MIR-Canon: Improving Code Diff Through Canonical Transformation

Puyan Lotfi
Apple Inc.

Hi I’m Puyan Lotfi and today I’ll be talking about MIR-Canon.

What is MIR?

- MIR (Machine IR) is the newer IR form for MachineInstrs.

* The first question many of you may have is, “what is mir?”
* MIR is the newer serializable/deserializable MachineIR for MachineInstrs

What is MIR?

```
bb.0:
  liveins: $w0, $x1, $x2
  %1:gpr64 = COPY $x1
  %0:gpr32 = COPY $w0
  %2:gpr32 = COPY $wzr
  ...
  %3:gpr64sp = COPY $sp
  %foo1:fpr64 = FMOVDi 28
  %foo2:fpr64 = LDRDui %stack.1, 0
  %foo3:fpr64 = LDRDui %stack.2, 0
  %foo4:fpr64 = FMULDrr %foo2, %foo3
  %foo5:fpr64 = FDIVDrr %foo4, %foo3
  %4:gpr32 = MOVi32imm 8
  %vreg234_0:gpr32 = COPY $w0
  ...
  %foo:gpr32 = MOVi32imm 0
  $w0 = COPY %foo
  $x2 = REG_SEQUENCE %vreg234_0, %subreg.dsub0, \
                     %vreg645646_1, %subreg.dsub1
  RET_ReallyLR implicit $x2
```


* This is what it looks like.
* It looks somewhere in-between LLVM IR and the old MachineInstr print output.

What is MIR-Canon?

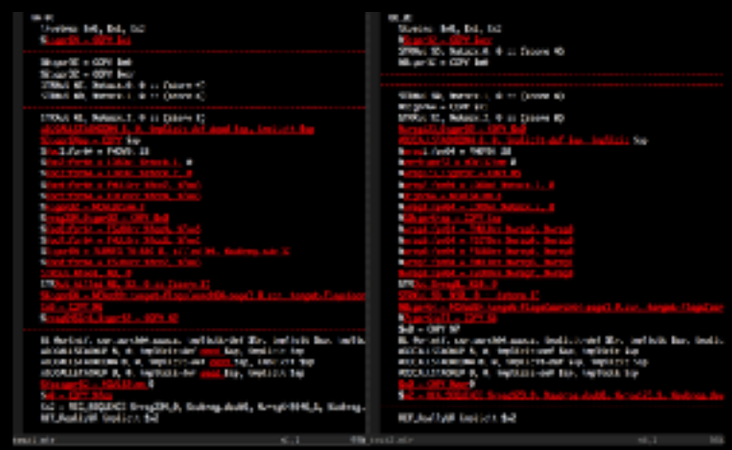
- A new pass that canonically transforms Machine IR (MIR).
- Reduces register naming and scheduling differences.
- The goal is to make semantic differences stand out.

* The next question many of you may have is “what is mir-canon?”

* MIR-Canon is a new machine pass built with the aim of doing canonical transformations on MIR code that canonicalize register naming and instruction placement for a given mir file.

* The goal is that any two similar programs processed with mir-canon would have their semantic differences stand out and have their non-semantic differences fade to the background when viewed in a diff tool.

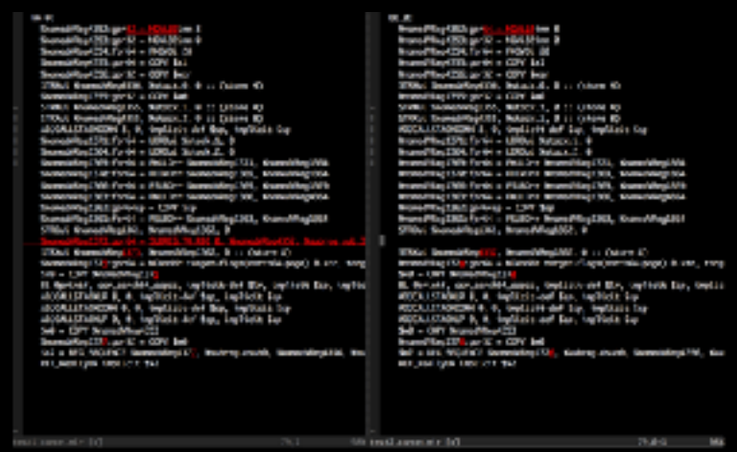
Problem Statement



* So a common workflow is to use some tool like vimdiff to compare two somewhat similar IR, MIR or assembly files for differences.

* Because of the nature of how register operands can have cascading naming differences and because of scheduling differences we often end up with a diff view that looks like this.

More Ideal Situation



A more ideal scenario is where the differences happen to be isolated largely to the parts that are semantically different and where the register operands and instruction orderings are mostly consistent across the entirety of the files being diff'ed. Notice that we now can clearly see the subreg_to_reg extension versus the MOVi64imm difference now.

MIR-Canon

Can be invoked through llc:

```
llc -run-pass mir-canonicalizer -o - foo.mir
```


Before we go any further I'd like to point out that mir-canon can be involved simply by giving -run-pass to llc with mir-canonicalizer.

Rationale

- Wanted a tool for improving the state of code diff as well as code verification.
  - Has to be more than just trivial sorting and renaming.
  - We did not want to build yet another diff tool.
  - Must preserve semantics.
- We wanted to improve the process of comparing MIR produced from identical IR applied with different passes.
- Initially used for GlobalISel vs DAGISel verification.

* So what were out reasons for building mir-canon?
* Well we wanted to improve the state of code diff and code verification. That means we need to preserve semantics and that trivial sorting and operand renaming is not enough.
* We make a major assumption that any two mir files we intend to compare likely came from the same source IR at some stage in the pipeline.
* Our initial primary application was at this point has been diffing and verifying IR thats been selected with GlobalISel versus SelectionDAG.

Considerations

- Analyze one file at a time as a generic machine pass.
- Leverage existing diff tools.
- Def-use graph used to represent program semantics?
- Could we alphabetical reorder based on dump output?

* Before implementing MIR-Canon an early decision was made to leverage existing diff tools, that MIR-Canon shouldn't be yet another diff tool.
* This means that analyzing one file at a time in a generic machine pass made sense.
* Other ideas that were considered were that the def-use graph can be used to arrive at a canonical form that still adheres to proper semantics.
* Also we thought a bit if the idea of alphabetical reordering of instructions based on dump output in certain situations to arrive at a more canonical result.

Getting to a more canonical form???

```
bb.0:
  liveins: $w0, $x1, $x2
  %1:gpr64 = COPY $x1
  %0:gpr32 = COPY $w0
  %2:gpr32 = COPY $wzr
  STRWui %2, %stack.0, 0 :: (store 4)
  STRWui %0, %stack.1, 0 :: (store 4)
  STRXui %1, %stack.2, 0 :: (store 8)
  ADJCALLSTACKDOWN 8, 0, implicit-def dead $sp, implicit $sp
  %3:gpr64sp = COPY $sp
  %foo1:fpr64 = FMOVDi 28
  %foo2:fpr64 = LDRDui %stack.1, 0
  %foo3:fpr64 = LDRDui %stack.2, 0
  %foo4:fpr64 = FMULDrr %foo2, %foo3
  %foo5:fpr64 = FDIVDrr %foo4, %foo3
  %4:gpr32 = MOVi32imm 8
  %vreg234_0:gpr32 = COPY $w0
  %foo6:fpr64 = FSUBDrr %foo4, %foo5
  %foo7:fpr64 = FMULDrr %foo6, %foo1
  %5:gpr64 = SUBREG_TO_REG 0, killed %4, %subreg.sub_32
  %foo8:fpr64 = FSUBDrr %foo7, %foo3
  STRDui %foo8, %3, 0
  STRXui killed %5, %3, 0 :: (store 8)
  %6:gpr64 = MOVaddr target-flags(aarch64-page) @.str,...
  $x0 = COPY %6
  %vreg645646_1:gpr32 = COPY %2
  BL @printf, csr_aarch64_aapcs, implicit-def $lr,...
  ADJCALLSTACKUP 8, 0, implicit-def dead $sp, implicit $sp
  ADJCALLSTACKDOWN 0, 0, implicit-def dead $sp, implicit $sp
  ADJCALLSTACKUP 0, 0, implicit-def dead $sp, implicit $sp
  %foo:gpr32 = MOVi32imm 0
  $w0 = COPY %foo
  $x2 = REG_SEQUENCE %vreg234_0, %subreg.dsub0,
                     %vreg645646_1, %subreg.dsub1
  RET_ReallyLR implicit $x2
```

