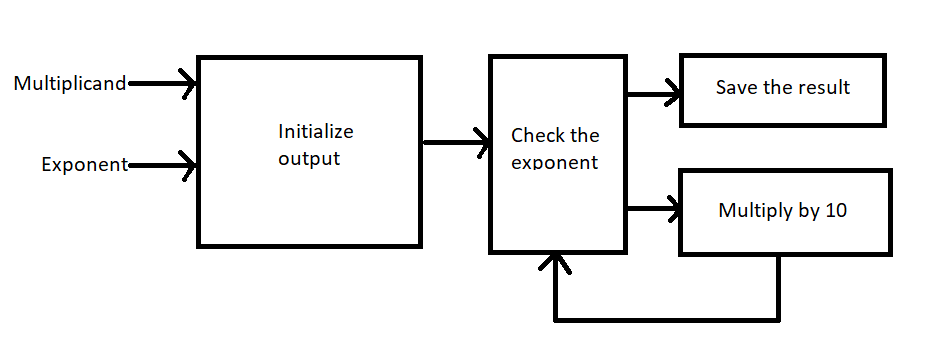
Microcontroller Based Systems

Author name: Muncan Simon

Task description:

* "Fast" multiplication of a 16 bit unsigned integer by an exponent of 10 (1, 10, 100, 1000, 10000).
* The number is in the internal memory, the exponent (0..4) is an input parameter of the subroutine.
* Fast multiplication means that we exploit the special properties of the exponent (e.g. 10=8+2, 100=64+32+2 etc.)
* Using ordinary multiplication is not an acceptable solution to this task.
* The result should be a 32 bit unsigned integer.
* Inputs: Multiplicand address (pointer), exponent of multiplier (value), result address (pointer)
* Output: Result starting at the given address

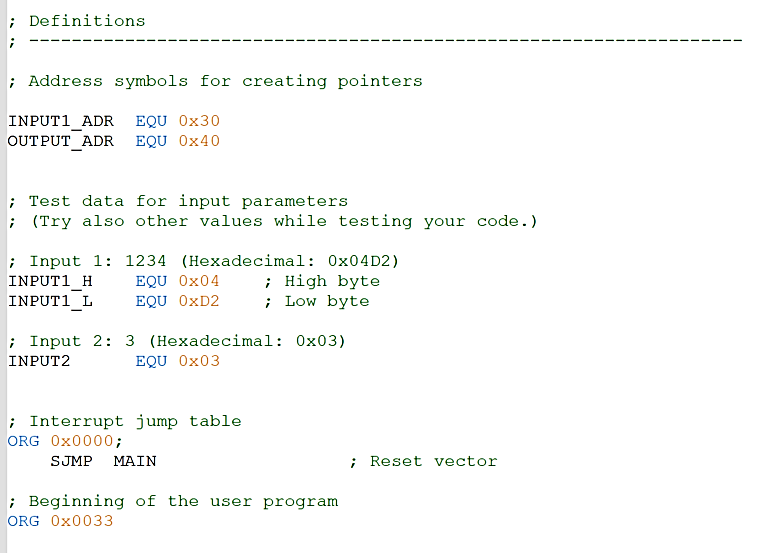
Block diagram



At the input we have multiplicand (16-bit) which we will multiply by 10 ^ exponent.

First we will save input number to the output address. Then we will check exponent and if its not zero we will multiply by 10. If its 0 we will save it to the output.

At the output we will have 32-bit number



Code Snippet 1.1

In the beginning, we have to define the constants that we will use in our program.

