EE2016 - Experiment 4

Nishanth Senthil Kumar EE23B049
Gopalkrishna Ramanujan EE23B029
Arjun Krishnaswamy EE23B009

Group Number : 27

September 14, 2024

1 Task 1

1.1 Objective

To write a program to find the (i) maximum and (ii) minimum of 10 numbers stored in flash.

1.2 Code

```
.EQU no_of_elements=10  ;defining the number of elements

.dseg
.org 0x60 ;allocating data in the SRAM to store the flash memory.
numbers: .BYTE no_of_elements

.cseg
LDI ZL,LOW(NUM<<1)  ;Storing the 2 byte address of the list
LDI ZH,HIGH(NUM<<1) ;As it is in program memory, we have to multiply by 2

LDI R16,no_of_elements ;R16 will be our counter in the loop
LDI YL,numbers         ;YL points to the data we had allocated in SRAM
 ↪  earlier

storing: ;we are transferring data from flash to SRAM
LPM R18,Z+
ST Y+,R18
DEC R16
BRNE storing

LDI YL,numbers ;Resetting the pointer to the beginning of SRAM
LDI R16,no_of_elements ;R16 will be our counter
LDI R17,0 ;R17 will store the maximum value, setting it to be 0
LDI R18,255        ;R18 will store the minimum value, setting it to 255

max_min:
LD R19,Y+  ;storing the current value in R19
CP R19,R18 ;Comparing and checking if it is smaller than the minimum value
BRCC smaller
```

```
29 MOV R18,R19 ;moving it into R18 if so
30
31 smaller:
32 CP R17,R19  ;Comparing and checking if it is bigger than the maximum value
33 BRCC larger
34 MOV R17,R19 ;Moving it into R19,if so
35
36 larger:
37 DEC R16        ;Decrementing the counter
38
39 BRNE max_min ;Running the for loop
40 NOP
41
42 .org 0x0100
43 NUM :.db 200,20,3,5,4,6,4,3,10,21
```

1.3 Explanation

- I am first defining "no_of_elements" as a label, to make it easy to change the number of elements if required.
- After that I have defined a data segment to allocate data to a label called "numbers" in the SRAM, to store the data from flash into SRAM.
- I am starting the code segment, where I store the address of the list "NUM" in the Z pointer register, multiply the register address by 2 as program memory has a word as an address.
- After that, I make a loop in order to transfer the memory in flash into SRAM.
- I define R17 as the register that will store the maximum value, and r18 as the register that will store the minimum value.
- I loop through the SRAM memory with the help of Y pointer, and then find the maximum and minimum by using CP and BRCC.

1.4 Some values stored in registers

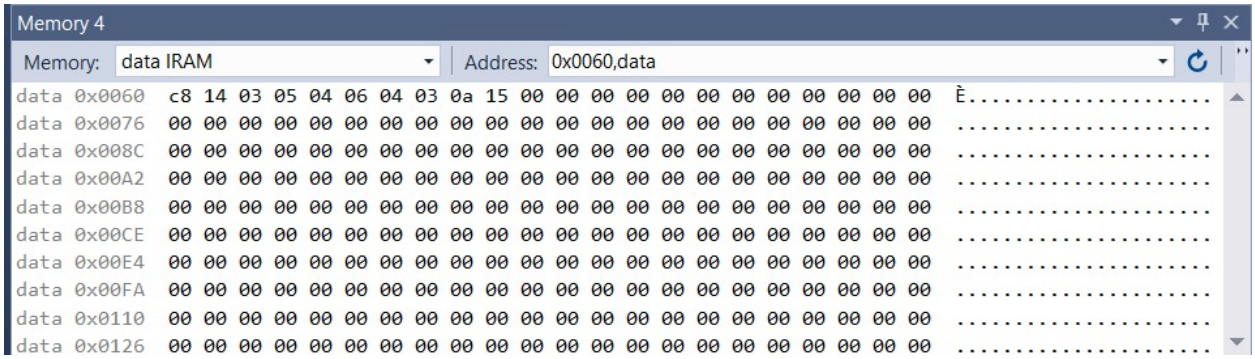


Figure 1: Values written to SRAM

R17	0xC8
R18	0x03

Figure 2: Final Results

2 Task 2

2.1 Objective

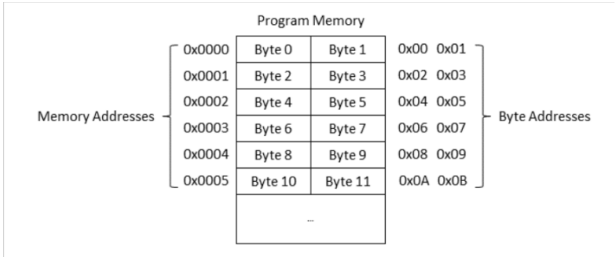
To load 10 numbers stored in flash memory and add them together

2.2 Code

```
.CSEG
.ORG 0x0000
    RJMP START          ; Reset vector jump to start

.ORG 0x0020            ; Define flash memory location for data
NUM:
    .DB 1, 2, 3, 4, 5, 6, 7, 8, 9, 10   ; Example data in flash memory

START:
    LDI ZH, HIGH(NUM<<1)   ; Load high byte of NUM address into ZH
    LDI ZL, LOW(NUM<<1)    ; Load low byte of NUM address into ZL
    LDI R16, 0             ; Clear R16 to store the sum
    LDI R18, 10            ; Counter for 10 numbers

LOOP:
    LPM R17, Z+            ; Load program memory into R17 and increment Z
    ADD R16, R17           ; Add the value in R17 to R16
    DEC R18                ; Decrement the counter
    BRNE LOOP              ; If counter is not zero, repeat the loop

DONE:
    NOP                    ; End of program (could add more here if needed)

```

2.3 Explanation

- **Understanding flash memory addressing:**



Flash memory is word addressed. Each word has 16 bits i.e 2 bytes. Thus assuming our bytes start from word location 0x0002, our first byte will have the byte address 0x04. It is clear that the byte address for the first byte in the word is equal to the corresponding

word address multiplied by 2. Sequentially incrementing this byte address by 1 gives us the byte addresses of the remaining numbers.

Registers ZL and ZH correspond to GPRs R30 and R31. Each stores one half of the word address transformed to the byte address. This is achieved using the left shift '¡¡1' which acts as a multiply by 2 operator.

Our pointer 'NUM' stores the word address of the first set of 2 numbers i.e 0x0002. LDI works with byte addresses thus LDI ZL=LOW(NUM¡¡1) stores the byte address '0x04' in ZL while ZH stores 0x00.

Thereafter in our instruction set, we use Z+ to increment our address to access data stored at byte locations 0x05, 0x06 etc.

- **Instruction Flow:** .ORG 0x0000 assigns 0x0000 as the memory address for the 'RJMP' which helps execute a particular block of code when the microcontroller starts.

  .ORG 0x0002 assigns the pointer NUM with the memory address 0x0002 which is the address of the first word in which data is stored.

  START: As explained, we create the pointers ZL and ZH ro access the flash data 'LDI R16, 0' initialises R16 with 0x00 and 'LDI R18, 10' creates our loop control counter.

  LOOP: 'LPM R17, Z+' loads the first number from program memory into R17 and then post increments Z to point to the next byte in program memory

  'ADD R16, R17' simply adds the contents of the two registers and stores the sum in R16, our accumulator.

  'DEC R18' decrements the content of R18 by 1

  'BRNE LOOP' is our loop control instruction

2.4 Images

Figure 3: The address of the first byte getting stored in the Z register.

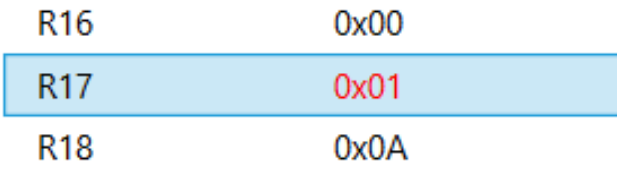


Figure 4: The first number getting loaded into R17

Figure 5: The final contents after the loop is over

3 Task 3

3.1 Objective

To sort 5 numbers stored in flash memory in arbitrary order and write the final results to data memory.

3.2 Code

```
.EQU array_size = 5          ; Define the size of the array as 5

.dseg
.org SRAM_START              ; Start of SRAM
sArr: .BYTE array_size       ; Reserve space in SRAM for 'array_size' bytes

.cseg
.ORG 0x0000                  ; Start of the code segment at address 0x0000

; Load the two-byte address of the NUM array (stored in flash memory) into
↪ the Z register
LDI ZL, LOW(NUM<<1)          ; Load the low byte of the NUM array's address
↪ into ZL
LDI ZH, HIGH(NUM<<1)         ; Load the high byte of the NUM array's address
↪ into ZH

; Initialize YL to point to the start of sArr in SRAM
LDI YL, sArr                 ; Load the SRAM address of sArr into YL
LDI R17, array_size          ; R17 is counter
storing:                     ; Label for the loop that transfers data from
↪ flash to SRAM
LPM R16, Z+                  ; Load a byte from flash memory
ST Y+, R16                   ; Store the byte from R16 into SRAM (sArr),
↪ increment Y
DEC R17                      ; Decrement the counter R17
BRNE storing

; Reset the counter and pointers for sorting
LDI R17, array_size-1        ; R17 is outer loop counter
LDI XL, sArr                 ; Load the start address of sArr into XL
LDI YL, sArr                 ; Load the start address of sArr into YL

loop1:                       ; Start of the outer loop (Bubble Sort)
    LDI R18, array_size-1    ; Load array_size-1 into R18 (inner loop
    ↪ counter)

    loop2:                   ; Start of the inner loop for comparing and
    ↪ swapping
        LD R20, Y+           ; Load the current element from sArr into R20,
        ↪ increment Y
        LD R21, Y            ; Load the next element into R21 (no increment)

        CP R21, R20          ; Compare the current and next elements (R21 with
        ↪ R20)
```

```
36          BRCC no_swap          ; If R21 >= R20, no need to swap, branch to
            ↪ no_swap
37
38          ; Swap R20 and R21 if R21 < R20
39          MOV R23, R20          ; Move R20 (current element) into R23 (temporary
            ↪ register)
40          MOV R20, R21          ; Move R21 (next element) into R20
41          MOV R21, R23          ; Move R23 (old current element) into R21
42
43      no_swap:                  ; No swap occurred, continue
44          DEC R18               ; Decrement the inner loop counter (R18)
45          ST X+, R20            ; Store the new value of R20 in sArr, increment
            ↪ X
46          ST X, R21             ; Store the new value of R21 in sArr
47
48          BRNE loop2            ; If R18 is not zero, branch to loop2 and
            ↪ continue
49
50      DEC R17                   ; Decrement the outer loop counter (R17)
51      LDI XL, sArr              ; Reset the X register to the start of sArr
52      LDI YL, sArr              ; Reset the Y register to the start of sArr
53
54      BRNE loop1                ; If R17 is not zero, branch to loop1 and
        ↪ continue the outer loop
55
56  ; Load the sorted values from SRAM for further use
57  LDS R20, 0x60                 ; Loading elements from SRAMinto R20
58  LDS R21, 0x61
59  LDS R22, 0x62
60  LDS R23, 0x63
61  LDS R24, 0x64
62  NOP
63
64  .org 0x0100                   ; Starting address in flash memory for NUM array
65  NUM: .db 0x05, 0x34, 0xFF, 0x11, 0X01 ; Define an array in flash memory
66
```

3.3 Explanation

- '.EQU' is an assembler directive that is used to assign a label a value, so I have defined the label "array_size" to be used throughout the code so that it can be easily modified to sort more number of numbers.
- '.DSEG' starts a data segment, it is required in order to allocate data in the SRAM, '.org' sets the location of the data segment to the beginning of the data allocated to SRAM(i.e SRAM_START), whose address is 0x60.
- The '.BYTE' directive allocated the given number of bytes to the label given to it, starting from what was set as the beginning address under the '.ORG' directive. So here, sArr is a label that points to address starting from 0x60 till 0x64, as array_size is 5.
- So basically, we have allocated 5 bytes in the SRAM to the label sArr.

- At the end of the code, I have written the input to the flash memory using the ".db" directive. The DB directive is used to reserve byte or bytes of memory locations in the available memory. Using '.org', we start allocating memory to the list NUM starting from the address 0x0100, which is where the Flash memory starts.
- The reason I am using '.org' everywhere is so that I know what data i have actually allotted to what, so I can access that data byte conveniently as I know the address.
- Now we start the code segment using ".cseg", and we have defined the starting address as 0x00.
- I have loaded the address of the list NUM to the Z pointer register, and since NUM is in the program memory(flash), each byte has a word address, so each number is stored in a 16 bit address, but every increment of the Z register will only traverse 1 bit, so we multiply the address by 2, every increment will traverse one word.
- However, in the case of data memory(like SRAM), the addresses are byte sized, so there is no need to multiply by 2, and there is no need of storing "LOW" and "HIGH" as the addresses are only byte sized.
- The Y register points to the data that I allocated in the SRAM, and I transfer the data from the flash memory to the data memory using a loop, decrementing R17.
- After storing, I implement the bubble sort algorithm. R17 and R18, are my counters, and I use two pointers to the same address, with the X and Y registers.
- The outer loop goes on for array_size times.
- In the inner loop, I load two consecutive numbers of the list to R20 and R21 using the Y pointer. I compare both of them using CP, and if the number stored in R20 is greater than R21, they get swapped, which is done with the help of the BRCC command.
- After that, I write the contents of R20 and R21(which would have swapped if R20 was greater) to the 2 address that were just accessed by the Y pointer (the Z pointer is incremented after Y, so it is still pointing to the previous address, and hence we can swap inplace in the SRAM). At the end of each iteration, both the pointers are set back to the beginning of the list.
- So when all the iterations are done, the list will be sorted in place in the SRAM in the addresses 0x60,0x61,0x62,0x63 and 0x64. I use the LDS command to directly load the data from there registers into R20,R21,R22,R23,R24.
- Note : If the number of elements are to be increased, an extra element can be added to the flash memory through the ".db" directive and the value of the arr_size can also be changed accordingly.

3.4 Contents in some Registers

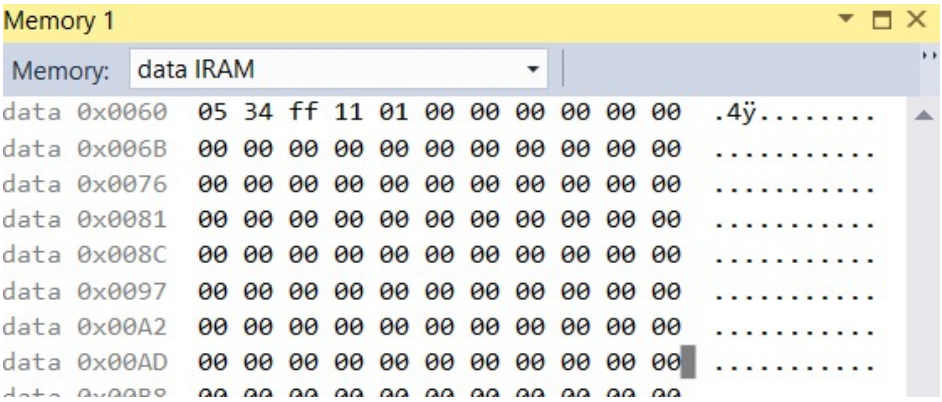


Figure 6: Data transferred from Flash to SRAM

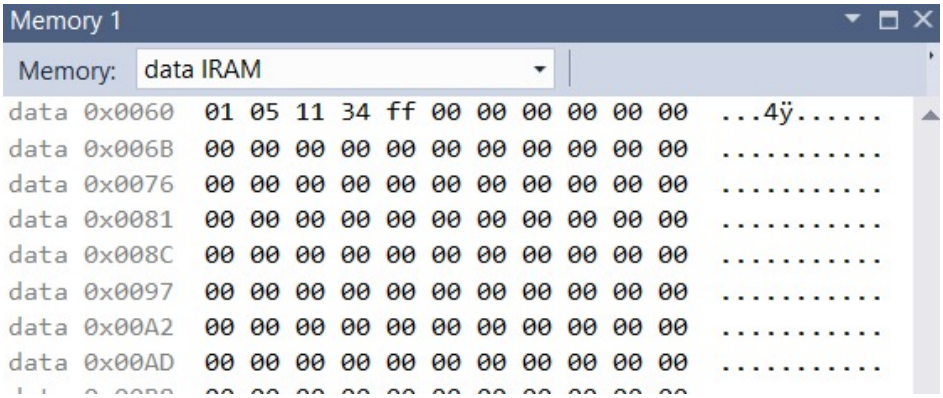


Figure 7: Data in SRAM after sorting

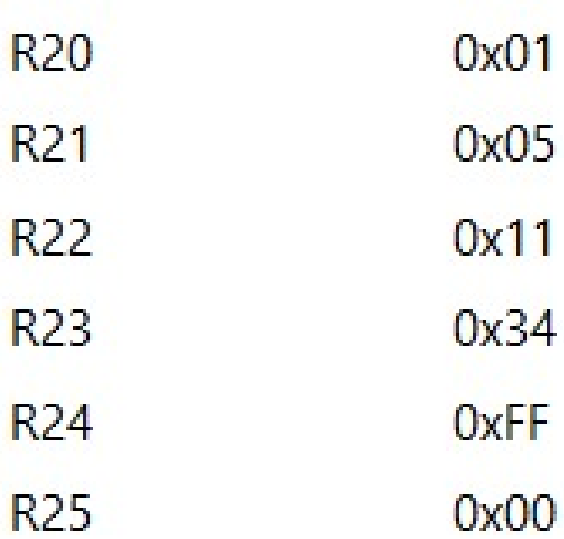


Figure 8: Data in registers,transferred from SRAM, after sorting (R20 to R24 store the relevant data

4 Programming and Debugging Experience

- We learnt how to use microchip studio, and how to use the debug command to test our program, and the various windows that give details on the values present in different registers.
- When we wrote only 5 bytes into flash memory (odd number of bytes), the assembler gave a "padding" warning. This warning just indicates that since each memory address in program memory can store 2 bytes, when you have a odd number of bytes, the assembler pads the one empty byte with 0's, and this wont affect the functionality of the code. This warning wont come up if we try to write even number of bytes into program memory.

5 References

- Slides uploaded on Moodle
- The AVR textbook by Mazidi
- AVR ASM Tutorials