

Assignment Solution Report

Contents

Summary	2
About x86 emulator	2
Variables	2
Declaring the array.....	3
Finding the Sum	4
Finding the Positive Sum.....	5
Finding the Zero-Crossing Count (ZCC).....	6

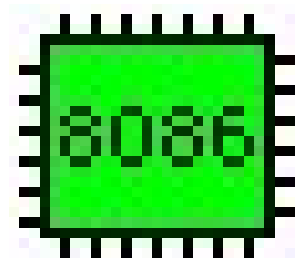
Summary

In this assignment, the x86 emulator software is used to develop an 8086 assembly program that performs different functionalities including the following:

- Declaring an array of integers
- Sum of integers
- Sum of Positive integers
- Zero-Crossing Count (ZCC)

About x86 emulator

In this assignment the 8086 Emulator tool was used to run the assembly code. EMU8086 - MICROPROCESSOR EMULATOR is a free emulator for multiple platforms. It provides its user with the ability to emulate old 8086 processors, which were used in Macintosh and Windows computers from the 1980s and early 1990s.



<https://emu8086-microprocessor-emulator.en.softonic.com/>

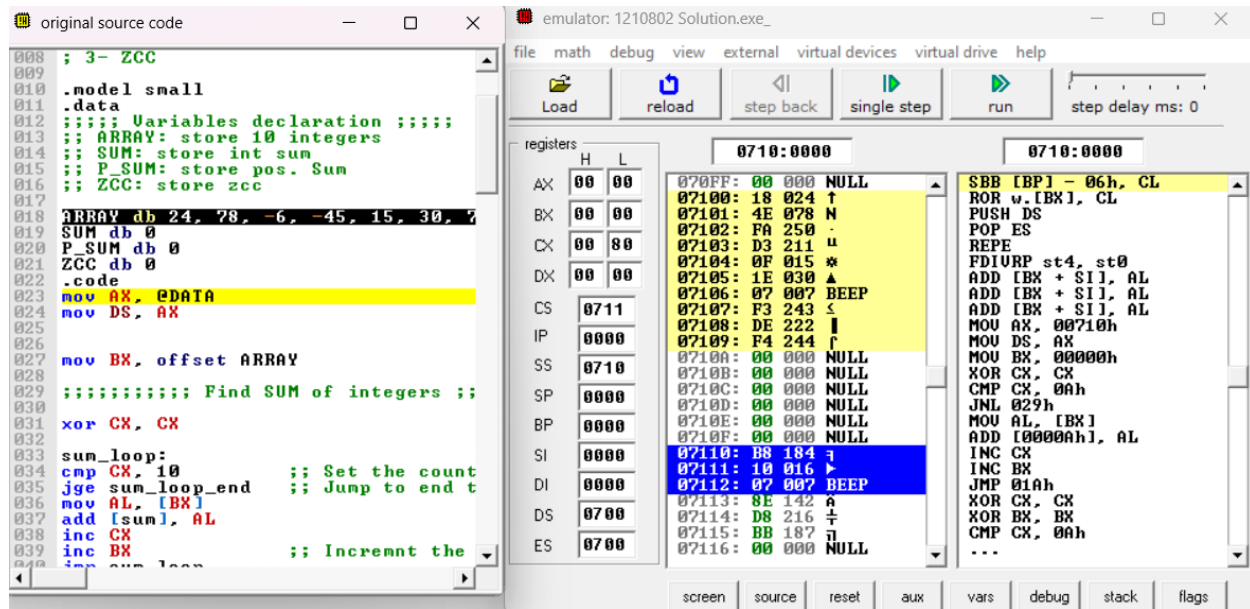
Variables

These are the variables used in the program:

- **ARRAY:** used to store the array of integers.
- **SUM:** used to store the sum of integers of **ARRAY**.
- **P_SUM:** used to store the sum of the **ARRAY** positive integers.
- **ZCC:** Used to store the Zero-Crossing Count of the **ARRAY** integers.

Declaring the array

First thing that this program does is declare an array of 10 signed integer numbers (8-bit) in the memory with their initial values. It was chosen to be half positive and half negative integers.



