

Part 4: "New Advanced Study of Magic Squares and Cubes"
 Chapter 4: Commentary Articles No.2: **Kanji Setsuda**
 "Various Arts and Tools for Studying Magic Squares"
**Section 7: How to Compose 'Complete Euler Squares' 4x4
 by Binary Number System**

0. Let's Compose 'Complete Euler Squares' 4x4.

Let's compose Pan-diagonal Magic Squares of order 4 by "Greco-Latinian Method" applied even to every pan-diagonal of the object, shall we? When we now know that we cannot compose all 'Complete Euler Squares' of order 4 by Positional Number System of Base 4, instead of giving it up, we are going to try to compose them by Base 2, that means 'Binary Number System'.

The name "Greco-Latinian Method" is already famous and known well wide to many researchers, but I myself have not yet been well informed and educated, and I have failed in getting what I have wanted to know exactly about it. At last I made up my mind to try to define everything by myself.

1. Definition of 'Complete Euler Squares' 4x4 by Binary System

What does the 'Complete Euler Square' of order 4 look like?

Won't you take your careful look at the next lists below?

[List 1: Pan-diagonal Magic Squares with /D4i:

Positive n/: Euler Type,				Negative n/: Non-Euler Type]			
1# -1/ -2/		2# -3/ -2/		3# -4/ -5/			
1 15 4 14	0303 0231	1 15 4 14	0303 0231	1 15 6 12	0312 0213		
12 6 9 7	2121 3102	8 10 5 11	1212 3102	14 4 9 7	3021 1302		
13 3 16 2	3030 0231	13 3 16 2	3030 0231	11 5 16 2	2130 2031		
8 10 5 11	1212 3102	12 6 9 7	2121 3102	8 10 3 13	1203 3120		
4# 6/ 7/		5# -8/ -5/		6# 9/ 7/			
1 15 6 12	0312 0213	1 15 10 8	0321 0213	1 15 10 8	0321 0213		
8 10 3 13	1203 3120	14 4 5 11	3012 1302	12 6 3 13	2103 3120		
11 5 16 2	2130 2031	7 9 16 2	1230 2031	7 9 16 2	1230 2031		
14 4 9 7	3021 1302	12 6 3 13	2103 3120	14 4 5 11	3012 1302		
7# -1/ -10/		8# -3/ -10/		9# 6/ 11/			
1 14 4 15	0303 0132	1 14 4 15	0303 0132	1 14 7 12	0312 0123		
12 7 9 6	2121 3201	8 11 5 10	1212 3201	8 11 2 13	1203 3210		
13 2 16 3	3030 0132	13 2 16 3	3030 0132	10 5 16 3	2130 1032		
8 11 5 10	1212 3201	12 7 9 6	2121 3201	15 4 9 6	3021 2301		
10# 9/ 11/		11# -5/ -4/		12# -5/ -8/			
1 14 11 8	0321 0123	1 12 6 15	0213 0312	1 12 7 14	0213 0321		
12 7 2 13	2103 3210	8 13 3 10	1302 3021	8 13 2 11	1302 3012		
6 9 16 3	1230 1032	11 2 16 5	2031 2130	10 3 16 5	2031 1230		
15 4 5 10	3012 2301	14 7 9 4	3120 1203	15 6 9 4	3120 2103		

[List 2: The same Solutions with /D2i: Binary Decompositions]

1# 1/ 2/ 3/ 4/				2# 2/ 1/ 3/ 4/			
1 15 4 14	0101 0101 0110 0011	1 15 4 14	0101 0101 0110 0011				
12 6 9 7	1010 0101 1001 1100	8 10 5 11	0101 1010 1001 1100				
13 3 16 2	1010 1010 0110 0011	13 3 16 2	1010 1010 0110 0011				
8 10 5 11	0101 1010 1001 1100	12 6 9 7	1010 0101 1001 1100				