Getting to a more canonical form???

```
bb.0:
  liveins: $w0, $x1, $x2
  %1:gpr64 = COPY $x1
  %0:gpr32 = COPY $w0
  %2:gpr32 = COPY $wzr
  STRWui %2, %stack.0, 0 :: (store 4)
  STRWui %0, %stack.1, 0 :: (store 4)
  STRXui %1, %stack.2, 0 :: (store 8)
  ADJCALLSTACKDOWN 8, 0, implicit-def dead $sp, implicit $sp
  %3:gpr64sp = COPY $sp
  %foo1:fpr64 = FMOVDi 28
  %foo2:fpr64 = LDRDui %stack.1, 0
  %foo3:fpr64 = LDRDui %stack.2, 0
  %foo4:fpr64 = FMULDrr %foo2, %foo3
  %foo5:fpr64 = FDIVDrr %foo4, %foo3
  %4:gpr32 = MOVi32imm 8
  %vreg234_0:gpr32 = COPY $w0
  %foo6:fpr64 = FSUBDrr %foo4, %foo5
  %foo7:fpr64 = FMULDrr %foo6, %foo1
  %5:gpr64 = SUBREG_TO_REG 0, killed %4, %subreg.sub_32
  %foo8:fpr64 = FSUBDrr %foo7, %foo3
  STRDui %foo8, %3, 0
  STRXui killed %5, %3, 0 :: (store 8)
  %6:gpr64 = MOVaddr target-flags(aarch64-page) @.str,...
  $x0 = COPY %6
  %vreg645646_1:gpr32 = COPY %2
  BL @printf, csr_aarch64_aapcs, implicit-def $lr,...
  ADJCALLSTACKUP 8, 0, implicit-def dead $sp, implicit $sp
  ADJCALLSTACKDOWN 0, 0, implicit-def dead $sp, implicit $sp
  ADJCALLSTACKUP 0, 0, implicit-def dead $sp, implicit $sp
  %foo:gpr32 = MOVi32imm 0
  $w0 = COPY %foo
  $x2 = REG_SEQUENCE %vreg234_0, %subreg.dsub0,
                     %vreg645646_1, %subreg.dsub1
  RET_ReallyLR implicit $x2
```

Getting to a more canonical form???

```
bb.0:
  liveins: $w0, $x1, $x2
  %1:gpr64 = COPY $x1
  %0:gpr32 = COPY $w0
  %2:gpr32 = COPY $wzr
  STRWui %2, %stack.0, 0 :: (store 4)
  STRWui %0, %stack.1, 0 :: (store 4)
  STRXui %1, %stack.2, 0 :: (store 8)
  ADJCALLSTACKDOWN 8, 0, implicit-def dead $sp, implicit $sp
  %3:gpr64sp = COPY $sp
  %foo1:fpr64 = FMOVDi 28
  %foo2:fpr64 = LDRDui %stack.1, 0
  %foo3:fpr64 = LDRDui %stack.2, 0
  %foo4:fpr64 = FMULDrr %foo2, %foo3
  %foo5:fpr64 = FDIVDrr %foo4, %foo3
  %4:gpr32 = MOVi32imm 8
  %vreg234_0:gpr32 = COPY $w0
  %foo6:fpr64 = FSUBDrr %foo4, %foo5
  %foo7:fpr64 = FMULDrr %foo6, %foo1
  %5:gpr64 = SUBREG_TO_REG 0, killed %4, %subreg.sub_32
  %foo8:fpr64 = FSUBDrr %foo7, %foo3
  STRDui %foo8, %3, 0
  STRXui killed %5, %3, 0 :: (store 8)
  %6:gpr64 = MOVaddr target-flags(aarch64-page) @.str,...
  $x0 = COPY %6
  %vreg645646_1:gpr32 = COPY %2
  BL @printf, csr_aarch64_aapcs, implicit-def $lr,...
  ADJCALLSTACKUP 8, 0, implicit-def dead $sp, implicit $sp
  ADJCALLSTACKDOWN 0, 0, implicit-def dead $sp, implicit $sp
  ADJCALLSTACKUP 0, 0, implicit-def dead $sp, implicit $sp
  %foo:gpr32 = MOVi32imm 0
  $w0 = COPY %foo
  $x2 = REG_SEQUENCE %vreg234_0, %subreg.dsub0,
                     %vreg645646_1, %subreg.dsub1
  RET_ReallyLR implicit $x2
```

```
bb.0:
  %namedVReg4352:gpr32 = MOVi32imm 8
  %namedVReg4353:gpr32 = MOVi32imm 0
  %namedVReg4354:fpr64 = FMOVDi 28
  %namedVReg4355:gpr64 = COPY $x1
  %namedVReg4356:gpr32 = COPY $wzr
  STRWui %namedVReg4356, %stack.0, 0 :: (store 4)
  %namedVReg1355:gpr32 = COPY $w0
  STRWui %namedVReg1355, %stack.1, 0 :: (store 4)
  STRXui %namedVReg4355, %stack.2, 0 :: (store 8)
  ADJCALLSTACKDOWN 8, 0, implicit-def $sp, implicit $sp
  %namedVReg1371:fpr64 = LDRDui %stack.1, 0
  %namedVReg1364:fpr64 = LDRDui %stack.2, 0
  %namedVReg1369:fpr64 = FMULDrr %namedVReg1371, %namedVReg1364
  %namedVReg1370:fpr64 = FDIVDrr %namedVReg1369, %namedVReg1364
  %namedVReg1366:fpr64 = FSUBDrr %namedVReg1369, %namedVReg1370
  %namedVReg1363:fpr64 = FMULDrr %namedVReg1366, %namedVReg4354
  %namedVReg1362:gpr64sp = COPY $sp
  %namedVReg1361:fpr64 = FSUBDrr %namedVReg1363, %namedVReg1364
  STRDui %namedVReg1361, %namedVReg1362, 0
  %namedVReg1373:gpr64 = SUBREG_TO_REG 0, %namedVReg4352, %subreg.sub_32
  STRXui %namedVReg1373, %namedVReg1362, 0 :: (store 8)
  %namedVReg1375:gpr64 = MOVaddr target-flags(aarch64-page) @.str, ...
  $x0 = COPY %namedVReg1375
  BL @printf, csr_aarch64_aapcs, implicit-def $lr, ...
  ADJCALLSTACKUP 8, 0, implicit-def $sp, implicit $sp
  ADJCALLSTACKDOWN 0, 0, implicit-def $sp, implicit $sp
  ADJCALLSTACKUP 0, 0, implicit-def $sp, implicit $sp
  $w0 = COPY %namedVReg4353
  %namedVReg1377:gpr32 = COPY $w0
  $x2 = REG_SEQUENCE %namedVReg1377, %subreg.dsub0,
                     %namedVReg4356, %subreg.dsub1
  RET_ReallyLR implicit $x2
```

Notice how the code on the right is sorted in some places in alphabetical order and that the operands are all incrementally named "namedVReg####."

Algorithmic Details

Techniques:

1. Virtual Register Renaming
2. Instruction Reordering
3. Code Folding

Design Decisions:

- ISA Agnostic: Works on all ISAs; no opcode rewriting.
- Local: For each basic block in Reverse Post Order.

* The way we arrive at a more canonical form is through the following techniques: Virtual Register Renaming, Instruction Reordering, and Code folding.
* All of these techniques are ISA Agnostic, meaning that there are no ISA specific details used. This so that mir-canon has the most broad impact across the most MIR Code across all targets in LLVM.
* All these techniques are also local only. We apply them only on a per basic block basis and we don't cross basic block boundaries. However, we do process all the basic blocks in a CFG in a canonical reverse post ordering.

Register Renaming

Def-Use Walk Virtual Register Renaming for a given basic block:

1. Scan basic block for side-effects (writes to phyregs or memory).

2. For each side-effecting instruction walk the def-use graph. Let the walk ordering determine a renaming scheme for the virtual registers encountered in the walk.

3. The high-level goal is to let the def-use chain determine the VReg names.

Register Renaming

```
STRWui %2, %stack.0, 0 :: (store 4)
STRWui %0, %stack.1, 0 :: (store 4)
STRXui %1, %stack.2, 0 :: (store 8)
ADJCALLSTACKDOWN 8, 0, ...
%3:gpr64sp = COPY $sp
%foo1:fpr64 = FMOVDi 28
%foo2:fpr64 = LDRDui %stack.1, 0
%foo3:fpr64 = LDRDui %stack.2, 0
%foo4:fpr64 = FMULDrr %foo2, %foo3
%foo5:fpr64 = FDIVDrr %foo4, %foo3
%4:gpr32 = MOVi32imm 8
%vreg234_0:gpr32 = COPY $w0
%foo6:fpr64 = FSUBDrr %foo4, %foo5
%foo7:fpr64 = FMULDrr %foo6, %foo1
%5:gpr64 = SUBREG_TO_REG ...
%foo8:fpr64 = FSUBDrr %foo7, %foo3
STRDui %foo8, %3, 0
STRXui killed %5, %3, 0 :: (store 8)
%6:gpr64 = MOVaddr ...
$x0 = COPY %6
```