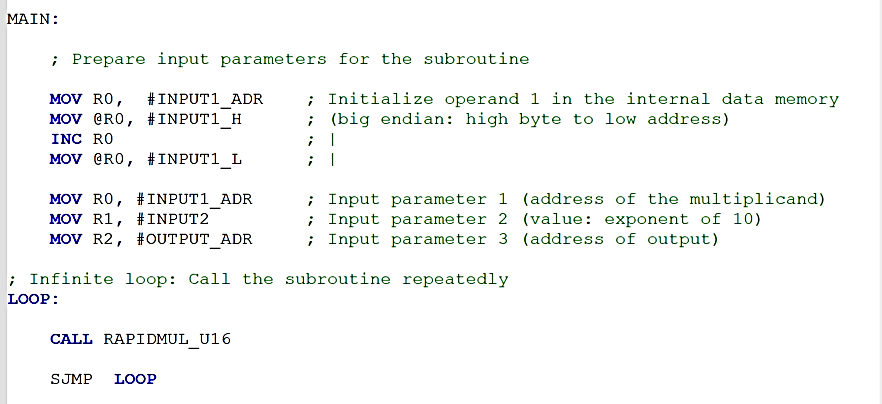
In the above code 1.1, we can see our input and output addresses.

Below address definition we have defined input test values.

Input 1 is the number that we want to multiply.

Input 2 is an exponent of the number 10, which we want to multiply with input 1.

After defining addresses and input values we should write our main program and loop.



Code Snippet 1.2

In the code above (1.2) we can see the main part and the loop part of our program.

In our main program, first we have to prepare parameters for the subroutine or we have to initialize our above defined constants.

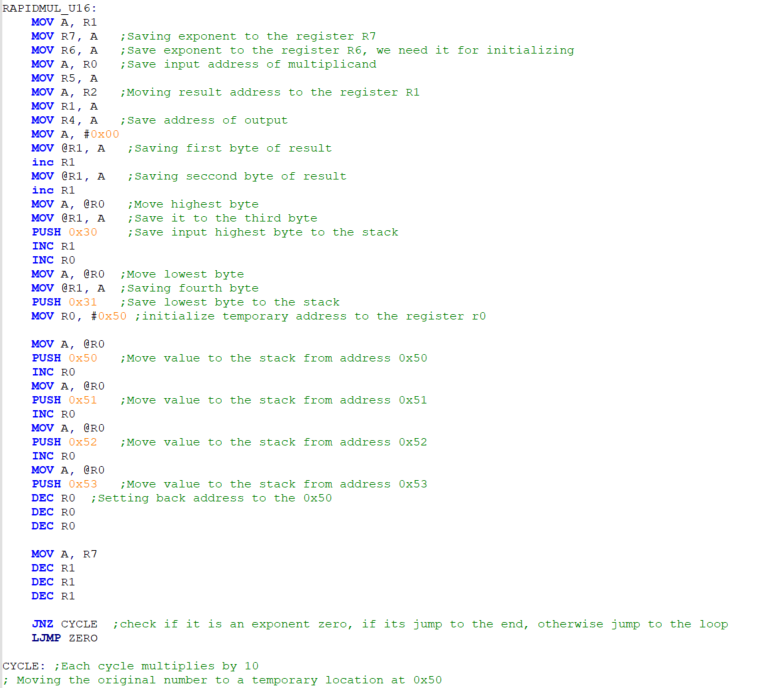
In the register R0 we put our input address, then at that address we initialized high bite of the given number.

Now we have to increment input address at 0x31 and put lower byte on that address.

After initializing lower byte we have to put our register R0 at the first given address 0x30.

Now we have to initialize our exponent of 10, then we have to initialize address of the beginning of our output.

In LOOP part of our program we will call subroutine RAPIDMUL\_U16. We will use function SJMP so we will jump at the beginning of a loop and again call our function.



Code Snippet 1.3

Finally our last and most important part of the program is subroutine RAPIDMUL\_U16.(Code 1.3)

First we will save our exponent to the register R7, because we will use that number later and we will overwrite register R1.

Next part of the code is initializing input number to the output address. And we are saving output address at register R1, so we can use @R1.

Now we will initialize temporary address at 0x50.

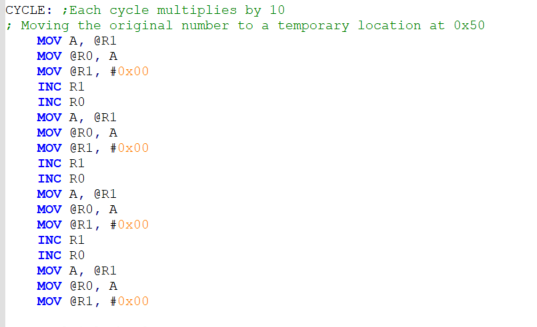
Because we cannot overwrite values from addresses 0x50-0x53 we will save values to the stack and return them at the end.