The screenshot displays an x86 emulator interface with two main windows. The left window, titled 'original source code', shows assembly code with several lines highlighted in yellow. The right window, titled 'emulator: 1210802 Solution.exe', shows the execution state, including registers, memory dump, and assembly instructions.

original source code

```
0008 ; 3- ZCC
0009
0010 .model small
0011 .data
0012 ;;;; Variables declaration ;;;;
0013 ;; ARRAY: store 10 integers
0014 ;; SUM: store int sum
0015 ;; P_SUM: store pos. Sum
0016 ;; ZCC: store zcc
0017
0018 ARRAY db 24, 78, -6, -45, 15, 30, 7
0019 SUM db 0
0020 P_SUM db 0
0021 ZCC db 0
0022 .code
0023 mov AX, @DATA
0024 mov DS, AX
0025
0026
0027 mov BX, offset ARRAY
0028
0029 ;;;;;;;;;; Find SUM of integers ;;
0030
0031 xor CX, CX
0032
0033 sum_loop:
0034 cmp CX, 10           ;; Set the count
0035 jge sum_loop_end    ;; Jump to end t
0036 mov AL, [BX]
0037 add [sum], AL
0038 inc CX
0039 inc BX               ;; Incremnt the
0040 jmp sum_loop
```

emulator: 1210802 Solution.exe

Registers:

	H	L
AX	00	00
BX	00	00
CX	00	80
DX	00	00
CS	0711	
IP	0000	
SS	0710	
SP	0000	
BP	0000	
SI	0000	
DI	0000	
DS	0700	
ES	0700	

Memory dump (0710:0000):

Address	Hex	Dec	Comment
070FF	00 000	NULL	
07100	18 024	↑	
07101	4E 078	N	
07102	FA 250	.	
07103	D3 211	u	
07104	0F 015	*	
07105	1E 030	▲	
07106	07 007	BEEP	
07107	F3 243	≤	
07108	DE 222	↓	
07109	F4 244	↑	
0710A	00 000	NULL	
0710B	00 000	NULL	
0710C	00 000	NULL	
0710D	00 000	NULL	
0710E	00 000	NULL	
0710F	00 000	NULL	
07110	B8 184	q	
07111	10 016	►	
07112	07 007	BEEP	
07113	8E 142	A	
07114	D8 216	†	
07115	BB 187	¶	
07116	00 000	NULL	

