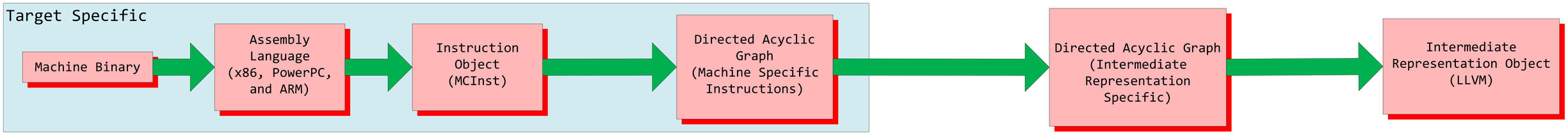
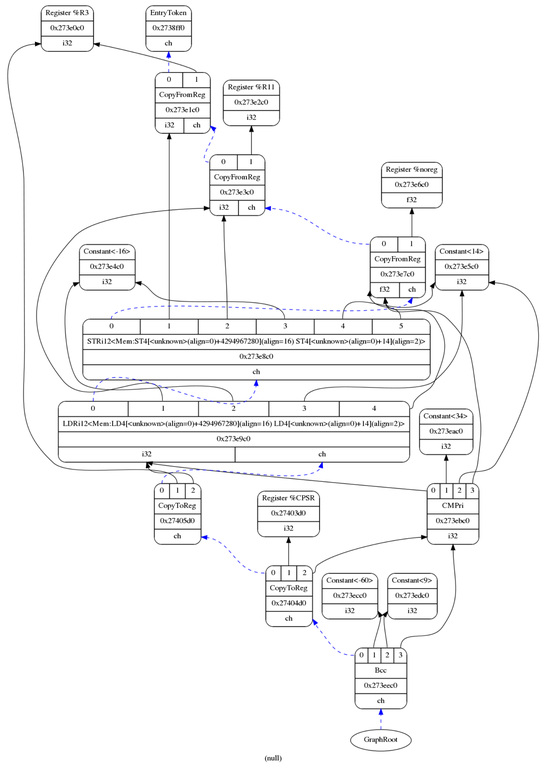
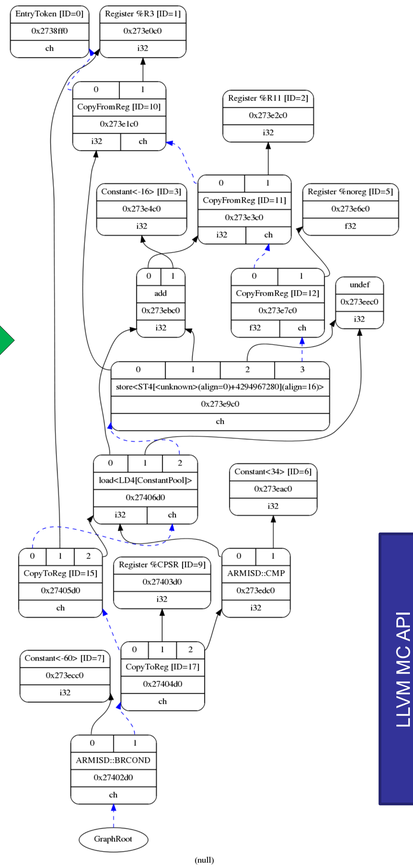
Fracture: Inverting LLVM's Target Independent Code Generator



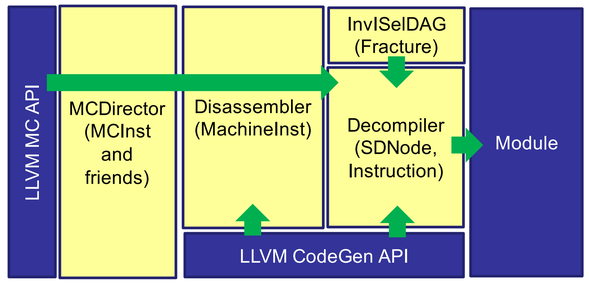

```
000000B8:   10 48 2D E9  push {r4, r11, lr}
000000BC:   08 B0 8D E2  add  r11, sp, #0x8
000000C0:   0C D0 4D E2  sub  sp, sp, #0xc
000000C4:   00 30 A0 E3  mov  r3, #0x0
000000C8:   10 30 0B E5  str  r3, [r11, #-16]
000000CC:   0A 00 00 EA  b    #0x28
```

```
%SP<def,tied1> = STMDB_UPD %SP<tied0>, pred:14,
                           pred:%noreg, %R4, %R11, %LR
%R11<def> = ADDri %SP, 8, pred:14, pred:%noreg, opt:%noreg
%SP<def> = SUBri %SP, 12, pred:14, pred:%noreg, opt:%noreg
%R3<def> = MOVi 0, pred:14, pred:%noreg, opt:%noreg
STRi12 %R3, %R11, 4294967280, pred:14, pred:%noreg
Bcc 40, pred:14, pred:%noreg
```






```
%0 = sub %SP, Constant<4>
store %R4, %0
%1 = sub %0, Constant<4>
store %R11, %1
%2 = sub %1, Constant<4>
store %LR, %2
%R11 = add %2, Constant<8>
%SP = sub %2, Constant<12>
%R3 = Constant<0>
%3 = add %R11, Constant<-16>
store %R3, %3
ARMISD::BRCOND Constant<40>
```