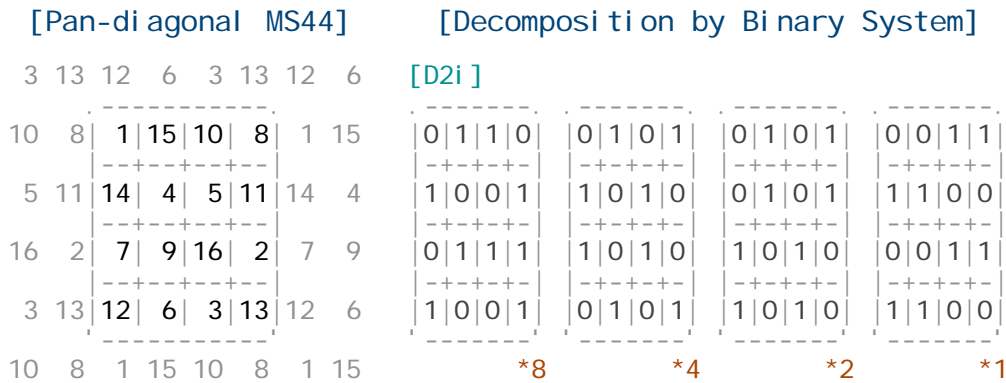
3#	1/	3/	2/	4/		4#	2/	3/	1/	4/
1 15 6 12	0101	0110	0101	0011	1 15 6 12	0101	0110	0101	0011	
14 4 9 7	1010	1001	0101	1100	8 10 3 13	0101	1001	1010	1100	
11 5 16 2	1010	0110	1010	0011	11 5 16 2	1010	0110	1010	0011	
8 10 3 13	0101	1001	1010	1100	14 4 9 7	1010	1001	0101	1100	
5#	3/	1/	2/	4/		6#	3/	2/	1/	4/
1 15 10 8	0110	0101	0101	0011	1 15 10 8	0110	0101	0101	0011	
14 4 5 11	1001	1010	0101	1100	12 6 3 13	1001	0101	1010	1100	
7 9 16 2	0110	1010	1010	0011	7 9 16 2	0110	1010	1010	0011	
12 6 3 13	1001	0101	1010	1100	14 4 5 11	1001	1010	0101	1100	
7#	1/	2/	4/	3/		8#	2/	1/	4/	3/
1 14 4 15	0101	0101	0011	0110	1 14 4 15	0101	0101	0011	0110	
12 7 9 6	1010	0101	1100	1001	8 11 5 10	0101	1010	1100	1001	
13 2 16 3	1010	1010	0011	0110	13 2 16 3	1010	1010	0011	0110	
8 11 5 10	0101	1010	1100	1001	12 7 9 6	1010	0101	1100	1001	
9#	2/	3/	4/	1/		10#	3/	2/	4/	1/
1 14 7 12	0101	0110	0011	0101	1 14 11 8	0110	0101	0011	0101	
8 11 2 13	0101	1001	1100	1010	12 7 2 13	1001	0101	1100	1010	
10 5 16 3	1010	0110	0011	1010	6 9 16 3	0110	1010	0011	1010	
15 4 9 6	1010	1001	1100	0101	15 4 5 10	1001	1010	1100	0101	
11#	2/	4/	1/	3/		12#	2/	4/	3/	1/
1 12 6 15	0101	0011	0101	0110	1 12 7 14	0101	0011	0110	0101	
8 13 3 10	0101	1100	1010	1001	8 13 2 11	0101	1100	1001	1010	
11 2 16 5	1010	0011	1010	0110	10 3 16 5	1010	0011	0110	1010	
14 7 9 4	1010	1100	0101	1001	15 6 9 4	1010	1100	1001	0101	
	*8	*4	*2	*1		*8	*4	*2	*1	

They show some solution examples of the pan-diagonal magic square of order 4: with the decomposition diagrams by Base 4(List 1) and the ones by Base 2(List 2).

As you see, no Complete Euler Squares can be found in the List 1, though some solutions have the properties of Latin Squares along with the 2 primary diagonals.

But in the List 2 all pan-diagonal magic squares of order 4 are always Complete Euler Squares decomposed by the Base 2 even with all pan-diagonals of each.

Watch the next diagrams below. The figure on the left most is a solution in classical style. The next four figures on the right show the Decomposed Layers of the same solution by binary system.



