A Laboratory Project on

**16-bit Decimal Calculator**

Submitted to Manipal University, Jaipur

Towards the partial fulfillment for the Award of the Degree of

**BACHELORS OF TECHNOLOGY**

In Computer and Communication Engineering

2018-2019

By

Amar Sharma

179303017



Under the guidance of

Dr. Naina Narang

**Department of Computer and Communication Engineering**

**School of Computing and Information Technology**

**Manipal University Jaipur**

**Jaipur, Rajasthan**

1. **Problem Statement**

* To implement a 16-bit decimal calculator for performing basic arithmetic operations between two decimal numbers.

1. **Introduction**

Mathematics is a building block upon which all of the sciences are dependent. Being able to perform simple arithmetic operations quickly and efficiently is a necessary tool in all scientific fields. Calculators were created in order to give people a simple, fast, and error free method of doing these calculations. I chose to prototype a calculator because they are one of the most basic and important tools for an engineer such as myself. Being able to design and understand the hardware of a calculator is a good starting point from which I can go on to design and understand more complicated devices.

The program is designed to act like a “16–bit Decimal Calculator” with the usual standard functions (addition, subtraction, multiplication, division, modulo, and power). This calculator will have the capability of performing arithmetic operations on 16-bit decimal numbers. It operates in base 10(Decimal) and is capable of accepting, displaying, and operating on any numbers in the range **0 to 65535** (**16-bit**). If answers occur that are not in this range an overflow will occur and the outputted number is not correct.

This calculator works by accepting three inputs from a user: a first number (from 0 to 65535), a second number (from 0 to 65535), and lastly, an operator (+, -, \*, /, %, ^).

**+ :** Addition

**- :** Subtraction

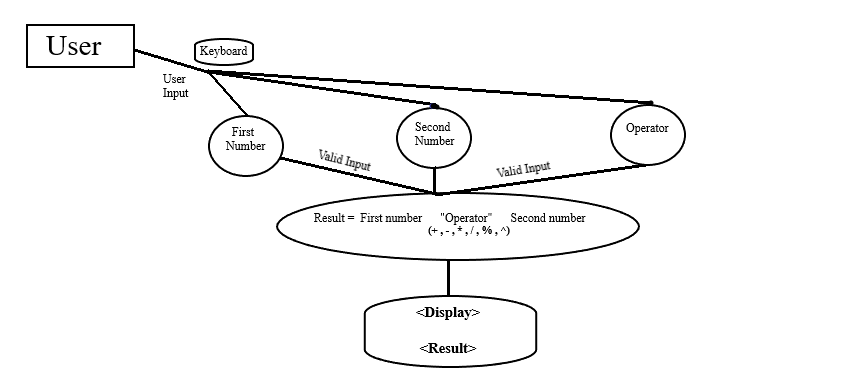
**\* :** Multiplication

**/ :** Division

**% :** Modulo

**^ :** Power

**2.a. Abstract Global Level Block Diagram**

****Figure 1 is a Global Level Block diagram which identifies the major functions of the calculator program and shows their relationship to each other.

**Figure 1**

1. **Modularizing the Problem**

**3.a. Specification**

**3.a.1. Number and Operator entry**

Enter first number using the keyboard (note: to use a numeric keypad make sure Num Lock is on.). As long as an enter key is not been pressed, digits will be added to the right-hand side of the current number. After enter key is pressed same procedure will be followed for second number and operator (note: only one operator at a time). This number will be displayed in the display area as it is composed. Number entry restarts after result display.

**3.a.2 Arithmetic Operations**

Operations are represented by symbols (+, -, \*, /, %, ^) on the calculator which specify the arithmetic operation to perform.

* **Addition (+)**

Choosing an addition operation, which will be marked with the character “+” will perform the addition of first number with second number and display the result of addition.

* **Subtraction (-)**

Choosing a subtract operation, which will be marked with the character “-” will subtract the second number from first number and display the result of subtraction.

* **Multiplication (\*)**

Choosing a multiplication operation, which will be marked with the character “\*” will multiply the first number with second number and display the result of multiplication.

* **Division (/)**

Choosing a division operation, which will be marked with the character “/” will divide the first number by second number and display the result of division.

* **Modulo (%)**

Choosing a modulo operation, which will be marked with the character “%” will divide the first number by second number and display the remainder obtained.