Register Renaming

Identify side-effecting instructions:

```
STRWui %2, %stack.0, 0 :: (store 4)
STRWui %0, %stack.1, 0 :: (store 4)
STRXui %1, %stack.2, 0 :: (store 8)
ADJCALLSTACKDOWN 8, 0, ...
%3:gpr64sp = COPY $sp
%foo1:fpr64 = FMOVDi 28
%foo2:fpr64 = LDRDui %stack.1, 0
%foo3:fpr64 = LDRDui %stack.2, 0
%foo4:fpr64 = FMULDrr %foo2, %foo3
%foo5:fpr64 = FDIVDrr %foo4, %foo3
%4:gpr32 = MOVi32imm 8
%vreg234_0:gpr32 = COPY $w0
%foo6:fpr64 = FSUBDrr %foo4, %foo5
%foo7:fpr64 = FMULDrr %foo6, %foo1
%5:gpr64 = SUBREG_TO_REG ...
%foo8:fpr64 = FSUBDrr %foo7, %foo3
STRDui %foo8, %3, 0
STRXui killed %5, %3, 0 :: (store 8)
%6:gpr64 = MOVaddr ...
$x0 = COPY %6
```

Register Renaming

Identify side-effecting instructions:

```
→   STRWui %2, %stack.0, 0 :: (store 4)
→   STRWui %0, %stack.1, 0 :: (store 4)
→   STRXui %1, %stack.2, 0 :: (store 8)
    ADJCALLSTACKDOWN 8, 0, ...
    %3:gpr64sp = COPY $sp
    %foo1:fpr64 = FMOVDi 28
    %foo2:fpr64 = LDRDui %stack.1, 0
    %foo3:fpr64 = LDRDui %stack.2, 0
    %foo4:fpr64 = FMULDrr %foo2, %foo3
    %foo5:fpr64 = FDIVDrr %foo4, %foo3
    %4:gpr32 = MOVi32imm 8
    %vreg234_0:gpr32 = COPY $w0
    %foo6:fpr64 = FSUBDrr %foo4, %foo5
    %foo7:fpr64 = FMULDrr %foo6, %foo1
    %5:gpr64 = SUBREG_TO_REG ...
    %foo8:fpr64 = FSUBDrr %foo7, %foo3
→   STRDui %foo8, %3, 0
→   STRXui killed %5, %3, 0 :: (store 8)
    %6:gpr64 = MOVaddr ...
→   $x0 = COPY %6
```

Register Renaming

Identify side-effecting instructions (memory stores or physreg writes):

```
→   STRWui %2, %stack.0, 0 :: (store 4)
→   STRWui %0, %stack.1, 0 :: (store 4)
→   STRXui %1, %stack.2, 0 :: (store 8)
    ADJCALLSTACKDOWN 8, 0, ...
    %3:gpr64sp = COPY $sp
    %foo1:fpr64 = FMOVDi 28
    %foo2:fpr64 = LDRDui %stack.1, 0
    %foo3:fpr64 = LDRDui %stack.2, 0
    %foo4:fpr64 = FMULDrr %foo2, %foo3
    %foo5:fpr64 = FDIVDrr %foo4, %foo3
    %4:gpr32 = MOVi32imm 8
    %vreg234_0:gpr32 = COPY $w0
    %foo6:fpr64 = FSUBDrr %foo4, %foo5
    %foo7:fpr64 = FMULDrr %foo6, %foo1
    %5:gpr64 = SUBREG_TO_REG ...
    %foo8:fpr64 = FSUBDrr %foo7, %foo3
→   STRDui %foo8, %3, 0
→   STRXui killed %5, %3, 0 :: (store 8)
    %6:gpr64 = MOVaddr ...
→   $x0 = COPY %6
```

Register Renaming

Identify side-effecting instructions (memory stores or physreg writes):

```
→   STRWui %2, %stack.0, 0 :: (store 4)
→   STRWui %0, %stack.1, 0 :: (store 4)
→   STRXui %1, %stack.2, 0 :: (store 8)
    ADJCALLSTACKDOWN 8, 0, ...
    %3:gpr64sp = COPY $sp
    %foo1:fpr64 = FMOVDi 28
    %foo2:fpr64 = LDRDui %stack.1, 0
    %foo3:fpr64 = LDRDui %stack.2, 0
    %foo4:fpr64 = FMULDrr %foo2, %foo3
    %foo5:fpr64 = FDIVDrr %foo4, %foo3
    %4:gpr32 = MOVi32imm 8
    %vreg234_0:gpr32 = COPY $w0
    %foo6:fpr64 = FSUBDrr %foo4, %foo5
    %foo7:fpr64 = FMULDrr %foo6, %foo1
    %5:gpr64 = SUBREG_TO_REG ...
    %foo8:fpr64 = FSUBDrr %foo7, %foo3
→   STRDui %foo8, %3, 0
→   STRXui killed %5, %3, 0 :: (store 8)
    %6:gpr64 = MOVaddr ...
→   $x0 = COPY %6
```

Register Renaming

Identify side-effecting instructions (memory stores or physreg writes):

```
→   STRWui %2, %stack.0, 0 :: (store 4)
→   STRWui %0, %stack.1, 0 :: (store 4)
→   STRXui %1, %stack.2, 0 :: (store 8)
    ADJCALLSTACKDOWN 8, 0, ...
    %3:gpr64sp = COPY $sp
    %foo1:fpr64 = FMOVDi 28
    %foo2:fpr64 = LDRDui %stack.1, 0
    %foo3:fpr64 = LDRDui %stack.2, 0
    %foo4:fpr64 = FMULDrr %foo2, %foo3
    %foo5:fpr64 = FDIVDrr %foo4, %foo3
    %4:gpr32 = MOVi32imm 8
    %vreg234_0:gpr32 = COPY $w0
    %foo6:fpr64 = FSUBDrr %foo4, %foo5
    %foo7:fpr64 = FMULDrr %foo6, %foo1
    %5:gpr64 = SUBREG_TO_REG ...
    %foo8:fpr64 = FSUBDrr %foo7, %foo3
→   STRDui %foo8, %3, 0
→   STRXui killed %5, %3, 0 :: (store 8)
    %6:gpr64 = MOVaddr ...
→   $x0 = COPY %6
```



```
→   STRWui %2, %stack.0, 0 :: (store 4)
→   STRWui %0, %stack.1, 0 :: (store 4)
→   STRXui %1, %stack.2, 0 :: (store 8)
→   STRDui %foo8, %3, 0
→   STRXui killed %5, %3, 0 :: (store 8)
→   $x0 = COPY %6
```

Register Renaming

Identify side-effecting instructions (memory stores or physreg writes):

```
→   STRWui %2, %stack.0, 0 :: (store 4)
→   STRWui %0, %stack.1, 0 :: (store 4)
→   STRXui %1, %stack.2, 0 :: (store 8)
    ADJCALLSTACKDOWN 8, 0, ...
    %3:gpr64sp = COPY $sp
    %foo1:fpr64 = FMOVDi 28
    %foo2:fpr64 = LDRDui %stack.1, 0
    %foo3:fpr64 = LDRDui %stack.2, 0
    %foo4:fpr64 = FMULDrr %foo2, %foo3
    %foo5:fpr64 = FDIVDrr %foo4, %foo3
    %4:gpr32 = MOVi32imm 8
    %vreg234_0:gpr32 = COPY $w0
    %foo6:fpr64 = FSUBDrr %foo4, %foo5
    %foo7:fpr64 = FMULDrr %foo6, %foo1
    %5:gpr64 = SUBREG_TO_REG ...
    %foo8:fpr64 = FSUBDrr %foo7, %foo3
→   STRDui %foo8, %3, 0
→   STRXui killed %5, %3, 0 :: (store 8)
    %6:gpr64 = MOVaddr ...
→   $x0 = COPY %6
```

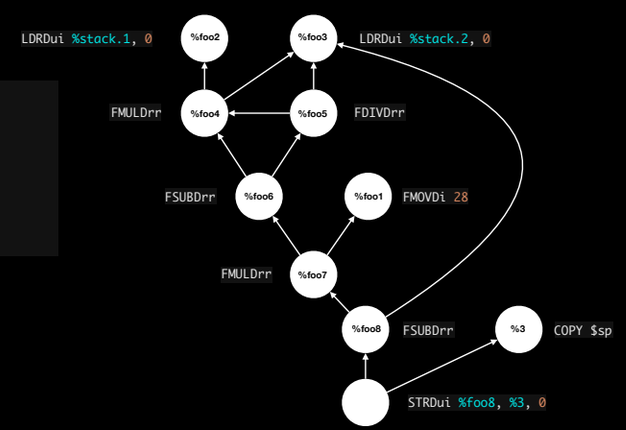
Process in bottom up order:

```
→   $x0 = COPY %6
→   STRXui killed %5, %3, 0 :: (store 8)
→   STRDui %foo8, %3, 0
→   STRXui %1, %stack.2, 0 :: (store 8)
→   STRWui %0, %stack.1, 0 :: (store 4)
→   STRWui %2, %stack.0, 0 :: (store 4)
```

