**Generic Operand inst instruction selection patterns:**

Case A)

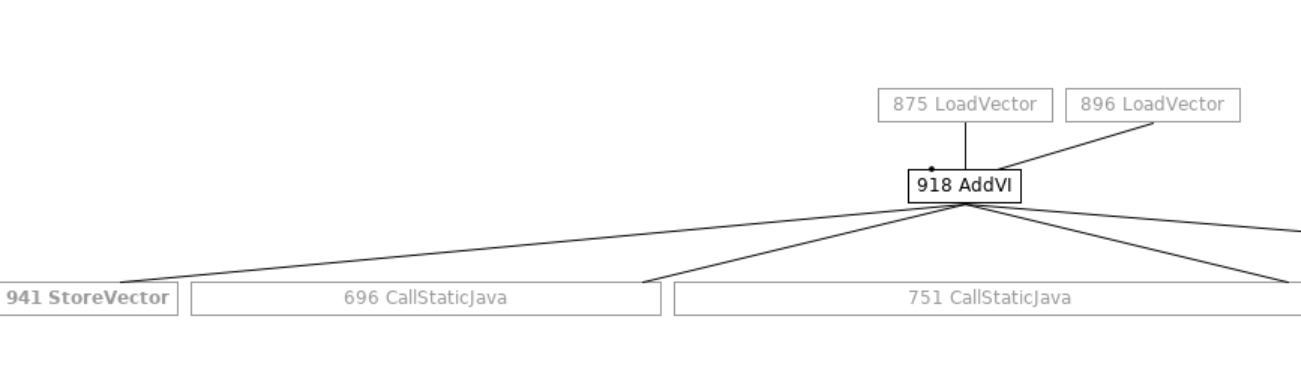
Single Addition operation:

IntVector IV1 = IntVector.fromArray(IntVector.SPECIES\_256, a, 0);

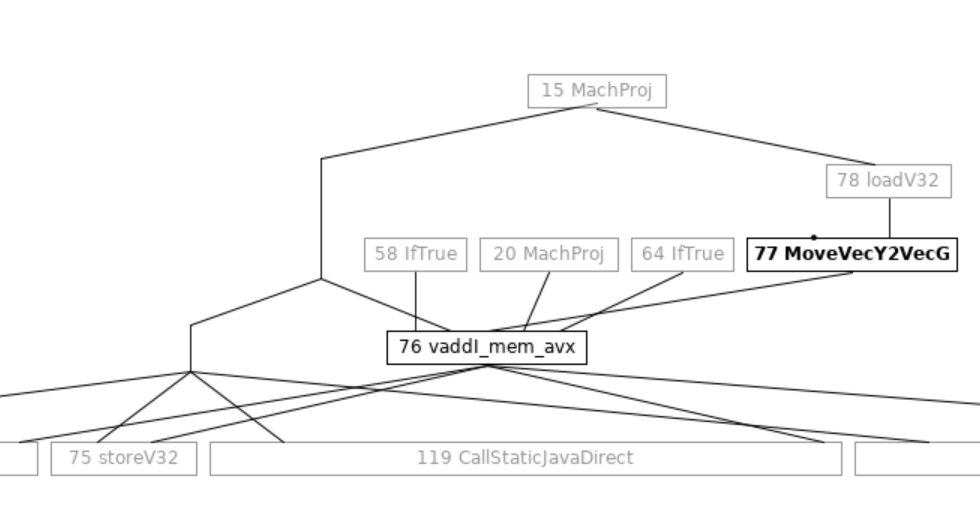
IntVector IV2 = IntVector.fromArray(IntVector.SPECIES\_256, b, 0);

IV1.add(IV2).intoArray(r,0);

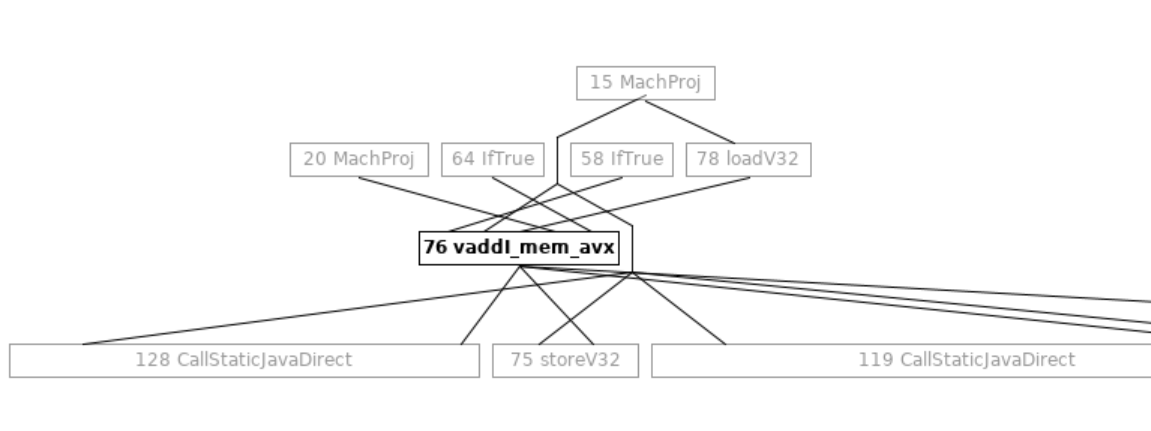
Ideal Graph:



Post Selection Graph:



Post remove Generic Register moves:



Case B)

Conditional Addition:

Source:

IntVector IV1 = IntVector.fromArray(IntVector.SPECIES\_256, a, 0);

IntVector IV2 = IntVector.fromArray(IntVector.SPECIES\_256, b, 0);

IntVector IV3 = IntVector.fromArray(IntVector.SPECIES\_256, a, 8);

IntVector IV4 = IntVector.fromArray(IntVector.SPECIES\_256, b, 8);

IntVector T1;

if (cond)

T1 = IV1.add(IV2);

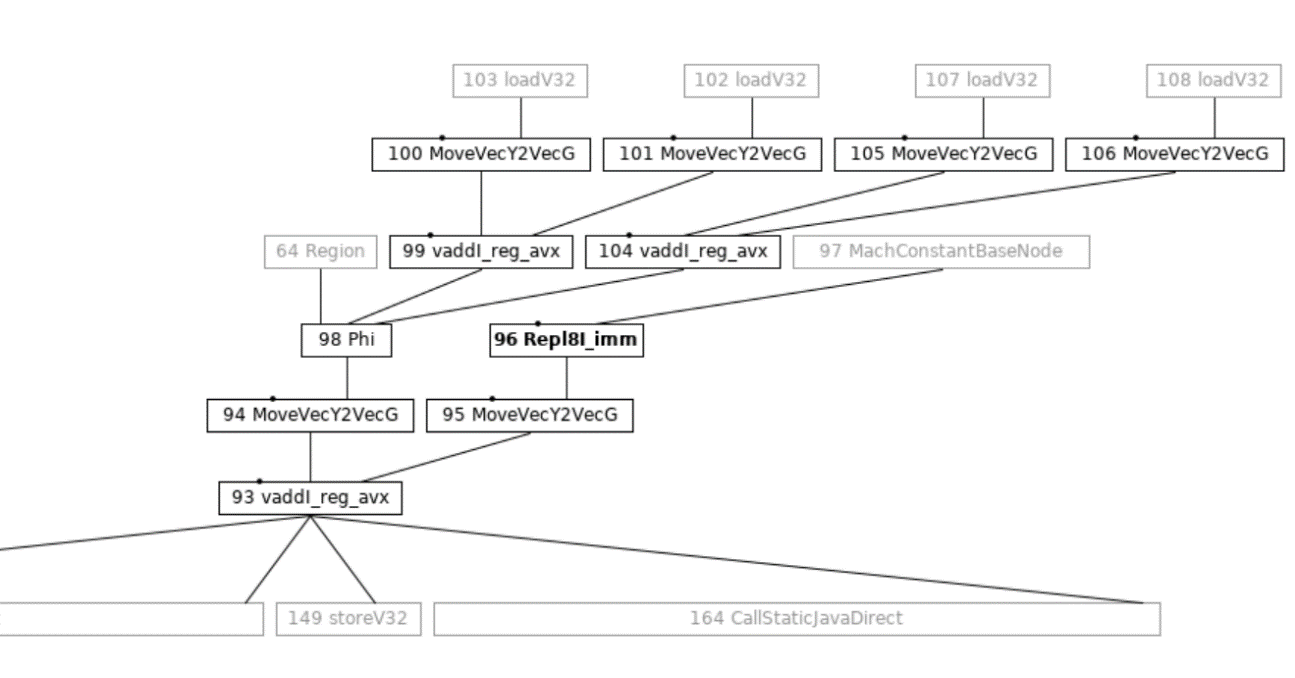
else

T1 = IV3.add(IV4);

