Проектиране на вградени системи
Лабораторно упражнение №4

Работа с кръслинкер LD. Преместване на изпълнимия код в RAM. Статични и динамични библиотеки.

==========

1. Вземете сорс файловете от директория **04_1** и ги копирайте в директорията на вашия проект от миналото упражнение. Заредете проекта в конфигурираната за работа с GDB развойна среда Eclipse (виж лаб. Упр. №3).
Добавете поле в Menuconfig за разрешаване на бутона. Съответната дефиниция трябва да се казва CONFIG_BUTTON_ENABLED.
Компилирайте и заредете проекта на демо платката. Ако всичко е минало успешно, диод LD3 ще мига, а при натискане на бутон “user” (синият) в терминала (/dev/ttyACM0, 9600-N-1) трябва да се изпише съобщението “Hello!\ n”.
ВНИМАНИЕ: Не забравяйте преди да натиснете бутона за дебъг да стартирате OpenOCD сървърът:

openocd -f board/stm32f7discovery.cfg

2. Разгледайте вътрешните (intrinsic) функции, поддържани от комерсиалният компилатор на ARM – armclang за NEON векторен модул за числа с плаваща запетая [1]. Спрямо inline Асемблерът, вътрешните функции са по-елегантен начин за достъпване на специфичен изчислителен ресурс на един микропроцесор – например операции с числа с плаваща запетая и двойна прецизност. Към настоящия момент на правенето на настоящото лабораторно упражнение GCC не поддържа всички вътрешни функции както комерсиалните ARM компилатори [2].

3. Променете програмата за изчисляване на средноквадратична стойност от лабораторното упражнение с GCC компилатора, така че да използва библиотечни функции за изчисляване на коренквадратен, умножение на квадрат и деление на числа с плаваща запетая.
За целта направете един отделен сорс и отделен хедърен файл, в който да поместите имплементацията на вашата функция. Включете хедърния файл “math.h” и добавете към линкера (в основния Makefile) аргумент за включване на стандартната библиотека за математика -l.

ВНИМАНИЕ: включете библиотеката в края на реда с командата за линкване (най-дясно).

Линкерът на GCC се нарича LD и аналогично на асемблера може да бъде извикван през GCC. По-важни аргументи на линкера са дадени в таблицата по-

долу [3] [4]. Забележете, че статичните библиотеки трябва да имат префикс “lib” и окончание “.a”, които обаче не се включват в аргумента.

Аргумент на LD	**Значение**
-L[директория-с-библиотеки]	Указва директория през GCC, където линкера ще търси обектовите файлове на библиотеката (не е задължителен, когато са стандартни библиотеки)
-l[име-на-статична-библиотека]	Указва името на обектовия файл (през GCC) на библиотеката. Не се пише префикса “lib” и вида на файла “.a”.
-T[име-на-линкерен-скрипт]	Включва конфигурационен файл за линкера (през GCC), в който са описани сегментите на паметта за дадената система. Трябва да е с разширение “.ld”.
-Wl,[линкерен-аргумент]	Предава аргумент на линкера LD през командата GCC.
-Wl,--start-group [няколко-линкерни-аргумента] -Wl,--end-group	Предава два или повече аргумента на линкера LD през командата GCC.
-M (или --print-map)	Създай файл с информация за това къде даден обектов файл ще се намира в паметта, кои библиотеки са включени, както и инициализацията на глобалните променливи (символи). Предава се през GCC.
--gc-sections	Изключи кода от обектовите файлове на функции, които не се използват никъде в останалта част от програмата. Тази опция **намалява значително обема** на изпълнимия файл и е валидна, само ако се предаде -ffunction-sections аргумента на компилатора.
--print-memory-usage	Принтирай използваната ROM/RAM памет.
-nostdlib	Търси директории на библиотеки единствено указани с аргумента -L. Директории, указани в линкерния скрипт ще бъдат игнорирани.
--print-multi-lib	Списък с всички поддържани набори

	от инструкции за конкретния порт на GCC. Предава се през GCC.

Примери:

Библиотека, която се казва m трябва да бъде записана във файл с име **libm.a** и да бъде включена към линкера по следния начин

arm-none-eabi-gcc -mcpu=cortex-m7 -nostartfiles --specs=nosys.specs -mfloat-abi=hard -mfpu=fpv5-d16 -Tscript.ld main.o -o main.axf **-lm**

Библиотека, която се казва gcc трябва да бъде записана във файл с име **libgcc.a** и да бъде включена към линкера по следния начин

arm-none-eabi-gcc -mcpu=cortex-m7 -nostartfiles --specs=nosys.specs -mfloat-abi=hard -mfpu=fpv5-d16 -Tscript.ld main.o -o main.axf **-lgcc**

и т.н.

