The Rust standard library's portable SIMD API

CI passing

Code repository for the Portable SIMD Project Group. Please refer to CONTRIBUTING.md for our contributing guidelines.

The docs for this crate are published from the main branch. You can read them here.

If you have questions about SIMD, we have begun writing a guide. We can also be found on Zulip.

If you are interested in support for a specific architecture, you may want stdarch instead.

Hello World

Now we're gonna dip our toes into this world with a small SIMD "Hello, World!" example. Make sure your compiler is up to date and using `nightly`. We can do that by running

```
rustup update -- nightly
```

or by setting up `rustup default nightly` or else with `cargo +nightly {build,test,run}`. After updating, run

```
cargo new hellosimd
```

to create a new crate. Edit `hellosimd/Cargo.toml` to be

```toml
[package]
name = "hellosimd"
version = "0.1.0"
edition = "2018"
[dependencies]
core_simd = { git = "https://github.com/rust-lang/portable-simd" }
```

and finally write this in `src/main.rs`:

```rust
use core_simd::*;
fn main() {
    let a = f32x4::splat(10.0);
    let b = f32x4::from_array([1.0, 2.0, 3.0, 4.0]);
    println!("{:?}", a + b);
}
```

Explanation: We import all the bindings from the crate with the first line. Then, we construct our SIMD vectors with methods like `splat` or `from_array`. Finally, we can use operators on them like `+` and the appropriate SIMD instructions will be carried out. When we run `cargo run` you should get `[11.0, 12.0, 13.0, 14.0]`.

Code Organization

Currently the crate is organized so that each element type is a file, and then the 64-bit, 128-bit, 256-bit, and 512-bit vectors using those types are contained in said file.

All types are then exported as a single, flat module.

Depending on the size of the primitive type, the number of lanes the vector will have varies. For example, 128-bit vectors have four `f32` lanes and two `f64` lanes.

The supported element types are as follows:

- **Floating Point:** `f32` , `f64`
- **Signed Integers:** `i8` , `i16` , `i32` , `i64` , `i128` , `isize`
- **Unsigned Integers:** `u8` , `u16` , `u32` , `u64` , `u128` , `usize`
- **Masks:** `mask8` , `mask16` , `mask32` , `mask64` , `mask128` , `masksize`

Floating point, signed integers, and unsigned integers are the primitive types you're already used to. The `mask` types are "truthy" values, but they use the number of bits in their name instead of just 1 bit like a normal `bool` uses.