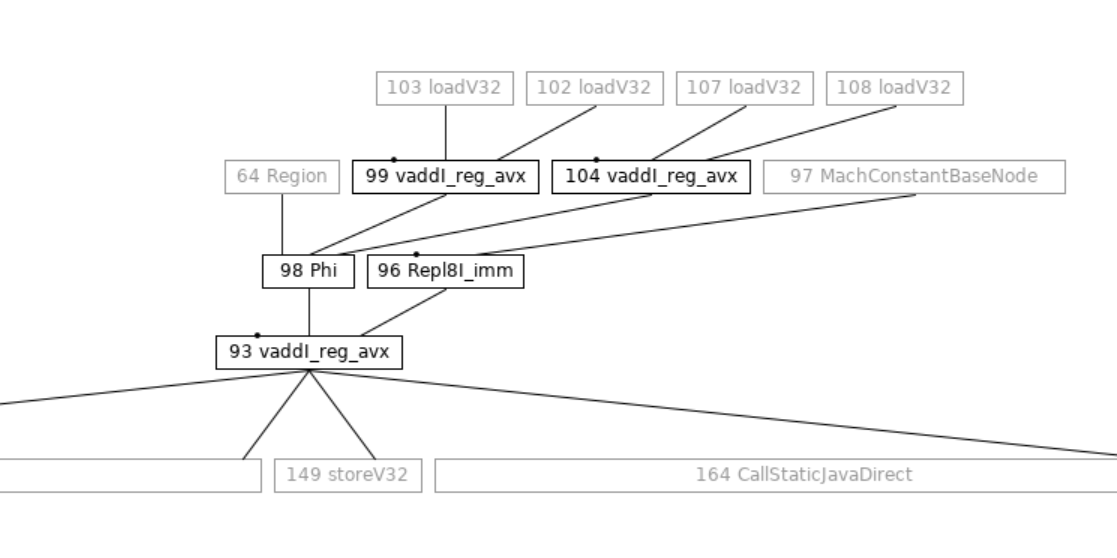
// Phi for T1

T1.add(10).intoArray(r,0);

Post Selection Graph:



Post remove Generic Register Moves X-form:



Case C)

Source:

IntVector IV1 = IntVector.fromArray(IntVector.SPECIES\_256, a, 0);

IntVector IV2 = IntVector.fromArray(IntVector.SPECIES\_256, b, 0);

IntVector IV3 = IntVector.fromArray(IntVector.SPECIES\_256, a, 4);

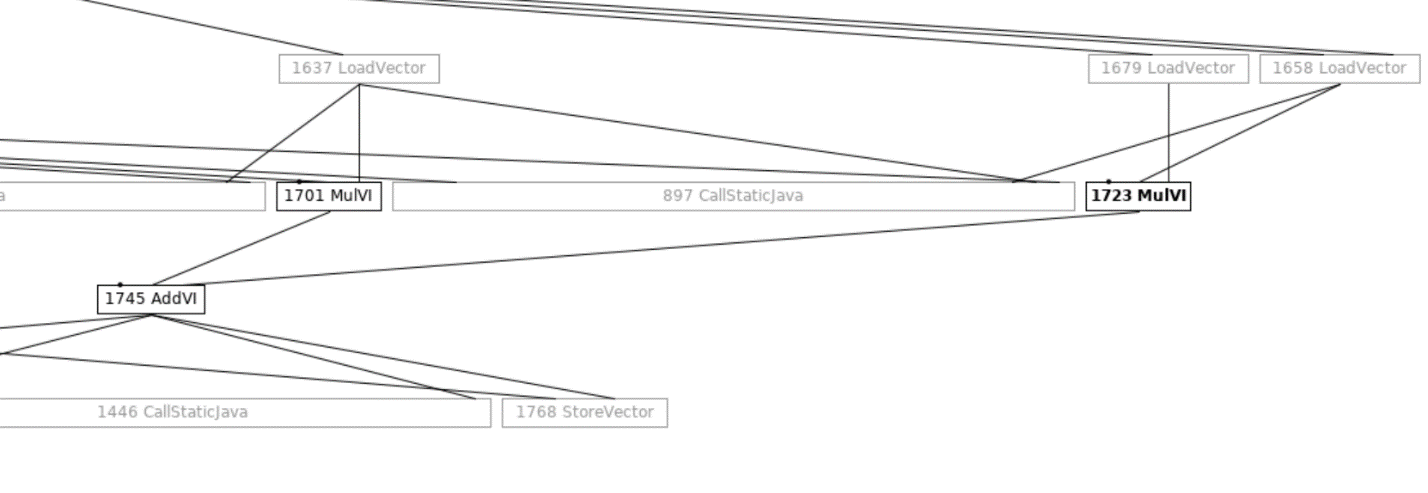
IntVector IV4 = IntVector.fromArray(IntVector.SPECIES\_256, b, 4);

IntVector T1 = IV1.mul(IV2);

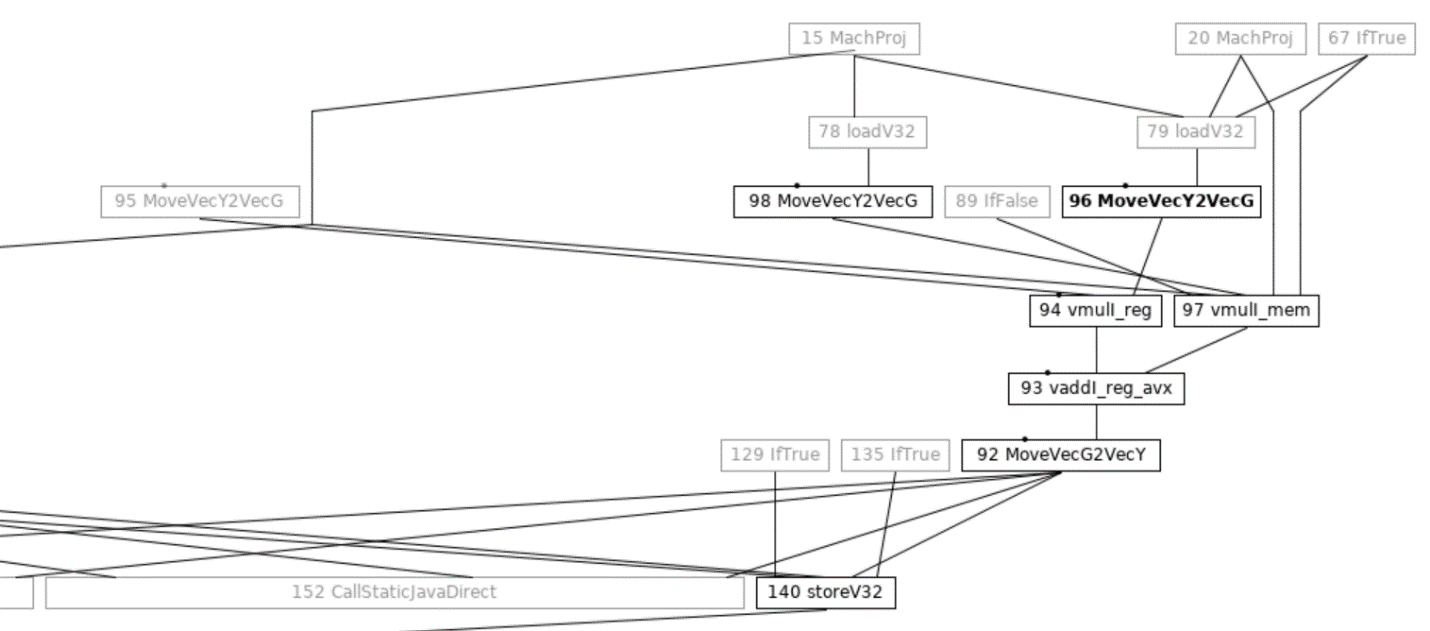
IntVector T2 = IV3.mul(IV4);