Technical Approach

- We mirrored the LLVM compilation process
  - This was sometimes easier, sometimes harder
    - Easy: Objects, graphs, etc already done
    - Hard: fighting LLVM API, decoding symbols/memory accesses, classic decompilation problems (which we haven't solved yet)
- Two major software components:
  - A "reverse" DAG manipulation table in TableGen
  - Libraries to handle LLVM API Interaction ("CodeInv")

Fracture TableGen Backend

- Borrowed a lot of code (modified some of it):

```
CodeGenDAGPatterns.h    CodeGenInstruction.cpp  CodeGenInstruction.h CodeGenIntrinsics.h
CodeGenMapTable.cpp     CodeGenRegisters.cpp        CodeGenRegisters.h    CodeGenSchedule.cpp
CodeGenSchedule.h     CodeGenTarget.cpp           CodeGenTarget.h       DAGISelMatcher.cpp
DAGISelMatcher.h      DAGISelMatcherEmitter.cpp      SDNodeInfo.cpp       SDNodeInfo.h
SetTheory.cpp      SetTheory.h         TGValueTypes.cpp   TableGen.cpp TableGenBackends.h
```

- CodeInvDAGPatterns
  - Helpers: InvTreePattern, InvTreePatternNode, InvTreePatternToMatch modeled after originals
  - Examines InOperandList and OutOperandList, then looks at Pattern and Fragments
  - No Glue, but kept chains

```
def SUBSri {
…
  dag OutOperandList = (outs GPR:$Rd);
  dag InOperandList = (ins GPR:$Rn, so_imm:$imm, pred:$p);
  list<dag> Pattern = [(set GPR:$Rd, CPSR, (anonymous.val.3663 GPR:$Rn, so_imm:$imm))];
…
def anonymous.val.3663 {
…
  dag Operands = (ops node:$LHS, node:$RHS);
  dag Fragment = (ARMsubc node:$LHS, node:$RHS);
…
```

```
/*6126*/  /*Scope*/ 30, /*->6157*/
/*6127*/    OPC_CheckOpcode, TARGET_VAL(ARM::SUBSri),
/*6130*/    OPC_MoveChild, 0,
/*6132*/    OPC_RecordNode,   // #0 = $Rn
/*6133*/    OPC_MoveParent,
/*6134*/    OPC_MoveChild, 1,
/*6136*/    OPC_CheckOpcode, TARGET_VAL(ISD::Constant),
/*6139*/    OPC_RecordNode,   // #1 = constant node
/*6140*/    OPC_MoveParent,
/*6141*/    OPC_MoveChild, 2,
/*6143*/    OPC_RecordNode,   // #2 = $p
/*6144*/    OPC_MoveParent,
/*6145*/    OPC_EmitNode, TARGET_VAL(ARMISD::SUBC), 0,
                1/*#VTs*/, MVT::i32, 2/*#Ops*/, 0, 1,  // Results = #3
/*6154*/    OPC_CompleteMatch, 1, 0,
            // Src: (SUBSri GPR:$Rn, (imm), pred:$p) - Complexity = ?
            // Dst: (ARMsubc GPR:$Rn, (imm))
```

- Notes:
  - Not optimized (OptimizeMatcher not called)
  - CPSR not in VT
  - Why care about predicate $p?
  - ARMISD::SUBC?

Ideas to Use Fracture

- Multiple code "views" (e.g., IDA Pro plugin) to see IR->MC when user doesn't know Target
- Use KLEE to analyze code
  - Lots of neat recent research could be applied here
  - Can use to solve indirect control transfer (ICT)
- Interpreter (lli) can run code
  - Extend to simulate entire embedded system if you can work out HW interaction
- Binary editor - retarget (for tractable programs), fix bugs, change functionality
- Decompiler (i.e., C-Backend)

Summary

- Modular (and fast?) conversion from machine code to IR
- Uses existing LLVM APIs and TableGen definitions
- Generic IR-based tools instead of one-off target specific solutions

DRAPER LABORATORY

Richard T. Carback III
rcarback@draper.com

November 6-7, 2013
LLVM Developer's Conference

Approved for Public Release, Distribution Unlimited

DISCLAIMER:

The views expressed are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

This research was sponsored by the Air Force Research Laboratory (AFRL), contract number FA8750-12-C-0261, as part of the DARPA High Assurance Cyber Military Systems (HACMS) program.