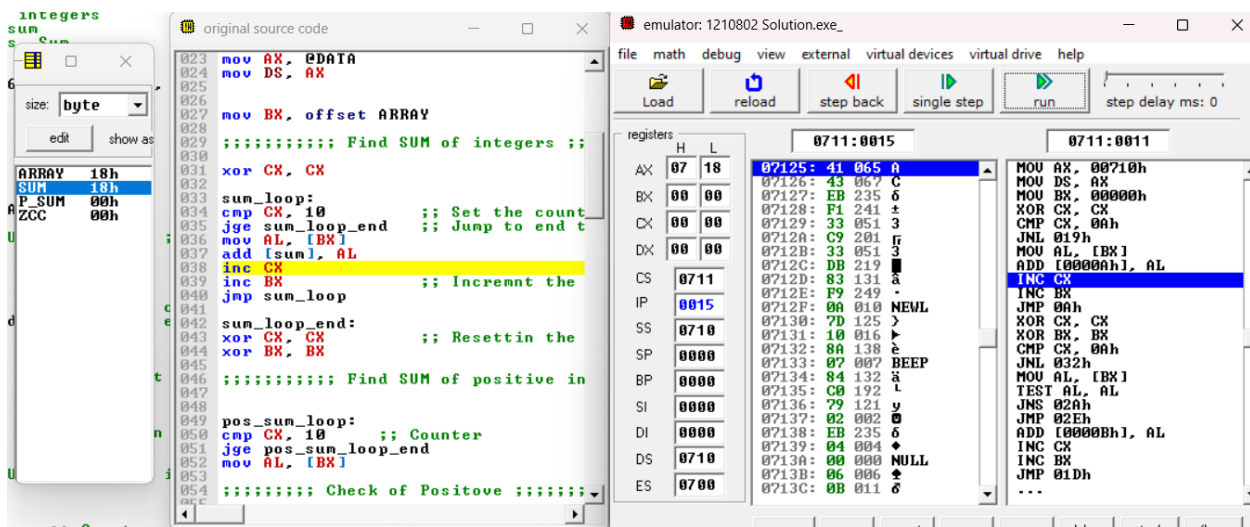
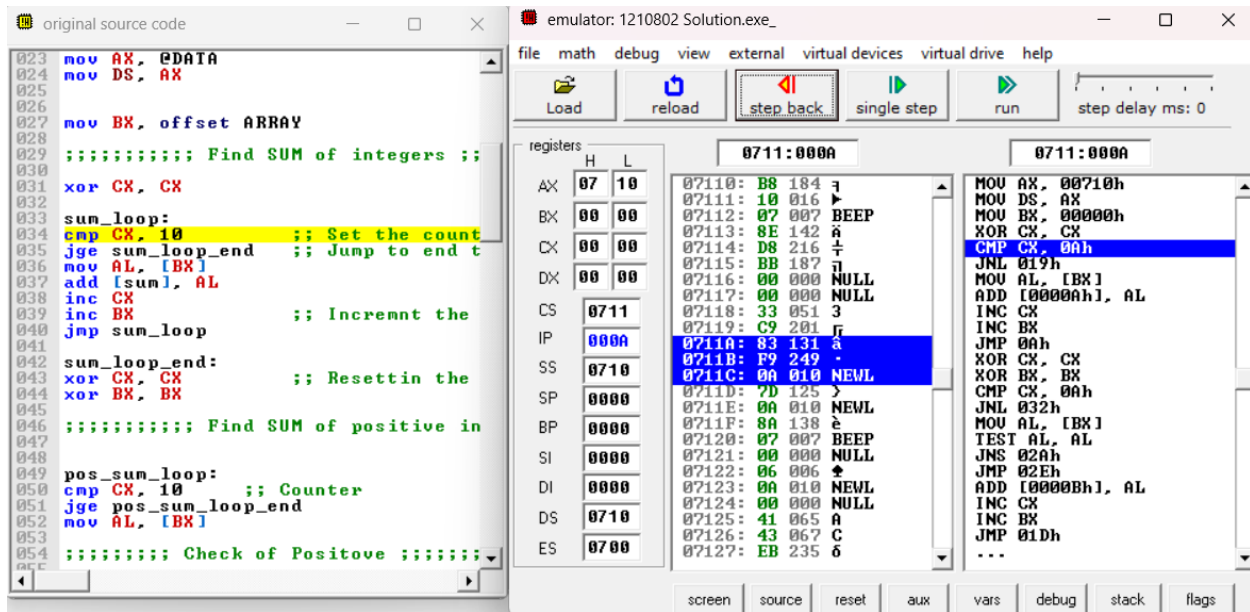
Assembly instructions (0710:0000):

```
SBB [BP] - 06h, CL
ROR w.[BX], CL
PUSH DS
POP ES
REPE
FDIURP st4, st0
ADD [BX + $1], AL
ADD [BX + $1], AL
ADD [BX + $1], AL
MOV AX, 00710h
MOV DS, AX
MOV BX, 00000h
XOR CX, CX
CMP CX, 0Ah
JNL 029h
MOV AL, [BX]
ADD [0000Ah], AL
INC CX
INC BX
XOR CX, CX
XOR BX, BX
CMP CX, 0Ah
...
```

It is seen on the labeled lines using the yellow color, that the array is declared for 10 elements.

