8051 ASSEMBLY LANGUAGE HOMEWORK DOCUMENTATION

Course name:	**Micro-controller Based Systems (VIAUAC06_EN)**
Student's Name:	**Das Anjan Kumar**
Neptun ID:	**D42DQA**
Lecturer(s):	**Kiss Domokos and Viktor Kovács**

‘I, Das Anjan Kumar, declare that this is my and only my own solution’

Task description:

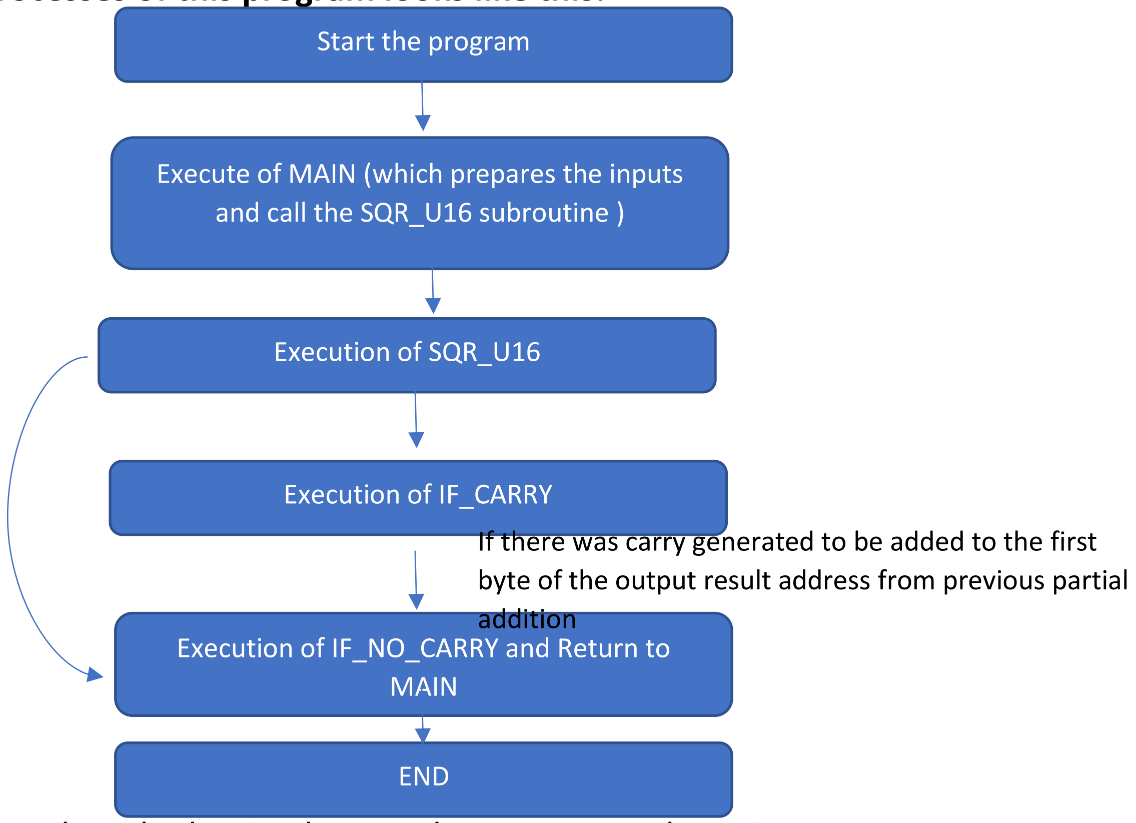
Square of a 16 bit unsigned integer passed through registers.

The result should be a 32 bit unsigned integer.

Input: The number in 2 registers, result address (pointer)

Output: Result starting at the given address

The main processes of this program looks like this:

