Міністерство освіти і науки України

Національний технічний університет України

«Київський Політехнічний Інститут імені Ігоря Сікорського»

Кафедра конструювання електронно-обчислювальної апаратури

Звіт

З лабораторної роботи №2

по курсу «Обчислювальні та МП засоби в РЕА-2»

На тему: «Програмування на С і ASM»

Виконав:

cтудент III курсу

групи ДК-81

Шунь .О.

Перевірив:

доц. Корнєв В.П.

cт. в. Бондаренко Н.О.

Київ – 2021

**Завдання**

Створити проект у IDE Keil μVision5 та програму, що виконує:

1. Копіювання рядка символів у другий рядок.

2. У другому рядку малі літери замінити на велікі, а цифри на пробіл.

**Теоретична частина**

У даній лабораторній роботі розглядається написання програми на мові С і водночас на мові асемблера. Таке поєднання має свої переваги, а саме: є можливість писати максимально оптимізований код програми, що може бути необхідним у критично важливих випадках; можливо реалізовувати функції, що не можуть бути написаними на С, тобто функцій, що безпосередньо маніпулюють вмістом стека або використовують спеціальні команди, які не можуть бути отримані за допомогою стандартних конструкцій мови. Також можна віддати перевагу написанні коду на асемблері при обмежених обсягах пам'яті.

Можливо по-різному реалізовувати структуру програми з двома мовами:

1. Використання вбудованого асемблера.

2. Виклик з програми на С функції або процедури, написаної на асемблері і реалізованої в окремому файлі.

3. Виклик з асемблерної програми процедури або функції, написаної на С.

При цьому передача і повернення параметрів у викликану функцію відбувається наступним чином:

1. Вхідні параметри передаються в регістрах R0-R3. Якщо ж обсяг переданих даних більше 4 \* 32 біт, то вони передаються через R0-R3 та стек.

2. Значення, що повертається, зазвичай зберігається в R0, а якщо це значення займає 2\*32 біта, то в R0-R1.

3. Регістри R0-R3 можуть змінюватися в функції, тоді як вміст R4-R11, R12 і LR слід зберігати при вході в підпрограму і відновлювати при виході з неї. Зазвичай для цього використовують стек.

**Структура програми**

Розглянемо структуру написаної програми. На рис. 1 зображена блок-схема програми. Розроблена програма має послідовну структуру з «пустим» безкінечним циклом наприкінці, написана з використанням вбудованого асемблера (inline assembly).

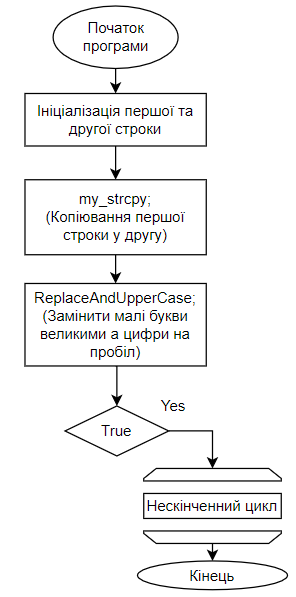


Рис. 1. Блок-схема програми

Процедура запуску програми винесена в окремий asm-файл mystartup.s:

1. Stack\_Size EQU 0x00000400
2. AREA STACK**,** NOINIT**,** READWRITE**,** ALIGN**=**3
3. Stack\_Mem SPACE Stack\_Size
4. \_\_initial\_sp
5. AREA RESET**,** DATA**,** **READONLY**
6. EXPORT \_\_Vectors
7. EXPORT Reset\_Handler **[**WEAK**]**
8. \_\_Vectors
9. DCD \_\_initial\_sp ; Top of Stack
10. DCD Reset\_Handler ; Reset Handler
11. \_\_Vectors\_End
12. AREA **|.text|** **,** CODE**,** **READONLY**
13. EXTERN main
14. Reset\_Handler PROC
16. B main
18. ENDP

21. END

Це так звана «преамбула», аналогічна як у першій лабораторній роботі.

У іншому файлі – main.c описана вся інша частина програми. Розглянемо основну функцію main, написану на С.

