

Base Converter Embedded in High level Language

Strengths of the Project:

- ✓ This program represents the best example of inline assembly.
- ✓ The program can handle 16 bit of input data properly and successfully.
- ✓ It can calculate results accurately.
- ✓ The program executable file can be run on any platform. Like Windows or Linux.

Shortcomings of the Project:

- ✓ The written code is not optimized as it should be.
- ✓ It can not convert signed numbers correctly.
- ✓ There are 6 total functions for conversion. Code to take input from user is implemented in each function separately according to function requirements. So, there are no function parameters. Which is not good (according to me) as I can not use that function again by providing only arguments. I must write whole code for that function again to use it, which is not optimized. So, in this regard a big future improvement can be made by taking input separately and provide them as parameters in functions.
- ✓ The intel syntax is used for inline assembly. So, to compile it successfully one must use MSVC (Microsoft Visual C++).
- ✓ The program can be improved by adding more conversions functions like we can add 6 more functions for Octal number system.

Testing:

- ✓ Every line of code was tested with the help of Disassembler, Registers panel and Memory viewer in debugger section of VS studio.
- ✓ After the completion of every function, it was tested against some random and predefined values. The result was crossed checked with the **online base converter's** results.
- ✓ After the completion of the project testing on whole project was performed. Switches were checked by giving random values as switches were used to navigate between different functions
- ✓ Restraint put on different types of inputs were checked by giving very large value greater than 16bit and very small value (negative number).
- ✓ All the results were cross checked by an online base converter.

Resources:

- <https://learn.microsoft.com/documentation> for resolving Errors
- Emu8086 official documentation.

To compile and test the code, use VS studio.

```

1. //=====Base Converter=====
2. //Zohaib Ahmad
3. //SP21-BCS-048
4.
5. #include <iostream>
6. #include <string>
7. #include <windows.h>
8. #include <cstdlib>
9. using namespace std;
10. //-----
11. char func_1_input[] = "0000";
12. int func_1_DecimalResult;
13. //-----
14. string func_2_hexResult;
15. int func_2_input;
16. short func_2_counter;
17. //-----
18. int func_3_input;
19. string func_3_BinaryResult;
20. short func_3_counter;
21. //-----
22. unsigned short func_4_decimalResult;
23. short func_4_counter;
24. //-----
25. char func_5_input[] = "0000";
26. int func_5_decimalvalue;
27. uint8_t func_5_counter;
28. //-----
29. unsigned short func_6_decimalvalue;
30. string func_6_hexResult;
31. short func_6_counter;
32.
33. //=====
34. int random_num() {
35.     int a = 1;
36.     int b = 15;
37.     int rn = (rand() % (b - a + 1)) + a;
38.     return rn;
39. }
40. //=====
41. void color(int y)
42. {
43.     SetConsoleTextAttribute(GetStdHandle(STD_OUTPUT_HANDLE)
44. ), y);
45. }
46. //1.=====
47. void De_to_hex()
48. {
49.     //string
50.     func_2_hexResult = "0000";
51.     func_2_counter = 0;
52.     // Take input between 0 to 65535
53.     do
54.     {
55.         std::cout << "Enter a decimal number (0-65535): ";
56.         cin >> func_2_input;
57.     } while (func_2_input < 0 || func_2_input > 65535);
58.
59.     __asm {
60.         //clearing registers
61.         xor EBX, EBX;
62.         xor EDX, EDX;
63.         xor EAX, EAX;
64.         xor EDI, EDI;
65.         xor ECX, ECX;
66.
67.         //Storing offset address of input variable in ESI register
68.         lea ESI, func_2_input;
69.         //Storing offset address of output string in EDI register
70.         lea EDI, func_2_hexResult;
71.
72.         //Adding 4 in EDI to get exact address of first byte of the
73.         result string
74.         add EDI, 4;
75.         //Moving input number in AX register
76.         mov AX, [ESI];
77.         //moving 16 in EBX
78.         mov EBX, 16;
79.
80.         divide_1:
81.         //Dividing EAX by EBX
82.         div EBX;
83.         //Pushing the remainder onto the stack
84.         push EDX;
85.         //Incrementing counter (which will be used to print the
86.         result)
87.         add func_2_counter, 1;
88.         //Incrementing ECX to use it later in loop
89.         inc ECX;
90.         //Comparing AX with 0
91.         cmp AX, 0;
92.         //Checking if AX is not zero jump to label divide_1;
93.         jne divide_1;
94.
95.         check_1:
96.         //Popping remainder values in EBX
97.         pop EBX;
98.         //Comparing with 9
99.         cmp BL, 9;
100.        //If BL is greater than 9 jump to label greater_1
101.        ja greater_1;
102.
103.        less_1:
104.        //Adding 48 to change the value to Ascii to print
105.        add BL, 48;
106.        //Storing Ascii value to result string
107.        mov[EDI], BL;
108.        //Incrementing address to move it to next Byte of string
109.        add EDI, 1;
110.
111.        loop check_1;
112.        jmp r;
113.
114.        greater_1:
115.        //If BL is greater than 9 adding addition 7 to get ascii
116.        values from A-F
117.        add BL, 7;
118.        jmp less_1;
119.
120.        r:
121.
122.    }
123.    std::cout << "Hexadecimal: ";
124.    color(2);
125.    for (int i = 0; i < func_2_counter; i++)
126.    {
127.        //Printing result string
128.        std::cout << func_2_hexResult[i];
129.    }
130.    color(15);
131.    std::cout << "" << endl;
132. }
133. //2.=====
134. void hex_to_de() {
135.     bool choice = false;
136.
137.     //Will take 4 character hex input.
138.     //will check whether the input is between 1-9 an A-F.
139.     //if not ask again.
140.
141.     do {
142.         std::cout << "Enter hexadecimal value : "
143.         << endl;
144.         cin >> func_1_input;
145.     } while (choice == false);
146. }