The next equations are set to be true among these figures:

$$\begin{aligned}
 1(\text{Dec}) &= 0*8 + 0*4 + 0*2 + 0*1 + 1; & 15(\text{Dec}) &= 1*8 + 1*4 + 1*2 + 0*1 + 1; \\
 10(\text{Dec}) &= 1*8 + 0*4 + 0*2 + 1*1 + 1; & 8(\text{Dec}) &= 0*8 + 1*4 + 1*2 + 0*1 + 1;
 \end{aligned}$$

$$\begin{aligned}
14(\text{Dec}) &= 1*8 + 1*4 + 0*2 + 1*1 + 1; & 4(\text{Dec}) &= 0*8 + 0*4 + 1*2 + 1*1 + 1; \\
5(\text{Dec}) &= 0*8 + 1*4 + 0*2 + 0*1 + 1; & 11(\text{Dec}) &= 1*8 + 0*4 + 1*2 + 0*1 + 1; \\
&\dots & & \\
3(\text{Dec}) &= 0*8 + 0*4 + 1*2 + 0*1 + 1; & 13(\text{Dec}) &= 1*8 + 1*4 + 0*2 + 0*1 + 1; \\
Vn(\text{Dec}) &= An*8 + Bn*4 + Cn*2 + Dn*1 + 1 \quad (n=1, 2, 3, 4, \dots, 15, 16)
\end{aligned}$$

(1) In each layer every row, every column and every pan-diagonal consist of {0,0,1,1}, {0,1,0,1}, {0,1,1,0}, {1,0,0,1}, {1,0,1,0} or {1,1,0,0} using '0' twice and also '1' as often. Therefore all sums of those rows, columns and pan-diagonals are calculated in the same form as:

$$\begin{aligned}
&\{0x2+1x2\}*8 + \{0x2+1x2\}*4 + \{0x2+1x2\}*2 + \{0x2+1x2\}*1 \\
&= \{0x2+1x2\}*\{8 + 4 + 2 + 1\} = 2x15 = 30(\text{Dec})
\end{aligned}$$

This makes every sum take the same value, that means 'magic constant'.

30(Dec) here is calculated and written in the 'mathematical' notation, which is made of the series of integers: 0~15. It is equivalent to 34(Dec) of classical notation, which is made of the series of natural numbers: 1~16.

(2) Each layer consists of eight '0' and as many '1' as the former. This property is always true whenever the condition (1) is true.

(3) Every combination of 4 values of the corresponding position in each layer must be any one of {0000, 0001, 0010, 0011, 0101, ..., , 1101, 1110, 1111(N2i)}, and neither repetition nor drop-off of any value must be taken.

This condition is logically equivalent to one of the most basic promises of our classical style that we must use the series of natural numbers 1~16 to make the object and use each number strictly once and must not use any number twice or more often in each solution.

Although they say Legendary Leonhard Euler(1707-1783) did not mention even about the pan-diagonals of 'Greco-Latin Squares' (Prof. M. Suzuki taught me), I want all pan-diagonals to accept the first condition of 'Complete Euler Squares' above and I would call the "Complete Euler Squares" of order 4 for those which take it as true.

2. Our Purpose

What I want to do now is to reconstruct all Pan-diagonal Magic Squares of order 4 only by these definitions and binary decompositions for each solution.

We need the set of 4 layers of binary decompositions for each.

We need to compose those 8 layers in all, the same set as listed above.

We need to know how to compose those layer units and how to pick up four pieces and combine them.

(1) First of all let's build all the necessary layer units from one.

(2) Let's pick up any 4 units to combine and calculate, and let's assume them as the set of 4 binary decompositions.

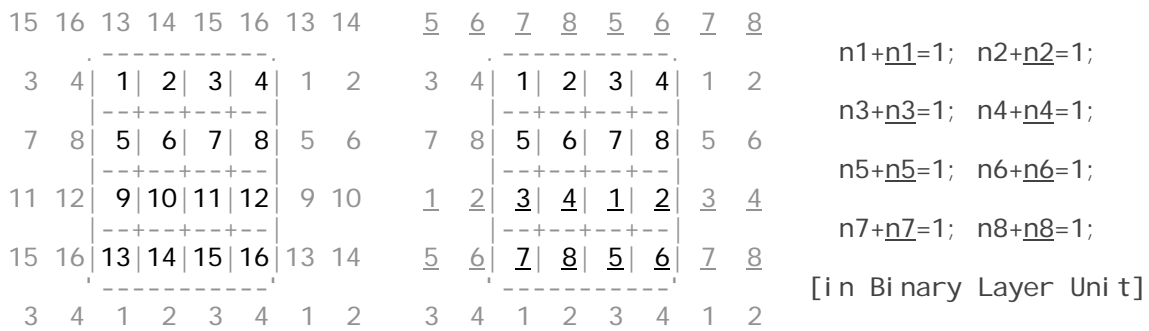
(3) We must know how to combine them exactly to make only correct solutions.