T1.add(T2).intoArray(r,0);

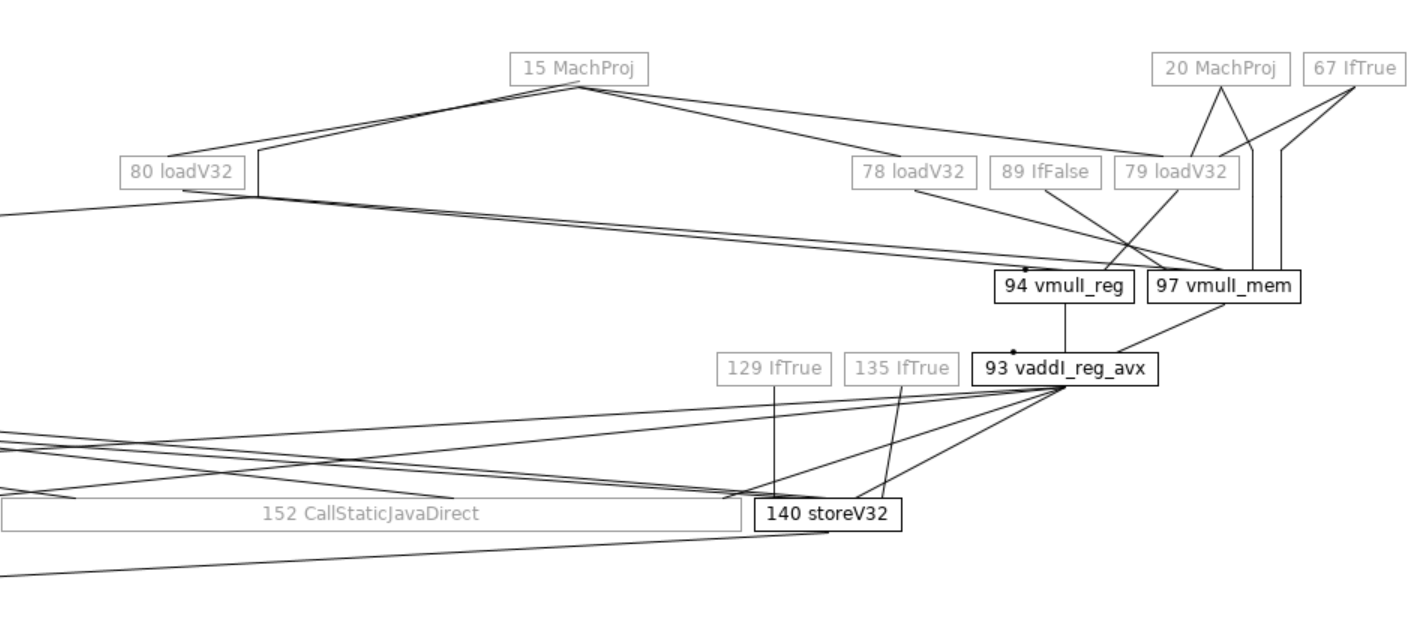
Ideal Graph:



Post Selection Graph:



Post remove Generic Register Moves X-form:



**Mixed feature instructions over higher targets:-**

UseAVX > 2 for addLanes (Short128):

Post selection graph for UseAVX > 2:

@ bci:12 Short128VectorTests::addLanesShort128VectorTestsMasked @ bci:176

289 loadN === 480 268 690 [[ 287 ]] narrowoop: bool[int:>=0]:exact \* !jvms: Short128Vector::addLanes @ bci:15 Short128Vector::addLanes @ bci:12 Short128VectorTests::addLanesShort128VectorTestsMasked @ bci:176

255 Phi === 250 601 253 [[ 253 306 316 326 336 346 356 377 ]] #int:8..max-56:www !jvms: Short128VectorTests::addLanesShort128VectorTestsMasked @ bci:159

692 Phi === 250 687 746 [[ 377 368 356 346 336 326 316 306 696 ]] #short[int:>=0]:exact \* !jvms: Short128Vector::addLanes @ bci:15 Short128Vector::addLanes @ bci:12 Short128VectorTests::addLanesShort128VectorTestsMasked @ bci:176

268 Phi === 145 269 266 [[ 267 266 280 287 289 296 306 316 326 336 346 356 368 377 ]] #memory Memory: @BotPTR \*+bot, idx=Bot; !jvms: Short128VectorTests::addLanesShort128VectorTestsMasked @ bci:146

499 Region === 499 142 [[ 499 140 280 281 296 306 316 326 336 346 356 368 377 389 ]] !jvms: Short128Vector::addLanes @ bci:15 Short128Vector::addLanes @ bci:12 Short128VectorTests::addLanesShort128VectorTestsMasked @ bci:176

288 MachTemp === 1 [[ 286 ]] !jvms: Short128Vector::addLanes @ bci:15 Short128Vector::addLanes @ bci:12 Short128VectorTests::addLanesShort128VectorTestsMasked @ bci:176

287 loadV8 === \_ 268 289 [[ 286 ]] vectord[8]:{bool} !jvms: Short128Vector::addLanes @ bci:15 Short128Vector::addLanes @ bci:12 Short128VectorTests::addLanesShort128VectorTestsMasked @ bci:176

316 loadV16 === 499 268 692 255 [[ 315 ]] vectorx[8]:{short} !jvms: Short128Vector::addLanes @ bci:15 Short128Vector::addLanes @ bci:12 Short128VectorTests::addLanesShort128VectorTestsMasked @ bci:176

286 loadmask8s === \_ 287 288 [[ 278 294 304 314 324 334 344 354 366 375 ]] vectorx[8]:{short} !jvms: Short128Vector::addLanes @ bci:15 Short128Vector::addLanes @ bci:12 Short128VectorTests::addLanesShort128VectorTestsMasked @ bci:176

315 MoveVecX2Leg === \_ 316 [[ 314 ]] !jvms: Short128Vector::addLanes @ bci:15 Short128Vector::addLanes @ bci:12 Short128VectorTests::addLanesShort128VectorTestsMasked @ bci:176

224 Repl8S\_zero === 1 [[ 218 232 278 294 304 314 324 334 344 354 366 375 ]] vectorx[8]:{short} !jvms: ShortVector::broadcast @ bci:23 Short128Vector::addLanes @ bci:4 Short128VectorTests::addLanesShort128VectorTestsMasked @ bci:109

314 vpblendvb8S === \_ 224 315 286 [[ 313 ]] vectorx[8]:{short} !jvms: Short128Vector::addLanes @ bci:15 Short128Vector::addLanes @ bci:12 Short128VectorTests::addLanesShort128VectorTestsMasked @ bci:176

# To suppress the following error report, specify this argument

# after -XX: or in .hotspotrc: SuppressErrorAt=/assembler\_x86.cpp:8855

#

# A fatal error has been detected by the Java Runtime Environment:

#

# Internal Error (/home/jatinbha/sandboxes/dev/src/hotspot/cpu/x86/assembler\_x86.cpp:8855), pid=172308, tid=172342

# assert(((dst\_enc < 16 && nds\_enc < 16 && src\_enc < 16) || (!attributes->is\_legacy\_mode()))) failed: XMM register should be 0-15