1. int main**(**void**)**
2. **{**
3. const char a**[]** **=** "zZ0!\*90Aa"**;**
4. char b**[**100**];**
6. my\_strcpy**(**a**,** b**);**
7. ReplaceAndUpperCase**(**b**);**
9. **while** **(**1**);**
10. **}**

Тут ініціалізується константна строка «а», задана в даному випадку набором символів «zZ0!\*90Aa». Далі ініціалізується строка «b», в яку буде скопійована строка «а». Відбуваєтсься виконання процедур копіювання строки та заміни символів. Наприкінці програма залишається у нескінченному циклі.

Розглянемо вищезгадані процедури детальніше.

Процедура копіювання строки:

1. \_\_asm void my\_strcpy**(**const char **\***src**,** char **\***dst**)**
2. {
3. **loop**
4. **LDRB** r2**,** **[**r0**]** **//** Завантажуємо у r2 байт з комірки пам'яті на яку показує r0 (src)
5. **ADDS** r0**,** #1 **//** Нарощуємо покажчик src на 1 **(**переводимо його на наступний байт**)** src
6. **STRB** r2**,** **[**r1**]** **//** Збережемо байт з r2 в комірці пам’яті**,** на яку вказує r1 **(**покажчик dst**)**
7. **ADDS**  r1**,** #1 **//** Нарощуємо покажчик dst
8. **CMP** r2**,** #0 **//** Чи дорівнював байт нулю**?** **/(**Кінець рядка src?**)**
9. **BNE** **loop** **//** Якщо ні **(**прапорець Z**=**0**/**"not equal" **-** "не дорівнює"**),** повторимо цикл
10. **BX** lr **//** Інакше **-** повернення з підпрограми
11. }

У дану процедуру передаються два параметри: const char **\***src**,** char **\***dst як посилання на перший символ першої та другої строки відповідно. Дані передаються через регістри r0 та r1.

Для копіювання знадобиться проміжний регістр, наприклад r2. Спочатку записуємо перший символ строки джерела (src) у проміжний регістр. Далі інкрементуємо показник src, оскільки повторно читати перший символ вже не потрібно. Тоді збережемо символ із проміжного регістру в комірці пам’яті, на яку вказує r1 (покажчик dst) – строку призначення. Далі інкрементуємо показник dst – надалі будемо записувати символ у наступний байт. В кінці перевіряємо, чи дорівнював байт нулю, що говорить про кінець строки. Якщо ні, то повторюємо всі дії спочатку, а якщо так, то виконуємо перехід по адресу, що знаходиться в регістрі повернення – lr. Адрес повернення автоматично записується в даний регістр при виклику процедури.

Процедура, що редагує строку:

1. \_\_asm void ReplaceAndUpperCase**(**char **\*str)**
2. {
3. cap\_loop
4. **LDRB** r1**,** **[**r0**]** **//** Завантажимо у r1 байт з комірки пам'яті, на яку показує r0 (\*str)
5. **//** **(**r0 **-** покажчик на рядок**,** переданий як аргумент функції**)**
6. **CMP** r1**,** #0 **//** Чи дорівнює байт нулю**?**
7. **BEQ** exit **//** Якщо так**,** то повернення з підпрограми
9. **CMP** r1**,** #'0' **//** Порівняємо байт з кодом символу "0" в ASCII таблиці
10. **BGE** higherThanZero **//** Якщо більше або дорівнює**,** перейдемо до перевірки чи це цифра
11. **B** cap\_skip **//** Інакше байт не потрапив у діапазон цифр і літер**,** пропускаємо
12. higherThanZero
13. **CMP** r1**,** #'9' **//** Порівняємо байт з кодом символу "9" в ASCII таблиці
14. **BLS** lowerThanNine **//** Якщо менше або рівне**,** перейдемо до заміни цифри на пробіл
15. **CMP** r1**,** #'a' **//** Інакше це може бути буква. Порівнюємо байт з кодом символу "а" в ASCII таблиці
16. **BGE** higherThan\_a\_ **//** Якщо значення більше або дорівнює**,** перевіряємо чи це літера
17. **B** cap\_skip **//** Інакше байт не потрапив у діапазон літер**,** пропускаємо
18. lowerThanNine
19. **MOV** r1**,** #0x0020 **//** Замінюємо цифру на пробіл
20. **STRB** r1**,** **[**r0**]** **//** Редагуємо строку
21. **B** cap\_skip **//** Перехід до наступного байту
23. higherThan\_a\_
24. **CMP** r1**,** #'z' **//** Порівняємо байт з кодом символу "z" в ASCII таблиці
25. **BHI** cap\_skip **//** Якщо більше**,** то це не літера**,** пропускаємо
26. **SUBS** r1**,**#32 **//** Інакше віднімемо різницю 32 **(**0х20**),** щоб перевести в код великої літери
27. **STRB** r1**,** **[**r0**]** **//** Збережемо код великої літери в пам'яті
29. cap\_skip
30. **ADDS** r0**,** r0**,** #1 **//** Нарощуємо покажчик **str**
31. **B** cap\_loop **//** Повторюємо цикл
32. exit
33. **BX** lr **//** Повернення з підпрограми
34. }