3. Composition of Layer Units

Even each layer of binary decompositions of C.E.S. must accept every definition of them, and it must also accept all the basic conditions of Pan-diagonal Magic Squares of order 4 in its own way. Since every pan-diagonal magic square 4x4 is always the 'composite and complete' magic square of order 4, all complementary pairs must be located in each layer just in the same manner as illustrated below:

[Basic Positions]

[Complementary Pairs]



$$\begin{aligned}
 n1+n2+n3+n4 &= n5+n6+n7+n8 = n9+n10+n11+n12 = n13+n14+n15+n16 = 2; \\
 n1+n5+n9+n13 &= n2+n6+n10+n14 = n3+n7+n11+n15 = n4+n8+n12+n16 = 2; \\
 n1+n6+n11+n16 &= n2+n7+n12+n13 = n3+n8+n9+n14 = n4+n5+n10+n15 = 2; \\
 n1+n8+n11+n14 &= n2+n5+n12+n15 = n3+n6+n9+n16 = n4+n7+n10+n13 = 2;
 \end{aligned}$$

$$\begin{aligned}
 n1+n2+n5+n6 &= n2+n3+n6+n7 = n3+n4+n7+n8 = n4+n1+n8+n5 = 2; \\
 n5+n6+n9+n10 &= n6+n7+n10+n11 = n7+n8+n11+n12 = n8+n5+n12+n9 = 2; \\
 n9+n10+n13+n14 &= n10+n11+n14+n15 = n11+n12+n15+n16 = n12+n9+n16+n13 = 2; \\
 n13+n14+n1+n2 &= n14+n15+n2+n3 = n15+n16+n3+n4 = n16+n13+n4+n1 = 2;
 \end{aligned}$$

Of course, all equations above must take any one of the following value patterns:
 $\{0, 0, 1, 1\}, \{0, 1, 0, 1\}, \{0, 1, 1, 0\}, \{1, 0, 0, 1\}, \{1, 0, 1, 0\}, \{1, 1, 0, 0\}.$

How many units can you make with such strict conditions as above?
 Let's try to make a sample of those units directly by hand.
 Suppose one of the primary diagonals is put as [Fig.1].

[Fig. 1]	[Fig. 2]	[Fig. 3]	[Fig. 4]
0 - - -	0 1 - -	0 1 1 0	0 1 0 1
- 0 - -	1 0 - -	1 0 0 1	1 0 1 0
- - 1 -	- - 1 0	0 1 1 0	1 0 1 0
- - - 1	- - 0 1	1 0 0 1	0 1 0 1

Then you can make [Fig.2] automatically by those conditions above.
 If you put '0' in n4 on the right shoulder, you can make [Fig.3] automatically.
 If you put '1' in n4, you can automatically make [Fig.4].
 But [Fig.4] proves to be the mirror-reflected image of [Fig.3]. Both of them are considered to be the different faces of the same thing.
 Is there only one unit available?

** 8 Units for Binary Layers **

1/	2/	3/	4/
0 1 0 1	0 1 0 1	0 1 1 0	0 0 1 1
1 0 1 0	0 1 0 1	1 0 0 1	1 1 0 0
1 0 1 0	1 0 1 0	0 1 1 0	0 0 1 1
0 1 0 1	1 0 1 0	1 0 0 1	1 1 0 0
5/	6/	7/	8/
1 1 0 0	1 0 0 1	1 0 1 0	1 0 1 0
0 0 1 1	0 1 1 0	1 0 1 0	0 1 0 1
1 1 0 0	1 0 0 1	0 1 0 1	0 1 0 1
0 0 1 1	0 1 1 0	0 1 0 1	1 0 1 0

[Count = 8]

I checked the solution list I had got before and counted how many patterns are used to make the layers of solutions.

I found we need the only eight units above in all and they are enough to make everything.