```

```

151.         for (int i = 0; i < 4; i++) {
152.             if ((func_1_input[i] >= '0' &&
func_1_input[i] <= '9') || (func_1_input[i] >= 'A' &&
func_1_input[i] <= 'F') || (func_1_input[i] >= 'a' &&
func_1_input[i] <= 'f')) {
153.                 choice = true;
154.             }
155.             else {
156.                 std::cout << "Enter correct hex Value" << endl;
157.                 choice = false;
158.                 break;
159.             }
160.         }
161.     } while (choice == false);
162.
163.
164.
165.     __asm
166.     {
167.         //Moving 4 in ECX to loop 4 times
168.         mov ECX, 4;
169.         //Clearing EBX
170.         xor EBX, EBX;
171.         mov EBX, 0;
172.
173.
174.     myl:
175.     //Comparing input characters with the value of ascii 'a'
176.         cmp[func_1_input + EBX], 'a';
177.     //jump if input character value is less than 'a'
178.         jl nocap;
179.     //Comparing characters of input with the value of ascii 'z'
180.         cmp[func_1_input + EBX], 'z';
181.     //jump if input character value is greater than 'z'
182.         ja nocap;
183.         //Else sub 32 to change it to capital
184.         sub[func_1_input + EBX], 32;
185.     nocap:
186.         add EBX, 1;
187.         loop myl;
188.
189.     //Converting ascii values to hex in memory
190.         xor ECX, ECX;
191.         xor EAX, EAX;
192.     //Taking length of variable 'func_1_input' and moving it to
    ECX
193.         mov ECX, length func_1_input;
194.     //sub 1 for getting correct length
195.         sub ECX, 1;
196.         xor EBX, EBX;
197.         mov EBX, 0;
198.     ToHex_1:
199.     //Moving first ascii character in BL
200.         mov AL, [func_1_input + EBX];
201.     //Comparing it with ascii '9'
202.         cmp AL, 39h;
203.     //jump if value is above ascii '9'
204.         ja If_A_to_F_1;
205.         //Else sub 30h from it to change it to hex
    value
206.         sub AL, 30h;
207.     //Store it back to its location
208.         mov[func_1_input + EBX], AL;
209.         add EBX, 1;
210.         jmp loop_1;
211.
212.
213.     //if value is above the value of '9'
214.     If_A_to_F_1:
215.     //Add 7h to move to ascii value 'A'
216.         sub AL, 7h;
217.         //sub 30h to change it into its hex value
218.         sub AL, 30h;
219.         //storing back to its location
220.         mov[func_1_input + EBX], AL;
221.         add EBX, 1;
222.         jmp loop_1;
223.
224.     loop_1:
225.         loop ToHex_1;
226.     }
227.
228.
229.
230.     __asm
231.     {
232.         //Clearing the registers
233.         xor EDX, EDX;
234.         xor EAX, EAX;
235.         xor EDI, EDI;
236.         xor ECX, ECX;
237.         //Storing Starting address of hex value in ESI
238.         lea ESI, func_1_input;
239.         //Storing address of result in EDI
240.         lea EDI, func_1_DecimalResult;
241.         mov ECX, 4;
242.         //moving 1 in BX
243.         mov BX, 1d;
244.         //Adjusting address of ESI (addressing last value)
245.         add ESI, 3;
246.
247.     labl2:
248.
249.         //Moving first Hex value in AL
250.         mov AL, [ESI];
251.         //Multiplying it with BX
252.         mul BX;
253.         //Pushing DX onto stack
254.         push DX;
255.         //Pushing AX onto stack
256.         push AX;
257.         //Pop value of stack in 32bit register EDX
258.         pop EDX;
259.         //storing result value in 'func_1_DecimalResult' variable
260.         mov[EDI], EDX;
261.         //Adding 2 in EDI to point to next word
262.         add EDI, 2;
263.         sub ESI, 1;
264.
265.         //Moving value of BX in AX
266.         mov AX, BX;
267.         mov BX, 16d;
268.         mul BX;
269.         mov BX, AX;
270.         xor EAX, EAX;
271.         loop labl2;
272.     }
273.
274.     __asm
275.     {xor EBX, EBX;
276.     xor EDX, EDX;
277.     xor EDI, EDI;
278.
279.     lea ESI, func_1_DecimalResult;
280.     mov ECX, 4;
281.
282.     labl1:
283.         //Adding all the word size value in variable
284.         mov BX, [ESI];
285.         add ESI, 2;
286.         add EDX, EBX;
287.         loop labl1;
288.
289.         lea EDI, func_1_DecimalResult;
290.         mov[EDI], EDX;
291.         xor ECX, ECX;
292.     }
293.
294.
295.     std::cout << "Decimal: ";
296.     color(2);
297.     std::cout << func_1_DecimalResult << endl;
298.     color(15);
299.
300.
301.
302. }
303.
304. //3.=====
305. void de_to_bi() {
306.
307.     //string
308.     func_3_BinaryResult = "$$$$$$$$$$$$$$";
309.     func_3_counter = 0;
310.
311.     // Take input between 0 to 65535
312.     do
313.     {
314.         std::cout << "Enter a decimal number (0-
65535): " << endl;
315.         cin >> func_3_input;
316.     } while (func_3_input < 0 || func_3_input > 65535);
317.
318.     __asm