Post Selection Graph with UseAVX = 1, 2 (vecX === legVecX)



**Problem:**

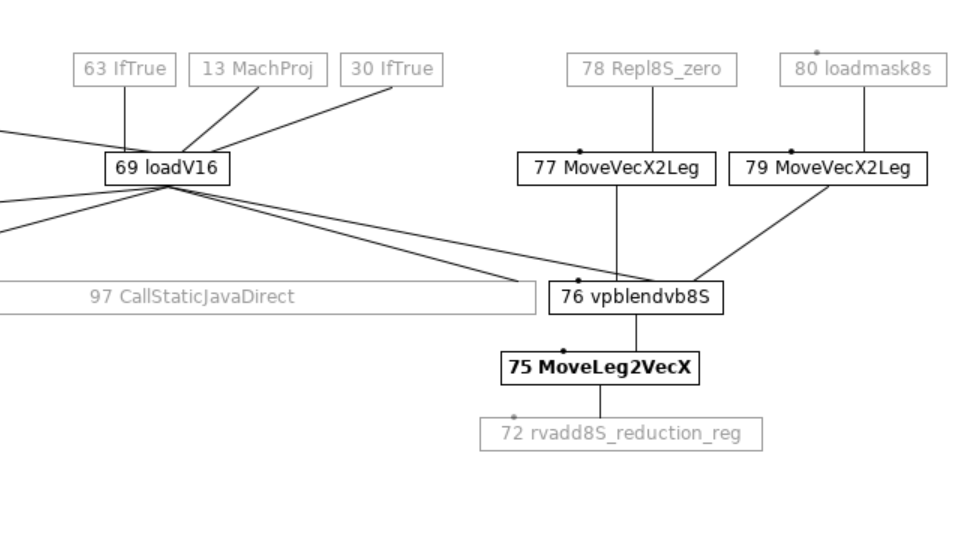
1. Mixed-feature instruction selection (AVX2-AVX512) over higher feature target (AVX512) may cause problem if definition operand is allocated from higher register bank and use operand expects register to be allocated from lower register-bank.
2. Constraining RA to allocate a definition operand from lower register bank over higher feature is done by associating it with legacy register classes (legVecX, legVecY, legVecZ).
3. Above constraint is added because some of the lower feature instruction are not present on higher feature targets and their corresponding patterns are missing for higher feature targets from AD file.
4. Selector should be able to insert appropriate RCC (register class convertors i.e. MoveVec\*ToLeg\* / MoveLeg\*ToVec\*). Instruction selection performs dynamic programming based selection where it picks the instruction with minimum cost.
5. Selector has two phases:-
   1. Labeling phase (bottom-up): Creates a State tree where each state node encapsulates valid transitions from given state to next state, operand type are used to select the valid move.
   2. Reduction phase (top-down): MachNode creation is done based on the minimum cost rule.
6. Current cost model looks fragile since most of the instruction have a default instr\_cost attribute.

Solution:

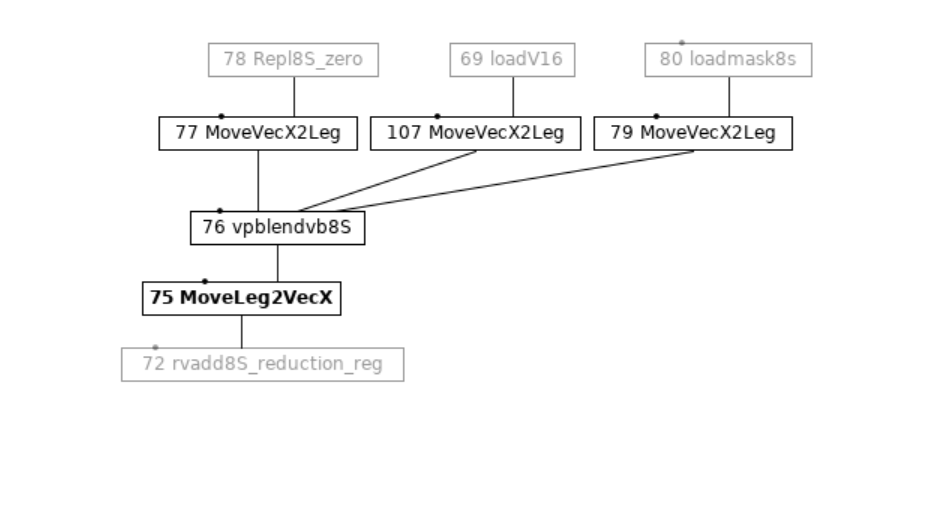
1. Additional post allocation **pass to fixup the def-use pairs where def’s register mask strictly overlaps the use’s register mask**. One stop solution for all similar leakages in code.

**Mixed-feature instruction handling over higher target:-**

Post-selection graph:



Post FixupMixedFeatureOperands stage:



**Register** **Allocation**

1. RA first gathers the live ranges for each definition, while doing this it also limits the allocation set for each operand.
2. It also limits the allocation set for inputs by back-propagating the register mask from the user instruction i.e. from use to its def operands.

**Node[] : 76 vpblendvb8S === \_ 77 107 79 [[ 75 ]] vectorx[8]:{short} !jvms: Short128Vector::addLanes @ bci:15 Short128Vector::addLanes @ bci:12 addlanes::workload @ bci:18**

**4 [XMM0-XMM0d,XMM1-XMM1d,XMM2-XMM2d,XMM3-XMM3d,XMM4-XMM4d,XMM5-XMM5d,XMM6-XMM6d,XMM7-XMM7d,XMM8-XMM8d,XMM9-XMM9d,XMM10-XMM10d,XMM11-XMM11d,XMM12-XMM12d,XMM13-XMM13d,XMM14-XMM14d,XMM15-XMM15d], #?(64) EffDeg: ? Def: N76 Cost: 0 Area: 0 Score:1e+35 Vector**

**77 MoveVecX2Leg === \_ 78 [[ 76 ]] !jvms: Short128Vector::addLanes @ bci:15 Short128Vector::addLanes @ bci:12 addlanes::workload @ bci:18**

**INITIAL INPUT MASK:**

**0 [R10-rS703], #!!!\_704\_vs\_704 EffDeg: ? Dead Cost: 0 Area: 0 Score:1e+35**

**FINAL INPUT MASK:**

**0 [XMM0-XMM0d,XMM1-XMM1d,XMM2-XMM2d,XMM3-XMM3d,XMM4-XMM4d,XMM5-XMM5d,XMM6-XMM6d,XMM7-XMM7d,XMM8-XMM8d,XMM9-XMM9d,XMM10-XMM10d,XMM11-XMM11d,XMM12-XMM12d,XMM13-XMM13d,XMM14-XMM14d,XMM15-XMM15d], #?(64) EffDeg: ? Dead Cost: 0 Area: 0 Score:1e+35**

**107 MoveVecX2Leg === \_ 69 [[ 76 ]] !jvms: Short128Vector::addLanes @ bci:15 Short128Vector::addLanes @ bci:12 addlanes::workload @ bci:18**

**INITIAL INPUT MASK:**

**0 [R10-rS703], #!!!\_704\_vs\_704 EffDeg: ? Dead Cost: 0 Area: 0 Score:1e+35**

**FINAL INPUT MASK:**

**0 [XMM0-XMM0d,XMM1-XMM1d,XMM2-XMM2d,XMM3-XMM3d,XMM4-XMM4d,XMM5-XMM5d,XMM6-XMM6d,XMM7-XMM7d,XMM8-XMM8d,XMM9-XMM9d,XMM10-XMM10d,XMM11-XMM11d,XMM12-XMM12d,XMM13-XMM13d,XMM14-XMM14d,XMM15-XMM15d], #?(64) EffDeg: ? Dead Cost: 0 Area: 0 Score:1e+35**