Заредете променената програма и поставете точка на прекъсване на ред, в който има математическа функция (например sqrt()). Влезте в дебъг режим, отворете таб на дисасемблера (Window → Show view → Disassembly) и натиснете Resume (F8), за да започне да се изпълнява програмата. Когато дебъгерът спре на реда с точката на прекъсване, натиснете бутона “Instruction Stepping Mode”. С бутона “Step into” (F5) влезте в тялото на стандартната функция и разгледайте инструкциите ѝ. Ако библиотеката е била коректно port-ната, рано или късно ще се стигне до използване на инструкции за числа с плаваща запетая (например в sqrt( ) ще видите инструкцията “vsqrt.f64 d0, d0” аналогично на примера с inline Асемблера, който направихте в предишно упражнение).

4. Заместете командата за линкване в top-level Makefile-а ви от GCC на LD. При такова директно извикване е необходимо изрично да се укажат директориите на всички библиотеки, включително и стандартните. За случая може да помогне линкерният аргумент -Map=./debug/main.map:

```
arm-none-eabi-gcc ... -Wl,-Map=./debug/main.map ...
```

който ще принтира във файла main.map всички библиотеки, които GCC използва вътрешно, когато минава през фазите асемблиране и линкване (потърсете раздела “Archive member included to satisfy reference by file / symbol”).

ВНИМАНИЕ: редът на включване на библиотеките има значение. Включете първо -lm, след това -lc и накрая -lgcc.

5. Добавете динамично заделяне на памет във вашата програма. То изисква

отделен сегмент в RAM паметта, наречен хийп. Копирайте сорс файловете от директория **04_5** във вашия проект. Добавете target за sbrk.c файла и компилирайте програмата. Заредете я във Flash и стартирайте дебъг сесия. Изпълнете програмата веднъж, за да научите стартовия адрес на HEAP сегмента. Изпълнете програмата втори път, като преди да заделите и напълните съответния масив изберете в Eclipse → Window → Show View → Other → Memory и натиснете Enter. От бутона “Add Memory Monitor” въведете началния адрес на HEAP региона. Изпълнявайте програмата стъпка по стъпка и наблюдавайте как заделената памет се пълни. Обърнете внимание на **допълнителната служебна информация**, която malloc( ) добавя преди и след заделената памет.

ВНИМАНИЕ: за да използвате функцията malloc е необходимо да използвате **runtime библиотеката** newlib (-lc). Тази библиотека може да се окаже много голяма за някои микроконтролери, затова се използва нейна редуцирана версия като укажем на линкера --specs=nosys.specs. Тази версия съдържа системни функции с празни тела, или функции с минимална имплементация. В този случай malloc( ) е имплементирана, но една нейна функция липсва – sbrk( ), която трябва да бъде добавена от програмиста.

Разучете вкъщи библиотеката crt0.

6. Опитайте да включите статична библиотека към вашия проект. Преименувайте main.c на main_ORIG.c за по-късно използване. Вземете библиотеката libstm32cube_fw_f7_v1_16_0.a и прилежащите ѝ хедъри от директория **04_6**. Във main.c е написана кратка програма за тест на библиотеката. Основната ѝ функция дублира направеното досега, с тази разлика че тук се викат API функции от HAL библиотеката на ST Microelectronics. Използвайте таблицата от страница 2, за да включите към линкера правилните аргументи (в top-level Makefile-а ви). Включете следните директории в пътя на компилатора:

./hal_lib/Inc
./hal_lib/CMSIS/Device/ST/STM32F7xx/Include
./hal_lib/BSP/STM32F769I-Discovery
./hal_lib/CMSIS/Device/ST/STM32F7xx/Include
./hal_lib/CMSIS/DSP/Include
./hal_lib/CMSIS/Include
./hal_lib/CMSIS/Device/ST/STM32F7xx/Include

Добавете следните макроси във вашата програма, като ги дефинирате от командния ред /виж лаб. упр. за компилатора GCC/:

STM32
STM32F7

USE_HAL_DRIVER
STM32F769NIHx
DEBUG
STM32F769I_DISCO
__FPU_PRESENT
ARM_MATH_CM7
STM32F769xx

Накрая добавете аргумента за включване на библиотека. За да компилирате проекта си, временно сложете в коментар функциите:

```
void exti0_line_interrupt_handler(void)    /във файла button.c/
void systick_handler(void)                 /във файла startup.c/
```

Заредете програмата във флаш и стартирайте дебъг сесия. Тествайте потребителския бутон, двата светодиода и UART1 (9600-8N1). Запишете си размера на програмата, показван от arm-none-eabi-size.