```

```

319. {
320.     //Clearing registers
321.     xor EBX, EBX;
322.     xor EDX, EDX;
323.     xor EAX, EAX;
324.     xor EDI, EDI;
325.     xor ECX, ECX;
326.
327.     //Loading address in ESI
328.     lea ESI, func_3_input;
329.     //Loading address in EDI
330.     lea EDI, func_3_BinaryResult;
331.     //Adjusting address of EDI
332.     add EDI, 4;
333.
334.     //Moving input value in AX
335.     mov AX, [ESI];
336.     mov EBX, 2;
337.
338.     divid_3:
339.         //Dividing by EBX
340.         div EBX;
341.         //Push remainder onto stack
342.         push EDX;
343.         add func_3_counter, 1;
344.         inc ECX;
345.         //Comparing with zero
346.         cmp AX, 0;
347.         //if not equal loop 'divid_3'
348.         jne divid_3;
349.
350.
351.         //saves result
352.     save_3:
353.         //pop in EBX register
354.         pop EBX;
355.         //Convert to ascii to print
356.         add BX, 30h;
357.         mov[EDI], BL;
358.         inc EDI;
359.         loop save_3;
360.     }
361.     std::cout << "Binary: ";
362.     color(2);
363.     for (int i = 0; i < func_3_counter; i++) {
364.         std::cout << func_3_BinaryResult[i];
365.     }
366.     std::cout << "" << endl;
367.     color(15);
368.
369. }
370.
371. //4.=====
372. void bi_to_de()
373. {
374.     func_4_decimalResult = 0;
375.     func_4_counter = 0;
376.     char func_4_input[] = "$$$$$$$$$$$$$$";
377.
378.     //input
379.     std::cout << "Enter Binary value : " << endl;
380.     cin >> func_4_input;
381.
382.     //Counting the 0 and 1
383.     for (int i = 0; i < 18; i++) {
384.         if (func_4_input[i] == '$')
385.         {
386.             break;
387.         }
388.         else {
389.             func_4_counter = func_4_counter
390. + 1;
391.         }
392.     }
393.     func_4_counter = func_4_counter - 1;
394.
395.
396.
397.     __asm
398.     {
399.
400.         //Clearing registers
401.         xor EDX, EDX;
402.         xor ECX, ECX;
403.         xor EAX, EAX;
404.         xor EDI, EDI;
405.
406.         //Loading address of 'func_4_input' in ESI
407.         lea ESI, func_4_input;
408.         //Loading address of 'func_4_decimalResult'
409.         in EDI
410.         lea EDI, func_4_decimalResult;
411.         mov BX, 1d;
412.         mov CX, func_4_counter;
413.         //for pointing the last byte of the char
414.         array
415.         add ESI, ECX;
416.         sub ESI, 1;
417.
418.     labl2:
419.         mov AL, byte ptr[ESI];
420.         //coverting to decimal
421.         sub AL, 30h;
422.         mul BX;
423.         push DX;
424.         push AX;
425.         pop EDX;
426.         add func_4_decimalResult, DX;
427.         sub ESI, 1;
428.
429.         mov AX, BX;
430.         mov BX, 2d;
431.         mul BX;
432.         mov BX, AX;
433.         xor EAX, EAX;
434.         loop labl2;
435.     }
436.
437.     //Result
438.     std::cout << "Decimal: ";
439.     color(2);
440.     std::cout << func_4_decimalResult << endl;
441.     color(15);
442.
443.     //5.=====
444.     void hex_to_bi() {
445.         func_5_counter = 0;
446.         char e[] = "$$$$$$$$$$$$$$";
447.
448.         //Will take 4 character hex input.
449.         //will check whether the input is between 1-9 an A-F.
450.         //if not ask again.
451.
452.         bool choice = false;
453.         do {
454.             std::cout << "enter hexadecimal value : "
455. << endl;
456.             cin >> func_5_input;
457.
458.             for (int i = 0; i < 2; i++) {
459.                 if ((func_5_input[i] >= '0' &&
460. func_5_input[i] <= '9') || (func_5_input[i] >= 'A' &&
461. func_5_input[i] <= 'F') || (func_5_input[i] >= 'a' &&
462. func_5_input[i] <= 'f')) {
463.                     choice = true;
464.                 }
465.                 else {
466.                     std::cout << "enter
467. correct" << endl;
468.                     choice = false;
469.                     break;
470.                 }
471.             } while (choice == false);
472.
473.             __asm
474.             {
475.                 mov ECX, 4;
476.                 xor EBX, EBX;
477.                 mov EBX, 0;
478.
479.                 myl2:
480.                 //Comparing input characters with the value of ascii 'a'
481.                 cmp[func_5_input + EBX], 'a';
482.                 //jump if input character value is less than 'a'
483.                 jl nocap1;
484.                 //Comparing characters of input with the value of ascii 'z'
485.                 cmp[func_5_input + EBX], 'z';
486.                 //jump if input character value is greater than 'z'
487.                 ja nocap1;