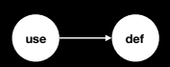
Register Renaming

For each side-effect we do a bottom Up def-use Graph Walk:

```
%3:gpr64sp = COPY $sp
%foo1:fpr64 = FMOVDi 28
%foo2:fpr64 = LDRDui %stack.1, 0
%foo3:fpr64 = LDRDui %stack.2, 0
%foo4:fpr64 = FMULDrr %foo2, %foo3
%foo5:fpr64 = FDIVDrr %foo4, %foo3
%foo6:fpr64 = FSUBDrr %foo4, %foo5
%foo7:fpr64 = FMULDrr %foo6, %foo1
%foo8:fpr64 = FSUBDrr %foo7, %foo3
STRDui %foo8, %3, 0
```

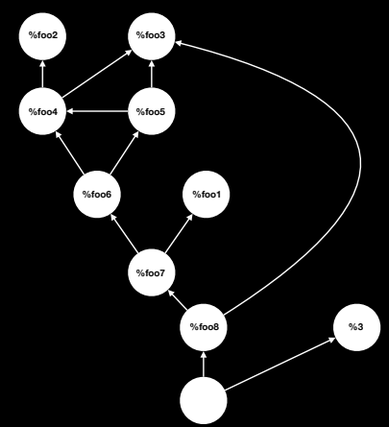


Note:



Register Renaming

For each side-effect we do a bottom Up def-use Graph Walk:



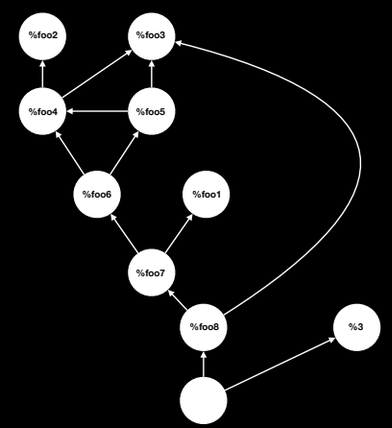
Naming strategy, VReg Number Cursor:

```
class Cursor {

  void SkipVRegs(unsigned I); // Skips VNumber Modulo M

  Cursor() {
    SkipVRegs(MRI.createIncompleteVirtualRegister());
  }

  unsigned createVReg(const TargetRegisterClass *RC);
};
```


* The mechanism we used to effectively keep state of what we are renaming the vregs to is this vreg number cursor.
* Initially it calls createIncompleteVirtualRegister on MRI to get a sense of how many vregs are allocated in the MIR so that we can figure out how much to initially round up an initial number used in naming the vregs.

Register Renaming

For each side-effect we do a bottom Up def-use Graph Walk:



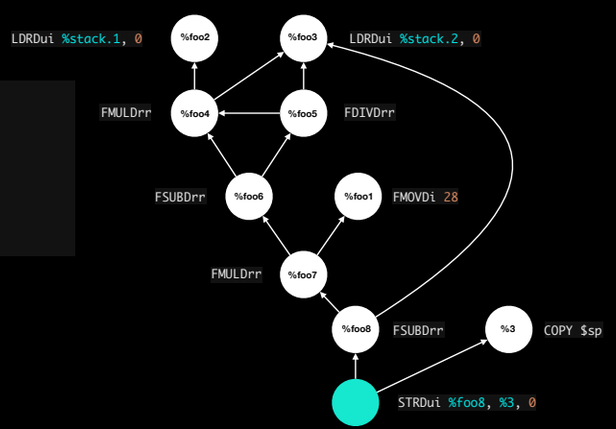
Naming strategy, VReg Number Cursor:

- The cursor rounds up vreg numbers to the same value for similar programs with high confidence.
- The cursor also increments the number used in the vreg name for each call to createVirtualRegister().
- The cursor also rounds up for every walk.

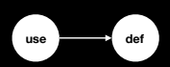
Register Renaming

For each side-effect we do a bottom Up def-use Graph Walk:

```
%3:gpr64sp = COPY $sp
%foo1:fpr64 = FMOVDi 28
%foo2:fpr64 = LDRDui %stack.1, 0
%foo3:fpr64 = LDRDui %stack.2, 0
%foo4:fpr64 = FMULDrr %foo2, %foo3
%foo5:fpr64 = FDIVDrr %foo4, %foo3
%foo6:fpr64 = FSUBDrr %foo4, %foo5
%foo7:fpr64 = FMULDrr %foo6, %foo1
%foo8:fpr64 = FSUBDrr %foo7, %foo3
STRDui %foo8, %3, 0
```



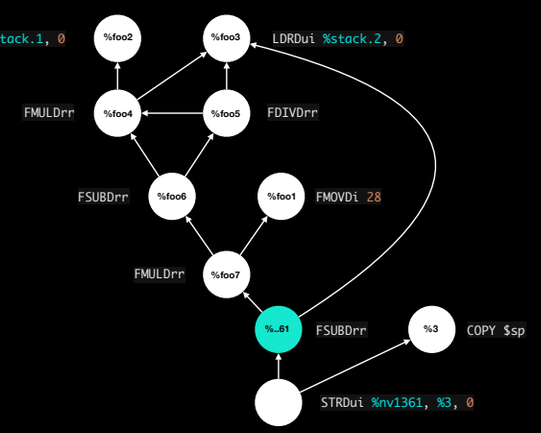
Note:



Register Renaming

For each side-effect we do a bottom Up def-use Graph Walk:

```
%3:gpr64sp = COPY $sp
%foo1:fpr64 = FMOVDi 28
%foo2:fpr64 = LDRDui %stack.1, 0
%foo3:fpr64 = LDRDui %stack.2, 0
%foo4:fpr64 = FMULDrr %foo2, %foo3
%foo5:fpr64 = FDIVDrr %foo4, %foo3
%foo6:fpr64 = FSUBDrr %foo4, %foo5
%foo7:fpr64 = FMULDrr %foo6, %foo1
%nv1361:fpr64 = FSUBDrr %foo7, %foo3
STRDui %nv1361, %3, 0
```



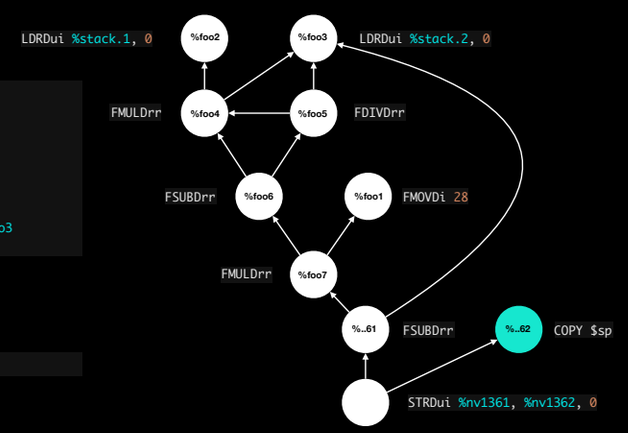
Note:

```
%nv#### is short for %namedVReg####
```

Register Renaming

For each side-effect we do a bottom Up def-use Graph Walk:

```
%nv1362:gpr64sp = COPY $sp
%foo1:fpr64 = FMOVDi 28
%foo2:fpr64 = LDRDui %stack.1, 0
%foo3:fpr64 = LDRDui %stack.2, 0
%foo4:fpr64 = FMULDrr %foo2, %foo3
%foo5:fpr64 = FDIVDrr %foo4, %foo3
%foo6:fpr64 = FSUBDrr %foo4, %foo5
%foo7:fpr64 = FMULDrr %foo6, %foo1
%nv1361:fpr64 = FSUBDrr %foo7, %foo3
STRDui %nv1361, %nv1362, 0
```



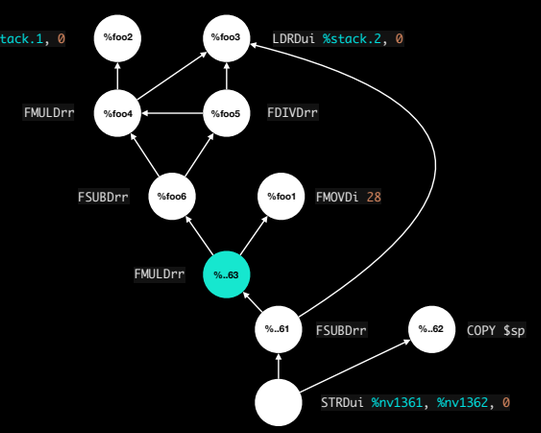
Note:

```
%nv#### is short for %namedVReg####
```

Register Renaming

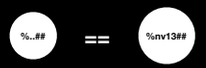
For each side-effect we do a bottom Up def-use Graph Walk:

```
%nv1362:gpr64sp = COPY $sp
%foo1:fpr64 = FMOVDi 28
%foo2:fpr64 = LDRDui %stack.1, 0
%foo3:fpr64 = LDRDui %stack.2, 0
%foo4:fpr64 = FMULDrr %foo2, %foo3
%foo5:fpr64 = FDIVDrr %foo4, %foo3
%foo6:fpr64 = FSUBDrr %foo4, %foo5
%nv1363:fpr64 = FMULDrr %foo6, %foo1
%nv1361:fpr64 = FSUBDrr %nv1363, %foo3
STRDui %nv1361, %nv1362, 0
```