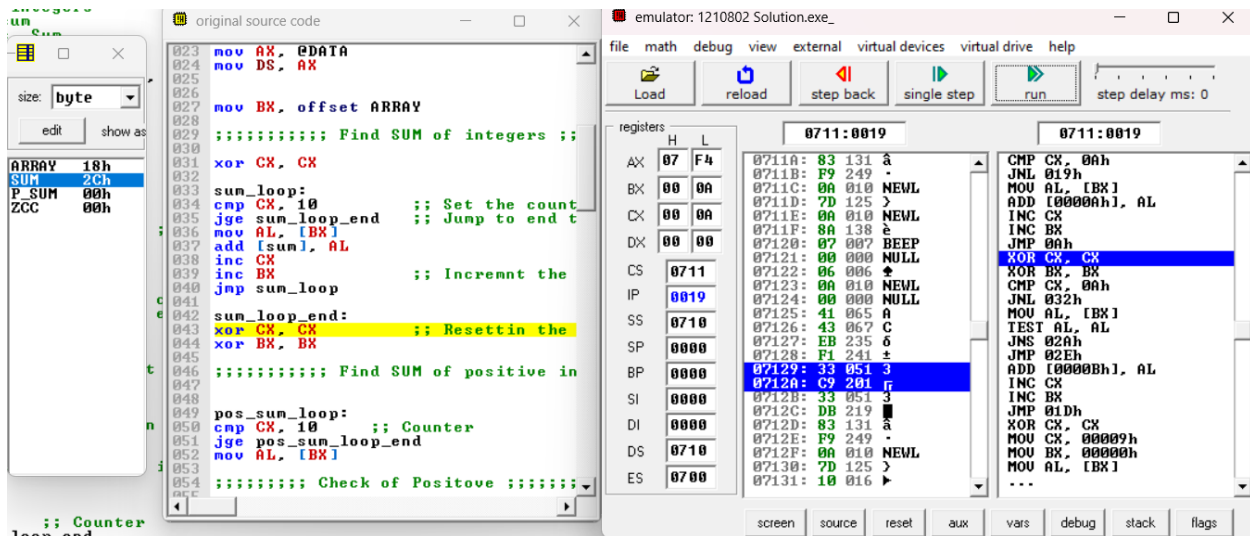
Finding the Sum

The second function that this program does is finding the arithmetic sum of all elements of the array and store it in the memory in a variable called **SUM**. This was done using the CX register and looping until it reaches 10.



It is noticed that after starting a single step of the sum loop that the SUM variable value has changed from 00h to 18h.

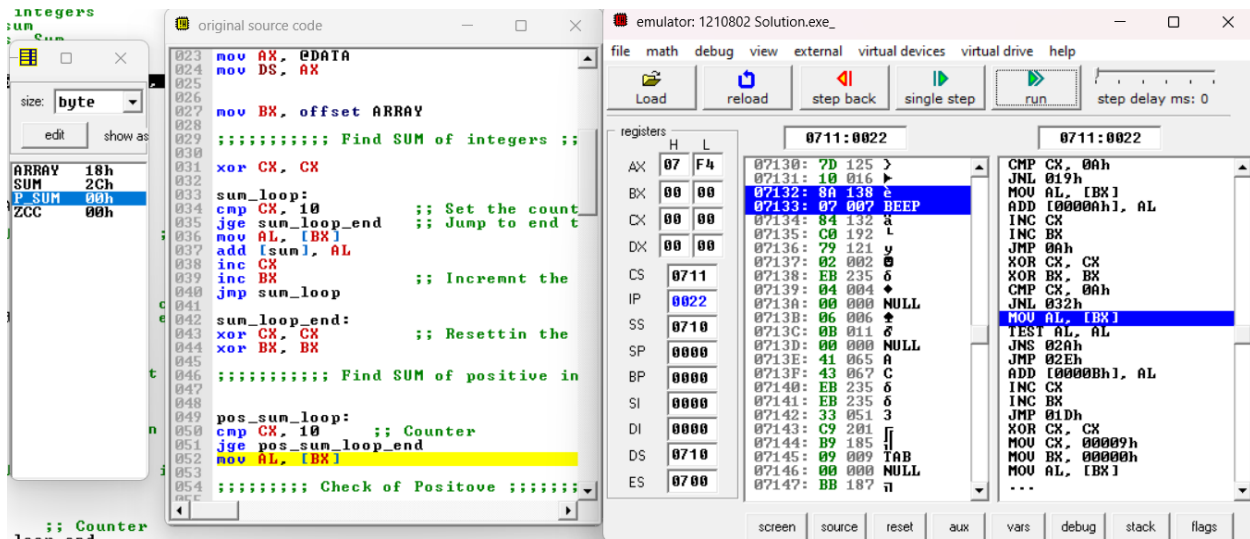
When the final element is added then we noticed the SUM variable value is 2Ch



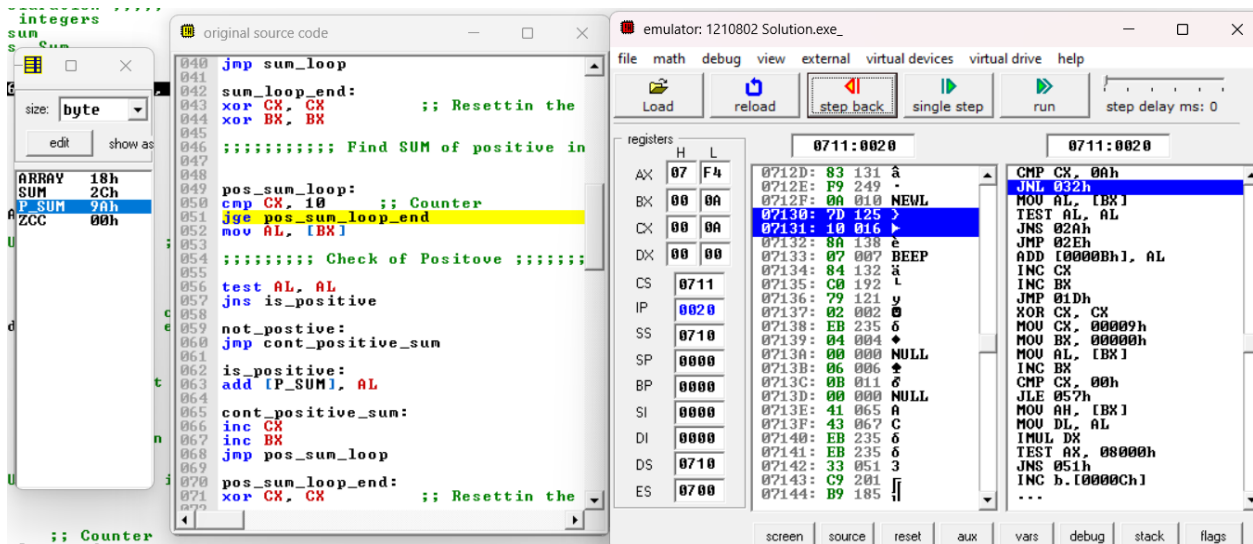
Which is a valid output, since the arithmetic sum of the elements equal 44 decimal equivalent to 2C in hexadecimal.

Finding the Positive Sum

Then the program finds the sum of the elements that are greater than zero in the array. And store it in a memory variable called P_SUM. This is done using the (test) mnemonic or instruction which is a bitwise AND.



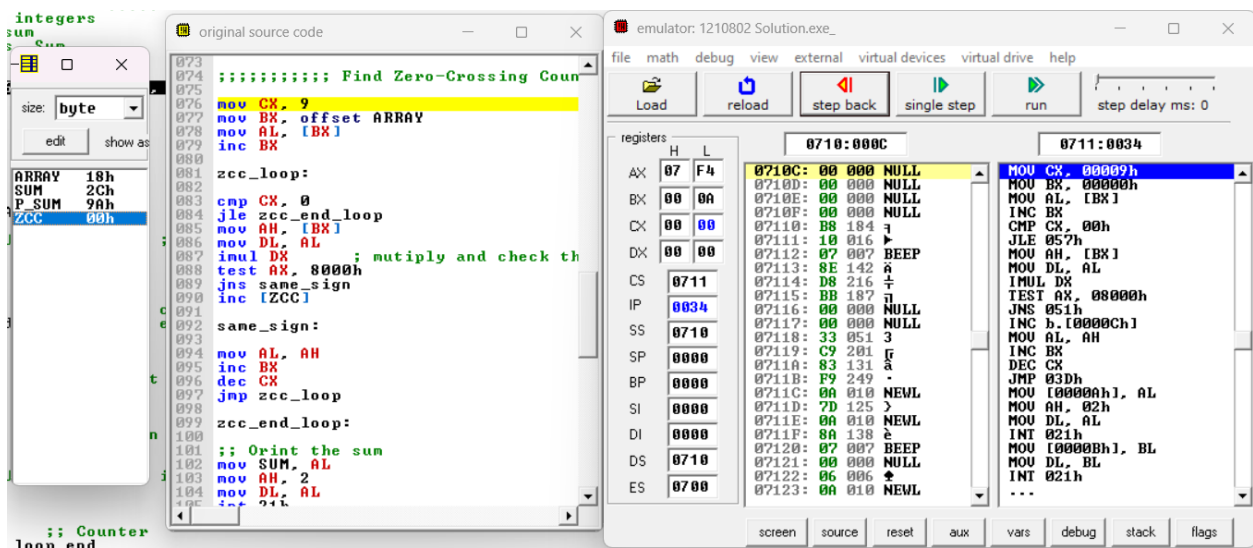
It is noticed that the P_SUM initial value is 00h. And it starts to change once the loop changes.