Any pair of {1/, 3/}, {2/, 4/}, {5/, 8/} and {6/, 7/} is mirror-reflected to each other.

You can easily transform 1/ -> 2/ -> 8/ -> 7/ -> 1/ by repeating simple rotation of rows, and transform 3/ -> 4/ -> 6/ -> 5/ -> 3/ by rotation of columns. They are just the unique movements any pan-diagonal magic squares would often show.

4. Take 4 Layers of Decompositions and Combine them.

These 8 units are so similar to one another that they should be considered as different faces of the same thing. But in reality they act as the individual units enough to make the set of 4 layers of decompositions for each different solution.

Let's make some experiments to know what we have to do with them.

Let's pick up any 4 units out of the 8 ones above and assume them as the set of 4 layers of decompositions. Let's combine them and compose a solution.

Let's begin with {1/,1/,1/,1/}, {1/,1/,1/,2/}, {1/,1/,1/,3/}, ..., {1/,1/,2/,1/}, {1/,1/,2/,2/}, ..., {1/,2/,3/,4/}, {1/,2/,3/,5/}, {1/,2/,3/,6/}, ..., {2/,1/,1/,1/}, ..., and let's end with {8/,8/,8/,6/}, {8/,8/,8/,7/} and {8/,8/,8/,8/}.

I tested all 4096 cases (8x8x8x8=4096), and could have got the correct answers as many as 384 at last and find the other ones wrong.

I made each solution from the 4 layers of decompositions by calculating under the next equation below.

$$V_n = A_n * 8 + B_n * 4 + C_n * 2 + D_n * 1 + 1 \quad (n = 1, 2, 3, 4, \dots, 15, 16)$$

And I tested every answer by the definition (3) to know if any repetition or drop-off of any value happened to appear in the same answer. It is true I had to remove a lot of wrong answers by that test.

I also had to remove the imitations with simple reflections and rotations of the correct answers out of 384 by the list-forming inequality conditions:

$$n_1 < n_{16}; \quad n_1 < n_4; \quad n_1 < n_{13}; \quad \text{and} \quad n_2 > n_5;$$

I could get the complete list of 48 pan-diagonal magic squares of order 4.

* List of Standard Solutions: Used Units//// List Number New[Old] *

1/	2/	3/	4/	1[1]	1/	2/	3/	5/	2[13]
0101	0101	0110	0011	1 15 4 14	0101	0101	0110	1100	2 16 3 13
1010	0101	1001	1100	12 6 9 7	1010	0101	1001	0011	11 5 10 8
1010	1010	0110	0011	13 3 16 2	1010	1010	0110	1100	14 4 15 1
0101	1010	1001	1100	8 10 5 11	0101	1010	1001	0011	7 9 6 12
1/	2/	4/	3/	3[7]	1/	2/	4/	6/	4[19]
0101	0101	0011	0110	1 14 4 15	0101	0101	0011	1001	2 13 3 16
1010	0101	1100	1001	12 7 9 6	1010	0101	1100	0110	11 8 10 5
1010	1010	0011	0110	13 2 16 3	1010	1010	0011	1001	14 1 15 4
0101	1010	1100	1001	8 11 5 10	0101	1010	1100	0110	7 12 6 9
1/	2/	5/	3/	5[25]	1/	2/	5/	6/	6[35]
0101	0101	1100	0110	3 16 2 13	0101	0101	1100	1001	4 15 1 14
1010	0101	0011	1001	10 5 11 8	1010	0101	0011	0110	9 6 12 7
1010	1010	1100	0110	15 4 14 1	1010	1010	1100	1001	16 3 13 2
0101	1010	0011	1001	6 9 7 12	0101	1010	0011	0110	5 10 8 11
1/	2/	6/	4/	7[31]	1/	2/	6/	5/	8[41]
0101	0101	1001	0011	3 13 2 16	0101	0101	1001	1100	4 14 1 15
1010	0101	0110	1100	10 8 11 5	1010	0101	0110	0011	9 7 12 6
1010	1010	1001	0011	15 1 14 4	1010	1010	1001	1100	16 2 13 3
0101	1010	0110	1100	6 12 7 9	0101	1010	0110	0011	5 11 8 10

