

**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
ІМЕНІ ІГОРЯ СИКОРСЬКОГО»**

**КАФЕДРА КОНСТРУЮВАННЯ ЕЛЕКТРОННО-ОБЧИСЛЮВАЛЬНОЇ
АПРАТУРИ**

ЗВІТ

з лабораторної роботи № 2
по курсу «Обчислювальні та МП засоби в РЕА-2»

Виконав:
студент гр. ДК-82
Дмитрук О.О.

Перевірів:
ст. викладач
Бондаренко Н.О.

Завдання

1. Реалізувати ф-ю копіювання рядку в зворотньому напрямку.
2. Реалізувати ф-ю обчислення суми 32-бітних без знакових елементів масиву. Результат 64-бітне беззнакове число.
3. Реалізувати ф-ю обчислення 2-х знакових 32-бітних чисел. Результат 32-бітне число.

Теоретична частина

Розглянемо теорію, необхідну для виконання лабораторної роботи.

Так як ми будемо використовувати процедури, то для правильного використання процедур, потрібно дотримувати таких правил як:

1. Вхідні параметри передаються в регістрах R0-R3. Якщо ж обсяг переданих даних більше $4 * 32$ біт (важливо не кількість переданих, а їх загальний розмір), то вони передаються через R0-R3 + 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.
4. Значення SP має бути вирівняно за подвійним словом
5. Якщо к-сть вхідних параметрів більша за 4, в такому випадку, виконається наступне:
Завантажитись вміст параметрів в регістри, тобто 4 максимум за 1 ітерацію. Після завантажитись в стек за допомогою команди PUSH вмісти регістрів R0-R3 (якщо у нас 4 невідміщені параметри) , після завантажитись наступні параметри в регістри R0-R3.

Виконання роботи

Далі буде представлення реалізація ф-ї копіювання рядку в зворотньому напрямку.

```

#include "define.h"

__asm void strcpy_and_reverse(const char *src, char *dst, long array_size){

    PUSH {r4}
    MOV r4, r0
    ADDS r0, r0, r2
    SUBS r0, r0, #2

loop
    LDRB r3, [r0]
    STRB r3, [r1]
    CMP r0, r4
    BEQ end
    SUBS r0, r0, #1
    ADDS r1, r1, #1
    B loop

end
    POP {r4}
    BX lr
}

```

Рис.1. реалізація ф-ї копіювання рядку в зворотньому напрямку.

В якості аргументів передаються адреси 2 масивів, та їхній розмір. Тобто в регістр R0 поміститься адреса початку масиву джерела, з якого будуть братись дані. В регістр R1 поміститься адреса початку масиву приймача, в який будуть записуватись дані. В рег. R2 запишеться розмір.

Функція працює наступним чином:

Спочатку вона виконує запис в стек вмісту R4, щоб зберегти його вміст. Далі виконується обчислення адрес таким чином, щоб в R0 містилась адреса кінцевого елементу 1 масиву, це необхідно для реалізації реверсивного копіювання.

Далі переходимо в блок loop, в якому і виконується копіювання. Завантажується в регістр R3 вміст масиву Source починаючи з кінця, після завантажується вміст R3 по адресі вмісту регістру R1, тобто в масив Destination, починаючи спочатку. Потім виконується перехід на інший елемент масиву, в масиві Source це декрементування, а в Destination, інкрементування. Виконуємо даний цикл, поки вміст рег. R0 та R4 не будуть рівні (R4 містить адресу початку масиву Source)

Далі виконуємо команду Pop, щоб витягнути збережені дані зі стеку, і виконуємо перехід BX lr, щоб повернутись до викликаної команди.

name	Location/Value	Type
main	0x00000000	int f()
a	0x20000564 "Hello World"	auto - uchar[12]
b	0x20000558 "dlroW olleH"	auto - uchar[12]
v	0x20000544	auto - int[5]

Рис.2. Результат виконання ф-ї

Далі буде представлена реалізація ф-ї обчислення суми 32-бітних без знакових елементів масиву. Результат 64-бітне беззнакове число.

```

1      AREA    FUNCTION, CODE, READONLY
2      EXPORT u_sum
3
4      u_sum PROC
5
6          push {LR, r4, r5 , r6}
7
8          mov r4, #0
9          mov r5, #0
10         mov r6, #0
11
12     loop
13         LDR r3, [r0]
14         ADDS r4, r4, r3
15         ADC r5, #0
16         ADD r0, r0, #4
17         ADD r6, r6, #1
18         CMP r6, r1
19         BNE loop
20
21         mov r0, r4
22         mov r1, r5
23
24         pop {PC, r4, r5 , r6}
25
26     ENDP
27
28     END

```

Рис.3. Реалізація ф-ї обчислення суми 32-бітних без знакових елементів масиву. Результат 64-бітне беззнакове число.

