Option

*# optimization*

OPT = -Og

Tối ưa trong quá trình biên dịch :

Vd: có –Og : size(led.elf) = 14e8, –Og : size(led.elf) = 1b14

CPU = -mcpu=cortex-m3

Lựa chọn cpu:

VD :

stm32f103 ~ -mcpu=cortext-m3

stm32f407 ~ -mcpu=cortext-m4

arm-none-eabi-objcopy

Tạo file object, binary:

arm-none-eabi-objcopy –O ihex

arm-none-eabi-objcopy –O binary –S

FPC : accelerate floating point computing

(gia tốc tính toán dấu phẩy động)

Kể từ khi tôi bắt đầu tiếp xúc với phát triền phần mền nhúng, gần như tất cả chúng là ARM Cortex M, cám ơn C đã giúp tôi không cần phải động đến assembly. Nhưng tôi cảm thấy tôi bỏ lỡ một số thứ. Đó là không thấy được vẽ đẹp của trinh biên dịch và CPU. Vì thế tôi đã quyết định để có một tuần buồn chán tìm kiếm thông tin. Thử nghiệm thực hành. Đi đến kết luận của cách khám phá CPU ARM, vẻ đẹp của kiến trúc, cũng như là C bí mật của trình biên dịch ngôn nhữ. Bởi vì tôi không thích phương pháp dạy của lớp học vi điều kiển trong lớp của tôi

1. Floating point storage

Con trỏ động được theo tiêu chuẩn IEEE 754 là được lưu trử trong máy tính. Một trường con trở thực ARM dựa theo việc triển khai tiêu chuẩn IEEE 754-1985.

IEEE 754 là chuẩn được chỉ định cái mà được lưu trữ dưới dạng của một con trỏ động có 3 thành phần.



Sign : dấu (1bit: 0~ positive, 1~ negative)

Exponent : phần nguyên

Fraction : phần thập phân

# Floating point support software library fplib

# ARM Cortex-M có 2 cách để tính toán dấu phẩy động trong bộ xử lí là : phần cứng, phần mền

Để bỏ qua FPU bộ xử lí của ARM cung cấp 1 thư viện phần mền Floating point support. Nó được sử dụng để tính toán floating point number : fplib  
fplib được cung cấp API cùng bắt đầu \_\_aeabi. Ví dụ

\_\_aeabi\_fadd： Calculate two float Type floating point number （float Occupy 4 Bytes ,32 position ）

\_\_aeabi\_dadd： Calculate two double Type floating point number （double Occupy 8 Bytes ,64 position ）

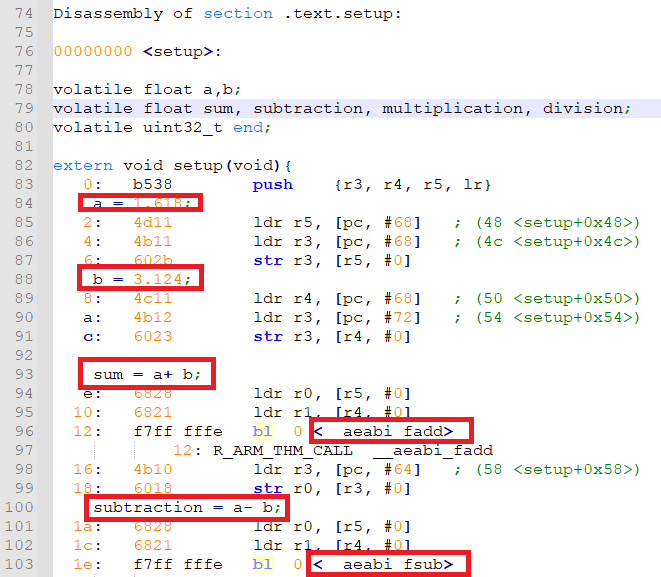
\_\_aeabi\_f2d：float Type to double type

\_\_aeabi\_d2f：double Type to float type

Bên cạnh dố, fplib thư viện cũng cunh cấp số dư. Căng bậc 2 là nhiều chức năng hoạt động con trở động khác. Nếu bạn cảm thấy thú vị hãy tham khảo tài liệu đến hết bài viết

