`wasm64-unknown-unknown`

Tier: 3

WebAssembly target which uses 64-bit memories, relying on the memory64 WebAssembly proposal.

Target maintainers

- Alex Crichton, https://github.com/alexcrichton

Requirements

This target is cross-compiled. The target supports `std` in the same manner as the `wasm32-unknown-unknown` target which is to say that it comes with the standard library but many I/O functions such as `std::fs` and `std::net` will simply return error. Additionally I/O operations like `println!` don't actually do anything and the prints aren't routed anywhere. This is the same as the `wasm32-unknown-unknown` target. This target comes by default with an allocator, currently dlmalloc which is ported to rust.

The difference of this target with `wasm32-unknown-unknown` is that it's compiled for 64-bit memories instead of 32-bit memories. This means that `usize` is 8-bytes large as well as pointers. The tradeoff, though, is that the maximum memory size is now the full 64-bit address space instead of the 4GB as limited by the 32-bit address space for `wasm32-unknown-unknown`.

This target is not a stable target. The memory64 WebAssembly proposal is stil in-progress and not standardized. This means that there are not many engines which implement the `memory64` feature and if they do they're likely behind a flag, for example:

- Nodejs - `--experimental-wasm-memory64`
- Wasmtime - `--wasm-features memory64`

Also note that at this time the `wasm64-unknown-unknown` target assumes the presence of other merged wasm proposals such as (with their LLVM feature flags):

- Bulk memory - `+bulk-memory`
- Mutable imported globals - `+mutable-globals`
- Sign-extending operations - `+sign-ext`
- Non-trapping fp-to-int operations - `+nontrapping-fptoint`

The `wasm64-unknown-unknown` target intends to match the default Clang targets for its `"C"` ABI, which is likely to be the same as Clang's `wasm32-unknown-unknown` largely.

> ***Note**: due to the relatively early-days nature of this target when working with this target you may encounter LLVM bugs. If an assertion hit or a bug is found it's recommended to open an issue either with rust-lang/rust or ideally with LLVM itself.*

This target does not support `panic=unwind` at this time.

Building the target

You can build Rust with support for the target by adding it to the `target` list in `config.toml`, and the target also requires `lld` to be built to work.

```
[build]
target = ["wasm64-unknown-unknown"]

[rust]
lld = true
```

Building Rust programs

Rust does not yet ship pre-compiled artifacts for this target. To compile for this target, you will either need to build Rust with the target enabled (see "Building the target" above), or build your own copy of `std` by using `build-std` or similar.

Note that the following `cfg` directives are set for `wasm64-unknown-unknown`:

- `cfg(target_arch = "wasm64")`
- `cfg(target_family = "wasm")`

Testing

Currently testing is not well supported for `wasm64-unknown-unknown` and the Rust project doesn't run any tests for this target. Testing support sort of works but without `println!` it's not the most exciting tests to run.

Cross-compilation toolchains and C code

Compiling Rust code with C code for `wasm64-unknown-unknown` is theoretically possible, but there are no known toolchains to do this at this time. At the time of this writing there is no known "libc" for wasm that works with `wasm64-unknown-unknown`, which means that mixing C & Rust with this target effectively cannot be done.