1/	3/	2/	4/		9[3]	1/	3/	2/	5/		10[15]
0101	0110	0101	0011	1	15 6 12	0101	0110	0101	1100	2	16 5 11
1010	1001	0101	1100	14	4 9 7	1010	1001	0101	0011	13	3 10 8
1010	0110	1010	0011	11	5 16 2	1010	0110	1010	1100	12	6 15 1
0101	1001	1010	1100	8	10 3 13	0101	1001	1010	0011	7	9 4 14
1/	3/	5/	2/		11[27]	1/	3/	5/	7/		12[37]
0101	0110	1100	0101	3	16 5 10	0101	0110	1100	1010	4	15 6 9
1010	1001	0011	0101	13	2 11 8	1010	1001	0011	1010	14	1 12 7
1010	0110	1100	1010	12	7 14 1	1010	0110	1100	0101	11	8 13 2
0101	1001	0011	1010	6	9 4 15	0101	1001	0011	0101	5	10 3 16

* List of Standard Solutions(to be continued): Used Units////[List Number] *

2/1/3/4/[2]	2/1/3/5/[14]	2/1/4/3/[8]	2/1/4/6/[20]	2/1/5/3/[26]	2/1/5/6/[36]
1 15 4 14	2 16 3 13	1 14 4 15	2 13 3 16	3 16 2 13	4 15 1 14
8 10 5 11	7 9 6 12	8 11 5 10	7 12 6 9	6 9 7 12	5 10 8 11
13 3 16 2	14 4 15 1	13 2 16 3	14 1 15 4	15 4 14 1	16 3 13 2
12 6 9 7	11 5 10 8	12 7 9 6	11 8 10 5	10 5 11 8	9 6 12 7
2/1/6/4/[32]	2/1/6/5/[42]	2/3/1/4/[4]	2/3/1/5/[16]	2/3/4/1/[9]	2/3/4/8/[21]
3 13 2 16	4 14 1 15	1 15 6 12	2 16 5 11	1 14 7 12	2 13 8 11
6 12 7 9	5 11 8 10	8 10 3 13	7 9 4 14	8 11 2 13	7 12 1 14
15 1 14 4	16 2 13 3	11 5 16 2	12 6 15 1	10 5 16 3	9 6 15 4
10 8 11 5	9 7 12 6	14 4 9 7	13 3 10 8	15 4 9 6	16 3 10 5
2/3/5/1/[28]	2/3/5/8/[38]	2/3/8/4/[33]	2/3/8/5/[43]	2/4/1/3/[11]	2/4/1/6/[23]
3 16 5 10	4 15 6 9	3 13 8 10	4 14 7 9	1 12 6 15	2 11 5 16
6 9 4 15	5 10 3 16	6 12 1 15	5 11 2 16	8 13 3 10	7 14 4 9
12 7 14 1	11 8 13 2	9 7 14 4	10 8 13 3	11 2 16 5	12 1 15 6
13 2 11 8	14 1 12 7	16 2 11 5	15 1 12 6	14 7 9 4	13 8 10 3
2/4/3/1/[12]	2/4/3/8/[24]	2/5/1/3/[45]	2/5/1/6/[47]	2/5/3/1/[46]	2/5/3/8/[48]
1 12 7 14	2 11 8 13	5 16 2 11	6 15 1 12	5 16 3 10	6 15 4 9
8 13 2 11	7 14 1 12	4 9 7 14	3 10 8 13	4 9 6 15	3 10 5 16
10 3 16 5	9 4 15 6	15 6 12 1	16 5 11 2	14 7 12 1	13 8 11 2
15 6 9 4	16 5 10 3	10 3 13 8	9 4 14 7	11 2 13 8	12 1 14 7
3/1/2/4/[5]	3/1/2/5/[17]	3/1/5/2/[29]	3/1/5/7/[39]	3/2/1/4/[6]	3/2/1/5/[18]
1 15 10 8	2 16 9 7	3 16 9 6	4 15 10 5	1 15 10 8	2 16 9 7
14 4 5 11	13 3 6 12	13 2 7 12	14 1 8 11	12 6 3 13	11 5 4 14
7 9 16 2	8 10 15 1	8 11 14 1	7 12 13 2	7 9 16 2	8 10 15 1
12 6 3 13	11 5 4 14	10 5 4 15	9 6 3 16	14 4 5 11	13 3 6 12
3/2/4/1/[10]	3/2/4/8/[22]	3/2/5/1/[30]	3/2/5/8/[40]	3/2/8/4/[34]	3/2/8/5/[44]
1 14 11 8	2 13 12 7	3 16 9 6	4 15 10 5	3 13 12 6	4 14 11 5
12 7 2 13	11 8 1 14	10 5 4 15	9 6 3 16	10 8 1 15	9 7 2 16
6 9 16 3	5 10 15 4	8 11 14 1	7 12 13 2	5 11 14 4	6 12 13 3
15 4 5 10	16 3 6 9	13 2 7 12	14 1 8 11	16 2 7 9	15 1 8 10