7. Премахнете неизползваните функции (dead code) от вашия изпълним код. За целта на компилатора предайте аргумента:

-ffunction-sections

а на линкера:

--gc-sections

и понеже функциите на статичната библиотека се викат само в main.c, то въпросните аргументи може да бъдат добавени само за въпросния файл. Тези аргументи принуждават компилатора да постави всяка една функция в отделен регион от паметта, а след това линкера прави анализ кои функции се викат, и кои не, и неизползваните ги премахва. Това може да доведе до драстично намаляване на изпълнимия файл, когато се използва само част от библиотеката – сравнете данните на arm-none-eabi-size от настоящата и миналата задача (в случая се наблюдава редукция от около 67 %).

8. Направете статична библиотека от вашите сорс файлове – led.c, printf.c, uart.c и button.c. За целта върнете оригиналният ви файл main_ORIG.c (от задача 4.5) на мястото на HAL файла.
За да компилирате проекта си, извадете от коментар функциите:

```
void exti0_line_interrupt_handler(void)    /във файла button.c/
void systick_handler(void)                 /във файла startup.c/
```

Статичната библиотека представлява архив от обектови файлове. Вашите

обектови файлове, съответсващи на въпросните сорс файлове са готови и се намират в директория ./debug. За да ги архивирате използвайте командата [6]:

arm-none-eabi-ar -r [libиме-на-библиотека.a] [файл–1.o] [файл–2.o] … [файл–n.o]
Добавете отделен target в top-level Makefile-а ви само за създаване на библиотеката (например make static_lib да прави .a файла).
След това добавете току-що създадената библиотека на мястото на обектовите ви файлове в target-а за линкване на main.axf.

ВНИМАНИЕ: понеже printf зависи от вътрешната функция __aeabi_uldivmod( ), която се намира в libgcc.a, то включването на вашата статична библиотека трябва задължително да бъде преди -lgcc.

Заредете и тествайте програмата ви. Ако всичко е минало успешно, тя трябва да работи по същия начин като в задача 4.5. Дебъгването на функциите от библиотеката ще съдържа сорс кода, но ако искате да създадете комерсиална библиотека, то трябва да премахнете аргумента -g при компилацията на съответните сорс файлове преди да ги архивирате. В този случай дебъгерът ще покаже само Асемблерни инструкции без C код.

9. Програмата arm-none-eabi-size може да работи върху архиви. Тогава тя показва пълен списък с обектовите файлове от архива, както и заеманото място в паметта на контролера от всеки един файл.
Добавете отделен target, който да извиква arm-none-eabi-size с входен файл – вашата библиотека.

10. Динамичните библиотеки се различават от статичните по това, че кодът в тях не се добавя към изпълнимия файл на потребителската програма и по това, че адресите в обектовите файлове са условни и се разбират чак когато операционната система зареди библиотеките в паметта. Програмата от ОС, която прави това се нарича динамичен линкер (dynamic linker / dynamic loader). Динамичният линкер изчислява адресите при стартирането на програмата, което забавя малко изпълнението на самата програма, но пък позволява по-лесното ѝ ъпдейтване.
Пример за създаване на динамична библиотека във вградена система, която работи с Linux е (логнете се на самата система) [8]:

gcc -c -fpic my_library.c -o my_library.o
gcc -shared -o libmy_library.so my_library.o
gcc main.c -L[път/до/директорията/на/libmy_library.so] -o main.bin -lmy_library
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:[път/до/директорията/на/libmy_library.so]
./main

11. Разгледайте линкерния файл на вашата програма. Опитайте се да преместите изпълнението на кода от Flash в SRAM (т.е. да преместите .text сегмента). Адресите на инструкциите могат да бъдат видяни в Eclipse от Windows → Show view → Other → Disassembly.

Всяка инструкция в обектовия файл притежава два вида адреси [9]: LMA (load memory address) и VMA (virtual memory address). LMA е адресът, на който инструкцията ще бъде записана при програмирането на микроконтролера. VMA е адресът, от който инструкцията ще бъде изпълнена от микропроцесора.

* * *

[1] “ARM Neon Intrinsics – Reference for ACLE Q3”, ARM Ltd, 2020.
[2] https://gcc.gnu.org/onlinedocs/gcc/ARM-C-Language-Extensions-_0028ACLE_0029.html
[3] Gatliff W., “An Introduction to the GNU Compiler and Linker”, Embedded Systems Programming Magazine, 2002.
[4] https://www.rowleydownload.co.uk/arm/documentation/gnu/ld/Options.html
[5] https://sourceware.org/newlib (Docs секцията)
[6] https://sourceware.org/binutils/docs/binutils/ar.html
[7] “Dynamic Linking with the ARM Compiler toolchain”, Application note 242, ARM Ltd, 2010.
[8] https://www.cprogramming.com/tutorial/shared-libraries-linux-gcc.html
[9] S. Chamberlain, I. Taylor, “The GNU Linker”, Free Software Foundation, Inc, 2009.

доц. д-р инж. Любомир Богданов, 2023 г.