Displaced stepping in GDB and its implementation in Thumb-2

齐尧

HelloGCC 2011
September 18, 2011

在这四十分钟里边

你能知道的

- 我叫齐尧
  qiyaoltc@gmail.com
- 什么是displaced stepping
- 如何为一些新的指令支持displaced stepping
- 将来的多核或者多线程调试可能是什么样子

你不可能知道的

- GDB里边的displaced stepping 的代码到底是怎样做的
- GDB里边的displaced stepping 为什么设计成这个样子
- 为了多核或者多线程调试，具体应该做什么

什么是*in-place stepping?*

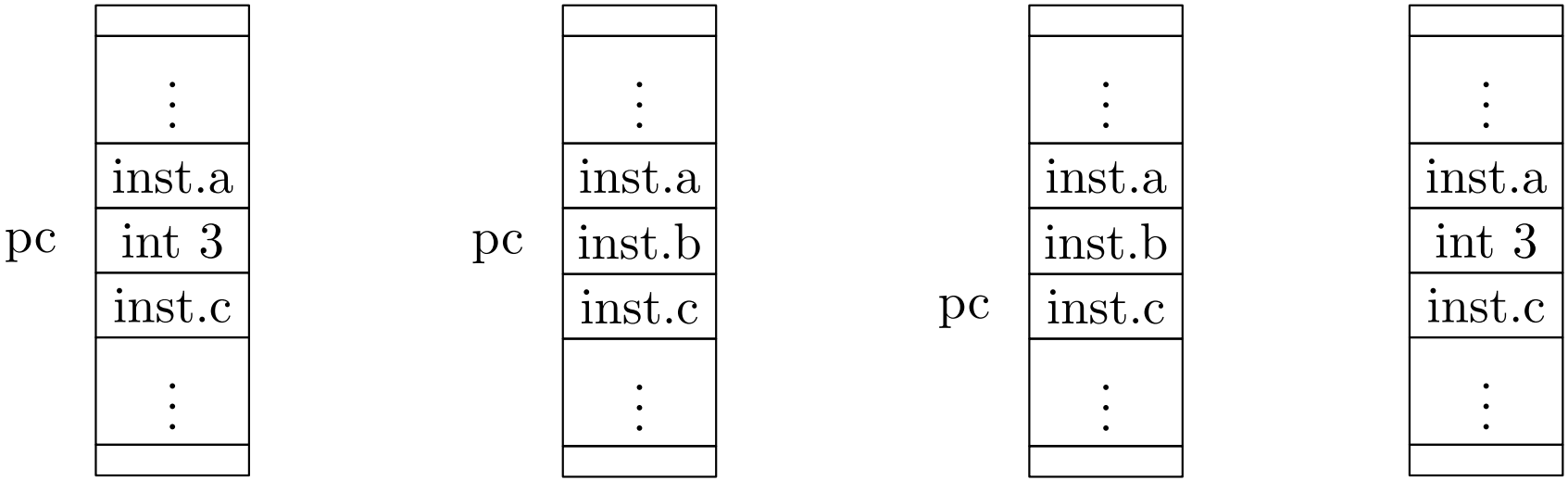


Figure: in-place stepping

in-place stepping 的问题 （我在HellGCC 2010讲过了）

- GDB 和其它调试器的缺省和基本方式
- 不适合调试多线程程序 (All-Stop Mode)
- 对程序的干扰比较大

All-Stop Mode vs Non-Stop Mode

- All-Stop Mode：当程序因为任何原因在GDB中停下来，那么 所有线程都会停下来。
  - stepping over breakpoint 的传统方式 in-place stepping 必须要求 All-Stop Mode
  - 至少是在 stepping over 的时候，是All-Stop。
  - 否则会有问题 （我在HelloGCC 2010讲过）
- Non-Stop Mode： 当程序因为任何原因在GDB中停下来，只有 当前线程停下来。
  - stepping over breakpoint 的时候需要displaced stepping