```

```

487. //Else sub 32 to change it to capital
488. sub[func_5_input + EBX], 32;
489. nocap1:
490. add EBX, 1;
491. loop myl2;
492.
493.
494. //Converting Ascii value to its hex values
495. xor ECX, ECX;
496. xor EAX, EAX;
497. mov ECX, length func_5_input;
498. sub ECX, 1;
499. xor EBX, EBX;
500. mov EBX, 0;
501. ak1:
502. mov AL, [func_5_input + EBX];
503. cmp AL, 39h;
504. ja delta1;
505. sub AL, 30h;
506. mov[func_5_input + EBX], AL;
507. add EBX, 1;
508. jmp alphas;
509.
510. delta1:
511. sub AL, 7h;
512. sub AL, 30h;
513. mov[func_5_input + EBX], AL;
514. add EBX, 1;
515. jmp alphas;
516.
517. alphas:
518. loop ak1;
519.
520.
521. //Converting Hex to Decimal
522. _asm
523. {
524.     xor EDX, EDX;
525.     xor EAX, EAX;
526.     xor EDI, EDI;
527.     xor ECX, ECX;
528.     lea ESI, func_5_input;
529.     lea EDI, func_5_decimalvalue;
530.     mov ECX, 4;
531.     mov BX, 1d;
532.     add ESI, 3;
533.
534. labl2:
535.
536.     mov AL, [ESI];
537.     mul BX;
538.     push DX;
539.     push AX;
540.     pop EDX;
541.     mov[EDI], EDX;
542.     add EDI, 2;
543.     sub ESI, 1;
544.
545.     mov AX, BX;
546.     mov BX, 16d;
547.     mul BX;
548.     mov BX, AX;
549.     xor EAX, EAX;
550.     loop labl2;
551. }
552. _asm
553. {
554.     xor EBX, EBX;
555.     xor EDX, EDX;
556.     xor EDI, EDI;
557.     lea ESI, func_5_decimalvalue;
558.
559.     mov ECX, 4;
560.
561. labl12:
562.     mov BX, [ESI];
563.     add ESI, 2;
564.     add EDX, EBX;
565.     loop labl12;
566.
567.     lea EDI, func_5_decimalvalue;
568.     mov[EDI], EDX;
569.
570.
571.
572. //Converting decimal value to
573. Binary
574.     xor EDX, EDX;
575.     xor EAX, EAX;
576.     xor EDI, EDI;
577.     xor ECX, ECX;
578.     lea ESI, func_5_decimalvalue;
579.     lea EDI, e;
580.
581.
582.     mov AX, [ESI];
583.     mov EBX, 2;
584.
585.     divid12:
586.     div EBX;
587.     push EDX;
588.     add func_5_counter, 1;
589.     inc ECX;
590.     cmp AX, 0;
591.     jne divid12;
592.
593.
594.     savel2:
595.     pop EBX;
596.     add BX, 30h;
597.     mov[EDI], BL;
598.     inc EDI;
599.     loop savel2;
600. }
601.
602. }
603.
604. std::cout << "Binary :";
605. color(2);
606. for (int i = 0; i < func_5_counter; i++) {
607.     std::cout << e[i];
608. }
609.
610. std::cout << "" << endl;
611. color(15);
612.
613. }
614.
615. //6.=====
616. void bi_to_hex() {
617.
618.     func_6_hexResult = "0000";
619.     func_6_counter = 0;
620.     func_6_decimalvalue = 0;
621.     short n3 = 0;
622.     char func_6_input[] = "$$$$$$$$$$$$$$$$";
623.
624. //Input
625. std::cout << "Enter Binary value : " << endl;
626. cin >> func_6_input;
627.
628. //Counting the 0 and 1
629. for (int i = 0; i < 18; i++) {
630.     if (func_6_input[i] == '$') {
631.         break;
632.     }
633.     else {
634.         n3 = n3 + 1;
635.     }
636. }
637.
638. n3 = n3 - 1;
639.
640.
641. //Converting binary To Decimal
642. _asm
643. {
644.     xor EDX, EDX;
645.     xor ECX, ECX;
646.     xor EAX, EAX;
647.     xor EDI, EDI;
648.     lea ESI, func_6_input;
649.     lea EDI, func_6_decimalvalue;
650.     mov BX, 1d;
651.     mov CX, n3;
652.     add ESI, ECX;
653.     sub ESI, 1;
654.
655. labl23:
656.
657.     mov AL, byte ptr[ESI];
658.     sub AL, 30h;
659.     mul BX;
660.     push DX;
661.     push AX;

