Apple / Linux Convergence Macros

This chapter documents the ongoing work in defining a macro suite that allows coding AARCH64 programs once with the ability to build correctly on Apple Silicon and Linux machines without change.

The work is ongoing and subject to change.

There are limits to what these macros can do. Variadic functions such as `printf()` must be handled via parallel code paths (i.e. use of `#if`).

Make assembly language file names end in .S

For widest compatibility, end your assembly language files in capital S rather than small s. This forces `gcc` to make use of the C preprocessor as there is no command line option to make it do so. `clang` (and a `gcc` derived from it) may or may not have a command line option to force the invocation of the preprocessor but ending your file names in capital S is universally appropriate.

Prepended underscores

A main difference unified by the macros is Apple's prepending of underscores to labels defined by libraries such as the CRT and certain other symbols like `main`.

So, `main` will not be found by the linker on Apple systems and `_main` will be an error on Linux systems.

The macros adjust for this.

There are some exceptions to the prepending rule on Apple such as making use of `FILE * stdin`. On Linux this would be `stdin`. On Mac OS you would expect `_stdin` but you'd be wrong... instead Apple uses `___stdinp`. Why? Because Apple.

There is an assumption here that labels created by you do not have prepended underscores. This can be a problem if this isn't the case. The solution may be to add a parallel set of macros that either do prepend or do not. This is an open question which we hope to get user input to resolve.

Note About Variadic Functions

Functions such as `printf()` do not have fixed signatures. That is, they may accept a variable number of parameters of varying types. Linux and Apple Silicon handle these functions quite differently.

This is explained at length in the chapter on variadic functions.

Macros of general use

First, we describe a number of macros which are the same on both Apple and Linux. These macros don't converge Apple and Linux. They're just nice to have.

PUSH_P, PUSH_R, POP_P and POP_R

These macros save some repetitive typing. For example:

```
PUSH_P  x29, x30
```

resolves to:

```
stp     x29, x30, [sp, -16]!
```

START_PROC and END_PROC

Place START_PROC after the label introducing a function.

Place END_PROC after the last `ret` of the function.

These resolve to: `.cfi_startproc` and `.cfi_endproc` respectively.

MIN and MAX

Handy more readable macros for determining minima and maxima. Note that the macro performs a `cmp` which subtracts `src_b` from `src_a` (discarding the results) in order to set the flags to be interpreted by the following `csel`.

Thank you to u/TNorthover for nudge to add the cmp directly into the macro.

Signature:

```
MIN     src_a, src_b, dest
```

The smaller of `src_a` and `src_b` is put into `dest`.

Signature:

```
MAX     src_a, src_b, dest
```

The larger of `src_a` and `src_b` is put into `dest`.

Loads and Stores

GLD_PTR

Loads the address of a label and then *dereferences* it where, on Apple the label is in the global space and on Linux is a relatively close label.

Signature:

```
GLD_PTR     xreg, label
```

When this macro finishes, the specified x register contains what 64 bit value lives at the specified label.

GLD_ADDR

Loads the address of the label into the specified x register. No dereferencing takes place. On Apple machines, the label will be found in the global space.

Signature:

```
GLD_ADDR    xreg, label
```

When this macro completes, the address of the label is in the x register.

LLD_ADDR

Similar to `GLD_ADDR` this macro loads the address of a "local" label.

Signature:

```
LLD_ADDR xreg, label
```

When this macro completes, the address of the label is in the x register.

LLD_DBL

Signature:

```
LLD_DBL xreg, dreg, label
```

When this macro completes, a double that lives at the specified local label will sit in the specified double register.

Note: No underscore is prepended.

See this sample program for an example.

LLD_FLT

Signature:

```
LLD_FLT xreg, sreg, label
```

When this macro completes, a float that lives at the specified local label will sit in the specified single precision register.

Note: No underscore is prepended.

See this sample program for an example.

Mark a label as global

Makes a label available externally.

Signature:

```
GLABEL label
```

An underscore is prepended.

Calling CRT functions

If you create your own function without an underscore, just call it as usual.

If you need to call a function such as those found in the C runtime library, use this macro in this way:

```
CRT     strlen
```

An underscore is prepended on the Mac.

Declaring `main()`

Put `MAIN` on a line by itself. Notice there is no colon.

An underscore is prepended on the Mac.

`errno`

The externally defined `errno` is accessed via a CRT function which isn't seen when coding in C and C++. The function is named differently on Mac versus Linux. To get the address of `errno` use:

```
ERRNO_ADDR
```

This macro makes the correct CRT call and leaves the address of `errno` in `x0`.

AASCIZ

```
AASCIZ    label, string
```

This macro invokes `.asciz` with the string set to `string` and the label set to `label`. In addition, this macro ensures that the string begins on a 4-byte-aligned boundary.