Addressing modes

Instructions

Make a document that contains answers to questions 1 and 2. Attach the document and the full source file of exercise 3 to your answer.

Exercise 1

Use Compiler Explorer (https://godbolt.org) to inspect how the code below is compiled into assembly language. Set the language to **C**, compiler to **armv7-a clang(trunk)** and paste the function below to the source window.

```
void tst(void)
{
    short a[8] = { 1, 2, 3, 4, 5, 6, 7, 8 };
    short *v = a;
    int k = 3;
    short temp = 0;
    temp = v[k];
    v[k] = v[k+1];
    v[k+1] = temp;
}
```

Study the code produced by the compiler and list all ARM assembly language data types and addressing modes used in tst(). Give an example of each addressing mode and data type by copying it from the disassembly window and name them.

Exercise 2

The table below shows contents of the memory.

Address	Value (hexadecimal)
9	BE
8	A5
7	DC
6	BA
5	A5
4	BE
3	BA
2	AB
1	AF
0	DE

a) Big endian processor reads a 32-bit number from address 2. Write the number in hexadecimal.
b) Little endian processor read a 16-bit number from address 8. Write the number in hexadecimal.
c) Big endian processor read a 32-bit number 0xBEA5BADC from memory. From which address the number was read?
d) Little endian processor reads an 8-bit number from address 1. Write the number in hexadecimal.
e) Big endian processor reads an 8-bit number from address 1. Write the number in hexadecimal.
f) Little endian processor writes a 32-bit number 0xCAFEADD1 to address 4. What is the content of each memory location after the write?

Exercise 3

M_0, M_1, M_2, and M_3 are memory addresses. Memory at each address contains a value to be used in computation.

Write a program that computes $M_0 = (M_0 + M_1 * M_1) * (M_3 + M_1 * M_1) + M_2$.

All ALU-operations are performed on registers. Data transfer is done between a register and a memory location. This is real assembly language that can (and must) be tested on RPI Pico before submitting your answer. Template CLion project can be found in the attached zip-file. **Attach your version of exercise3.c to your answer.**

Following instructions are available:

- LDR Rd, [Rn] – Read a word from memory address that is held in Rn to register Rd.
- STR Rn, [Rd] – Store a word from register Rn to memory address that is held in Rd.
- MOV Rd, Rn – Copy a word from register Rn to register Rd.
- ADD Rd, Rn, Rm - Add register Rn to register Rm. Result is stored in register Rd. (Rd = Rn + Rm).
- MUL Rd, Rn, Rd – Multiply register Rn with register Rd. Result is stored in register Rd. (Rd = Rn * Rd).

You can use registers R0 – R7. In the test program the memory addresses are passed to your program in registers R0 – R3.

R0 contains M_0, R1 contains M_1, etc. Note that in this case the values that are passed to your program in registers R0 – R3 are addresses (pointers). To deference a pointer (= to fetch the value that the pointer points to) you need to use LDR to copy the value from memory to another register.

You can overwrite the values in registers R0 – R7 if needed. You need to store the result of computation to the address M0, that was passed to your program in R0, so you need to keep this pointer safe throughout your computations. This time your program does not return a value because we store the result using a pointer.

Example:
LDR R7, [R0] // copies a word from memory address in register R0 to register R7.
ADDS R4, R7, R2 // R4 = R7 + R2

Function definition. Full template code is attached to the exercise:

```
__attribute__(( naked )) void asm_test(int *a, int *b, int *c, int *d)
{
  // write your code between push and pop instructions
  asm volatile
  (
              "push {r4, r5, r6, r7} \n" // do not remove

              "ldr r7, [r0] \n" // example assembly code - replace with your own code
              "adds r4, r7, r2 \n" // example assembly code replace with your own code
              // add more code here

              "pop {r4, r5, r6, r7} \n" // do not remove
              "bx lr \n" // do not remove
  );
}
```