Google

Asm Goto with Outputs

Linux Plumbers Conf 2020 - LLVM MC

Bill Wendling

Asm goto

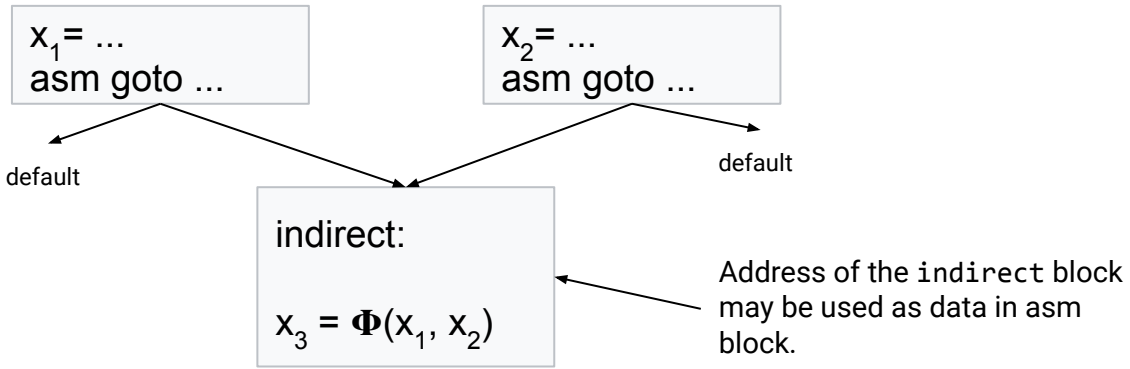
- `asm goto` allows assembly code to jump to one or more C labels.
- Motivating example: We want on rare occasions to call the `trace` function; on other occasions we'd like to keep the overhead to the absolute minimum. We can patch the `nop` instruction at run time by finding data stored in this section to be an unconditional branch to the stored label.

```
#define TRACE1(NUM)                                 \
    do {                                            \
         asm goto ("0: nop;"                        \
                   ".pushsection trace_table;"      \
                   ".long 0b, %l0;"                 \
                   ".popsection"                    \
                   : : : : trace#NUM);              \
         if (0) { trace#NUM: trace(); }             \
    } while (0)

#define TRACE TRACE1(__COUNTER__)
```

Asm goto with outputs

- We extended clang's implementation of asm goto to support outputs.
  - GCC didn't implement asm goto with outputs, due to an internal restriction of the compiler: control transfer instructions cannot have outputs.
- Outputs are supported only on the fallthrough path.
  - Supporting outputs on the indirect branches is very messy. E.g. it's not clear how to resolve PHI nodes.



Why would outputs be useful?

- Allows `asm goto` to behave the same as a normal asm block on the default / fallthrough path.
- Allows the programmer to optimize code further:
  - Now that they no longer need to use memory for outputs.
  - Improve the programmers ability to reuse labels as exceptional cases.
  - Reduce the amount of generated code—e.g. `unsafe_get_user()`.

Ambiguous cases

1. Multiple `asm goto` statements with the same target, but non-mutually satisfiable output constraints.
   a. I maintain that `asm goto` statements *shouldn't* jump to the same basic block, but normal transformations may make it impossible to enforce that assertion.
2. Jumping to labels where the output variable is out of scope.
   a. Shouldn't be able to refer to out of scope variables, but maybe something gross like this.

```
int foo() {
    int y;
    asm goto("..." : "=r"(y) : : : label);

    int x = bar();
    if (0) { label: y = x; }
    return y;
}
```

Design details

- Clang allows terminating IR (**I**ntermediate **R**epresentation) instructions to have outputs.
  - This is how exception handling is modeled.
- IR is converted to MIR (**M**achine **IR**), which allows for multiple terminators at the end of blocks.
- ASM goto's representation as a terminator in MIR didn't fit well with clang's back-end restrictions—i.e. there cannot be a non-terminator after a terminator.
  - Difficult to represent moving values from an asm goto call into registers before the end of the block, because there cannot be non-terminators (`MOV` instructions) after terminators.
    - Could place moves in separate fallthrough block, but "live in" analysis isn't ran until late in MIR processing.
  - Live range splits may need to spill after an asm goto, resulting again in a non-terminator after terminator violation.
- Ultimately, we decided that the asm goto representation in MIR *shouldn't* be a terminator.
  - However, we must ensure that uses of non-output variables on the indirect branches are defined *before* the asm block.

CONFIG_CC_HAS_ASM_GOTO_OUTPUT

https://lkml.org/lkml/2018/2/14/656

```
Side note: one thing that limits "asm goto" in gcc is the fact that
you can't have outputs.

But extending on what gcc does, and allowing outputs (possibly valid
in the fall-through case only, not in the cases where it jumps away to
a label) would be a big improvement on what gcc does. Linus
```

Commit 587f17018a2c (“Kconfig: add config option for asm goto w/ outputs”)

```
    [ This is not used anywhere yet, and currently released compilers don't
      support this yet, but it's coming, and I have some local experimental
      patches to take advantage of it when it does   - Linus ]
```

RFC

- It's mutually beneficial for the gcc and clang communities to collaborate on Linux support.
- Both compilers bring different things to the table:
  - Warnings, sanitizers, code health tools, ideas for language extensions, etc.
- Clang-built Linux (https://clangbuiltlinux.github.io/) is a renewed effort to make clang a first-class citizen in the Linux world.

Please join us!