Note:

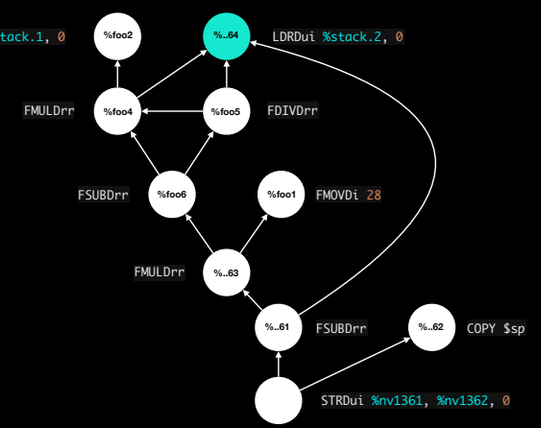
```
%nv#### is short for %namedVReg####
```



Register Renaming

For each side-effect we do a bottom Up def-use Graph Walk:

```
%nv1362:gpr64sp = COPY $sp
%foo1:fpr64 = FMOVDi 28
%foo2:fpr64 = LDRDui %stack.1, 0
%nv1364:fpr64 = LDRDui %stack.2, 0
%foo4:fpr64 = FMULDrr %foo2, %nv1364
%foo5:fpr64 = FDIVDrr %foo4, %nv1364
%foo6:fpr64 = FSUBDrr %foo4, %foo5
%nv1363:fpr64 = FMULDrr %foo6, %foo1
%nv1361:fpr64 = FSUBDrr %nv1363, %nv1364
STRDui %nv1361, %nv1362, 0
```



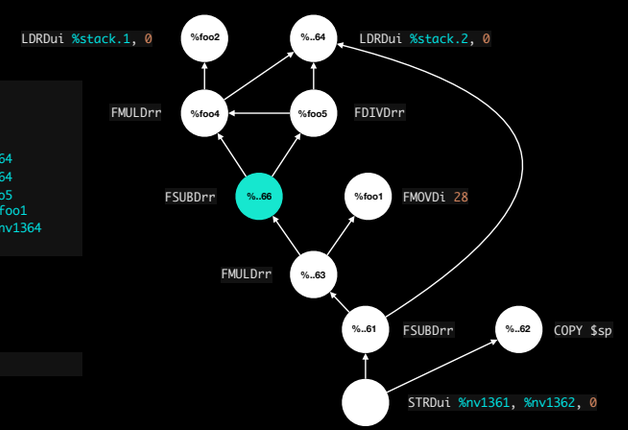
Note:

```
%nv#### is short for %namedVReg####
```

Register Renaming

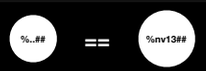
For each side-effect we do a bottom Up def-use Graph Walk:

```
%nv1362:gpr64sp = COPY $sp
%foo1:fpr64 = FMOVDi 28
%foo2:fpr64 = LDRDui %stack.1, 0
%nv1364:fpr64 = LDRDui %stack.2, 0
%foo4:fpr64 = FMULDrr %foo2, %nv1364
%foo5:fpr64 = FDIVDrr %foo4, %nv1364
%nv1366:fpr64 = FSUBDrr %foo4, %foo5
%nv1363:fpr64 = FMULDrr %nv1366, %foo1
%nv1361:fpr64 = FSUBDrr %nv1363, %nv1364
STRDui %nv1361, %nv1362, 0
```



Note:

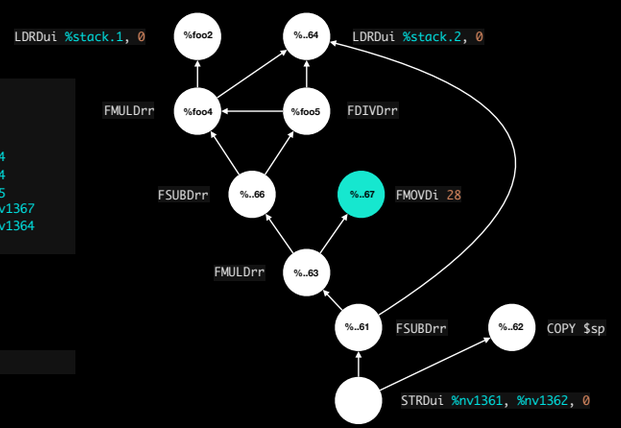
```
%nv#### is short for %namedVReg####
```



Register Renaming

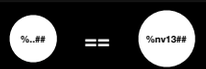
For each side-effect we do a bottom Up def-use Graph Walk:

```
%nv1362:gpr64sp = COPY $sp
%nv1367:fpr64 = FMOVDi 28
%foo2:fpr64 = LDRDui %stack.1, 0
%nv1364:fpr64 = LDRDui %stack.2, 0
%foo4:fpr64 = FMULDrr %foo2, %nv1364
%foo5:fpr64 = FDIVDrr %foo4, %nv1364
%nv1366:fpr64 = FSUBDrr %foo4, %foo5
%nv1363:fpr64 = FMULDrr %nv1366, %nv1367
%nv1361:fpr64 = FSUBDrr %nv1363, %nv1364
STRDui %nv1361, %nv1362, 0
```



Note:

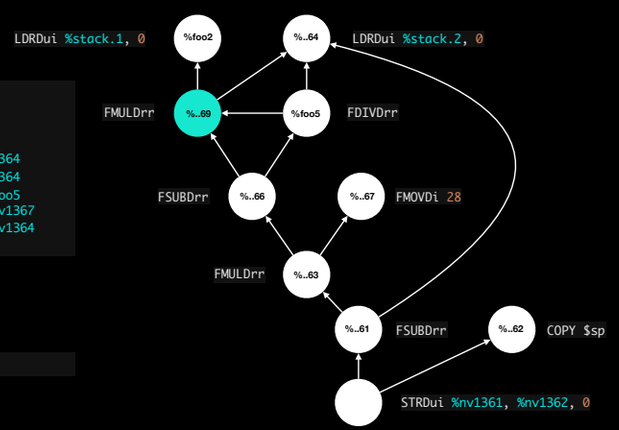
```
%nv#### is short for %namedVReg####
```



Register Renaming

For each side-effect we do a bottom Up def-use Graph Walk:

```
%nv1362:gpr64sp = COPY $sp
%nv1367:fpr64 = FMOVDi 28
%foo2:fpr64 = LDRDui %stack.1, 0
%nv1364:fpr64 = LDRDui %stack.2, 0
%nv1369:fpr64 = FMULDrr %foo2, %nv1364
%foo5:fpr64 = FDIVDrr %nv1369, %nv1364
%nv1366:fpr64 = FSUBDrr %nv1369, %foo5
%nv1363:fpr64 = FMULDrr %nv1366, %nv1367
%nv1361:fpr64 = FSUBDrr %nv1363, %nv1364
STRDui %nv1361, %nv1362, 0
```



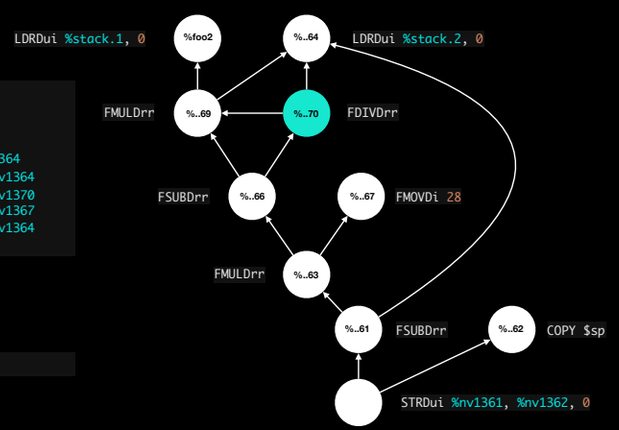
Note:

```
%nv#### is short for %namedVReg####
```

Register Renaming

For each side-effect we do a bottom Up def-use Graph Walk:

```
%nv1362:gpr64sp = COPY $sp
%nv1367:fpr64 = FMOVDi 28
%foo2:fpr64 = LDRDui %stack.1, 0
%nv1364:fpr64 = LDRDui %stack.2, 0
%nv1369:fpr64 = FMULDrr %foo2, %nv1364
%nv1370:fpr64 = FDIVDrr %nv1369, %nv1364
%nv1366:fpr64 = FSUBDrr %nv1369, %nv1370
%nv1363:fpr64 = FMULDrr %nv1366, %nv1367
%nv1361:fpr64 = FSUBDrr %nv1363, %nv1364
STRDui %nv1361, %nv1362, 0
```



