Bazel

Bazel Constraints for RISC-V

Please read Bazel Code of Conduct before commenting.

- **Authors**: tpudlik@google.com
- **Status**: Draft | **In review** | Approved | Rejected | In progress | Implemented
- **Reviewers**: aiuto@google.com
- **Created**: 2023-02-28
- **Updated**: 2023-03-07
- **Discussion thread**: <link>

Overview

This document proposes adding RISC-V-related constraint settings and constraint values to https://github.com/bazelbuild/platforms.

PR: https://github.com/bazelbuild/platforms/pull/63

Proposed additions

At a high level:

- `@platforms//cpu` values will represent the base instruction sets.
- `@platforms//cpu/riscv/extensions` will contain constraint_settings representing the ratified RISC-V extensions.

Extensions representation

For each ratified extension, we will introduce a constraint_setting with two values, indicating the presence or absence of the extension. For example:

```python
# Does the CPU possess extension M (multiplication and division)?
constraint_setting(
  name = "M_setting",
  default_constraint_value = ":no_M",
)
```

```
# CPU does possess the M extension
constraint_value(
    name = "M",
    constraint_setting = "M_setting",
)

# CPU does not possess the M extension
constraint_value(
    name = "no_M",
    constraint_setting = "M_setting",
)
```

Shorthand extensions

The RISC-V specification defines "shorthand extensions" which represent lists of extensions. For example, the "Zk extension" is actually just shorthand for Zbkb, Zbkc, Zbkx, Zknd, Zkne, Zknh, Zksed, Zksh, Zkn, Zks, Zkt, Zkr.

The official shorthand extensions will be represented as lists of constraint settings, defined in `cpu/riscv/extensions/shorthand.bzl`.

Implementation details

Plain BUILD file

I decided to explicitly write out all the constraint_settings and constraint_values in the BUILD file. An alternative would have been to introduce a macro or list comprehension generating them from the list of extension names. My sense was that this would have made the code easier to write and review, but less readable for the end-users, and so was a poor tradeoff.

Default constraint values

Every extension constraint possesses a default value, corresponding to the extension being absent. This should make for clearer select statements in user code (only the two clearly exhaustive cases, ":M" and ":no_M", need to be considered: every platform falls into one of these categories). But note that this removes the flexibility of defining platforms that leave the presence of an extension unspecified.

Alternatives considered

RISC-V profiles

Profiles are common sets of extensions. However, profile semantics is complex: extensions within a profile may be mandatory or optional. It's not obvious if a profile should be represented as,

- A list of mandatory extensions
- A list of mandatory and optional extensions
- A platform listing the mandatory (and optional?) extensions, intended to be inherited from by platforms defined by users.

Furthermore, the profiles' specification has not yet been officially ratified by the RISC-V Foundation.

In view of this, we're not adding any representation of RISC-V profiles to the canonical Bazel constraints at this time.

Extension versions

RISC-V extensions are versioned, but no representation of extension versioning is added in this proposal. The unversioned constraint_values added in this proposal should be interpreted as referring to the latest ratified standard version.

One way to represent extension versions would be to create a distinct constraint_value for each version:

Python

```
# Does the CPU possess extension M (multiplication and division)?
constraint_setting(
  name = "M_setting",
  default_constraint_value = ":no_M",
)

# CPU does possess the M extension at the latest ratified version.
constraint_value(
    name = "M",
    constraint_setting = "M_setting",
)

# CPU does possess the M extension, version 2.0.
constraint_value(
    name = "M_2_0",
    constraint_setting = "M_setting",
)

# CPU does possess the M extension, version 1.0.
constraint_value(
```

```
    name = "M_1_0",
    constraint_setting = "M_setting",
)

# CPU does not possess the M extension
constraint_value(
    name = "no_M",
    constraint_setting = "M_setting",
)
```

This seemed like an excessive level of detail. If needed, versioned constraint_values can be added in the future. (Note, though, that adding versions after the fact may lead to issues with exhaustive select statements that consider only the ":M" and ":no_M" cases.)

Open questions

Privileged architecture

I looked at the first sections of the ISA Manual Volume II. There's a distinction between the "Machine ISA", "Supervisor ISA", and "Hypervisor ISA" on the one hand, and the extensions "Svnapot", "Svpbmt", and "Svinval" on the other. Which of these entities represent capabilities, and should be mapped to Bazel constraint_settings? Should the supported modes (M, S, U) be mapped to constraint_settings instead?

Specific CPUs as base platforms?

Specific physical CPUs can be represented as platforms containing a base instruction set and extensions. We could add Bazel platforms representing well-known CPUs to the @platforms repository, for users to inherit from in defining their own platforms.

This would be a novel addition to the repository, since it currently only defines constraint settings, not base platforms for inheritance.

Document History

Date	Description
2023-03-01	First proposal


Bazel