Для роботи процедури знадобиться допоміжний регістр r1. Передача параметрів аналогічна до першої процедури. Але передається один аргумент – посилання на перший символ строки «b».

Спочатку завантажимо у допоміжний регістр перший символ строки і перевіримо його на нуль. Якщо він рівний нулю, тоді це кінець строки – виходимо із процедури. Якщо це не нуль, то порівняємо його з цифрою нуль, оскільки ця цифра має найменший код серед потрібних нам символів. Якщо код символу виявився меншим за код цифри нуль, то даний символ не може бути ні цифрою ні буквою, пропускаємо його, перейшовши по мітці cap\_skip. Тут просто інкрементуємо покажчик str та переходимо до наступного символу. Якщо ж код символу більший або рівний, ніж код цифри нуль, то переходимо по мітці higherThanZero і перевіряємо чи даний символ цифра, порівнюючи його з кодом найбільшої цифри – дев’ять. Якщо код символу, що перевіряємо, менший або рівний, ніж у цифри дев’ять, то це цифра. Переходимо по мітці lowerThanNine, замінюємо символ на пробіл, замінюємо відповідний символ у строці джерелі, переходимо до наступного символу. Якщо ж значення символу більше за код цифри 9, то перевіряємо чи більше або рівне це значення за код літери а. Якщо ні, то це не літера, пропускаємо. Якщо так, то переходимо на мітку higherThan\_a\_ і перевіряємо верхню границю – чи код даного символу більший за код літери z. Якщо так, то це не літера, пропусаємо. Інакще це літера, тоді віднімемо від коду даного символу десяткове 32 аби перевести літеру у верхній регістр, замінюємо відповідний символ у строці джерелі, переходимо до наступного символу.