**79 MoveVecX2Leg === \_ 80 [[ 76 ]] !jvms: Short128Vector::addLanes @ bci:15 Short128Vector::addLanes @ bci:12 addlanes::workload @ bci:18**

**INITIAL INPUT MASK:**

**0 [R10-rS703], #!!!\_704\_vs\_704 EffDeg: ? Dead Cost: 0 Area: 0 Score:1e+35**

**FINAL INPUT MASK:**

**0 [XMM0-XMM0d,XMM1-XMM1d,XMM2-XMM2d,XMM3-XMM3d,XMM4-XMM4d,XMM5-XMM5d,XMM6-XMM6d,XMM7-XMM7d,XMM8-XMM8d,XMM9-XMM9d,XMM10-XMM10d,XMM11-XMM11d,XMM12-XMM12d,XMM13-XMM13d,XMM14-XMM14d,XMM15-XMM15d], #?(64) EffDeg: ? Dead Cost: 0 Area: 0 Score:1e+35**

**== END OF INPUTS**

**Node[] : 77 MoveVecX2Leg === \_ 78 [[ 76 ]] !jvms: Short128Vector::addLanes @ bci:15 Short128Vector::addLanes @ bci:12 addlanes::workload @ bci:18**

**4 [XMM0-XMM0d,XMM1-XMM1d,XMM2-XMM2d,XMM3-XMM3d,XMM4-XMM4d,XMM5-XMM5d,XMM6-XMM6d,XMM7-XMM7d,XMM8-XMM8d,XMM9-XMM9d,XMM10-XMM10d,XMM11-XMM11d,XMM12-XMM12d,XMM13-XMM13d,XMM14-XMM14d,XMM15-XMM15d], #?(64) EffDeg: ? Def: N77 Cost: 0 Area: 0 Score:1e+35 Vector**

**78 Repl8S\_zero === 1 [[ 77 ]] vectorx[8]:{short} !jvms: Short128Vector::addLanes @ bci:15 Short128Vector::addLanes @ bci:12 addlanes::workload @ bci:18**

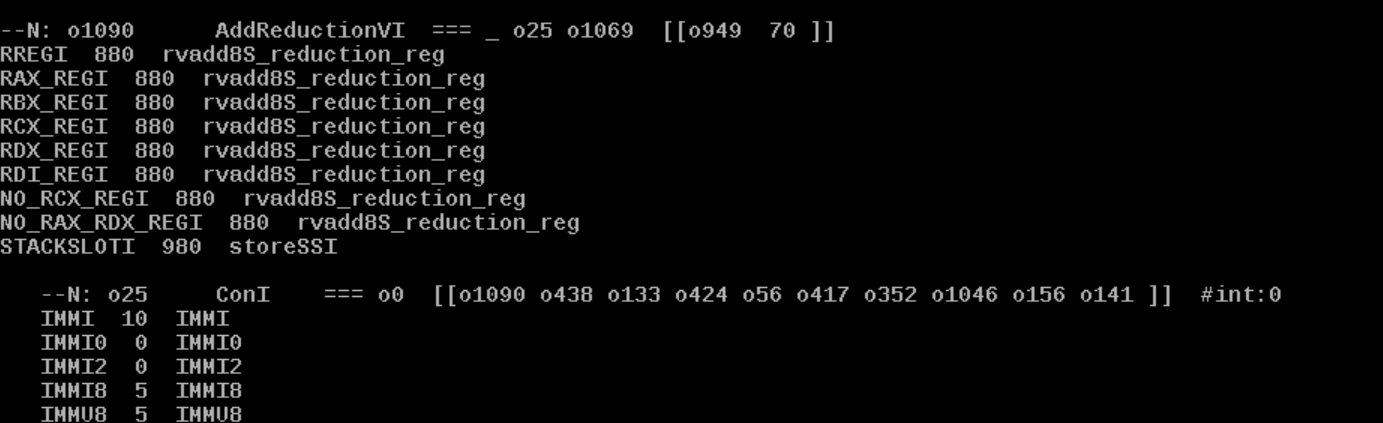
**INITIAL INPUT MASK:**

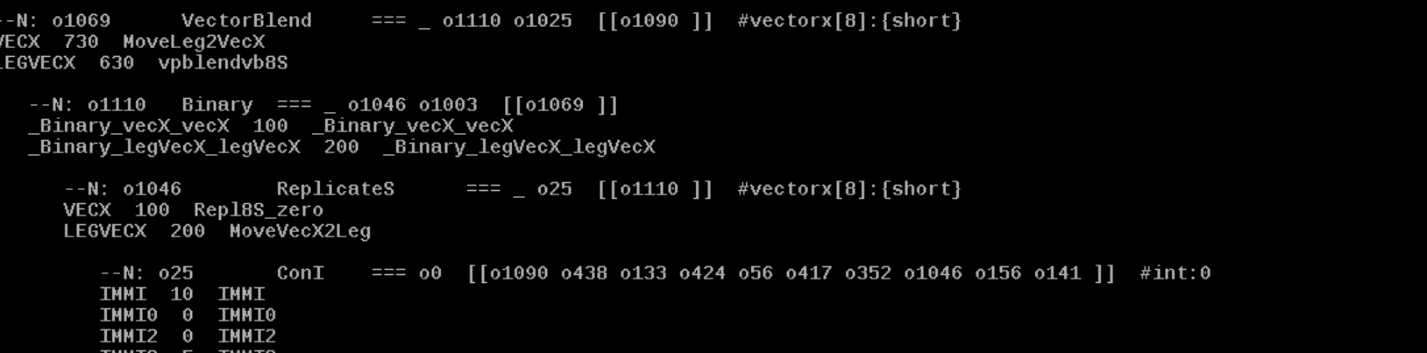
**4 [XMM0-XMM0d,XMM1-XMM1d,XMM2-XMM2d,XMM3-XMM3d,XMM4-XMM4d,XMM5-XMM5d,XMM6-XMM6d,XMM7-XMM7d,XMM8-XMM8d,XMM9-XMM9d,XMM10-XMM10d,XMM11-XMM11d,XMM12-XMM12d,XMM13-XMM13d,XMM14-XMM14d,XMM15-XMM15d,XMM16-XMM16d,XMM17-XMM17d,XMM18-XMM18d,XMM19-XMM19d,XMM20-XMM20d,XMM21-XMM21d,XMM22-XMM22d,XMM23-XMM23d,XMM24-XMM24d,XMM25-XMM25d,XMM26-XMM26d,XMM27-XMM27d,XMM28-XMM28d,XMM29-XMM29d,XMM30-XMM30d,XMM31-XMM31d], #?(128) EffDeg: ? Def: N78 Cost: 0 Area: 0 Score:1e+35 Vector**