```

```

662.         pop     EDX;
663.         add func_6_decimalvalue, DX;
664.         sub     ESI, 1;
665.
666.         mov     AX, BX;
667.         mov     BX, 2d;
668.         mul     BX;
669.         mov     BX, AX;
670.         xor     EAX, EAX;
671.         loop    lab123;
672.     }
673.     std::cout << func_6_decimalvalue << endl;
674.
675.     //Converting Decimal to Hex
676.     __asm {
677.         xor     EBX, EBX;
678.         xor     EDX, EDX;
679.         xor     EAX, EAX;
680.         xor     EDI, EDI;
681.         xor     ECX, ECX;
682.         lea     ESI, func_6_decimalvalue;
683.         lea     EDI, func_6_hexResult;
684.         add     EDI, 4;
685.
686.         mov     AX, [ESI];
687.         mov     EBX, 16;
688.
689.     Divide_6:
690.         div     EBX;
691.         push    EDX;
692.         add     func_6_counter, 1;
693.         inc     ECX;
694.
695.         cmp     AX, 0;
696.         jne     Divide_6;
697.
698.     check_6:
699.         pop     EBX;
700.         cmp     BL, 9;
701.         ja      greater_6;
702.
703.     less_6:
704.         add     BL, 48;
705.         mov     [EDI], BL;
706.         add     EDI, 1;
707.         loop    check_6;
708.         jmp     r1;
709.
710.     greater_6:
711.         add     BL, 7;
712.         jmp     less_6;
713.
714.     r1:
715.     }
716.     std::cout << "Hexadecimal: ";
717.     color(2);