* **Power (^)**

Choosing a power operation, which will be marked with the character “^” will cause, second number of times a first number is multiplied by itself and display the result.

**3.a.3 Display result**

After choosing an appropriate operation, the result of operation will be displayed on the screen and the calculator will be restarted. Thus ready for the entry of the next numbers and operator.

**3.a.4 Response to undesired events**

* + - * **Division by zero**

If the user requests division by zero, the application will put the message “ERROR: DIVIDE BY ZERO” in the display area and reset number entry.

* + - * **Entering invalid value**

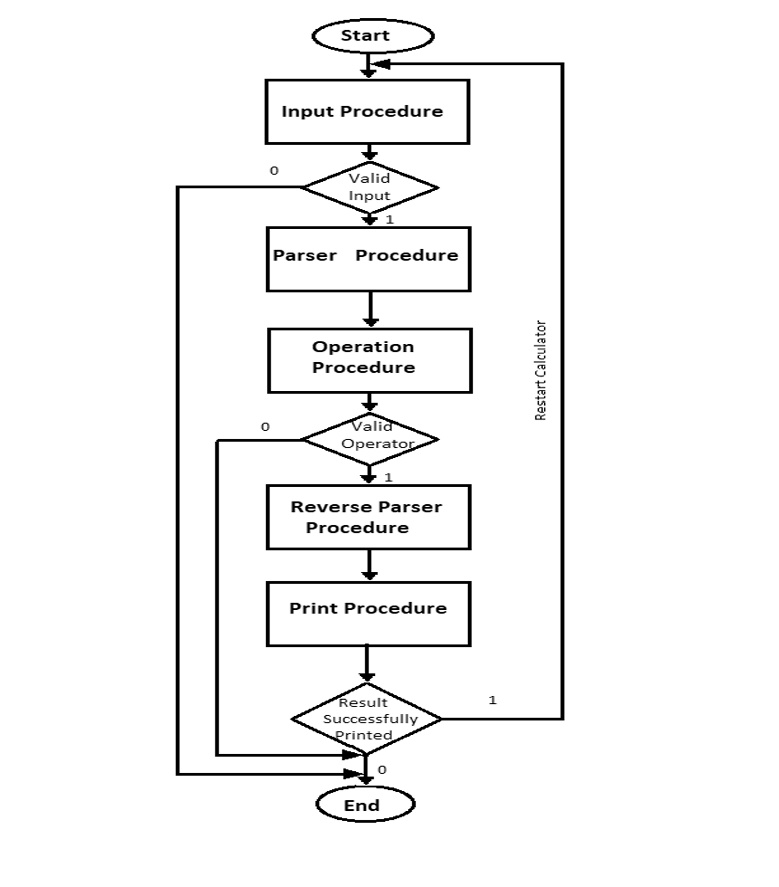
If the user enters a number which contains characters which are not digits into the display area, the calculator will display the message “INVALID INPUT” and reset number entry.

**3.b Software**

The program is written in assembly language for 8086 microprocessor using EMU8086, EMU8086 is a Microprocessor emulator with integrated 8086 Assembler and free tutorial. Emulator runs programs on a virtual machine, it emulates real hardware, such as screen, memory and input/output devices. So, the program shall be usable on any system which supports the EMU8086 emulator, and shall not require any particular hardware or software.

1. **Control Program**

**4.a. Flow Chart for Program Control**

****

* **Input procedure:**
* Read two numbers (In form of characters)
* Converts character to digit (SUB ASCII value of ‘0’)
* Stores digits in memory
* **Parser procedure:**
* Fetch digits from memory
* Converts digits into number (Ex. 3,2 = 2 + 3\*10 = 32)
* **Operation procedure:**
* Read one operator
* Perform arithmetic operation
* Stores the result
* **Reverse parser procedure:**
* Converts the resultant number to digits (ADD ASCII value of ‘0’)
* Store the digits for printing result.
* **Print procedure:**
* Prints the result on screen.

