Validating and defending QEMU TCG targets

Alex Bennée
alex.bennee@linaro.org

KVM Forum 2014
Introduction

- ARMv8
  - Not just more bits
  - New Instruction Set
- Lots of interest in the community
  - Not a lot of available HW
- Strong demand for QEMU solution
The Challenge

- Mostly new code
- Can we get it right first time?
Estimating defect rates

- Coverity estimate for FLOSS of our size: 0.65/kloc
  - assumes "many eyeballs" review
- Estimate based on target-arm/translate.c
  - 9.5 kloc
  - ~100 "fixes" applied in commits
  - defect rate of 10.96/kloc
## Size of the problem

<table>
<thead>
<tr>
<th>Architecture</th>
<th>Lines of Code</th>
<th>DR:0.65</th>
<th>DR:10.96</th>
</tr>
</thead>
<tbody>
<tr>
<td>i386</td>
<td>21118</td>
<td>13.7</td>
<td>231.5</td>
</tr>
<tr>
<td>ppc</td>
<td>11317</td>
<td>7.4</td>
<td>124.0</td>
</tr>
<tr>
<td>arm</td>
<td>14029</td>
<td>9.1</td>
<td>153.8</td>
</tr>
<tr>
<td>aarch64</td>
<td>16874</td>
<td>11.0</td>
<td>184.9</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>63338</strong></td>
<td><strong>41.2</strong></td>
<td><strong>694.2</strong></td>
</tr>
</tbody>
</table>
AArch64 kernel+userspace boot

GCC Code Coverage Report
Directory: target-arm/
Date: 2014-10-10
Lines: 2996
Exec 18089
low: < 75.0%
CPU Specific files

GCC Code Coverage Report

Directory: target-arm/

File: target-arm/cpu.h

Date: 2014-10-10

Exec Total

Lines: 98 180
Branches: 33 125
translate-a64.c

GCC Code Coverage Report
Directory: target-arm/
File: target-arm/translate-a64.c
Date: 2014-10-10
Lines: 1676
Exec:
Total:
Branches: 613
2796
The rest

GCC Code Coverage Report

Directory: target-arm/

Date: 2014-10-10

Lines: 2996 18089

low: < 75.0 %
RISU

"Random Instruction Sequences for Userspace"
RISU System Architecture

Validating Platform (Apprentice)
- Test Sequence
- RISU

Reference Platform (Master)
- Test Sequence
- RISU

TCP Socket Connection
The Test Sequence

- Raw binary containing machine code
- Loaded and executed by RISU
Contents of the Sequence

- Setup code
- Test instructions
- Pseudo RISU operations
RISU Ops

- Architecture specific
  - Encoded in a reserved opcode
  - Multiple operations are needed
- The RISU Operations are:
  - Compare Registers/Memory
  - Set/Get Memory Pointers
  - Signal end of test
Typical execution sequence
Test Patterns

• Generate a pseudo-random sequence based on the pattern

```bash
./risugen --numinsns 100000 --pattern "ADDx.* A64" aarch64.risu addx.risu.bin
```

• Define an instruction format with fields and constraints

```plaintext
# C3.5.1 Add/subtract (extended register)
# 31 30 29 28 27 26 25 24 |23 22| 21 | 20  16 15 13 12 10 9  5 4  0
# sf op S 0 1 0 1 1 | opt | 1 | Rm opt imm3 Rn Rd
# NB: rn == 31 is perfectly valid, however RISU doesn't generate instructions that
# use the SP as that can cause problems with different SPs across systems
ADDx A64 sf:1 00 01011 00 1 rm:5 option:3 imm:3 rn:5 rd:5 \ !constraints { $rn != 31 && $rd != 31 && $imm <= 4; }
# ReservedValue: break the (imm <= 4) constraint
ADDx_RES A64 sf:1 00 01011 00 1 rm:5 option:3 imm:3 rn:5 rd:5 \ !constraints { $imm > 4; }
```
Load/Store Test Pattern

./risugen --numinsns 100000 --pattern "STRhr.*A64" --pattern "LDRhr.*A64" arch64.risu ldstr.risu.bin

# C3.3.10 Load/store register (register offset)
# 31 30 29 28 27 26 25 24 23 22 21 20 16 15 13 12 11 10 9 5 4 0
# size 1 1 1 V 0 0 opc 1 Rm opt S 1 0 Rn Rt
# XXX opt=011 for now (LSL), other options NIY.
# XXX the constraint rn != rm is our limitation, not imposed by arch.
STRhr A64 01 111000 00 1 rm:5 011 shft:1 10 rn:5 rt:5 \\
!constraints { $rn != 31 && $rn != $rt && $rm != $rt && $rn != $rm; } \\
!memory { align(2); reg_plus_reg_shifted($rn, $rm, $shft ? 1 : 0); }

