void Run(PipelineData* data, Zone* temp_zone) {
    RegAllocator allocator(data->top_tier_register_allocation_data(),
                           RegisterKind::kGeneral, temp_zone);
    allocator.AllocateRegisters();
  }
};
```

```
@src/compiler/backend/register-allocator.h
class LinearScanAllocator final : public RegisterAllocator
```

AllocateRegistersForTopTier

- Some of instructions have architecture-specific operand information
- `MeetRegisterConstraintsPhase`

  Keep those arch-spec operand
- `BuildLiveRangesPhase`

  LiveRange - main data structure
- `AllocateGeneralRegistersPhase<LinearScanAllocator>`

  map vitural registers to actual registers

MeetRegisterConstraintsPhase

```
InstructionOperand* ConstraintBuilder::AllocateFixed(
    UnallocatedOperand* operand, int pos, bool is_tagged, bool is_input) {
  ...
  if (operand->HasFixedSlotPolicy()) {
    allocated = AllocatedOperand(AllocatedOperand::STACK_SLOT, rep,
                                 operand->fixed_slot_index());
  } else if (operand->HasFixedRegisterPolicy()) {
     ...
    allocated = AllocatedOperand(AllocatedOperand::REGISTER, rep,
                                 operand->fixed_register_index());
  } else if (operand->HasFixedFPRegisterPolicy()) {
     ...
  } else {
    UNREACHABLE();
  }
  ...
  InstructionOperand::ReplaceWith(operand, &allocated);
  ...
  return operand;
}
```

BuildLiveRangesPhase

```
TopLevelLiveRange* LiveRangeBuilder::LiveRangeFor(InstructionOperand* operand,
                                                  SpillMode spill_mode) {
  if (operand->IsUnallocated()) {
    return data()->GetOrCreateLiveRangeFor(
        UnallocatedOperand::cast(operand)->virtual_register());
  } else if (operand->IsConstant()) {
    return data()->GetOrCreateLiveRangeFor(
        ConstantOperand::cast(operand)->virtual_register());
  } else if (operand->IsRegister()) {
    return FixedLiveRangeFor(
        LocationOperand::cast(operand)->GetRegister().code(), spill_mode);
  } else if (operand->IsFPRegister()) {
    LocationOperand* op = LocationOperand::cast(operand);
    return FixedFPLiveRangeFor(op->register_code(), op->representation(),
                               spill_mode);
  } else {
    return nullptr;
  }
}
```

LinearScanAllocator

```
void LinearScanAllocator::ProcessCurrentRange(LiveRange* current,
                                              SpillMode spill_mode) {
  ...
  FindFreeRegistersForRange(current, free_until_pos);
  if (!TryAllocatePreferredReg(current, free_until_pos)) {
    ...
  }
  ...
}
```

```
bool LinearScanAllocator::TryAllocatePreferredReg(
    LiveRange* current, const Vector<LifetimePosition>& free_until_pos) {
  int hint_register;
  if (current->RegisterFromControlFlow(&hint_register) ||
      current->FirstHintPosition(&hint_register) != nullptr ||
      current->RegisterFromBundle(&hint_register)) {
    ...
  }
  return false;
}
```

Summary

- keep fixed registers in previous select instruction
- no target-specific implementation for the register allocator
- could not meet constraints of C-extension instructions ?
- learn register constraints in select instruction ?

Reference

- https://docs.google.com/document/d/1aeUugkWCF1biPB4tTZ2KT3mmRSDV785yWZhwzlJe5xY

V8 Register allocation

文件 编辑 查看 工具 帮助

申请编辑

V8 Turbofan Register Allocation Design

This is a high-level overview of the design of register allocation in the Turbofan compiler in V8. It focuses on those large picture concerns that span more than one type/method and are cumbersome to describe through code comments. It provides design rationale for some of the less obvious design choices.

Overview

Register allocation runs after instruction selection, and is followed by frame elision, jump threading[1] and finally code generation. The key inputs to register allocation are the reverse postorder (RPO) control flow graph (CFG), in split-edge form, produced by instruction selection (isel), and the list of architecture-specific registers available for allocation. It consists of a number of phases, which are all coordinated from `Pipeline::AllocateRegisters`, in `pipeline.cc`.

PLCT Lab

Thanks!

Liqiang TAO
taolq@outlook.com