1. **Source Code**
2. org 100h
3. .DATA
4. msg1 DB 0AH,0DH, "Enter first Number : $"
5. msg2 DB 0AH,0DH, "Enter second Number :$”
6. msg3 DB 0AH,0DH, "Enter operator (+,-,\*,/,%,^) : $"
7. result DB 0AH,0DH, "Result is : $"
8. proj DB 0AH,0DH, " -------- MPMC PROJECT - CALCULATOR -------$"
9. myname DB 0AH,0DH, " -------------- AMAR SHARMA ---------------$"
10. reg\_no DB 0AH,0DH, " --------------- 179303017 ----------------$"
11. section DB 0AH,0DH," ---------------- CCE - A -----------------$"
12. done DB 0AH,0DH, "---------------------------------$"
13. done2 DB 0AH,0DH, "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*$"
14. invalid\_message DB 0AH,0DH, "INVALID INPUT$"
15. num1 dw 00h
16. num2 dw 00h
17. overflow db 00h
19. .CODE
20. include 'emu8086.inc'
21. LEA DX,done2
22. MOV AH,09H
23. INT 21H
24. LEA DX,proj
25. MOV AH,09H
26. INT 21H
27. LEA DX,myname
28. MOV AH,09H
29. INT 21H
30. LEA DX,reg\_no
31. MOV AH,09H
32. INT 21H
33. LEA DX,section
34. MOV AH,09H
35. INT 21H
36. LEA DX,done2
37. MOV AH,09H
38. INT 21H
39. calculator:
40. MOV AX,@DATA
41. MOV DS,AX
42. CALL input
43. CALL parser
44. CALL operation
46. MOV [SI],'&'
48. call reverse\_parser
49. call print\_result
51. input PROC ;/////////////////////////////// input procedure /////////////////////////////////;
53. MOV [SI],'&'
54. LEA DX,msg1
55. MOV AH,09H
56. INT 21H
57. input1:
58. MOV AH,01H
59. INT 21H
60. CMP AL,13d
61. JZ print\_message2
62. MOV AH,AL
63. SUB AH,'0'
64. JC invalid
65. MOV AH,AL
66. MOV DH,'9'
67. SUB DH,AH
68. JC invalid
69. SUB AL,'0'
70. INC SI
71. MOV [SI],AL
72. JMP input1
73. print\_message2:
74. INC SI
75. MOV [SI],'&'
76. LEA DX,msg2
77. MOV AH,09H
78. INT 21H
79. input2:
80. MOV AH,01H
81. INT 21H
82. CMP AL,13d
83. JZ exit
84. MOV AH,AL
85. SUB AH,'0'
86. JC invalid
87. MOV AH,AL
88. MOV DH,'9'
89. SUB DH,AH
90. JC invalid
91. SUB AL,'0'
92. INC SI
93. MOV [SI],AL
94. JMP input2
95. exit:
96. ret
98. invalid:
99. LEA DX,invalid\_message
100. MOV AH,09H
101. INT 21H
102. hlt
103. ENDP ;//////////////////////////// END of input procedure //////////////////////////////;
105. parser PROC ;////////////////////////////// parser procedure //////////////////////////////////;
107. MOV CX,01d
108. MOV BX,00H
110. parse2:
111. MOV AX,00H
112. MOV AL,[SI]
113. MUL CX
114. ADD BX,AX
115. MOV AX,CX
116. MOV CX,10d
117. MUL CX
118. MOV CX,AX
119. DEC SI
120. CMP [SI],'&'
121. JNZ parse2
123. MOV [num2],BX
124. MOV BX,00H
125. MOV DX,00h
126. DEC SI
127. MOV CX,01d
129. parse1:
130. MOV AX,00H
131. MOV AL,[SI]
132. MUL CX
133. ADD BX,AX
134. MOV AX,CX
135. MOV CX,10d
136. MUL CX
137. MOV CX,AX
138. DEC SI
139. CMP [SI],'&'
140. JNZ parse1
142. MOV [num1],BX
143. MOV AX,[num1]
144. MOV BX,[num2]
145. ret
146. ENDP ;/////////////////////////////// END of parser procedure ///////////////////////////;

