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. There are macros to adjust for this.

There are some exceptions 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? Apple.

Macros of general use

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.

`MIN  x0, x1, x2`

resolves to:

`csel  x2, x0, x1, GT` putting the minimum of `x0` and `x1` into `x2`.

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.

Apple version:

```
.macro  GLD_PTR     xreg, label     // Dereference a global *
        adrp        \xreg, _\label@GOTPAGE
        ldr         \xreg, [\xreg, _\label@GOTPAGEOFF]
.endm
```

Linux version:

```
.macro  GLD_PTR     xreg, label     // Dereference a global *
        ldr         \xreg, =\label
        ldr         \xreg, [\xreg]
.endm
```

LLD_ADDR

Load the value of a "local" label.

Apple version:

```
.macro  LLD_ADDR xreg, label        // Load a local address
        adrp    \xreg, \label@PAGE
        add     \xreg, \xreg, \label@PAGEOFF
.endm
```

Linux version:

```
.macro  LLD_ADDR xreg, label
        ldr         \xreg, =\label
.endm
```

Extern a global label

Makes a label available externally.

Apple version:

```
.macro GLABEL label
        .global _\label
.endm
```

Linux version:

```
.macro GLABEL label
        .global \label
.endm
```

For example:

```
    GLABEL  main
```

Calling 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 in this way:

```
    CRT     strlen
```

Declaring `main()`

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