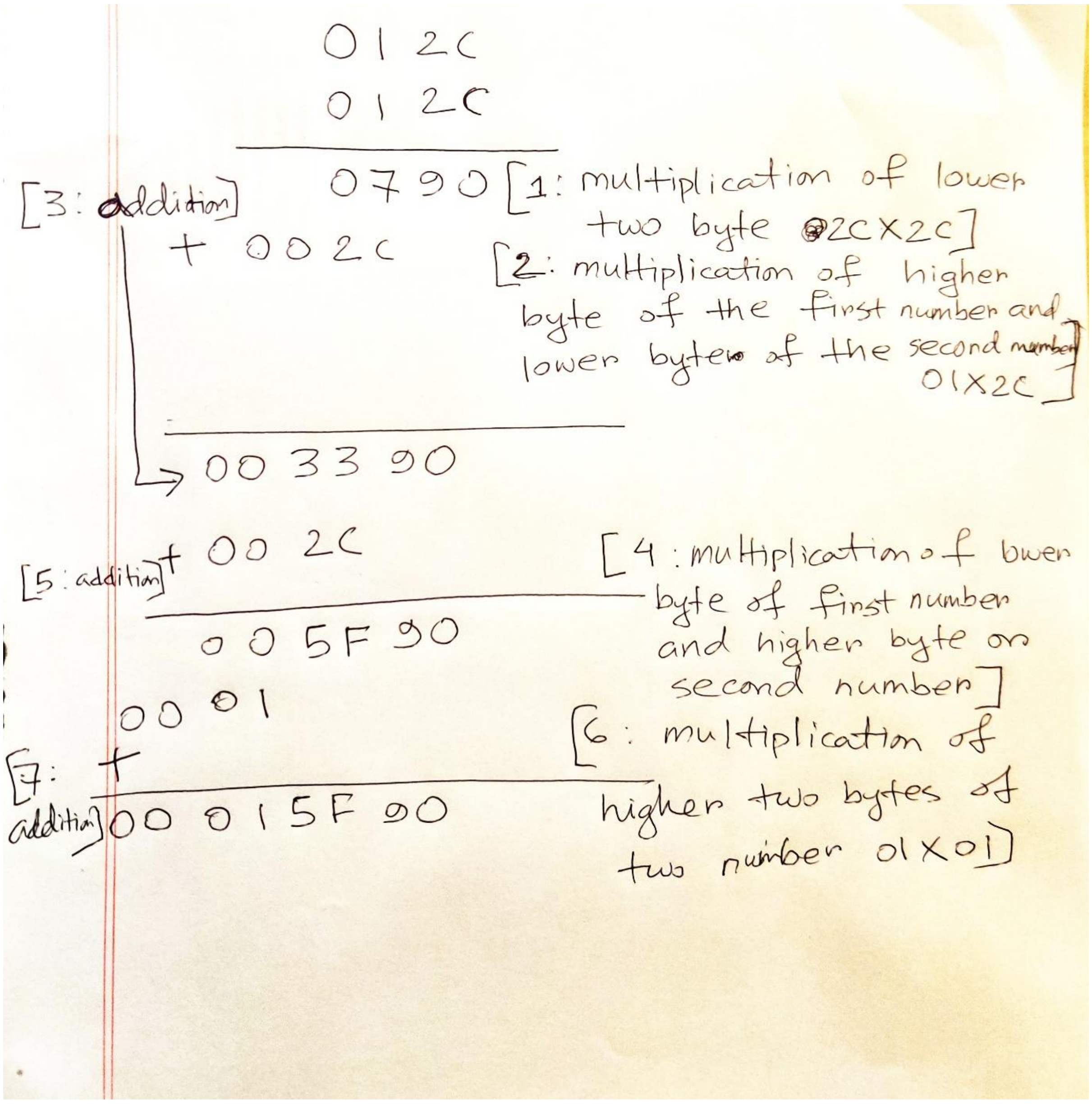

Start the program
Execute of MAIN (which prepares the inputs and call the SQR_U16 subroutine)
Execution of SQR_U16
Execution of IF_CARRY
If there was carry generated to be added to the first byte of the output result address from previous partial addition
Execution of IF_NO_CARRY and Return to MAIN
END

To solve the task we had to implement the SQR_U16 subroutine.

The higher byte of the 16 bit number is stored in register R0 and the lower byte is in the register R2 , and we have used the register R1 to point the address where the first byte of the result will be stored. Here the contents of R0, R2 is 0x01 and 0x2C which is the hexadecimal form of 300 . And register R1 will point to the output address 40H. We will save the consecutive bytes of the result in memory location 41H, 42H, 43H.

Execution and Description of the subroutine SQR_16:

Firstly, we need to understand the process of multiplication of two 16 bit number to understand this subroutine, because I have applied this method to find the result. Here is an example with step by step explanation of 16 bit multiplication:

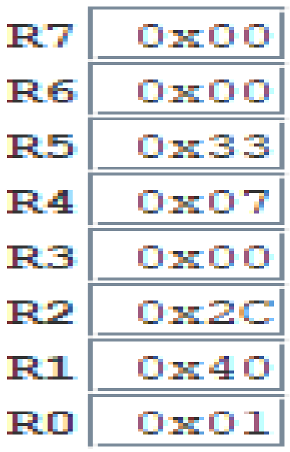


```
                 01 2C
                 01 2C
            -------------
                 07 90   [1: multiplication of lower
[3: addition]                two byte 2CX2C]
      +       00 2C      [2: multiplication of higher
                            byte of the first number and
                            lower byte of the second number
                                                   01X2C]
      ---------------------
      00 33 90
[5: addition] + 00 2C                  [4: multiplication of lower
      ---------------------                byte of first number
         00 5F 90                          and higher byte on
                                           second number]
   00 01                            [6: multiplication of
[7:  +                                  higher two bytes of
addition] ------------------            two number 01X01]
   00 01 5F 90
```