```

```

718.         for (int i = 0; i < func_6_counter; i++) {
C:\Users\Zohaib Ahmad\Desktop\Micro_project\micro\Debug\micro.exe
-----
=====MAIN=====
1. Decimal to Hexadecimal.
2. Hexadecimal to Decimal.
3. Decimal to Binary.
4. Binary to Decimal.
5. Hexadecimal to Binary.
6. Binary to hexadecimal.
7. To clear the screen.
8. Exit ):
1
-----
Enter a decimal number (0-65535): 65535
Hexadecimal: FFFF
-----
=====MAIN=====
1. Decimal to Hexadecimal.
2. Hexadecimal to Decimal.
3. Decimal to Binary.
4. Binary to Decimal.
5. Hexadecimal to Binary.
6. Binary to hexadecimal.
7. To clear the screen.
8. Exit ):
719.         std::cout << func_6_hexResult[i];
720.     }
721.     std::cout << " " << endl;
722.     color(15);
723.
724. }
725. //=====
727.
728.
729. int main()
730. {
731.     int option;
732.     do {
733.         std::cout << "-----"
734.         << endl;
735.         color(random_num());
736.         std::cout << "\t=====MAIN=====
737.         << endl;
738.         std::cout << "\t1. Decimal to Hexadecimal."
739.         << endl;
740.         std::cout << "\t2. Hexadecimal to Decimal."
741.         << endl;
742.         std::cout << "\t3. Decimal to Binary." <<
743.         << endl;
744.         std::cout << "\t4. Binary to Decimal." <<
745.         << endl;
746.         std::cout << "\t5. Hexadecimal to Binary."
747.         << endl;
748.         std::cout << "\t6. Binary to hexadecimal."
749.         << endl;
750.         std::cout << "\t7. To clear the screen." <<
751.         << endl;
752.         std::cout << "\t8. Exit ): " << endl;
753.
754.         cin >> option;
755.         color(15);
756.         std::cout << "-----"
757.         << endl;
758.
759.         switch (option)
760.         {
761.             case 1:
762.                 De_to_hex();
763.                 break;
764.             case 2:
765.                 hex_to_de();
766.                 break;
767.             case 3:
768.                 de_to_bi();
769.                 break;
770.             case 4:

```

```
763.             bi_to_de();
764.
765.             break;
766.         case 5:
767.             hex_to_bi();
768.
769.             break;
770.         case 6:
771.             bi_to_hex();
772.             break;
773.         case 7:
774.             system("cls");
775.             break;
776.         default:
777.             break;
```

```
778.             }
779.         } while (option != 8);
780.
781.         return 0;
782.     }
783.
784.
785.
```

Result: