Instruction set and datapath

General instructions

Prepare a single document that contains all your answers. Indicate clearly the question you are answering and follow any additional instructions given in the exercise in question.

M_0, M_1, M_2, and M_3 are memory locations that contain values to be used in the computation.

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

Exercise 1 and Exercise 2 give you details of which instruction set to use.

Try to minimize the number of instructions needed for computation. Remember that you need to store the result of the calculation at the end.

Exercise 1

Stack based architecture

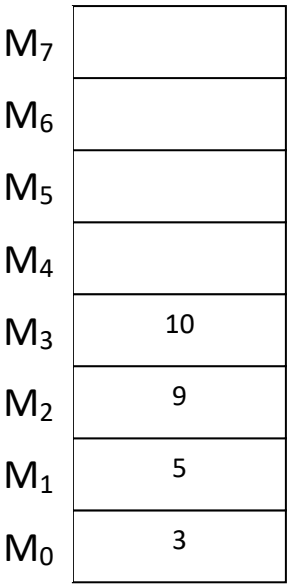
For this "processor" we don't have a simulator. Just write the code using your favorite text editor and **attach the code to your answer**.

All operands are implicit in a stack-based architecture. ALU instructions don't have to specify operands – they are popped from top of the stack and the result is pushed at the top of the stack. Note that you can leave your intermediate results in the stack to wait for further processing and push other values on top of them.

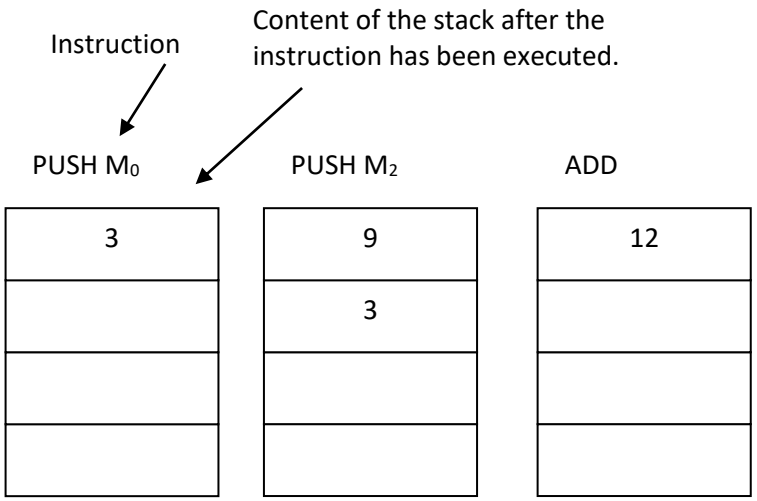
The following instructions are available:

- PUSH X –Pushes parameter (X) into the stack.
- POP X – Pops the top value from the stack and writes it to X.
- ADD – Pops two arguments from the stack, adds them and pushes result into the stack
- MUL – Pops two arguments from the stack, multiplies them and pushes result into the stack

Example of a stack-based program:



M_0, M_1, … M_N are values in the computer memory.



Exercise 2

Cortex-M3 assembly language

Perform the computation from exercise 1 using a subset of Cortex-M3 assembly language. The function that does the computation must be called asm_test(). The function takes the values of M0-M3 as parameters and the compiler places that parameters in registers R0 – R3 in the same order as they are passed. In this exercise you are going to write a real assembly language function that can (and must) be tested on LPCXpresso before submitting your answer. Use the attached exercise2.c as a starting point. **Attach the whole exercise2.c to your answer.**

All ALU-operations are performed on registers. Data transfer is done between a register and a memory location.

Following instructions are available – do not use any other instructions in this exercise:

- MOV Rd, Rn – Copy a word from register Rn to register Rd.
- ADDS 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).
  Note: the result overwrites one of the source registers!

You can use registers R0 – R7. In the LPCXpresso test program the values from M0 – M3 are passed to your program in registers R0 – R3. R0 contains value from M_0, R1 contains value from M_1, etc. You can overwrite the values in registers R0 – R7 if needed. Your assembly code must copy the result of computations to R0 before returning from the function.

Make changes only in function asm_test between the lines that are indicated by comments. Note that each assembly language line must be in double quotes and there must be a linefeed character at the end of each line before the closing double quote.

```
__attribute__(( naked )) int asm_test(int v0, int v1, int v2, int v3)
{
  // write your code between push and pop instructions
  asm volatile
  (
            "push {r4, r5, r6, r7} \n" // do not remove
            // do not add any code before this comment
            "mov r7, r3 \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

            // do not add any code after this comment
            "pop {r4, r5, r6, r7} \n" // do not remove
            "bx lr \n" // do not remove
  );
}

void fail() {
            print("Failed\n"); // set a break point here
            while(1);
}
void ok() {
            print("All ok\n"); // set a break point here
            while(1);
}
```

First argument goes to R0, second goes to R1 etc.

When testing your assembly code set a break point in the fail() and ok() functions to detect errors in the final test. Use instruction stepping mode for initial testing to inspect program behavior after each instruction. Your program works correctly if fail() is not called after any of the tests. Note that print() does not display the text in this configuration. We add that line only to be able set a break point on the line.

Exercise 3

Datapath

I recommend using Firefox for the following two exercises. Other browsers may not display all digits properly when you switch to binary mode.

Part A

Go to http://users.dickinson.edu/~braught/kands/KandS2/datapath.html and use the web-based simulator to calculate the sum of three numbers in registers R0, R1 and R2 into R3. Set the initial values of the three registers from your birthday. R0 = day, R1 = month, R2 = year. Values in R0, R1 and R2 may not be modified in your calculations. **Add screenshots that show the settings and the result of both intermediate calculation and the final result to your answer.**

Part B

Go to http://users.dickinson.edu/~braught/kands/KandS2/micromachine.html and use the web-based simulator to perform the following calculation:
M[7] = (M[0] | M[1]) – M[5]
M[x] means value of RAM at address x.
Initialize the RAM manually with the following values:
M[5] the first three digits of your student number
M[0] the last two digits of your student number.
M[1] the two middle digits (the ones that haven't been used already).

Attach a screenshot that shows the values in the memory after your program has been run and the microprogram memory to your anwswer. Adjust the screen size so that empty space is minimized in your screenshot.
The green arrow can be used to copy the current datapath settings into corresponding microprogam row. Note the computer stops either at the end of the program memory or when it encounters a microprogram line stating with xx.