Discovered in Android clang version 5.0.300080 from android-ndk-r16

Red text built with -Oz, incorrectly clobbers $r6 during stack unwinding.
Green text built with -Os, no problem

The following is the faulty ARM exception handling table for the function below:

```
$ readelf -u
Unwind table index '.ARM.exidx' …
…
0xc10
<_Z4workRKNSt6__ndk112basic_stringIcNS_11char_traitsIcEENS_9allocatorIcEEEE>:
@0x24480x80978408
  Compact model index: 10
  0x97      vsp = r7
  0x41      vsp = vsp - 8
  0x84 0x0e0x08 pop {r5, r6, r7, r14}
  0xb0      finish
  0xb0      finish
…
```

In the event of an exception, the procedure to unwind this frame for the Oz code is to restore registers $r{5, 6, 7, 14}, whereas the correct behaviour should be either to restore these stack slots as $r{2, 3, 7, 14} as per the function's epilogue, or to use the compact encoding (0x80978408) and just restore $r7 and $r14 (tested and shown to work).

Disassembly of interesting parts of function (with faulty EH table):

```
test`work:
<+0>:  push   {r5, r6, r7,{r7, lr}
<+2>:  add    r7, sp, #0x8
<+2>:  mov    r7, sp
<+4>:  sub    sp, #0x8
```

The Oz version elides the separate adjustment of $sp by pushing two dummy registers. Os version doesn't, rather it pushes clobbered register $r7 and makes a separate adjustment to $sp.

```
<+4><+6>:  ldr    r1, [pc, #0x3c]#0x40]
<+6><+8>:  add    r1, pc
<+8><+10>: ldr    r1, [r1]
<+10><+12>: ldr    r1, [r1]
<+12><+14>: str    r1, [sp, #0x4]
```

Saving the stack protector on the stack at the location previously occupied by $r6

```
…
<+30><+32>: str    r2, [sp]
```

Saving a variable on the stack at the location previously occupied by $r5

```
…
<+42><+44>: bl     0xb80
```

Call to a function that may throw an exception.

```
…
<+62>: addeq  sp, #0x8
<+60><+64>: popeq  {r2, r3, r7,{r7, pc}
```

Return from frame: Oz version restores modified stack entries to caller-saved scratch registers r2 and r3 (ie, different to what was originally pushed).