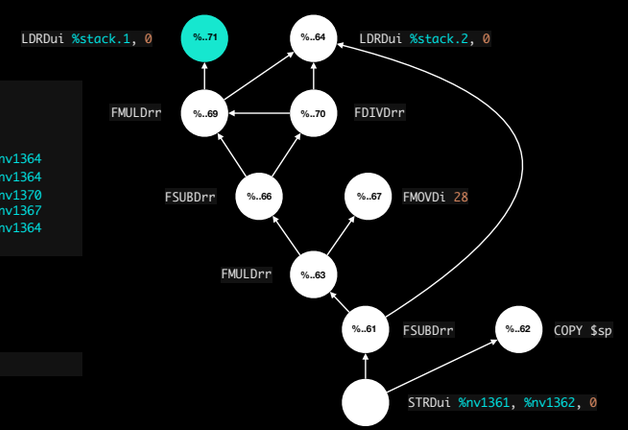
Note:

```
%nv#### is short for %namedVReg####
```

Register Renaming

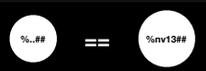
For each side-effect we do a bottom Up def-use Graph Walk:

```
%nv1362:gpr64sp = COPY $sp
%nv1367:fpr64 = FMOVDi 28
%nv1371:fpr64 = LDRDui %stack.1, 0
%nv1364:fpr64 = LDRDui %stack.2, 0
%nv1369:fpr64 = FMULDrr %nv1371, %nv1364
%nv1370:fpr64 = FDIVDrr %nv1369, %nv1364
%nv1366:fpr64 = FSUBDrr %nv1369, %nv1370
%nv1363:fpr64 = FMULDrr %nv1366, %nv1367
%nv1361:fpr64 = FSUBDrr %nv1363, %nv1364
STRDui %nv1361, %nv1362, 0
```



Note:

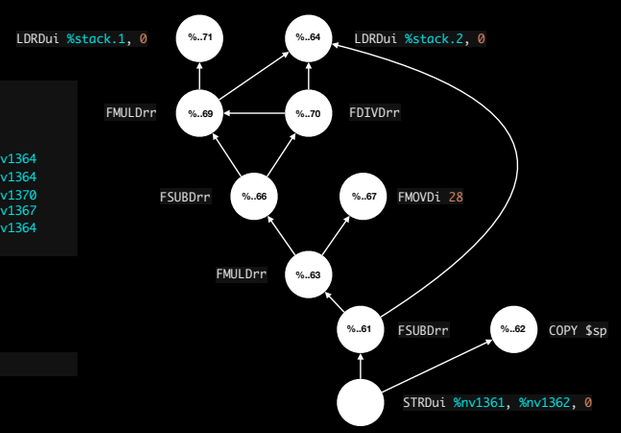
```
%nv#### is short for %namedVReg####
```



Register Renaming

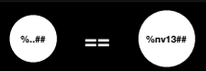
For each side-effect we do a bottom Up def-use Graph Walk:

```
%nv1362:gpr64sp = COPY $sp
%nv1367:fpr64 = FMOVDi 28
%nv1371:fpr64 = LDRDui %stack.1, 0
%nv1364:fpr64 = LDRDui %stack.2, 0
%nv1369:fpr64 = FMULDrr %nv1371, %nv1364
%nv1370:fpr64 = FDIVDrr %nv1369, %nv1364
%nv1366:fpr64 = FSUBDrr %nv1369, %nv1370
%nv1363:fpr64 = FMULDrr %nv1366, %nv1367
%nv1361:fpr64 = FSUBDrr %nv1363, %nv1364
STRDui %nv1361, %nv1362, 0
```



Note:

```
%nv#### is short for %namedVReg####
```



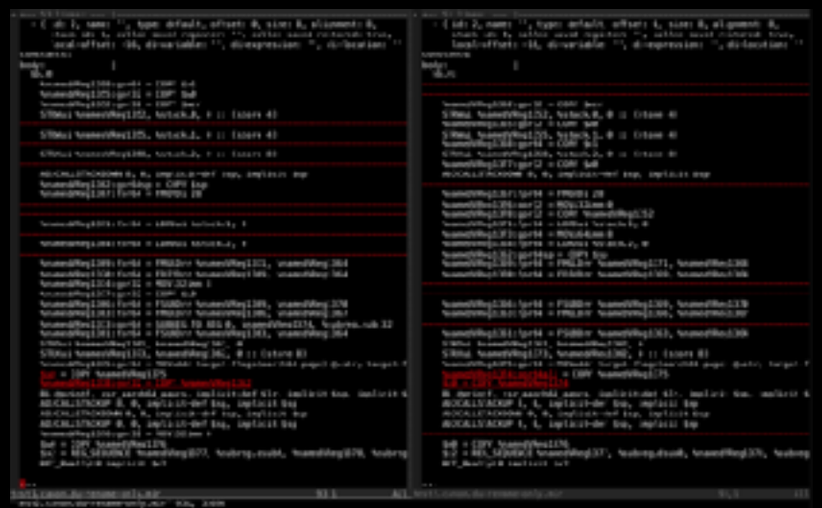
This transformation should be invariant for the same def-use graph topology. So as long as we landed originally on the same initial vreg name to start off, then all of the vregs should be renamed the same way.

Register Renaming

- Works because we process side-effecting instruction def-use graphs in a canonical order:
  - Canonical order for each basic block (RPO).
  - Canonical order for each side-effecting instruction.
- Major assumptions for two similar programs:
  - Side-effects should be roughly the same.
  - CFGs should be roughly the same.

Register Renaming: Results

Register Renaming: Results



Instruction Reordering

- Def-Use Distance Reduction:

  Moves defs of a common user closer to the user.

- Works because there are less differences when defs and uses are closer together versus being interspersed throughout a given basic block.

Def-Use Distance Reduction

- Collects defs for every user's use, and then moves defs as close to the user as possible.

```
%vreg234_0:gpr32 = COPY $w0
%foo6:fpr64 = FSUBDrr %foo4, %foo5
%foo7:fpr64 = FMULDrr %foo6, %foo1
%5:gpr64 = SUBREG_TO_REG 0, killed %4, %subreg.sub_32
%foo8:fpr64 = FSUBDrr %foo7, %foo3
STRDui %foo8, %3, 0
STRXui killed %5, %3, 0 :: (store 8)
%6:gpr64 = MOVaddr target-flags(aarch64-page) @.str,...
$x0 = COPY %6
%vreg645646_1:gpr32 = COPY %2
BL @printf, csr_aarch64_aapcs, implicit-def $lr,...
ADJCALLSTACKUP 8, 0, implicit-def dead $sp, implicit $sp
ADJCALLSTACKDOWN 0, 0, implicit-def dead $sp, implicit $sp
ADJCALLSTACKUP 0, 0, implicit-def dead $sp, implicit $sp
%foo:gpr32 = MOVi32imm 0
$w0 = COPY %foo
$x2 = REG_SEQUENCE %vreg234_0, %subreg.dsub0, %vreg645646_1, %subreg.dsub1
```

Def-Use Distance Reduction

- Collects defs for every user's use, and then moves defs as close to the user as possible.

```
def →  %vreg234_0:gpr32 = COPY $w0
       %foo6:fpr64 = FSUBDrr %foo4, %foo5
       %foo7:fpr64 = FMULDrr %foo6, %foo1
       %5:gpr64 = SUBREG_TO_REG 0, killed %4, %subreg.sub_32
       %foo8:fpr64 = FSUBDrr %foo7, %foo3
       STRDui %foo8, %3, 0
       STRXui killed %5, %3, 0 :: (store 8)
       %6:gpr64 = MOVaddr target-flags(aarch64-page) @.str,...
       $x0 = COPY %6
def →  %vreg645646_1:gpr32 = COPY %2
       BL @printf, csr_aarch64_aapcs, implicit-def $lr,...
       ADJCALLSTACKUP 8, 0, implicit-def dead $sp, implicit $sp
       ADJCALLSTACKDOWN 0, 0, implicit-def dead $sp, implicit $sp
       ADJCALLSTACKUP 0, 0, implicit-def dead $sp, implicit $sp
       %foo:gpr32 = MOVi32imm 0
       $w0 = COPY %foo
user → $x2 = REG_SEQUENCE %vreg234_0, %subreg.dsub0, %vreg645646_1, %subreg.dsub1
```

Def-Use Distance Reduction

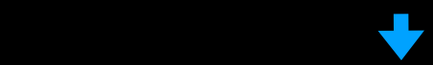
- Collects defs for every user's use, and then moves defs as close to the user as possible.

```
def →  %vreg234_0:gpr32 = COPY $w0
       ...
def →  %vreg645646_1:gpr32 = COPY %2
       ...
user → $x2 = REG_SEQUENCE %vreg234_0, %subreg.dsub0, %vreg645646_1, %subreg.dsub1
```

Def-Use Distance Reduction

- Collects defs for every user's use, and then moves defs as close to the user as possible.

```
def →  %vreg234_0:gpr32 = COPY $w0
       ...
def →  %vreg645646_1:gpr32 = COPY %2
       ...
user → $x2 = REG_SEQUENCE %vreg234_0, %subreg.dsub0, %vreg645646_1, %subreg.dsub1
```



Def-Use Distance Reduction

- Collects defs for every user's use, and then moves defs as close to the user as possible.

```
def →  %vreg234_0:gpr32 = COPY $w0
       ...
def →  %vreg645646_1:gpr32 = COPY %2
       ...
user → $x2 = REG_SEQUENCE %vreg234_0, %subreg.dsub0, %vreg645646_1, %subreg.dsub1
```

```
       ...
def →  %namedVReg1377:gpr32 = COPY %2
def →  %namedVReg1378:gpr32 = COPY $w0
user → $x2 = REG_SEQUENCE %namedVReg1378, %subreg.dsub0, %namedVReg1377, %subreg.dsub1
```

Def-Use Distance Reduction: Results

Before:

After:

Instruction Reordering

- Independent Instruction Hoisting:

  Moves a given instruction that can be placed at any point all in the same place as long a they are prior to the first user.