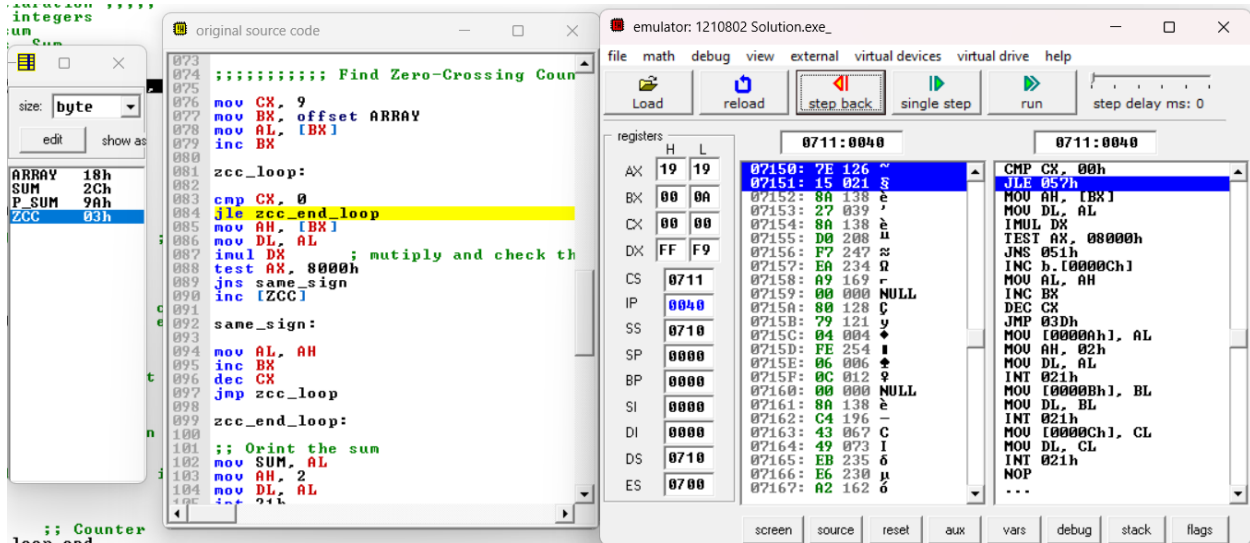
It is noticed that the final output is valid for the positive SUM which is supposed to be for the running ARRAY 154 in decimal that is 9A in hexadecimal.

Finding the Zero-Crossing Count (ZCC)

Finally the program finds the Zero-Crossing Count (ZCC) of the array elements. Which is computed by counting the number of times the successive samples of the array change their signs. For example, for an array $A = \{+1, -2, -8, +3, -4, +5, +6\}$, the ZCC is 4, i.e. the samples cross zero-axes 4 times. And the value is stored in the variable called ZCC.



After running the loop:



In the running example array there exists 4 Zero-Crossing values. So the final output is valid that is stored in ZCC variable 03 in Hexadecimal.