LDRhr A64 01 111000 01 1 rm:5 011 shft:1 10 rn:5 rt:5 \\
!constraints { $rn != 31 && $rn != $rt && $rm != $rt && $rn != $rm; } \\
!memory { align(2); reg_plus_reg_shifted($rn, $rm, $shft ? 1 : 0); }
Load/Store Generated Code

Get offset into memory ptr

1: mov x0, #0x154 ; Random aligned offset
2: .inst 0x000005af3 ; RISU_OP_GETMEMBLOCK

Ensure base + index point at real memory

3: sub x27, x0, x10
4: mov x0, #0x0

Do load instruction

5: dsb sy
6: ldrh w6, [x27,x10]
7: dsb sy

Recalulate offset

8: .inst 0x000005af3 ; RISU_OP_GETMEMBLOCK
9: sub x27, x27, x0

Trigger RISU compare operations

10: .inst 0x000005af4 ; RISU_OP_COMPAREMEM
11: .inst 0x000005af0 ; RISU_OP_COMPARE
Limitations

- No system instructions
- Unable to test branching
- Avoids manipulating the SP
Porting
RISU Binary

- Boilerplate
  - recv_and_compare_register_info
- Helper Functions
  - advance_pc
  - report_match_status
- Signal Context Code
  - reginfo_init/is_eq/dump/report_mismatch
  - architecture value masks
Code Generator

- Setup code generation
- Pre/post amble code for memory blocks
- Encode RISU Operations
Instruction Templates

- Largest amount of effort
- Machine readable source would be handy
- Otherwise a PDF which cut&pastes well ;-)
  - Group instructions together
Case Study: QEMU TCG AArch64 Implementation
SUSE Work

- RFC AArch64 implementation
- Organic development to support linux-user build farm
Our approach

- Clean slate
- Follow the ARM ARM decoding structure
- Bootstrap to run RISU
Implementing the instructions

- Tested the boot-strapped instructions with RISU
- We divided the remaining groups between
  - Peter Maydell
  - Claudio Fontana
  - Myself
- Implemented the whole group
  - sometimes with Graf/Matz reference
  - always tested with RISU
Timeline

- September 2013
  - LCU13 planning
  - RISU prototype for AArch64
- November 2013
  - I joined Linaro ;-) 
- April 2014
  - QEMU 2.0
  - AArch64 linux-user (no crypto)
- August 2014
  - QEMU 2.1
  - AArch64 System Emulation
  - AArch64 Crypto Instructions
Reminder: Kernel Boot

GCC Code Coverage Report
Directory: target-arm/
Date: 2014-10-10
low: < 75.0%
Lines: Exec Total
2996 18089
Current RISU AArch64 Test Sequence

GCC Code Coverage Report

Directory: target-arm/

Date: 2014-10-10

Exec Total

Lines: 6783 18089

low: < 75.0 %
Post QEMU 2.0 bugs

- 5 A64 Specific Candidates
SQXTUN

- "Fix un-allocated test of scalar SQXTUN"
  - Discovered by user testing on master
  - Mea culpa - RISU would have caught this but for me
Dead Code Removal

- "Fix dead ?: in handle_simd_shift_fpint_conv()"
  - Dead code, could never execute
System Instructions

- "Fix return address for A64 BRK instructions"
  - RISU Limitation
- "fix TLB flush instructions"
  - Kernel system instruction
Supporting RISU on LAVA CI

- Multi-node testing setups are a pain
- Added support for record/playback
  - This allows for a simple stand-alone RISU test
Conclusions
Testing is key

- RISU was key to our successful delivery of AArch64 work
- qemu-aarch64 quickly adopted
  - very few complaints
Coverage Analysis

- Verify your tests exercising the right bit
- Identify areas which need more testing
Recommendation

• "I'm writing a new ISA front end, should I use RISU?"
  ■ YES
• Mature TCG ISAs can benefit as well
  ■ Debugging
  ■ Regression testing
• Defend functionality with CI
  ■ Know about regressions as they happen
CI

- QEMU's CI efforts are decentralised
- Most CI is build focused
  - Buildbot
  - Travis
- System specific CI testing is rare
  - Run manually by maintainers?
  - Linaro is committed to improving using LAVA
Future work for RISU

- Up-streaming of record/playback code
- Support for SP/PC related instructions
- Expand RISU to a non-ARM architecture?
Questions?