什么是*displaced stepping?*

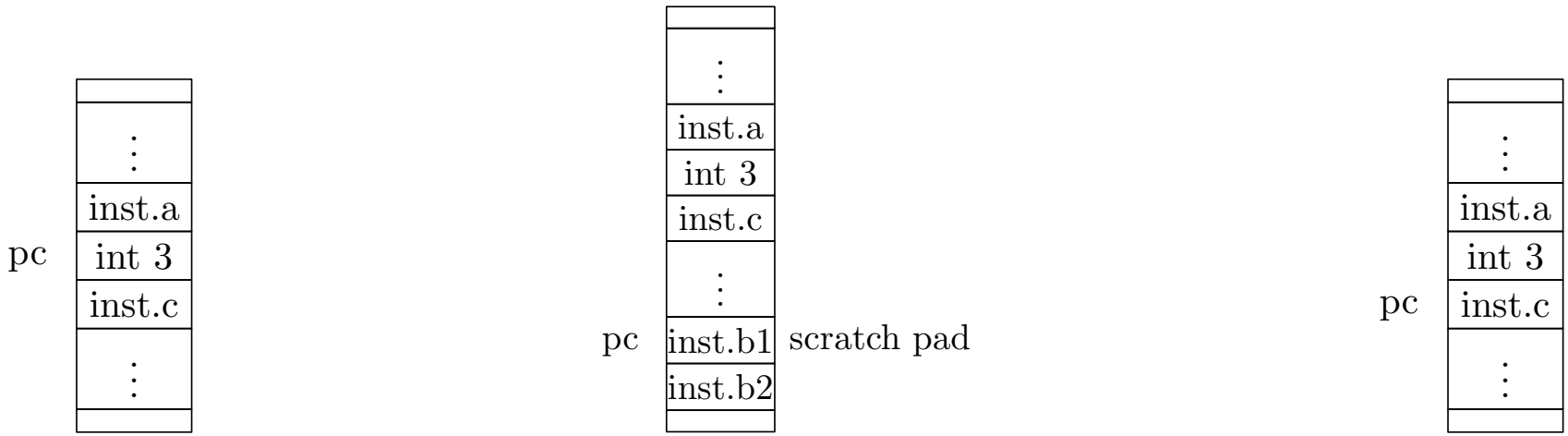
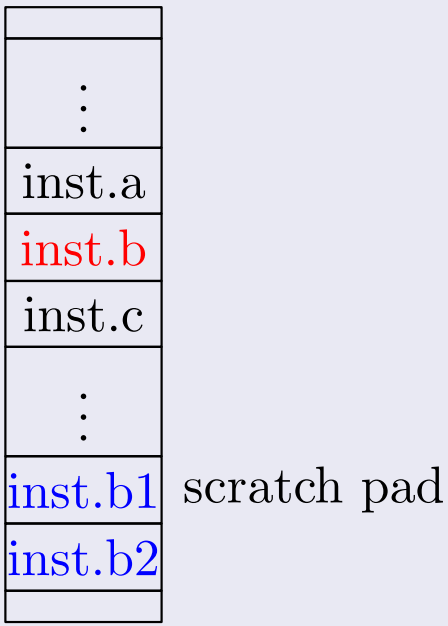


Figure: displaced stepping

displaced stepping 的考虑

- 把和inst.b**等价**的指令inst.b1,inst.b2拷贝到`scratch pad`，然后执行。
- 如何保证 inst.b == inst.b1 + inst.b2?
- `scratch pad`在哪里?

Displaced stepping的设计考虑



设计考虑

- 在内存空间中找到一段，可以执行代码，同时不能影响到程序其他部分的执行；`_start`;
- 对于任意一条指令inst.b，找到若干条与其在语义上完全一致的指令；和位置无关的指令是可以直接替代的，比如`mov r0, r1;`

Displaced stepping in GDB

过程

- 确定scratchpad的位置和长度 `gdbarch_max_insn_length` `gdbarch_displaced_step_location`
- 保存scratch pad内容，准备把需要的指令拷贝过去
- 分析当前指令，生成需要拷贝到scratch pad的指令，并且保存需要的寄存器 `gdbarch_displaced_step_copy_insn`
- 拷贝指令到scratch pad，加上断点指令。设置pc，让程序继续执行
- 命中断点后，恢复scratch pad，进行cleanup，恢复寄存器等等。`gdbarch_displaced_step_fixup`

Displaced stepping for Thumb-2

- 指令解码
- 选择替代指令
- 选择合适的cleanup操作

LDR Rd #imm8

- 选择替代指令 LDR R0, [R2, R3]
- 保存R0, R2, 和R3,
- 在R2写入PC，在R3中写入#imm8
- 在scratchpad执行 LDR R0, [R2, R3]
- 把R0的内容读出，写到Rd，恢复R0, R2, 和R3。

BL #imm

- 选在替代指令 NOP
- 计算目标地址DEST
- 在scratchpad执行 NOP
- 把PC的内容设置为DEST，同时更新LR

Displaced stepping for Thumb-2

实现工作

- 把每一条Thumb指令，做上一页的分析和操作，用代码实现。
- 代码量在两千行左右
- 从2010-12 到 2011-09

总结经验

- 需要十分熟悉处理器的指令
- 需要及其细心地阅读指令手册
- 尽早了解社区对这个工作的要求或者期望

多线程调试或者多核处理器调试展望

传统调试 *vs* 多核多线程

- 传统调试基于这样的假设:
  - 错误可重现
  - 调试对程序无干扰或者少干扰
- 多核多线程程序:
  - 不确定性，问题也许不能重现
  - 常规的测试没有意义，或者意义减少

多线程调试或者多核处理器调试展望

调试手段的变化（个人愚见）

- 更高的抽象，更细的粒度。调式的粒度可能需要在某个core上；但是需要有高层的抽象
- 更多的tracing， 更少的peeking。采用高效的trace，采集数据。
- 更多的有算法的分析，更少的手工检查。利用先进的算法分析采集到的数据，推导出程序中的问题。 Race detection or deadlock detection
- 更多的**工具引导人**，更少的**人使用工具**。由工具重现问题的环境，然后再有人去检查程序的问题。Deadlock causing
- visualization和重新定义**程序性能**和profiling。

The End

Q & A