  The top of the basic block is the most convenient placement.

Independent instructions are instructions that would mean the same thing regardless of where they are places in the basic block.

Independent Instruction Hoisting

- Determines if an instruction's semantics is independent of its placement in the basic block and if so then hoist to the top of the block.

```
→   %1:gpr64 = COPY $x1
→   %0:gpr32 = COPY $w0
→   %2:gpr32 = COPY $wzr

    ...
→   %foo1:fpr64 = FMOVDi 28
    ...
→   %4:gpr32 = MOVi32imm 8
    ...
→   %foo:gpr32 = MOVi32imm 0
    ...
    RET_ReallyLR implicit $x2
```

```
→   %1:gpr64 = COPY $x1
→   %0:gpr32 = COPY $w0
→   %2:gpr32 = COPY $wzr
→   %foo1:fpr64 = FMOVDi 28
→   %4:gpr32 = MOVi32imm 8

    ...
    RET_ReallyLR implicit $x2
```

Independent Instruction Hoisting

- Determines if an instruction's semantics is independent of its placement in the basic block and if so then hoist to the top of the block.

```
→   %1:gpr64 = COPY $x1
→   %0:gpr32 = COPY $w0
→   %2:gpr32 = COPY $wzr

    ...
→   %foo1:fpr64 = FMOVDi 28
    ...
→   %4:gpr32 = MOVi32imm 8
    ...
→   %foo:gpr32 = MOVi32imm 0
    ...
    RET_ReallyLR implicit $x2
```

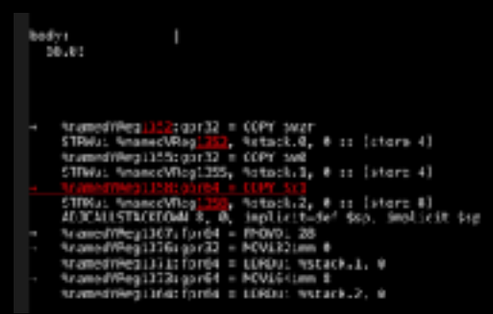
```
→   %namedVReg4352:gpr32 = MOVi32imm 8
→   %namedVReg4353:gpr32 = MOVi32imm 0
→   %namedVReg4354:fpr64 = FMOVDi 28
→   %namedVReg4355:gpr64 = COPY $x1
→   %namedVReg4356:gpr32 = COPY $wzr

    ...
    RET_ReallyLR implicit $x2
```

Independent Instruction Hoisting

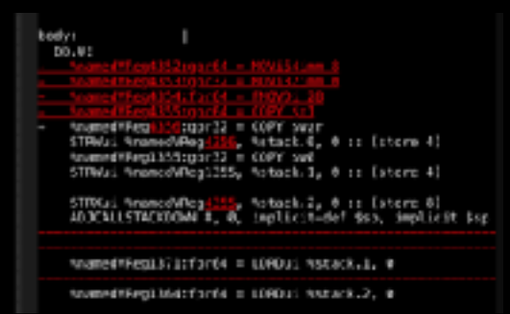
- Independent instruction hoisting improves diff quality by cleaning up a given basic block of an entire class of instructions that could be intersperse throughout the basic block in any order with the same semantics.
- It makes code more canonical to move these to one place and sort them in one way (alphabetically).

Independent Instruction Hoisting: Results

Before:



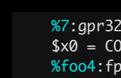
After:



COPY Folding

- If a COPY reads from and writes to a vreg with the same RegClass, fold it.

```
%7:gpr32 = COPY %8
$x0 = COPY %7
%foo4:fpr64 = FMULDrr %foo2, %7
```



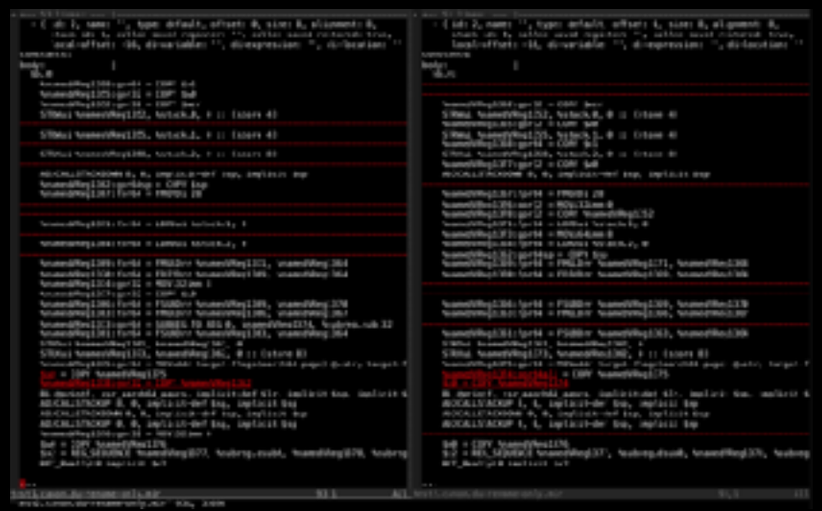
```
$x0 = COPY %8
%foo4:fpr64 = FMULDrr %foo2, %8
```

COPY Folding: Results

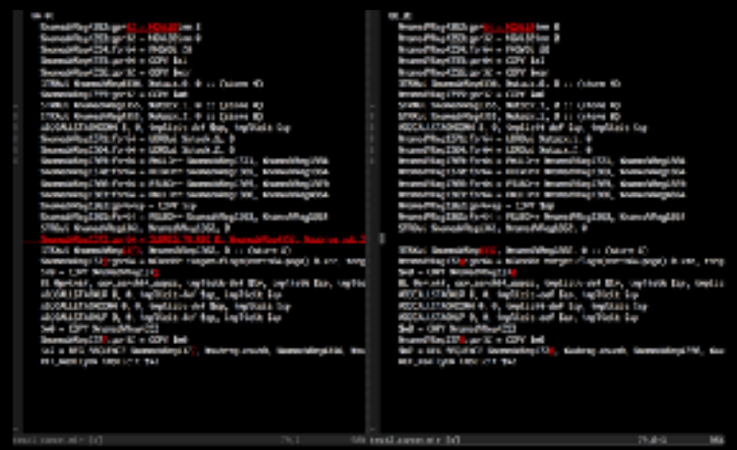
Before:

After:

Instruction Reordering: Results



Instruction Reordering: Results



LLVM Usage and Modifications

- Made use of llvm Machine Function Pass.
- Implemented Named VRegs in MIR: used by cursor.
- Rely heavily on SSA form.

Usage

vimdiff GlobalISel vs SDISel:

```
llc -S -stop-before peephole-opt $file.ll -o - | \
  llc -x mir -run-pass mir-canonicalizer -o canon.sdisel.mir

llc -S -stop-before peephole-opt $file.ll -o - -global-isel -global-isel-abort=0 | \
  llc -x mir -run-pass mir-canonicalizer -o canon.gisel.mir

vimdiff canon.sdisel.mir canon.gisel.mir
```


One way we used mir-canon was to canonicalize the mir files before opening the results in vimdiff.

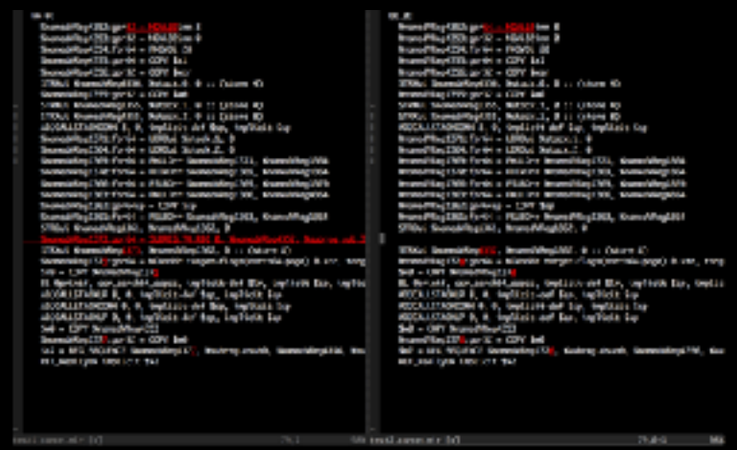
Usage

Checksum diff to classify common diffs:

```
llc -S -stop-before peephole-opt $file.ll -o - | \
  llc -x mir -run-pass mir-canonicalizer -o canon.sdisel.mir

llc -S -stop-before peephole-opt $file.ll -o - -global-isel -global-isel-abort=0 | \
  llc -x mir -run-pass mir-canonicalizer -o canon.gisel.mir

chk=`diff canon.sdisel.mir canon.gisel.mir | scrapediff.sh | md5`

echo "$chk $file.ll" >> mir-diff-logs/$chk.log
```


Another way we used mir-canon was to canonicalize the mir prior to scraping the diff for differing opcodes hashing the scrapped differences inorder to build a frequency list of differences hashes. MIR-Canon makes this possible because by reducing non-semantic differences we can see more of the semantic differences that are shared across different programs in a given test suite.

