**Diagram

Description automatically generated**

**8051 Assembly Assignment**

**Documentation**

**Microcontroller Based Systems**

**BMEVIAUAC06**

**Student: Khongmeng Kormoua**

**Neptun ID: I3MLPQ**

**Lecturers: Kiss Domokos**

**Viktor Kovács**

**Date: 15 October 2022**

**Budapest University of Technology and Economics**

**Department of Automation and Applied Informatics**

**2022**

**“I, Khongmeng Kormoua, declare that this is my and only my own solution”**

# Task description

* Search for the largest element in a number sequence (array) stored in the code memory.
* Every element is an 8-bit unsigned integer.
* Inputs: Start address of the array (pointer), number of elements
* Output: Largest element in a register

# Solution

* For this task I will work on the array in the code memory directly, the task does not require to copy the data from code memory to internal memory. That is why I implemented my solution based on the requirement of the task.

## Detailed Implementation of the subroutine “FIND\_MAX\_NUMBER”

* Inputs: - DPTR - start address of the array (first element of the array).
* R7 - number of elements.
* Output: R3 - maximum number from the array
* First, when enter the subroutine the subroutine will save all the contents of the memory to the stack using the PUSH instruction as shown below.

1. PUSH AR2
2. PUSH AR4
3. PUSH AR5
4. PUSH AR6
5. PUSH AR7
6. PUSH PCON
7. PUSH IE
8. PUSH IP
9. PUSH PSW
10. PUSH ACC
11. PUSH B
12. PUSH DPH
13. PUSH DPL

* After that the subroutine will read the first element from the array as below.

1. CLR A
2. MOVC A, @A+DPTR

* since this is the first element, this is automatically the biggest value for now since this is the only value that the subroutine has so far. Therefore, the subroutine will assign this value to R3 (which is the output) immediately

1. MOV R3, A

* Also, the first element has been loaded and taken care of so the pointer to the array “DPTR” and the remaining size of the array will be incremented and decremented respectively

1. INC DPTR
2. DEC R7

* Now all the preparation is ready for a subroutine loop to process the data if there is a data remain in the array to be processed. This loop will be called “SUBROUTINELOOP” and right after going inside the loop, the subroutine will read a new value from the array and save it to temporary variable which in this case is register R4

1. SUBROUTINELOOP:
2. CLR A
3. MOVC A, @A+DPTR
4. MOV R4, A

* Here comes to the trickiest part of the subroutine, how to compare and decide which one is the biggest. For this, I could use other compare instruction, but I just decided to use subtraction instruction “SUBB A, Rn”. Which will subtract Rn from A. if A > Rn then the result is saved in A. if A == Rn then result will be 0 and save to A. if A < Rn then overflow will happen, and the carry bit will be set. The idea to solve our task which is to find out which one is bigger is to put the biggest that we know so far to A then subtract the new element from A. if the carry bit is set. It means that the new element is bigger than the current biggest element. On the other hand, if the carry bit is not set, then it means that the new element is less than the current biggest element. For each case the subroutine will jump to a specific label accordingly

1. MOV A, R3
2. SUBB A, R4
3. JC NEWVALUEISBIGGER
4. JMP NEWVALUEISLESS

* Now after we can decide where to jump to, it is straight forward how to implement in each case. For the case that we value is bigger than we just copy the new value of R4 to R3 (the biggest element that we know so far). On the other case, when the new value is less than the current biggest value, there is nothing to do in other word just skip the assigning part from R3 to R4.

1. NEWVALUEISBIGGER:
2. MOV B, R4
3. MOV R3, B
4. NEWVALUEISLESS:

* From this point, since the subroutine has already taken care of the new element. The subroutine is going to Increment the DPTR, and Decrement R7. One more thing to do is to check if R7 is zero or not. If it is that means, there is nothing left in the array to process furthermore. If R7 is not zero yet, that means there is still some element left in the array. Therefore, the program should jump back to the subroutine loop and execute the whole process again.

1. INC DPTR
2. DJNZ R7, SUBROUTINELOOP

* When the processing of the subroutine loop is done, then one more thing to do before return to the main program is to load all register’s contents back from the stack. Note that it is necessary to load them back in the reverse order when the subroutine save them.

1. POP DPL
2. POP DPH
3. POP B
4. POP ACC
5. POP PSW
6. POP IP
7. POP IE
8. POP PCON
9. POP AR7
10. POP AR6
11. POP AR5
12. POP AR4
13. POP AR2
14. RET

## Flow chart of “FIND\_MAX\_NUM”

* Below is the flow chard diagram of the subroutine “FIND\_MAX\_NUMBER” regarding each execution and decision to fulfill the task description

Diagram

Description automatically generated

**Subroutine:** “FIND\_MAX\_NUM”

**Inputs:** R7 - length of the array

DPTR - start address of the array

**Output:** R3 - Max value from the array.

## Simulation result

**A register and memory contents right after assembly**

A picture containing table

Description automatically generated

The array contents is start at address 0070 as defined

At address 0000, there is only one instruction there which is SJMP instruction

Initially all the register value has no much meaning for us just to note that all of them are 0 in this case

All of our instructions begin at address 0010 as expected

Table

Description automatically generated with low confidence

The array pointer is currently pointing to the last element of the array

R3 now contain the max value from the array which is 0x55

**A register and memory contents after iterating through the array**

# Reference

* 8051 Instruction Set (Short Version).pdf
* Microcontroller Based System Lecture and Practice materials
* Personal consultation with professors
* www.win.tue.nl/~aeb/comp/8051/set8051.html