Then we will set output address at 0x40.

We will use register R6 to save exponent so we can return it later.

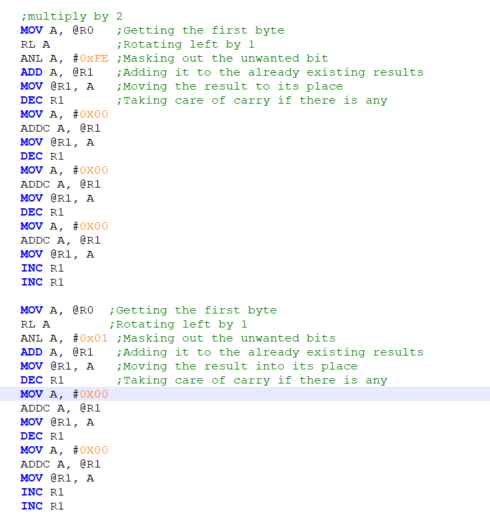
We will use instruction PUSH to save input bytes to the stack.

With fun JNZ we will check if exponent is not zero and if it’s not zero we will jump to the label CYCLE. If it’s zero we will jump to the label ZERO with fun LJMP.



Code Snippet 1.4

The code above is moving original number to the temporary address, because we will use number from a temp address for multiplication.

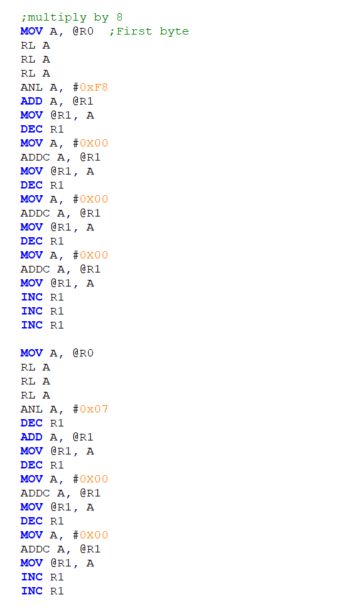


Code Snippet 1.5

The code above is multiplying input number with 2. Multiplying is done by left shifting by one. We use first lower bite from a temp address and we are adding it to the output. Then we should take care of carry if there is any and we will save it to the highest bytes.

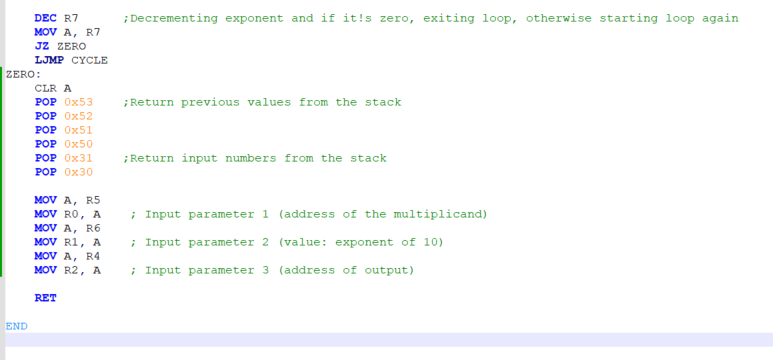
Then we should again use lowest byte and left shift it by one. We will use mask and will take just last digit and we will add it to the next byte.

We will use same logic to the other 3 bytes.



Code Snippet 1.6

We will use same logic and for multiplying by 8. But we will left shift 3 times instead of one. We will use same logic and for other 3 bytes.



Code Snippet 1.7

At the end we will decrement register R7, our exponent. Then we will move it the accumulator. Check it if it’s zero and if it’s we will jump to the label ZERO, if its not we will jump to the label CYCLE with long jump and we will do same code again.

At label ZERO we will return input values to the first place, because code will run again so we have to get same input parameters at the same place and we cannot overwrite others addresses that we used in our fun.

With fun RET we will return to the LOOP.