

MIR-Canon: Improving Code Diff Through Canonical Transformation

Puyan Lotfi
Apple Inc.

1

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.

2

* 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 = FMULDr %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.

4

* 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

<pre> 100 # 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179 180 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198 199 200 201 202 203 204 205 206 207 208 209 210 211 212 213 214 215 216 217 218 219 220 221 222 223 224 225 226 227 228 229 230 231 232 233 234 235 236 237 238 239 240 241 242 243 244 245 246 247 248 249 250 251 252 253 254 255 256 257 258 259 260 261 262 263 264 265 266 267 268 269 270 271 272 273 274 275 276 277 278 279 280 281 282 283 284 285 286 287 288 289 290 291 292 293 294 295 296 297 298 299 300 301 302 303 304 305 306 307 308 309 310 311 312 313 314 315 316 317 318 319 320 321 322 323 324 325 326 327 328 329 330 331 332 333 334 335 336 337 338 339 340 341 342 343 344 345 346 347 348 349 350 351 352 353 354 355 356 357 358 359 360 361 362 363 364 365 366 367 368 369 370 371 372 373 374 375 376 377 378 379 380 381 382 383 384 385 386 387 388 389 390 391 392 393 394 395 396 397 398 399 400 401 402 403 404 405 406 407 408 409 410 411 412 413 414 415 416 417 418 419 420 421 422 423 424 425 426 427 428 429 430 431 432 433 434 435 436 437 438 439 440 441 442 443 444 445 446 447 448 449 450 451 452 453 454 455 456 457 458 459 460 461 462 463 464 465 466 467 468 469 470 471 472 473 474 475 476 477 478 479 480 481 482 483 484 485 486 487 488 489 490 491 492 493 494 495 496 497 498 499 500 501 502 503 504 505 506 507 508 509 510 511 512 513 514 515 516 517 518 519 520 521 522 523 524 525 526 527 528 529 530 531 532 533 534 535 536 537 538 539 540 541 542 543 544 545 546 547 548 549 550 551 552 553 554 555 556 557 558 559 560 561 562 563 564 565 566 567 568 569 570 571 572 573 574 575 576 577 578 579 580 581 582 583 584 585 586 587 588 589 590 591 592 593 594 595 596 597 598 599 600 601 602 603 604 605 606 607 608 609 610 611 612 613 614 615 616 617 618 619 620 621 622 623 624 625 626 627 628 629 630 631 632 633 634 635 636 637 638 639 640 641 642 643 644 645 646 647 648 649 650 651 652 653 654 655 656 657 658 659 660 661 662 663 664 665 666 667 668 669 670 671 672 673 674 675 676 677 678 679 680 681 682 683 684 685 686 687 688 689 690 691 692 693 694 695 696 697 698 699 700 701 702 703 704 705 706 707 708 709 710 711 712 713 714 715 716 717 718 719 720 721 722 723 724 725 726 727 728 729 730 731 732 733 734 735 736 737 738 739 740 741 742 743 744 745 746 747 748 749 750 751 752 753 754 755 756 757 758 759 760 761 762 763 764 765 766 767 768 769 770 771 772 773 774 775 776 777 778 779 780 781 782 783 784 785 786 787 788 789 790 791 792 793 794 795 796 797 798 799 800 801 802 803 804 805 806 807 808 809 810 811 812 813 814 815 816 817 818 819 820 821 822 823 824 825 826 827 828 829 830 831 832 833 834 835 836 837 838 839 840 841 842 843 844 845 846 847 848 849 850 851 852 853 854 855 856 857 858 859 860 861 862 863 864 865 866 867 868 869 870 871 872 873 874 875 876 877 878 879 880 881 882 883 884 885 886 887 888 889 890 891 892 893 894 895 896 897 898 899 900 901 902 903 904 905 906 907 908 909 910 911 912 913 914 915 916 917 918 919 920 921 922 923 924 925 926 927 928 929 930 931 932 933 934 935 936 937 938 939 940 941 942 943 944 945 946 947 948 949 950 951 952 953 954 955 956 957 958 959 960 961 962 963 964 965 966 967 968 969 970 971 972 973 974 975 976 977 978 979 980 981 982 983 984 985 986 987 988 989 990 991 992 993 994 995 996 997 998 999 1000 1001 1002 1003 1004 1005 1006 1007 1008 1009 1010 1011 1012 1013 1014 1015 1016 1017 1018 1019 1020 1021 1022 1023 1024 1025 1026 1027 1028 1029 1030 1031 1032 1033 1034 1035 1036 1037 1038 1039 1040 1041 1042 1043 1044 1045 1046 1047 1048 1049 1050 1051 1052 1053 1054 1055 1056 1057 1058 1059 1060 1061 1062 1063 1064 1065 1066 1067 1068 1069 1070 1071 1072 1073 1074 1075 1076 1077 1078 1079 1080 1081 1082 1083 1084 1085 1086 1087 1088 </pre>

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 invoked 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 GlobalSel vs DAGSel verification.

8

- * So what were our 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 that's been selected with GlobalSel 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?

9

- * 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 3, 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.oapcs, 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
%l: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 3, 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 = FMULDrn %foo2, %foo3
%foo5:fpr64 = FDIVDrn %foo4, %foo3
%4:gpr32 = MOVi32imm 8
%vreg234_0:gpr32 = COPY $w0
%foo6:fpr64 = FSUBDrn %foo4, %foo5
%foo7:fpr64 = FMULDrn %foo6, %foo1
%5:gpr64 = SUBREG_TO_REG 0, killed %4, %subreg.sub_32
%foo8:fpr64 = FSUBDrn %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.oapcs, 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 = FMVDUI 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 = MOV132imm 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,...
$w0 = 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 = MOV132imm 0
$w0 = COPY %foo
$w2 = REG_SEQUENCE %vreg234_0, %subreg.dsub0,
%vreg645646_1, %subreg.dsub1
RET_ReallyLR implicit $x2
```



```
bb.0:
%namedVReg4352:gpr32 = MOV132imm 8
%namedVReg4353:gpr32 = MOV132imm 0
%namedVReg4354:fpr64 = FMVDUI 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
%namedVReg1365: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, ...
$w0 = 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
$w2 = 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.

13

- * 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:

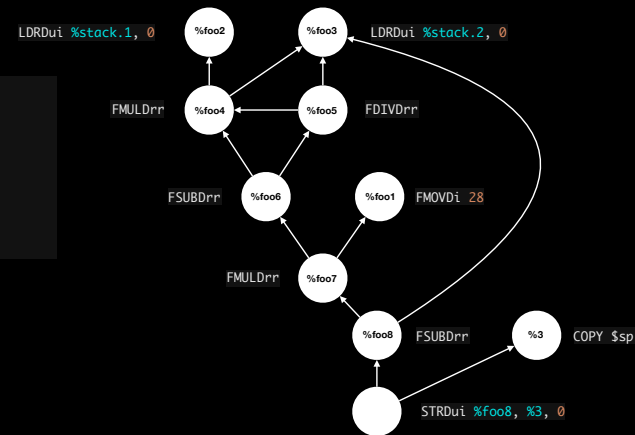
```
→ $x0 = COPY %6
→ STRXui killed %5, %3, 0 :: (store 8)
→ STRDui %foo8, %3, 0
→ STRXui %1, %stack.2, 0 :: (store 8)
→ STRWui %0, %stack.1, 0 :: (store 4)
→ STRWui %2, %stack.0, 0 :: (store 4)
```

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:

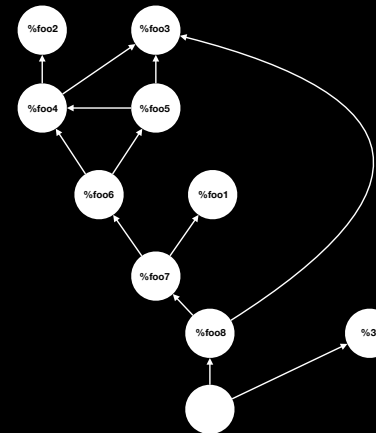


Register Renaming

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);  
};
```



23

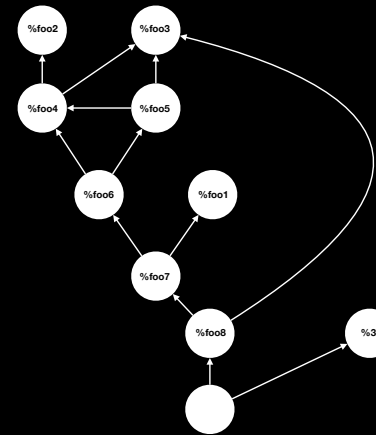
- * The mechanism we used to effectively keep state of what we are renaming the vregs to is this vreg number cursor.
- * Initially it calls `createIncompleteVirtualRegister` on `MRI` to get a sense of how many vregs are allocated in the MIR so that we can figure out how much to initially round up an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

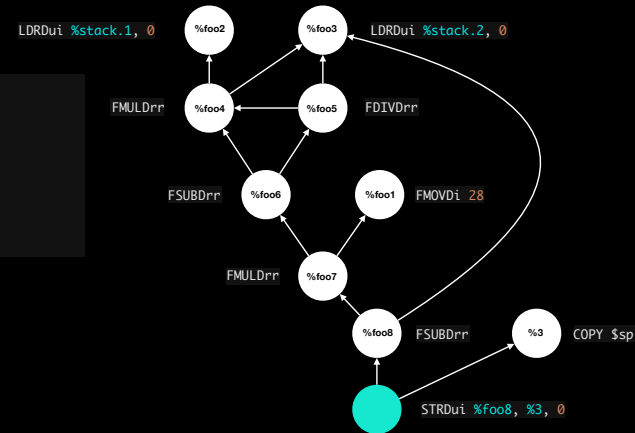


Register Renaming

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:



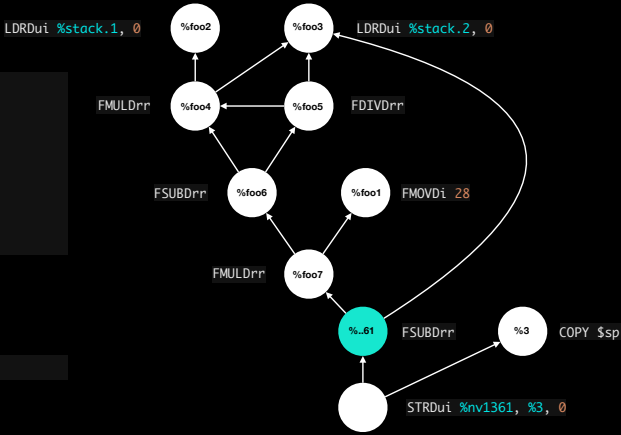
Register Renaming

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

```
%3:gpr64sp = COPY $sp
%foo1:fpr64 = FMOVEDi 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###



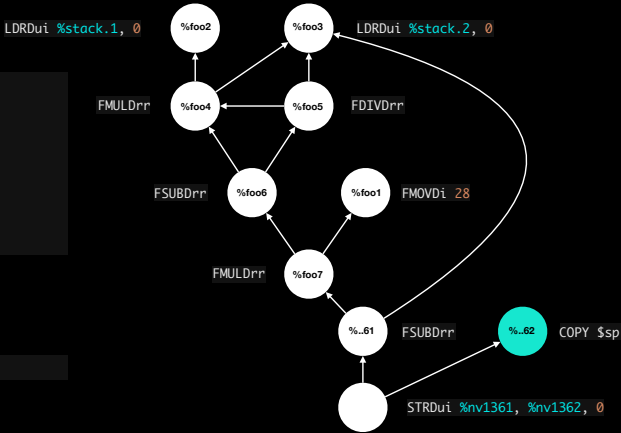
Register Renaming

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###



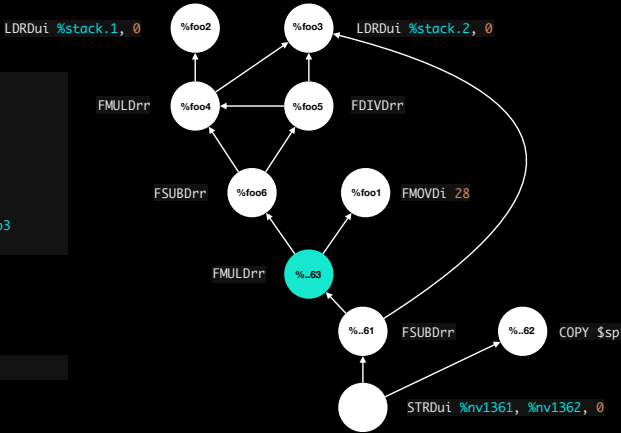
Register Renaming

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###



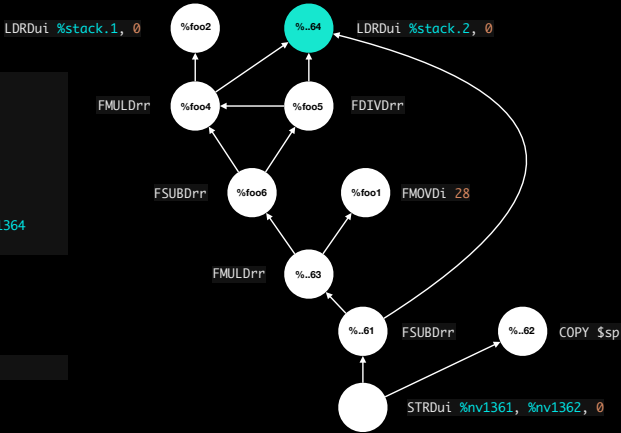
Register Renaming

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####



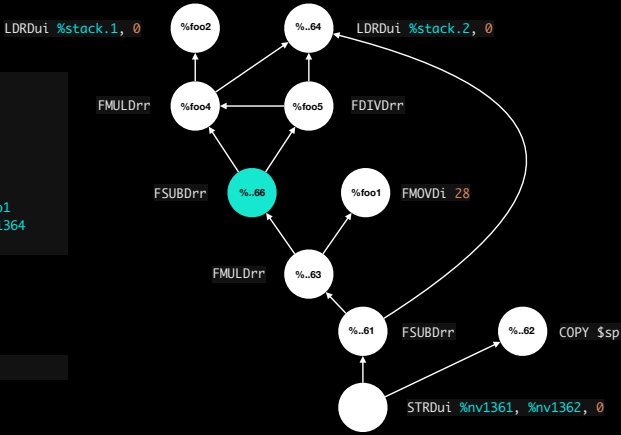
Register Renaming

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###



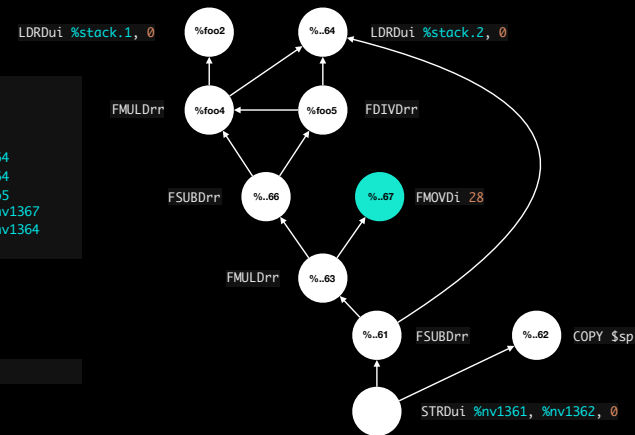
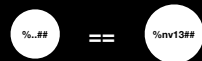
Register Renaming

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

```
%nv1362:qpr64sp = COPY $sp
%nvn1367:fpr64 = FMOVDI 28
%foo2:fpr64 = LDRDUI %stack.1, 0
%nvn1364:fpr64 = LDRDUI %stack.2, 0
%foo4:fpr64 = FMULDrr %foo2, %nv1364
%foo5:fpr64 = FDIVDrr %foo4, %nv1364
%nvn1366:fpr64 = FSUBDrr %foo4, %foo5
%nvn1363:fpr64 = FMULDrr %nv1366, %nv1367
%nvn1361:fpr64 = FSUBDrr %nv1363, %nv1364
STRDUI %nv1361, %nv1362, 0
```

Note:

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



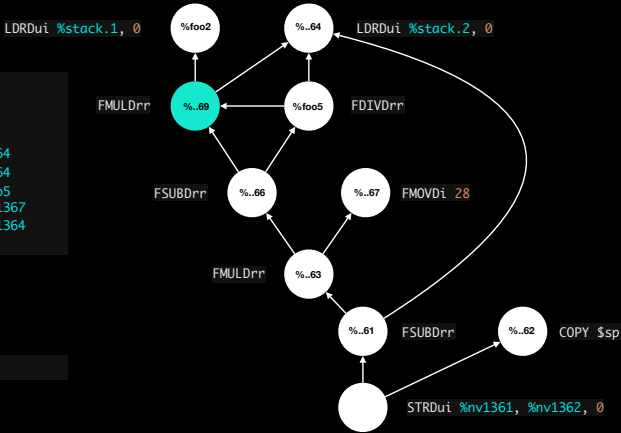
Register Renaming

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

```
%nv1362:gpr64sp = COPY $sp
%n1367:fpr64 = FMOVDi 28
%foo2:fpr64 = LDRDui %stack.1, 0
%n1364:fpr64 = LDRDui %stack.2, 0
%n1369:fpr64 = FMULDrr %foo2, %nv1364
%foo5:fpr64 = FDIVDrr %nv1369, %nv1364
%n1366:fpr64 = FSUBDrr %nv1369, %foo5
%n1363:fpr64 = FMULDrr %nv1366, %nv1367
%n1361:fpr64 = FSUBDrr %nv1363, %nv1364
STRDui %nv1361, %nv1362, 0
```

Note:

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



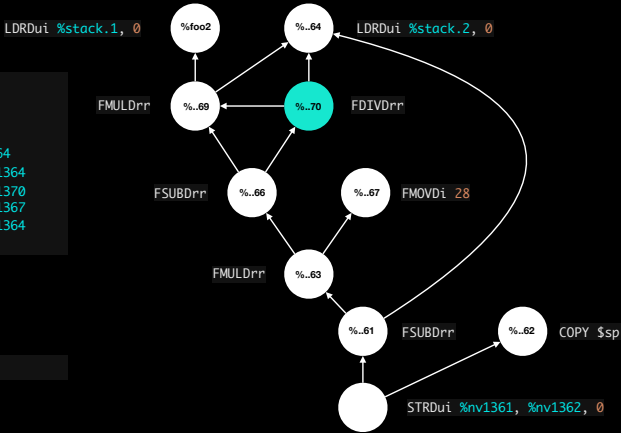
Register Renaming

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

```
%nv1362:gpr64sp = COPY $sp
%n1367:fpr64 = FMOVDi 28
%foo2:fpr64 = LDRDui %stack.1, 0
%n1364:fpr64 = LDRDui %stack.2, 0
%n1369:fpr64 = FMULDrn %foo2, %nv1364
%n1370:fpr64 = FDIVDrn %nv1369, %nv1364
%n1366:fpr64 = FSUBDrn %nv1369, %nv1370
%n1363:fpr64 = FMULDrn %nv1366, %nv1367
%n1361:fpr64 = FSUBDrn %nv1363, %nv1364
STRDui %nv1361, %nv1362, 0
```

Note:

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



Register Renaming

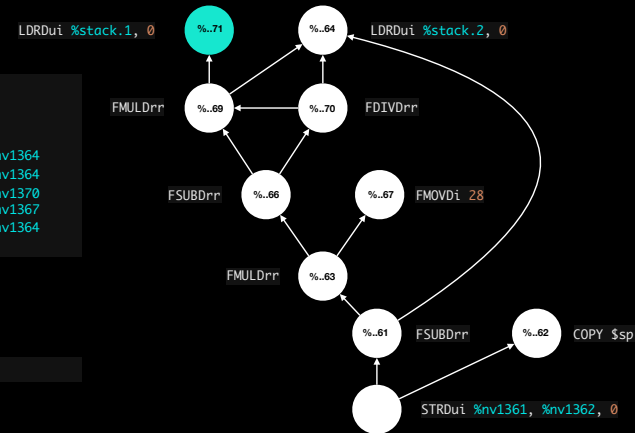
For each side-effect we do a bottom Up def-use Graph Walk:

```
%nv1362:gpr64sp = COPY $sp
%n1367:fpr64 = FMOVDi 28
%n1371:fpr64 = LDRDui %stack.1, 0
%n1364:fpr64 = LDRDui %stack.2, 0
%n1369:fpr64 = FMULDrn %nv1371, %nv1364
%n1370:fpr64 = FDIVDrn %nv1369, %nv1364
%n1366:fpr64 = FSUBDrn %nv1369, %nv1370
%n1363:fpr64 = FMULDrn %nv1366, %nv1367
%n1361:fpr64 = FSUBDrn %nv1363, %nv1364
STRDui %nv1361, %nv1362, 0
```

Note:

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

%..## == %nv13##



Register Renaming

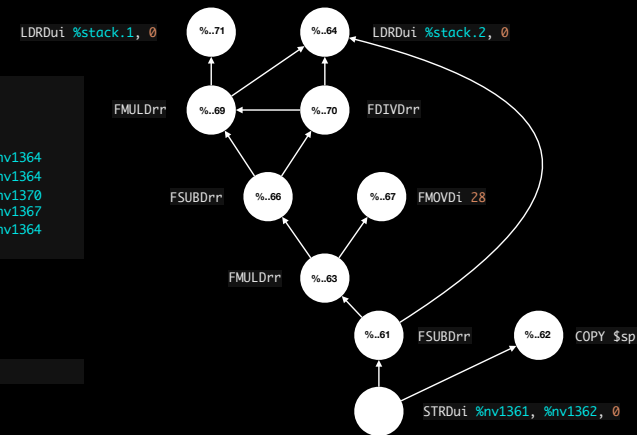
For each side-effect we do a bottom Up def-use Graph Walk:

```
%nv1362:gpr64sp = COPY $sp
%n1367:fpr64 = FMOVDi 28
%n1371:fpr64 = LDRDui %stack.1, 0
%n1364:fpr64 = LDRDui %stack.2, 0
%n1369:fpr64 = FMULDr %nv1371, %nv1364
%n1370:fpr64 = FDIVDr %nv1369, %nv1364
%n1366:fpr64 = FSUBDr %nv1369, %nv1370
%n1363:fpr64 = FMULDr %nv1366, %nv1367
%n1361:fpr64 = FSUBDr %nv1363, %nv1364
STRDui %nv1361, %nv1362, 0
```

Note:

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

%.## == %nv13##



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

[illegible]

Register Renaming: Results

[illegible]

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 %x2
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:

```

...
STORED %namedReg1156, %stack.2, 0 : (STORE R)
%namedReg1156:copr32 = COPY %w6
ABSTRACT_STORED %R, 0, implicit-def %sp, implicit %sp
%namedReg1156:for64 = PROVEI 20
%namedReg1156:copr32 = MOV32Imm 0
%namedReg1156:copr32 = COPY %namedReg1156
%namedReg1156:for64 = LDRDUI %stack.1, 0
%namedReg1156:copr64 = MOV64Imm 0
...

$u2 = REG_SEQUENCE %namedReg1177, %subreg.sub0,
%namedReg1177, %subreg.sub1
RET_ReallyLR implicit %u2

```

After:

```

...
STORED %namedReg1156, %stack.2, 0 : (STORE R)
ABSTRACT_STORED %R, 0, implicit-def %sp, implicit %sp
%namedReg1156:for64 = MOV64Imm 20
%namedReg1156:copr32 = MOV32Imm 0
%namedReg1177:for64 = LDRDUI %stack.1, 0
%namedReg1177:copr64 = MOV64Imm 0
...
%namedReg1177:copr32 = COPY %w6
%namedReg1177:copr32 = COPY %namedReg1156
$u2 = REG_SEQUENCE %namedReg1177, %subreg.sub0,
%namedReg1177, %subreg.sub1
RET_ReallyLR implicit %u2

```

Instruction Reordering

- Independent Instruction Hoisting:

Moves a given instruction that can be placed at any point all in the same place as long as 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 placed 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:

```
body:
  DO, #1

  → NamedReg132:qpr32 = COPY %wzr
  STP.W, %namedReg132, %stack.0, 0 to [store 4]
  → NamedReg135:qpr32 = COPY %wzr
  STP.W, %namedReg135, %stack.1, 0 to [store 4]
  → NamedReg138:qpr64 = COPY %x1
  STP.W, %namedReg138, %stack.2, 0 to [store 8]
  ADDX.W, %stack.0, 8, 0, implicit: def %sps, implicit: %sp
  → NamedReg139:qpr64 = PROVD, 20
  → NamedReg136:qpr32 = MOVLSImm, 0
  → NamedReg131:qpr64 = LDRD.W, %stack.1, 0
  → NamedReg133:qpr64 = MOVLSImm, 8
  → NamedReg134:qpr64 = LDRD.W, %stack.2, 0
```

After:

```
body:
  DO, #1
  → NamedReg132:qpr64 = MOVLSImm, 0
  → NamedReg135:qpr32 = MOVLSImm, 0
  → NamedReg134:qpr64 = PROVD, 20
  → NamedReg133:qpr64 = COPY %x1
  → NamedReg130:qpr32 = COPY %wzr
  STP.W, %namedReg130, %stack.0, 0 to [store 4]
  → NamedReg135:qpr32 = COPY %wzr
  STP.W, %namedReg135, %stack.1, 0 to [store 4]
  STP.W, %namedReg133, %stack.2, 0 to [store 8]
  ADDX.W, %stack.0, 8, 0, implicit: def %sps, implicit: %sp
  → NamedReg131:qpr64 = LDRD.W, %stack.1, 0
  → NamedReg134:qpr64 = LDRD.W, %stack.2, 0
```

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 = FMULDr %foo2, %7
```



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

COPY Folding: Results

Before:

```
%namedVReg1375rgr64 = H0Vader target_flags/00r054 pag  
= %namedVReg1375rgr64 = COPY %namedVReg1375  
$x0 = COPY %namedVReg1375
```

After:

```
%namedVReg1375rgr64 = H0Vader target_flags/00r054 pag  
$x0 = COPY %namedVReg1375
```

Instruction Reordering: Results

[illegible]

Instruction Reordering: Results