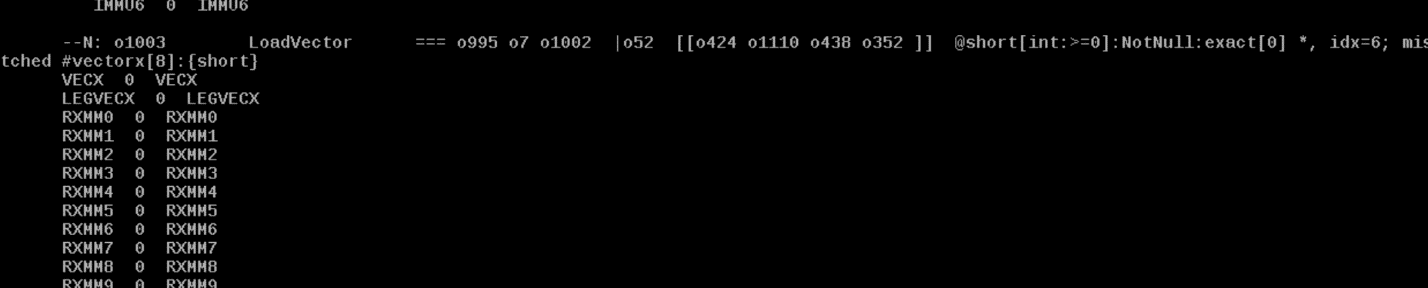
**FINAL INPUT MASK:**

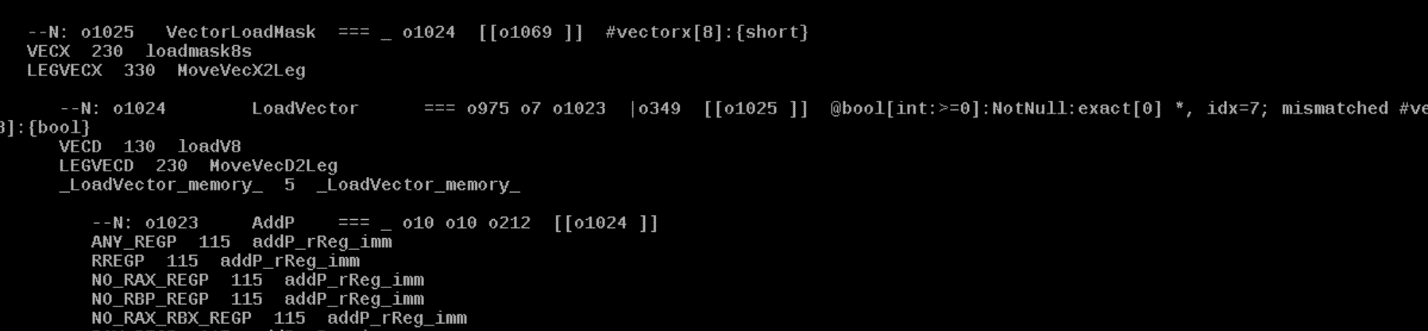
**4 [XMM0-XMM0d,XMM1-XMM1d,XMM2-XMM2d,XMM3-XMM3d,XMM4-XMM4d,XMM5-XMM5d,XMM6-XMM6d,XMM7-XMM7d,XMM8-XMM8d,XMM9-XMM9d,XMM10-XMM10d,XMM11-XMM11d,XMM12-XMM12d,XMM13-XMM13d,XMM14-XMM14d,XMM15-XMM15d,XMM16-XMM16d,XMM17-XMM17d,XMM18-XMM18d,XMM19-XMM19d,XMM20-XMM20d,XMM21-XMM21d,XMM22-XMM22d,XMM23-XMM23d,XMM24-XMM24d,XMM25-XMM25d,XMM26-XMM26d,XMM27-XMM27d,XMM28-XMM28d,XMM29-XMM29d,XMM30-XMM30d,XMM31-XMM31d], #?(128) EffDeg: ? Def: N78 Cost: 0 Area: 0 Score:1e+35 Vector**

**Dissecting selector labelling phase state tree dump for addLanesMasked128 operation:**

****

****

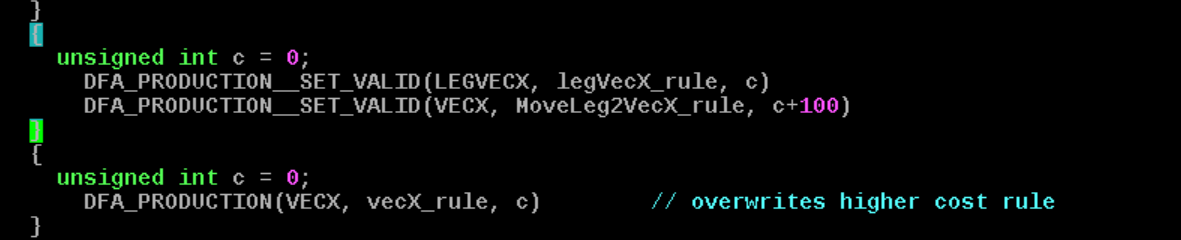
****

****

**Problem originates from labelling phase: LoadVector is shared node and state node corresponding to it that’s of a register load operation.**

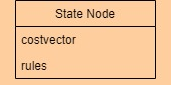
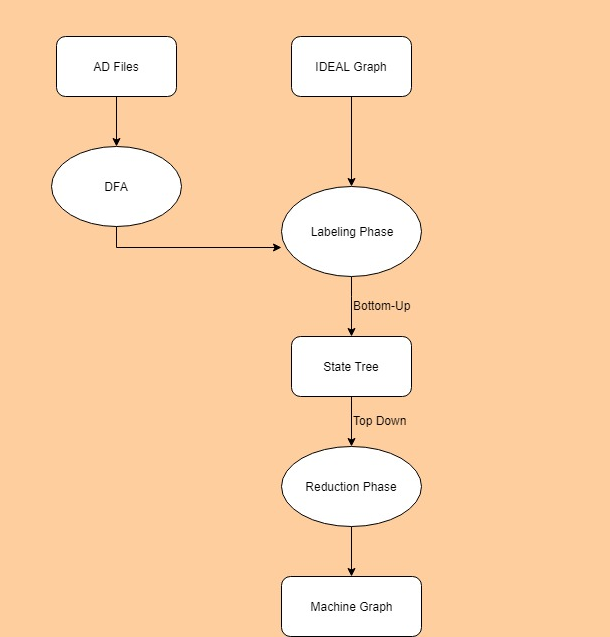
Post-selection FixupMixedFeatureOperands should take care of such scenarios.

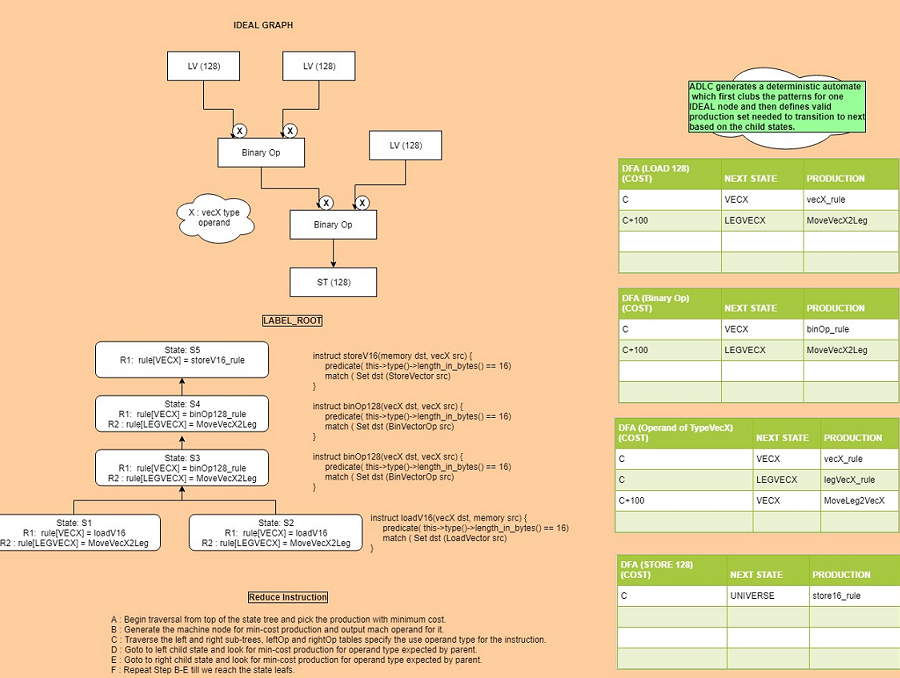
**ADLC generated DFA anomaly:**

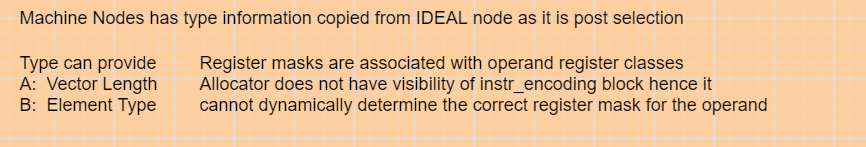


MoveLeg2VecX\_rule will never get executed since if user machine node expects a vecX operand then vecX\_rule production will be reduced (since its lower cost) and if user expects legVecX operand then legVecX\_rule production will be reduced to form a machine node.