So,

- Firstly , it is important to know that we can perform multiplication in 8051 only in register A and B, that’s why we always have to load the content on these two registers.
- Also, before that we need to add three to register R1, because we will get the last byte of the result first and we need to store in the last output address byte, so will be decrease it as we store the results
- We have to load the content of R2 , that means the lower byte of the number in register A and B
- Then we will perform the multiplication [Step 1, in the above picture]of A and B , and as we know higher order byte of the result will be stored into B and lower order byte to register A
- Now we can shift the value of A to memory location 43H which is pointer by the register R1, because we will not need this for any further addition and this will be the last byte of our final result. Then we will decrease the value of R1 , because we need to store the later result byte
- But we need the higher byte of the multiplication to add with later multiplication product , so we save it to register R4
- Now we have to implement the second multiplication [Step 2]
- For that we have to load the lower byte again in register A and then we have to load the higher byte in register B and multiply
- Then, we will add [Step 3] the lower byte of the product to the content we saved in R4
- Now, we will move the content of to register R5 for further addition later with another partial multiplication
- The higher byte of the partial multiplication is added to the carry of lower byte if generated of partial products, after that we also need to clear the carry flag. Also need to store the partial in R6 for future addition
- The next multiplication [Step 4] will be carried away between lower byte of the first number and higher byte of the second number

- After getting the result of previous multiplication we have to again add partial products [Step 5]
- Also, now in register A we have the 2nd most lower order byte (the 3rd 8 bits) of the final result which will not be used anymore, so we store it into 42H]
- Then again for the higher order addition of partial products we have store then in Accumulator, after addition we clear the carry flag and we move the content of A to R7 for later addition
- The last multiplication is between the higher byte [Step 6] and then we again do the addition of the partial products [Step 7], first with the lower byte of the multiplication (which will be stored to 41H) and then the higher byte addition with carry if any
- Lastly, we store the result of final addition to address (40H) pointed by register R1
- Also, we have created two other subroutine, one is IF_CARRY, it will only be used if there is carry generated in the 5th step of addition , where this carry is needed to be added to the first result byte, that is why using this subroutine we will increment the value of the first result address output.
- And the other subroutine IF_NO_CARRY, is the normal continuation of the multiplication if there is not carry generated in the last two byte addition of the 5th step

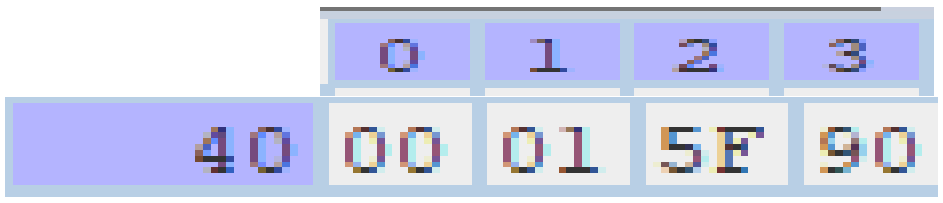


R7	0x00
R6	0x00
R5	0x33
R4	0x07
R3	0x00
R2	0x2C
R1	0x40
R0	0x01

Above we can see the value of the registers , here we have used every register except R3 . We have used the R0 and R2 register for storing the higher and lower byte of our 16 bit number. We used register R1 for storing the address where our result will start. And we used R4, R5, R6 and R7 for storing number that we used for later addition.

Simulation and Evaluation of the result:

We have used the EdSim51DI software for simulation and testing of our code.



	0	1	2	3
40	00	01	5F	90

As to our expectation, we can see that the result of the square of a 16 bit number (in our case 0x012C) , a 32 bit number, which is saved in the memory following big endian: high byte to low address . The result is 0x00015F90 , the first byte is stored in location 40H, second byte in 41H, third byte in 42H and last byte to 43H.

I have tested with other numbers as well and the program works perfectly fine with other numbers.

Reference: The knowledge of lectures was quite enough to solve this task. Though I have browsed a few websites to know exactly how 16 bit multiplication worked, but I have wrote the code myself following every step of 16 bit multiplication.