```

38:
39:
40:
41:
42:
43:
44:
45:
46:
47:
48:
49:
50:
51:
52:
53:
54:
55:
56:
57:
58:
59:
60:
61:
62:
63:
64:
65:
66:
67:
68:
69:
70:
71:
72:
73:
74:
75:
76:
77:
78:
79:
80:
81:
82:
83:
84:
85:
86:
87:
88:
89:
90:
91:
92:
93:
94:
95:
96:
97:
98:
99:
100:
101:
102:
103:
104:
105:
106:
107:
108:
109:
110:
111:
112:
113:
114:
115:
116:
117:
118:
119:
120:
121:
122:
123:
124:
125:
126:
127:
128:
129:
130:
131:
132:
133:
134:
135:
136:
137:
138:
139:
140:
141:
142:
143:
144:
145:
146:
147:
148:
149:
150:
151:
152:
153:
154:
155:
156:
157:
158:
159:
160:
161:
162:
163:
164:
165:
166:
167:
168:
169:
170:
171:
172:
173:
174:
175:
176:
177:
178:
179:
180:
181:
182:
183:
184:
185:
186:
187:
188:
189:
190:
191:
192:
193:
194:
195:
196:
197:
198:
199:
200:
201:
202:
203:
204:
205:
206:
207:
208:
209:
210:
211:
212:
213:
214:
215:
216:
217:
218:
219:
220:
221:
222:
223:
224:
225:
226:
227:
228:
229:
230:
231:
232:
233:
234:
235:
236:
237:
238:
239:
240:
241:
242:
243:
244:
245:
246:
247:
248:
249:
250:
251:
252:
253:
254:
255:
256:
257:
258:
259:
260:
261:
262:
263:
264:
265:
266:
267:
268:
269:
270:
271:
272:
273:
274:
275:
276:
277:
278:
279:
280:
281:
282:
283:
284:
285:
286:
287:
288:
289:
290:
291:
292:
293:
294:
295:
296:
297:
298:
299:
300:
301:
302:
303:
304:
305:
306:
307:
308:
309:
310:
311:
312:
313:
314:
315:
316:
317:
318:
319:
320:
321:
322:
323:
324:
325:
326:
327:
328:
329:
330:
331:
332:
333:
334:
335:
336:
337:
338:
339:
340:
341:
342:
343:
344:
345:
346:
347:
348:
349:
350:
351:
352:
353:
354:
355:
356:
357:
358:
359:
360:
361:
362:
363:
364:
365:
366:
367:
368:
369:
370:
371:
372:
373:
374:
375:
376:
377:
378:
379:
380:
381:
382:
383:
384:
385:
386:
387:
388:
389:
390:
391:
392:
393:
394:
395:
396:
397:
398:
399:
400:
401:
402:
403:
404:
405:
406:
407:
408:
409:
410:
411:
412:
413:
414:
415:
416:
417:
418:
419:
420:
421:
422:
423:
424:
425:
426:
427:
428:
429:
430:
431:
432:
433:
434:
435:
436:
437:
438:
439:
440:
441:
442:
443:
444:
445:
446:
447:
448:
449:
450:
451:
452:
453:
454:
455:
456:
457:
458:
459:
460:
461:
462:
463:
464:
465:
466:
467:
468:
469:
470:
471:
472:
473:
474:
475:
476:
477:
478:
479:
480:
481:
482:
483:
484:
485:
486:
487:
488:
489:
490:
491:
492:
493:
494:
495:
496:
497:
498:
499:
500:
501:
502:
503:
504:
505:
506:
507:
508:
509:
510:
511:
512:
513:
514:
515:
516:
517:
518:
519:
520:
521:
522:
523:
524:
525:
526:
527:
528:
529:
530:
531:
532:
533:
534:
535:
536:
537:
538:
539:
540:
541:
542:
543:
544:
545:
546:
547:
548:
549:
550:
551:
552:
553:
554:
555:
556:
557:
558:
559:
560:
561:
562:
563:
564:
565:
566:
567:
568:
569:
570:
571:
572:
573:
574:
575:
576:
577:
578:
579:
580:
581:
582:
583:
584:
585:
586:
587:
588:
589:
590:
591:
592:
593:
594:
595:
596:
597:
598:
599:
600:
601:
602:
603:
604:
605:
606:
607:
608:
609:
610:
611:
612:
613:
614:
615:
616:
617:
618:
619:
620:
621:
622:
623:
624:
625:
626:
627:
628:
629:
630:
631:
632:
633:
634:
635:
636:
637:
638:
639:
640:
641:
642:
643:
644:
645:
646:
647:
648:
649:
650:
651:
652:
653:
654:
655:
656:
657:
658:
659:
660:
661:
662:
663:
664:
665:
666:
667:
668:
669:
670:
671:
672:
673:
674:
675:
676:
677:
678:
679:
680:
681:
682:
683:
684:
685:
686:
687:
688:
689:
690:
691:
692:
693:
694:
695:
696:
697:
698:
699:
700:
701:
702:
703:
704:
705:
706:
707:
708:
709:
710:
711:
712:
713:
714:
715:
716:
717:
718:
719:
720:
721:
722:
723:
724:
725:
726:
727:
728:
729:
730:
731:
732:
733:
734:
735:
736:
737:
738:
739:
740:
741:
742:
743:
744:
745:
746:
747:
748:
749:
750:
751:
752:
753:
754:
755:
756:
757:
758:
759:
760:
761:
762:
763:
764:
765:
766:
767:
768:
769:
770:
771:
772:
773:
774:
775:
776:
777:
778:
779:
780:
781:
782:
783:
784:
785:
786:
787:
788:
789:
790:
791:
792:
793:
794:
795:
796:
797:
798:
799:
800:
801:
802:
803:
804:
805:
806:
807:
808:
809:
810:
811:
812:
813:
814:
815:
816:
817:
818:
819:
820:
821:
822:
823:
824:
825:
826:
827:
828:
829:
830:
831:
832:
833:
834:
835:
836:
837:
838:
839:
840:
841:
842:
843:
844:
845:
846:
847:
848:
849:
850:
851:
852:
853:
854:
855:
856:
857:
858:
859:
860:
861:
862:
863:
864:
865:
866:
867:
868:

```

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 GlobalSel 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

[illegible]

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      f3800a19c8216dcb9e4a3a3adb7a3aa.log
8313f94ea653e23a000be451da400633.log      f450b592c471d3a75513ea8fb77f66ef.log
8627b10b559bb8375b1ab604f8e19bef.log      f66a0ae738da0e8f75716e312fd33f9c.log
89822ef04a2e454fe44c72adf8717e1a.log      fad89503b53a3c60f6e0a46a6fd137f9.log
8f6c696a93dcb2139bc3a81da9c2b74e.log      ffbf310bc252adb2f2800a52c2e6f4904.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
```

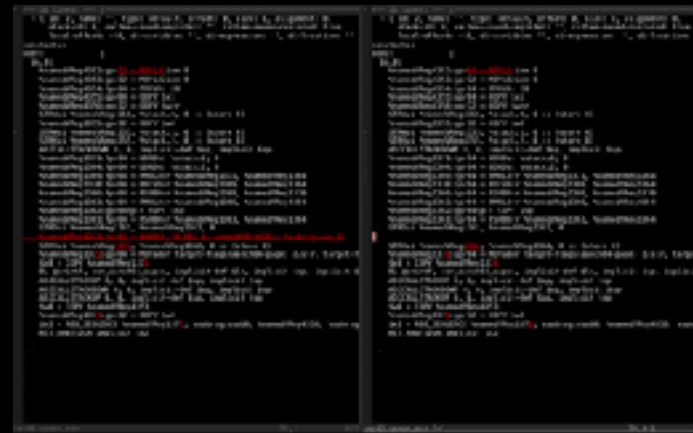
Future Work

Future Work

- Implement Symbolic VReg Renaming.