[Count of Solutions = 48]

* Monitor List of Solution Correspondences *

?: 0
 1: 1
 25: 1

[OK!]

5. What Makes Us Compose Wrong Solutions?

I do not want to close my report right here, though I could get the complete list of 48 solutions of pan-diagonal magic squares 4x4.

I haven't yet known why I had to make wrong answers as many as 3712 in the process. Don't you think what a bad waste of energy and time it is?

We have to check some wrong answers and analyze them precisely.
Please watch the next parts of primitive data listed below.

* Examples of Primitive Data: Used Units//// List Number New[Old] *

1/	1/	1/	1/	1[??]	1/	1/	1/	2/	2[??]
0101	0101	0101	0101	1 16 1 16	0101	0101	0101	0101	1 16 1 16
1010	1010	1010	1010	16 1 16 1	1010	1010	1010	0101	15 2 15 2
1010	1010	1010	1010	16 1 16 1	1010	1010	1010	1010	16 1 16 1
0101	0101	0101	0101	1 16 1 16	0101	0101	0101	1010	2 15 2 15
1/	1/	1/	3/	3[??]	1/	1/	1/	4/	4[??]
0101	0101	0101	0110	1 16 2 15	0101	0101	0101	0011	1 15 2 16
1010	1010	1010	1001	16 1 15 2	1010	1010	1010	1100	16 2 15 1
1010	1010	1010	0110	15 2 16 1	1010	1010	1010	0011	15 1 16 2
0101	0101	0101	1001	2 15 1 16	0101	0101	0101	1100	2 16 1 15
1/	1/	1/	5/	5[??]	1/	1/	1/	6/	6[??]
0101	0101	0101	1100	2 16 1 15	0101	0101	0101	1001	2 15 1 16
1010	1010	1010	0011	15 1 16 2	1010	1010	1010	0110	15 2 16 1
1010	1010	1010	1100	16 2 15 1	1010	1010	1010	1001	16 1 15 2
0101	0101	0101	0011	1 15 2 16	0101	0101	0101	0110	1 16 2 15
1/	1/	1/	7/	7[??]	1/	1/	1/	8/	8[??]
0101	0101	0101	1010	2 15 2 15	0101	0101	0101	1010	2 15 2 15
1010	1010	1010	1010	16 1 16 1	1010	1010	1010	0101	15 2 15 2
1010	1010	1010	0101	15 2 15 2	1010	1010	1010	0101	15 2 15 2
0101	0101	0101	0101	1 16 1 16	0101	0101	0101	1010	2 15 2 15
1/	1/	2/	1/	9[??]	1/	1/	2/	2/	10[??]
0101	0101	0101	0101	1 16 1 16	0101	0101	0101	0101	1 16 1 16
1010	1010	0101	1010	14 3 14 3	1010	1010	0101	0101	13 4 13 4
1010	1010	1010	1010	16 1 16 1	1010	1010	1010	1010	16 1 16 1
0101	0101	1010	0101	3 14 3 14	0101	0101	1010	1010	4 13 4 13
1/	1/	2/	3/	11[??]	1/	1/	2/	4/	12[??]
0101	0101	0101	0110	1 16 2 15	0101	0101	0101	0011	1 15 2 16
1010	1010	0101	1001	14 3 13 4	1010	1010	0101	1100	14 4 13 3
1010	1010	1010	0110	15 2 16 1	1010	1010	1010	0011	15 1 16 2
0101	0101	1010	1001	4 13 3 14	0101	0101	1010	1100	4 14 3 13