В якості параметрів передається адреса початку масиву в регістр R0, та к-сть елементів масиву в R1. Результат поміщаємо в регістр R0 та R1. Ф-я працює наступним чином:

1. Зберігаємо вміст регістрів LR, R4,R5,R6

2. Обнуляємо регістри R4,R5,R6
3. Виконуємо блок loop, в якому обчислюється додавання. Регістр R4 виступає в ролі регістру акумулятору. Тобто він постійно буде накопичуватись значеннями масиву. Після команди ADDS (додавання з врахуванням знаку, необхідно щоб флажки встановлювались) виконуємо додавання з врахуванням флажка carry, Необхідно щоб виконати перенос у випадку переповнення R4. Далі виконується перехід на наступний елемент масиву, інкрементування counter-у, і перевірка чи пройшли весь масив.
4. У випадку проходження всього масиву, виконується запис результату в регістри R0 та R1. В R0 записується молодші 32 біти результату, в R1 , старші 32 біти.

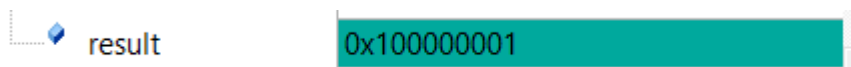
Результат виконання ф-ї:

```

unsigned int x[2] = {4294967295,2};

result = u_sum(x, sizeof(x)/4);

```



The screenshot shows a debugger window with a variable named 'result' highlighted. The value of 'result' is displayed as 0x1000000001 in a teal box.

Рис.4. Результат виконання ф-ї u_sum

Далі буде представлена реалізація ф-ї обчислення 2-х знакових 32-бітних чисел. Результат 32-бітне число.

```

1      AREA      FUNCTION, CODE, READONLY
2      EXPORT sum
3      sum PROC
4          MOV     r2, r0
5          MOV     r3, r1
6          EOR     r0, r0
7          EOR     r1, r1
8      loop_sign
9          LDR     r4, [r2]
10         ADDS    r0, r4
11         BVC     pass_ovf
12         ADDS    R1, R1, #1
13      pass_ovf
14         ADD     r2, #4
15         SUB     r3, #1
16         CMP     r3, #0
17         BNE     loop_sign
18         BX      lr
19     ENDP
20     END

```

Рис.5. реалізація ф-ї обчислення 2-х знакових 32-бітних чисел. Результат 32-бітне число.

Параметри передаються такі ж як і в попередній ф-ї. Ф-я працює наступним чином:

1. Виконуємо запис в регістри R2 та R3 вміст регістрів R0 ,R1 відповідно. Після обнуляємо R0 та R1. Результат будемо зразу ж записувати в них.

2. Виконуємо блок loop_sign, в якому виконується додавання знакових чисел. Далі у випадку успішного додавання, без переповнень , тобто без встановлення флажку 'V', виконуємо бранч на мітку pass_ovf, де виконуємо перехід на іншу строку і декремент лічильнику циклу.

Важливо зауважити, що у випадку переповнення, в регістр R1 запишеться 1, що оповістить нас, що відбулось переповнення. Також потрібно пам'ятати що флаг Carry відповідає за перенесення та за займ , при відніманні.

```
int z[2] = {2147483647,1};
```

result	0x180000000
--------	-------------

Рис.6. Результат знакового додавання

```
#include "define.h"

int main(void) {

    char a[12] = "Hello World";
    char b[12];
    int y[5] = {9, 11, 13, 15, 17};
    unsigned int x[2] = {4294967295,2};
    int z[2] = {2147483647,1};
    long long result;

    strcpy_and_reverse(a, b, sizeof(a));

    result = u_sum(x, sizeof(x)/4);
    result = sum(z, sizeof(z)/4);
    printf ("%i",&result);
    while(1);
}
```

Рис.7. Головна ф-я main