Future Work

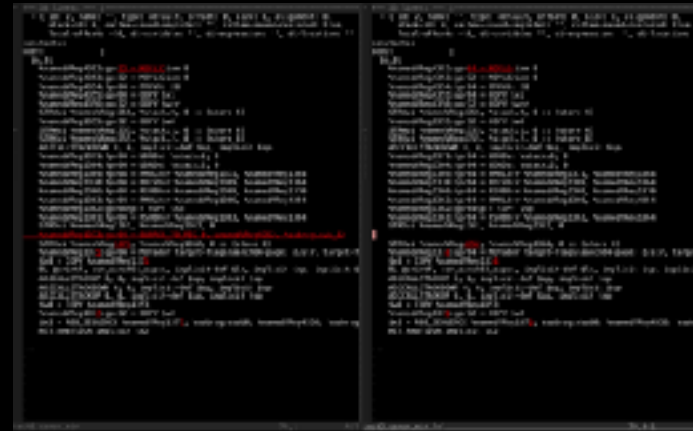
- Implement Symbolic VReg Renaming.



Future Work

- Implement Symbolic VReg Renaming.

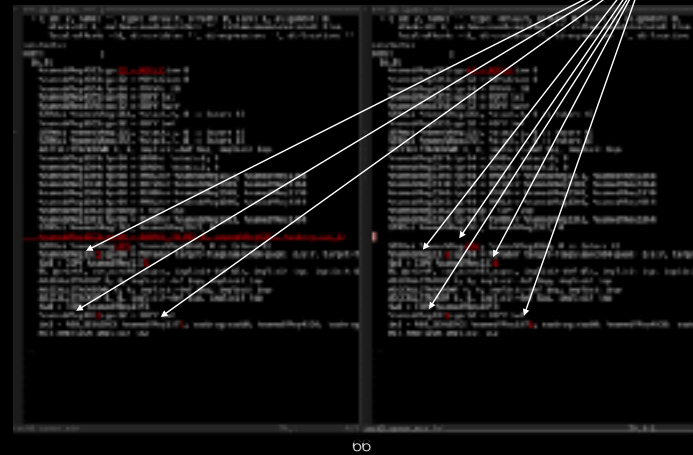
Numbered naming can result in off by one naming.



Future Work

- Implement Symbolic VReg Renaming.

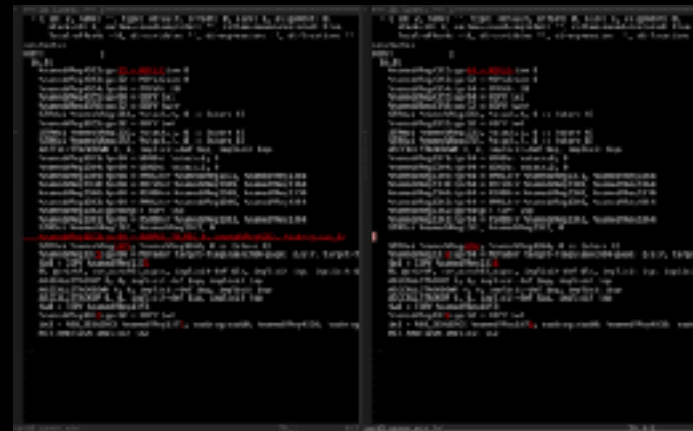
Numbered naming can result in off by one naming.



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

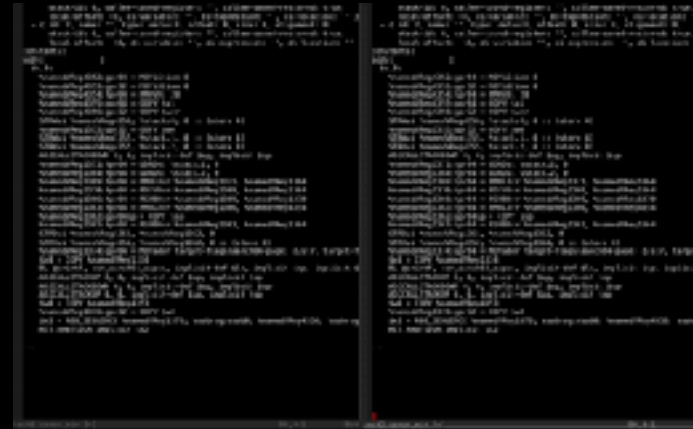
- Implement Symbolic VReg Renaming.



b/

Future Work

- Implement Symbolic VReg Renaming.



68

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.

70

- * 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