149. operation proc ;//////////////////////////////// operation procedure //////////////////////////////;
150. MOV CX,AX
151. LEA DX,msg3
152. MOV AH,09H
153. INT 21H
154. MOV AH,01H
155. INT 21H
157. CMP AL,'+'
158. JZ addition
159. CMP AL,'-'
160. JZ subtraction
162. CMP AL,'\*'
163. JZ multiplication
165. CMP AL,'/'
166. JZ division
168. CMP AL,'%'
169. JZ mod
171. CMP AL,'^'
172. JZ pow
174. LEA DX,invalid\_message
175. MOV AH,09H
176. INT 21H
177. hlt
179. addition:
180. MOV AX,CX
181. MOV DX,00h
182. ADD AX,BX
183. ADC AX,DX
184. RET
185. subtraction:
186. MOV AX,CX
187. SUB AX,BX
188. JC ov
189. JNC nov
190. ov:NEG AX
191. MOV [overflow],01h
192. RET
193. nov:RET
195. multiplication:
196. MOV AX,CX
197. MOV DX,00H
198. MUL BX
199. RET
200. division:
201. MOV AX,CX
202. MOV DX,00H
203. ADD BX,DX
204. JZ DbyZ
205. DIV BX
206. RET
207. DbyZ: print ' ERROR : DIVIDE BY ZERO'
208. JMP calculator
209. mod:
210. MOV AX,CX
211. MOV DX,00H
212. ADD BX,DX
213. JZ DbZ
214. DIV BX
215. MOV AX,DX
216. DbZ: RET
217. pow:
218. MOV AX,CX
219. MOV CX,BX
220. ADD CX,00h
221. JZ Lc
222. SUB CX,01h
223. JZ La
224. JNZ Lb
225. La: ret
226. Lb: MOV BX,AX
227. MOV DX,00h
228. L1: MUL BX
229. LOOP L1
230. ret
231. Lc: MOV AX,01h
232. ret
234. ENDP ;///////////////////////////// END OF operation procedure ////////////////////////////;

237. reverse\_parser PROC ;//////////////////////////// reverse\_parser procedure ///////////////////////////////;
239. r\_parse:
240. MOV DX,00h
241. MOV BX,10d
242. DIV BX
243. ADD DL,'0'
244. INC SI
245. MOV [SI],DL
246. ADD AX,00h
247. JNZ r\_parse

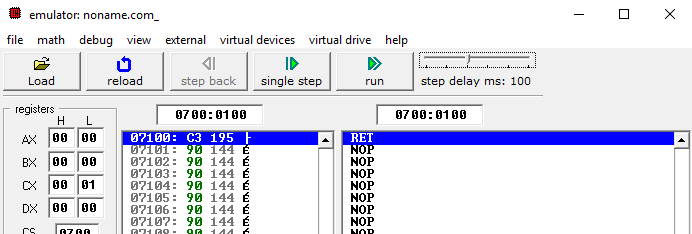
250. ENDP ;///////////////////////// END of reverse\_parser procedure ///////////////////////////;

253. print\_result PROC ;//////////////////////////// print\_result procedure /////////////////////////////////;
255. LEA DX,result
256. MOV AH,09H
257. INT 21H
258. MOV CL,01h
259. CMP CL,[overflow]
260. MOV [overflow],00h
261. JZ print\_minus
262. JNZ print
263. print\_minus: MOV DL,'-'
264. MOV AH,02H
265. INT 21H
267. print:
268. MOV DL,[SI]
269. MOV AH,02H
270. INT 21H
271. DEC SI
272. CMP [SI],'&'
273. JNZ print
275. LEA DX,done
276. MOV AH,09H
277. INT 21H
279. JMP calculator
280. ENDP ;//////////////////////////// END of print\_result procedure //////////////////////////;
281. **Testing and Debugging**

Testing and Debugging is done by using the following approaches:

**4.a. Using EMU8086 Tool**

There is a tool in emu8086 emulator by which you can slow down the instruction execution speed so, you can check the content of registers after each instruction which helps to debug the code.



**4.b. Using 8086 Type 1 Interrupt**

**Single Step Interrupt-Type1:**When we tell a system to single step, it will execute one instruction and stop. We can then examine the contents of registers and memory locations.   
  
In other words, when in single step mode a system will stop after it executes each instruction and wait for further direction from user. The 8086 trap flag and type 1 interrupt response make it quite easy to implement a single step feature direction.

**4.c. Removing syntax errors**

**Syntax errors are corrected at the time of assembling the code.**

1. **Conclusion and Future Scope**

The project has been successfully completed by having established the user friendly interface with the help of Emu8086.It allows user to perform basic arithmetic operations on 16-bit decimal number (range 0-65535) in an easy way. If we are able to introduce friendly interface for complicated tasks then it gives user what he wants, that will be ultimate success of our attempts.

At the same time there is some scope for improvement in the feature. It can be possible to make it more user friendly by adding more variety of functions to it and also by increasing its range (Ex. 32-bit ,64-bit etc).