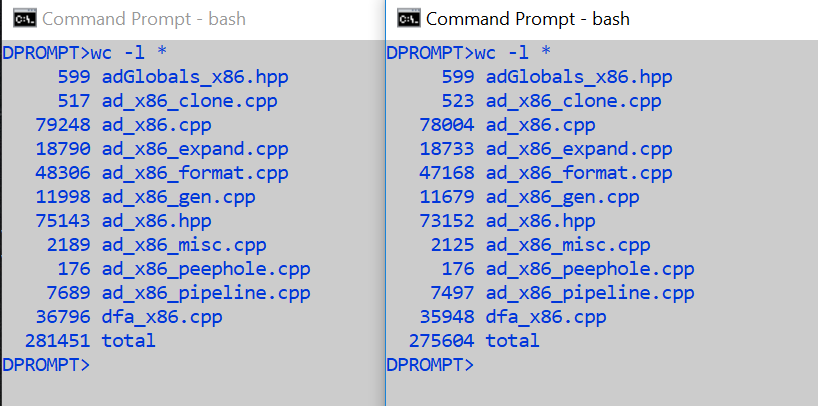
**Instruction Selection Flow Graph**

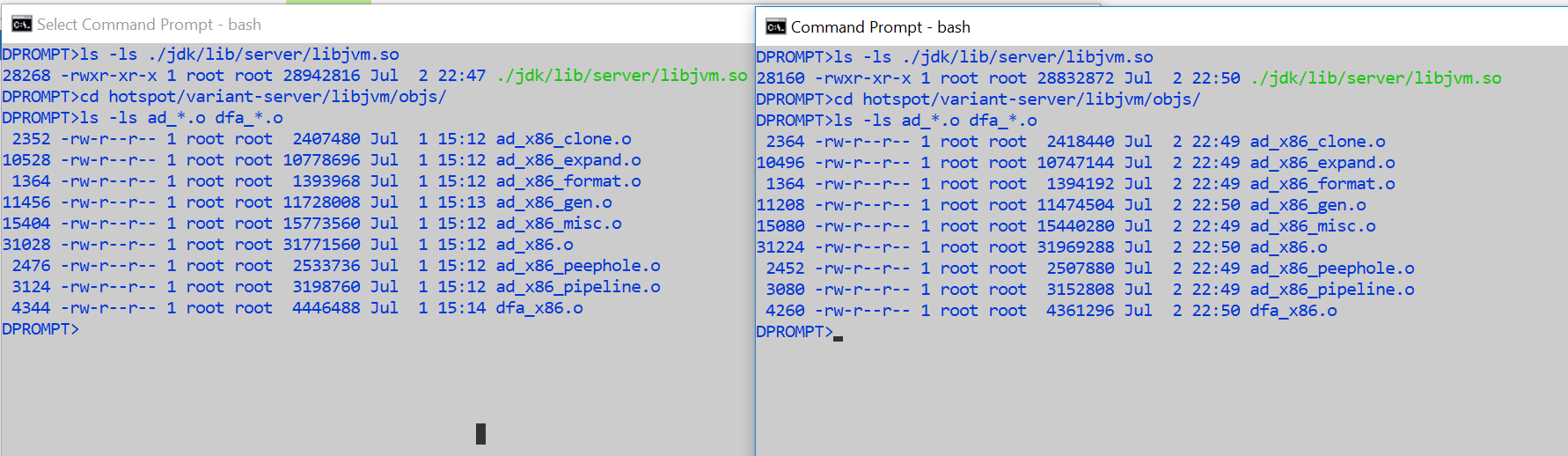




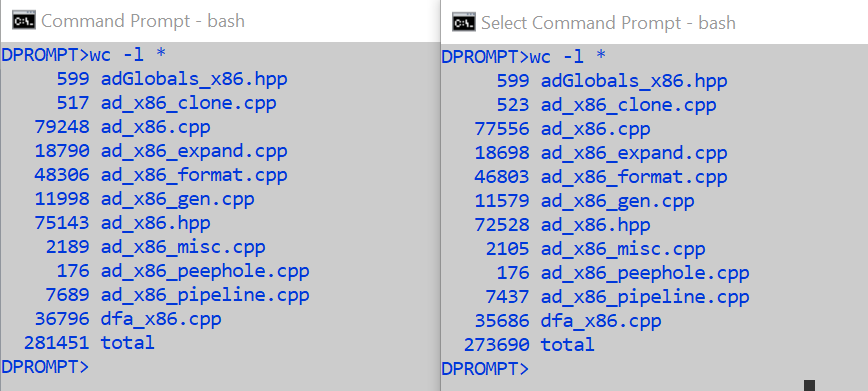


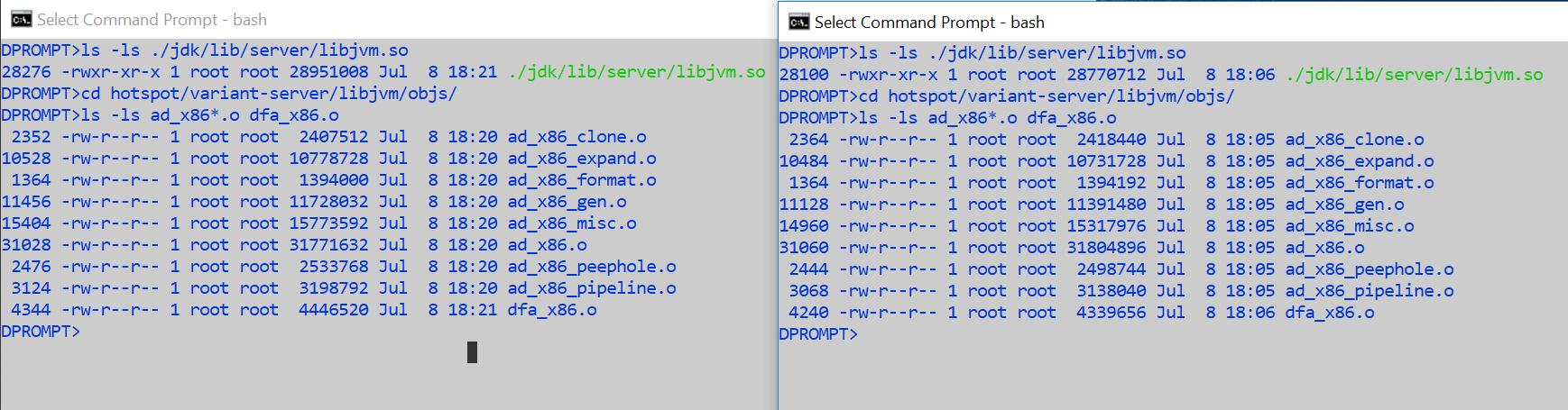
**Generic operands: ADLC generated files sizes, but libjvm.so size still not reduced.**



Stopped emitting MachineNode::Name() in release mode from adlc generated .o files.

libjvm.so size post broadcast operation (byte type) handling using generic operand.