* Khi thực hiện các phép toán số thực -> chuyển sang assembly -> MCU gọi thư viện fblib để xử lí

Xuất file asm : arm-none-eabi-objdump –Sx file.o > file.asm



1. Test code và mức độ tối ưa hoá

Được thực hiện bằng các option

-Og, -O0, -O1, -O2, -O3

Tối ưa hoá (giảm size của chương trình khi biên dịch), càng cao thì sự tối ưa hoá càng cao. Tối ưa hoá để giảm bộ nhớ cho vi điều kiển, nhưng sẽ có một số dòng lệnh cần thiết mà bị bỏ, dẫn đến code hoạt động sai

1. Lựa chọn FPU

|  |
| --- |
| -mfloat-abi=*name*  Specifies which floating-point ABI to use. Permissible values are: ‘soft’, ‘softfp’ and ‘hard’.  Specifying ‘soft’ causes GCC to generate output containing library calls for floating-point operations. ‘softfp’ allows the generation of code using hardware floating-point instructions, but still uses the soft-float calling conventions. ‘hard’ allows generation of floating-point instructions and uses FPU-specific calling conventions.  The default depends on the specific target configuration. Note that the hard-float and soft-float ABIs are not link-compatible; you must compile your entire program with the same ABI, and link with a compatible set of libraries. |

Chỉ định cái mà ABI (giao diện nhị phân ứng dụng) con trỏ động. Những cấu hình : “soft”, “softfp”, “hard”.

Việc chỉ định “soft” đã kiến GCC khởi tạo các lệnh thư viện chứa đầu ra cho sự hoạt động của con trở.

“softfp” cho phép sự khởi tạo của mã bằng cách sử dụng các lệnh dấu phẩy động phần cứng.

Nhưng vẫn sử dụng “soft-float” gọi là quy ước.

‘hard’ cho phép khởi tạo của hoạt động dấu phẩy động và sử dụng FPU cụ thể gọi là quy ước.

Sự định nghĩa phụ thuộc vào cấu hình mục tiêu riêng. Lưu ý rằng hard-float và soft-float ABIs không được tương thích với liên kế, bạn cần biên dịch toàn bộ chương trình của bạn cùng với ABI tượng tự và liên kết cùng với những bộ thư viện tương thích

-mfloat-abi = “soft”

-mfloat-abi = “softfp”

-mfloat-abi = “hard”

-mthumb

-marm

Select between generating code that executes in ARM and Thumb states. The default for most configurations is to generate code that executes in ARM state, but the default can be changed by configuring GCC with the --with-mode=*state* configure option.

You can also override the ARM and Thumb mode for each function by using the target("thumb") and target("arm") function attributes (see [ARM Function Attributes](https://gcc.gnu.org/onlinedocs/gcc/ARM-Function-Attributes.html#ARM-Function-Attributes)) or pragmas (see [Function Specific Option Pragmas](https://gcc.gnu.org/onlinedocs/gcc/Function-Specific-Option-Pragmas.html#Function-Specific-Option-Pragmas)).

Lựa chọn giữa việc khởi tạo mã cái mà được thực hiện trên trạng thái ARM và Thumb . Định nghĩa cho hầu hết các cấu hình là để khởi tạo mã cái mà thực thi trên trạng thái ARM nhưng định nghĩa có thể bị thay đổi bằng cấu hình GCC còng với cấu hình tuỳ chọn –with-mide=state

Bạn có thể ghi đề chế độ ARM và Thumb cho mỗi hàm bằng cách sử dụng “target (“thumb”)” và target(“arm”) những thuộc tính hàm (xem [ARM Function Attributes](https://gcc.gnu.org/onlinedocs/gcc/ARM-Function-Attributes.html#ARM-Function-Attributes) ) hoặc pragma (tên riêng của c, xem [Function Specific Option Pragmas](https://gcc.gnu.org/onlinedocs/gcc/Function-Specific-Option-Pragmas.html#Function-Specific-Option-Pragmas))