Usage



Usage

Output of scrapediff.sh for our example:

```
{ MOVi32imm, SUBREG_TO_REG }, { MOVi64imm }
```

Usage

Output of scrapediff.sh for our example:

```
{ MOVi32imm, SUBREG_TO_REG }, { MOVi64imm }
```

```
> echo "{ MOVi32imm, SUBREG_TO_REG }, { MOVi64imm }" | md5
e83501f2db1ef398605300b3713230e9
```

Usage

Checksum diff to classify common diffs:

```
[ 0 jobs - plotfi@grendel:$ ]
~/tmp/mir-diff-logs
> ls | tail
e83501f2db1ef398605300b3713230e9.log     f292d30a1bf7d81def9fe1b2863fac92.log
7efe5fe5429fdc321de717c4b9fe7991.log     f3800a19c8216dcb9e4a3a3a3db7a3aa.log
8313f94ea653e23a000be451da400633.log     f450b592c471d3a75513ea8fb77f66ef.log
8627b10b559bb8375b1ab604f8e19bef.log     f66a0ae738da0e8f75716e312fd33f9c.log
89822ef04a2e454fe44c72adf8717e1a.log     fad89503b53a3c60f6e0a46a6fd137f9.log
8f6c696a93dcb2139bc3a81da9c2b74e.log     ffbf310bc252adbf2800a52c2e6f4904.log

[ 0 jobs - plotfi@grendel:$ ]
~/tmp/mir-diff-logs
> wc -l * | sort | tail
     133 e83501f2db1ef398605300b3713230e9.log
     224 37ac572559c9858920e4bfe394054266.log
     368 34096bc993e8a34856783ebf4119b5b4.log
     412 c8149f477536c364e5adb7427bf8ae40.log
     703 bc4ca5e630850c6a84aa9fe4262d75db.log
     997 84632f4e3577233fb570944454c2a299.log
```

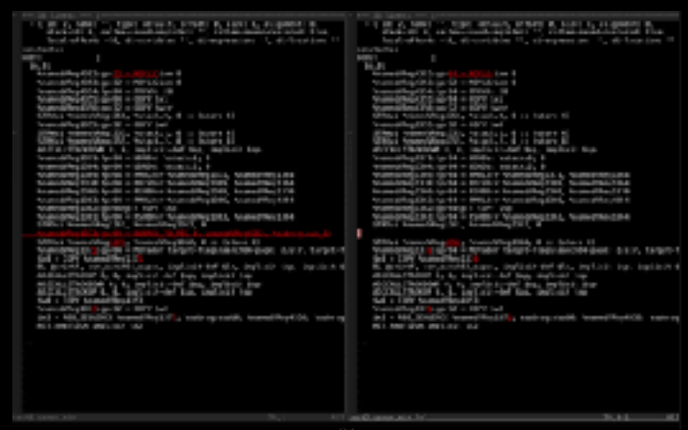
Future Work

Future Work

- Implement Symbolic VReg Renaming.

Future Work

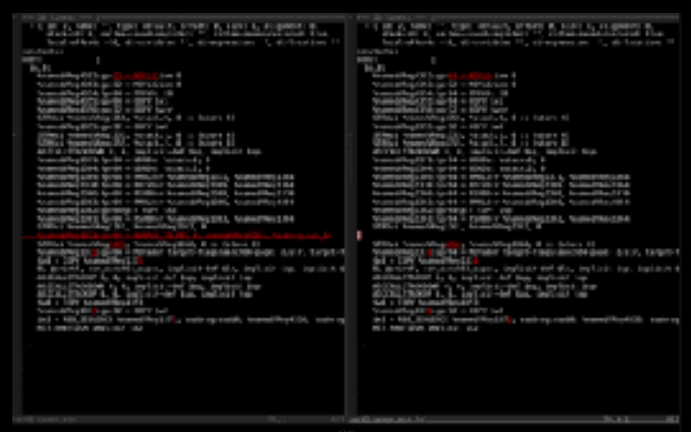
- Implement Symbolic VReg Renaming.



Future Work

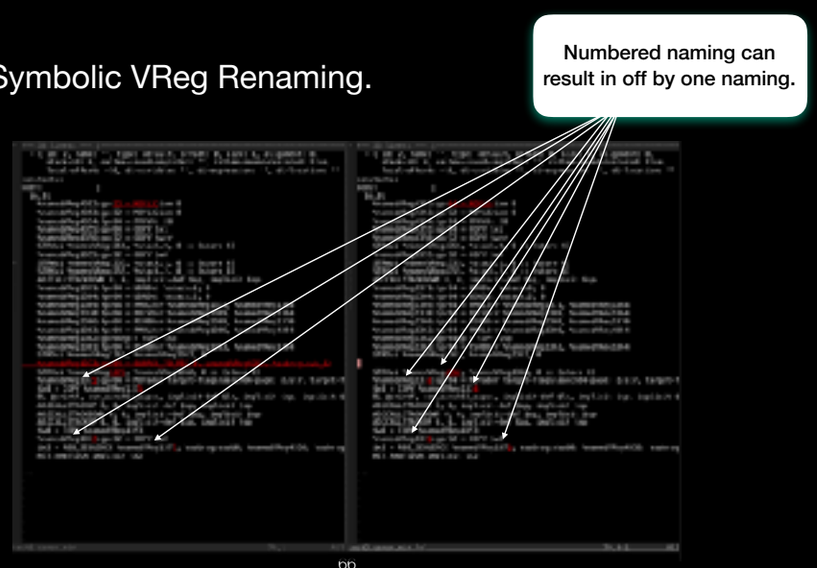
- Implement Symbolic VReg Renaming.

Numbered naming can result in off by one naming.



Future Work

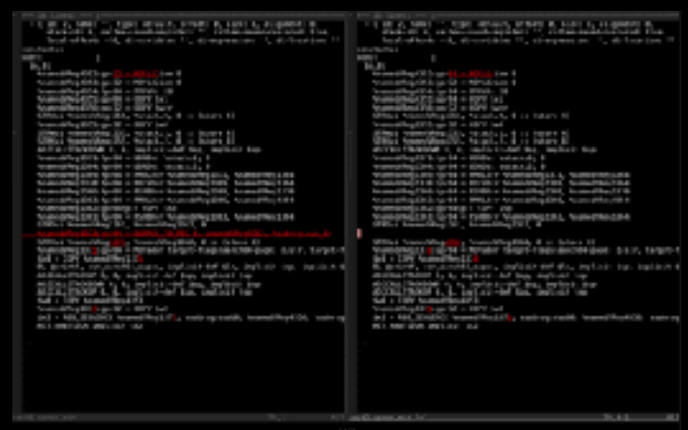
- Implement Symbolic VReg Renaming.



With number naming we cant always perfectly skip vreg numbers and land always on the same place. There are always corner cases where some differences may cascade for example with this subreg to reg.

Future Work

- Implement Symbolic VReg Renaming.



Future Work

- Implement Symbolic VReg Renaming.

This slide shows the mir-canon result of the same identical two programs where the one with the subreg to reg had been changed to use the movi64imm directly. They are now identical with no cascading vreg naming differences either.

Future Work

- Implement Symbolic VReg Renaming.

Future Work

- Implement Symbolic VReg Renaming.
- Explore ISA Specific Canonicalization.
- Explore Global Canonicalization Techniques.
- CodeGen Hardening.

* A good example of ISA Specific canonicalization would be always using the same instruction to clear or zero out a register. On Aarch64 you could do a movimm 0 or a copy from a zero register. Canonically transforming here would be always replacing these with the same instruction.
* Global canonicalization would include things like moving independent instructions to the top of the entry basic block instead of just moving to the top of the current basic block. We could also potentially do our def-use renaming walk across basic block through phi nodes some day.
* CodeGen hardening would be just testing and fixing corner case bugs in asm or binaries generated that had MIRCanonicalizationPass as part of their pipeline. There is some interest in using Canonicalization as a pre-pass for outlining to enhance outlining effectiveness for say -Oz.

MIR-Canon is currently in top of tree LLVM:

- lib/CodeGen/MIRCanonicalizerPass.cpp

Ready to Use:

```
llc -run-pass mir-canonicalizer -o - foo.mir
```

Questions