**Дослідження виконання програми у Keil μVision5**

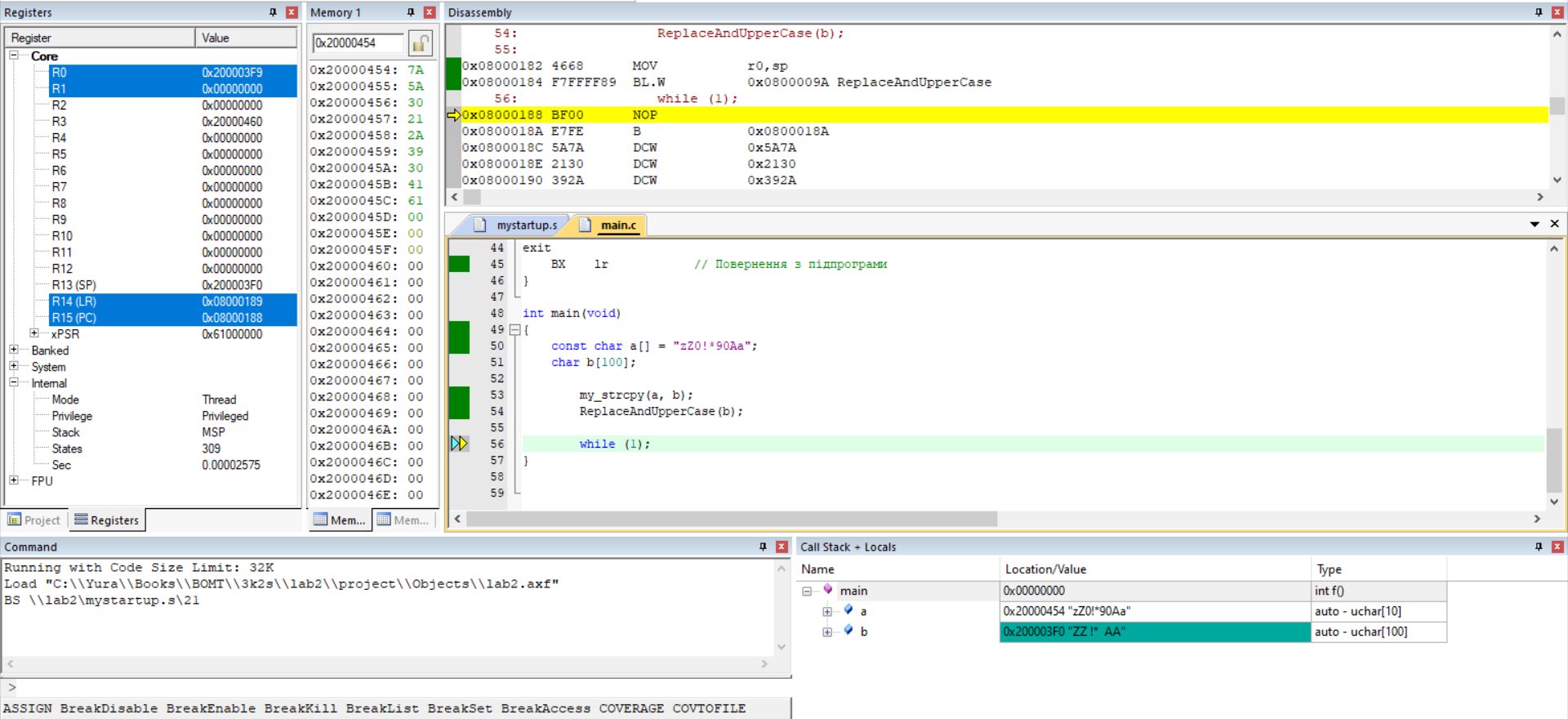
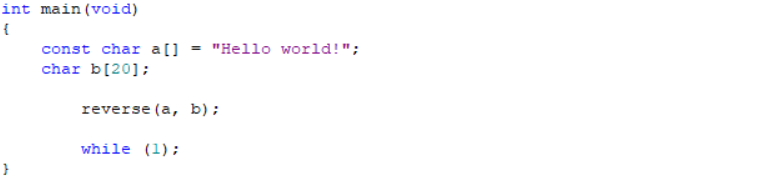
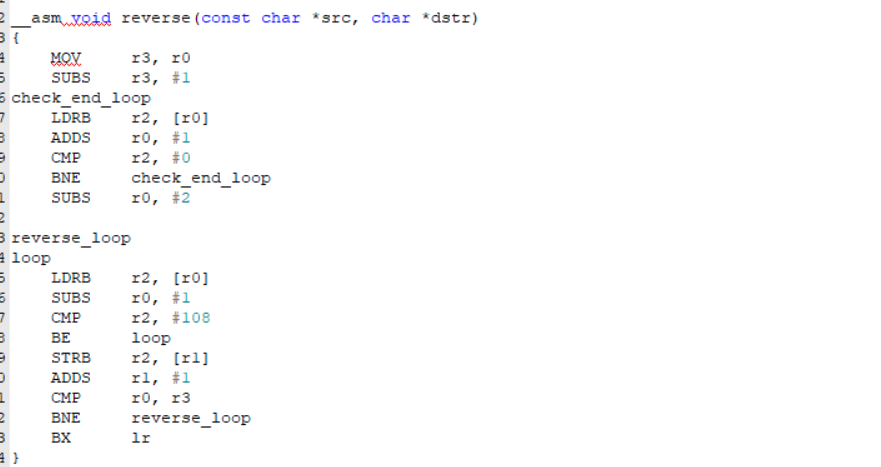
****

Рис. 2. Стан програми після виконання всіх команд.

В результаті виконання програми бачимо (рис. 2.), що строка «а» була скопійована у строку «b», яка була відредагована відповідно до завдання. Можемо бачити адресу в пам’яті, по якій знаходиться перший символ даних строк та їх вміст (написано у блоці Call Stack + Locals в стовпці Location/Value). У блоці Memory 1 видно як розміщується строка «а» в пам’яті.

**Текст програми**

****

**Висновок**

В результаті виконання даної лабораторної роботи я навчився писати програму одночасно на мові С та асемблера. Дізнався про можливі три техніки змішування коду С і асемблера та яким чином здійснюється передача параметрів в функцію, що викликається, і повернення результату її роботи в програму, яка її викликала.