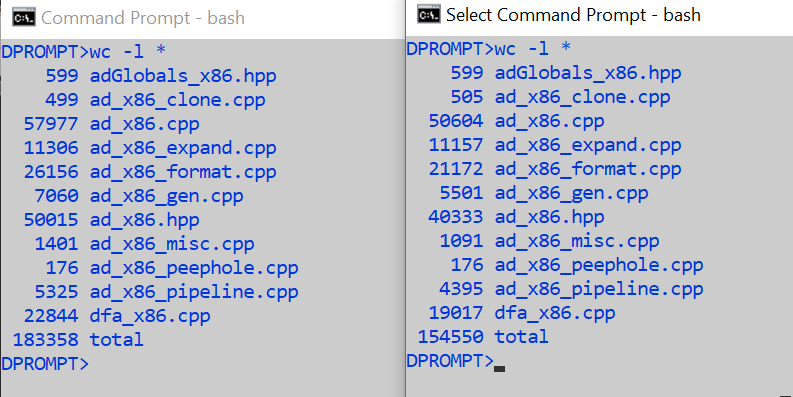


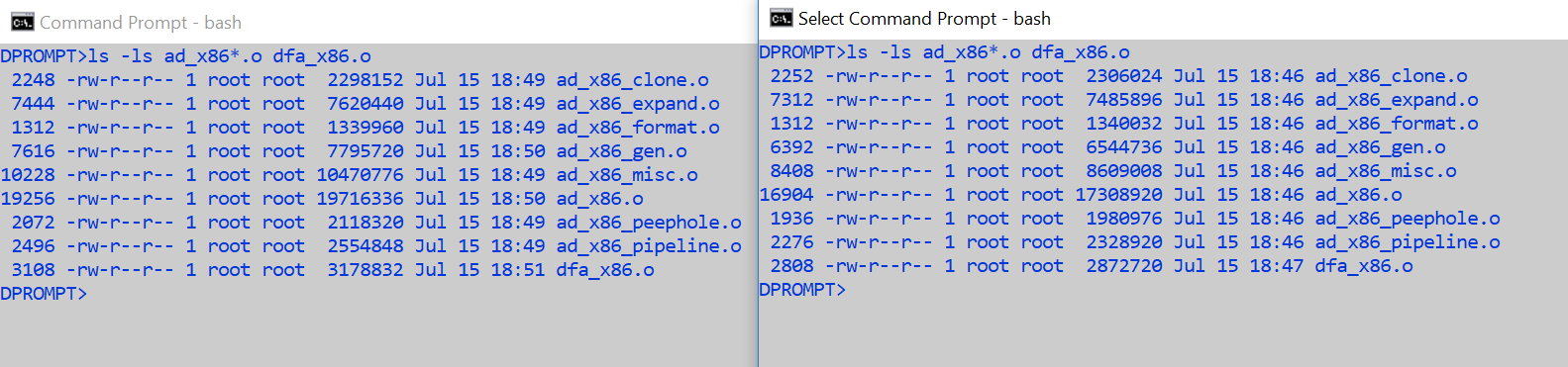
Mainline generic operand support stats:

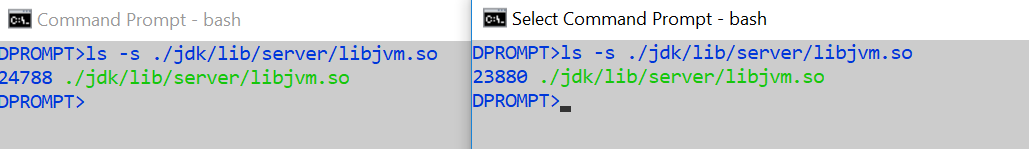
Total Vector instructions: 493

Instructions converted to generic operands: 385 reduced to 134 generic operands instructions patterns

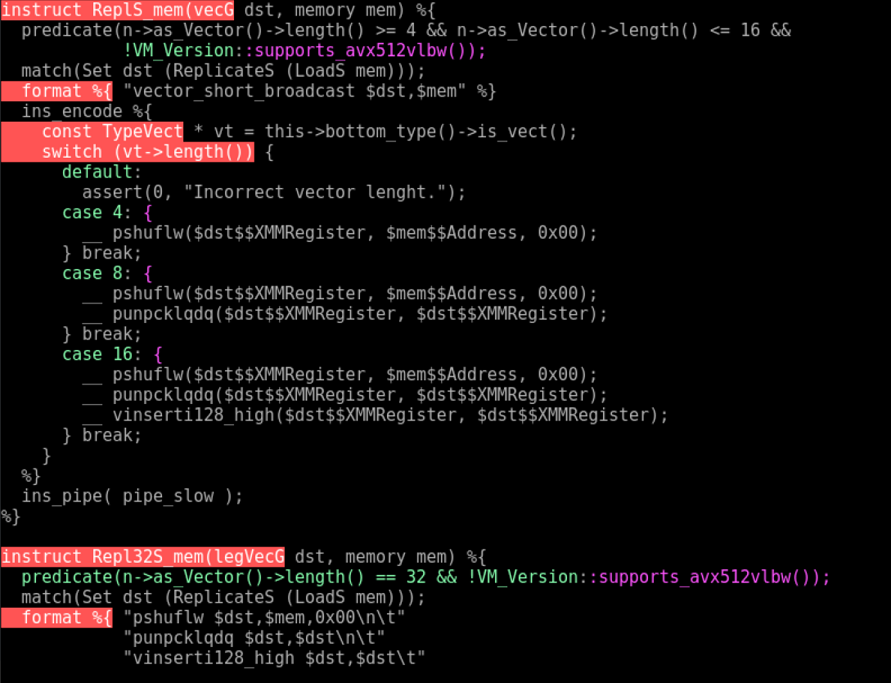
Remaining specific operands patterns: 108



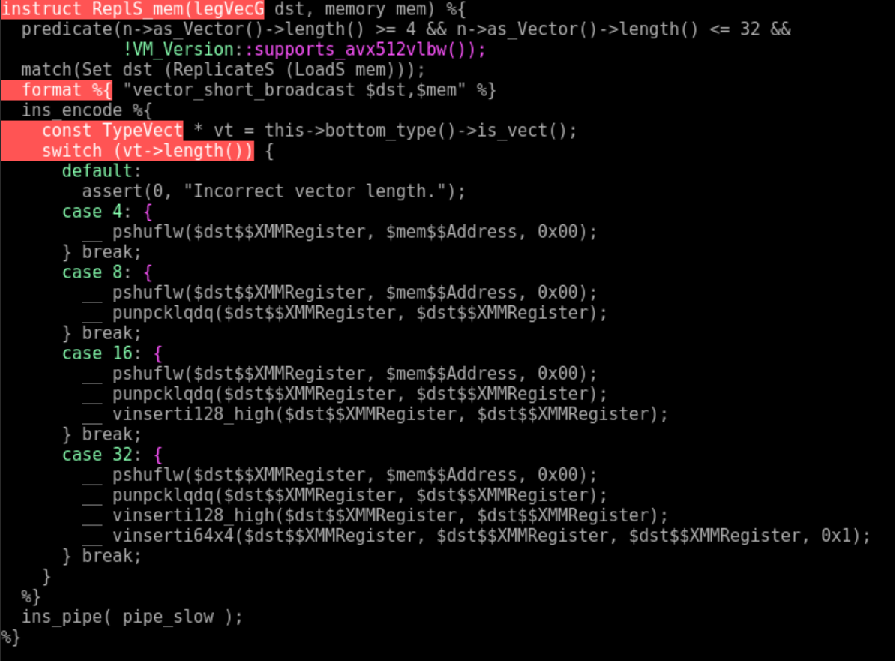




**Further collapsing of instruction patterns**



TO



**Useful rules: operand equivalence relations:**

1) legVecG covers following existing operands :-

1. SSE : vecX
2. AVX/AVX2 : vecX , vecY
3. AVX512 : vecZ (0-15) [legVecZ]

Post Selector stage will convert legVecG to either legVecX or legVecY or legVecG, following operands equivalencies will still hold good.

Over SSE : legVecX == vecX

OVer AVX/AVX2 : legVecX == vecX and legVecY == vecY

Over AVX512 : legVecZ == vecZ(0-15)

2) vecG covers following existing operands:-

a) SSE : vecX

b) AVX/AVX2 : vecX , vecY

c) AVX512 : vecZ (0-31)