* Examples of Primitive Data: Used Units////[Old Number] *

1/2/2/7/[??]	1/2/2/8/[??]	1/2/3/1/[??]	1/2/3/2/[??]	1/2/3/3/[??]	1/2/3/4/[1]
2 15 2 15	2 15 2 15	1 16 3 14	1 16 3 14	1 16 4 13	1 15 4 14
10 7 10 7	9 8 9 8	12 5 10 7	11 6 9 8	12 5 9 8	12 6 9 7
15 2 15 2	15 2 15 2	14 3 16 1	14 3 16 1	13 4 16 1	13 3 16 2
7 10 7 10	8 9 8 9	7 10 5 12	8 9 6 11	8 9 5 12	8 10 5 11
1/2/3/5/[25]	1/2/3/6/[??]	1/2/3/7/[??]	1/2/3/8/[??]	1/2/4/1/[??]	1/2/4/2/[??]
2 16 3 13	2 15 3 14	2 15 4 13	2 15 4 13	1 14 3 16	1 14 3 16
11 5 10 8	11 6 10 7	12 5 10 7	11 6 9 8	12 7 10 5	11 8 9 6
14 4 15 1	14 3 15 2	13 4 15 2	13 4 15 2	14 1 16 3	14 1 16 3
7 9 6 12	7 10 6 11	7 10 5 12	8 9 6 11	7 12 5 10	8 11 6 9
1/2/4/3/[7]	1/2/4/4/[??]	1/2/4/5/[??]	1/2/4/6/[31]	1/2/4/7/[??]	1/2/4/8/[??]
1 14 4 15	1 13 4 16	2 14 3 15	2 13 3 16	2 13 4 15	2 13 4 15
12 7 9 6	12 8 9 5	11 7 10 6	11 8 10 5	12 7 10 5	11 8 9 6
13 2 16 3	13 1 16 4	14 2 15 3	14 1 15 4	13 2 15 4	13 2 15 4
8 11 5 10	8 12 5 9	7 11 6 10	7 12 6 9	7 12 5 10	8 11 6 9
1/2/5/1/[??]	1/2/5/2/[??]	1/2/5/3/[49]	1/2/5/4/[??]	1/2/5/5/[??]	1/2/5/6/[73]
3 16 1 14	3 16 1 14	3 16 2 13	3 15 2 14	4 16 1 13	4 15 1 14
10 5 12 7	9 6 11 8	10 5 11 8	10 6 11 7	9 5 12 8	9 6 12 7
16 3 14 1	16 3 14 1	15 4 14 1	15 3 14 2	16 4 13 1	16 3 13 2
5 10 7 12	6 9 8 11	6 9 7 12	6 10 7 11	5 9 8 12	5 10 8 11

$1/2/5/7/[??]$ $1/2/5/8/[??]$ $1/2/6/1/[??]$ $1/2/6/2/[??]$ $1/2/6/3/[??]$ $1/2/6/4/[55]$
 4 15 2 13 4 15 2 13 3 14 1 16 3 14 1 16 3 14 2 15 3 13 2 16
 10 5 12 7 9 6 11 8 10 7 12 5 9 8 11 6 10 7 11 6 10 8 11 5
 15 4 13 2 15 4 13 2 16 1 14 3 16 1 14 3 15 2 14 3 15 1 14 4
 5 10 7 12 6 9 8 11 5 12 7 10 6 11 8 9 6 11 7 10 6 12 7 9
 $1/2/6/5/[79]$ $1/2/6/6/[??]$ $1/2/6/7/[??]$ $1/2/6/8/[??]$ $1/2/7/1/[??]$ $1/2/7/2/[??]$
 4 14 1 15 4 13 1 16 4 13 2 15 4 13 2 15 3 14 3 14 3 14 3 14
 9 7 12 6 9 8 12 5 10 7 12 5 9 8 11 6 12 5 12 5 11 6 11 6
 16 2 13 3 16 1 13 4 15 2 13 4 15 2 13 4 14 3 14 3 14 3 14 3
 5 11 8 10 5 12 8 9 5 12 7 10 6 11 8 9 5 12 5 12 6 11 6 11
 $1/3/4/5/[??]$ $1/3/4/6/[??]$ $1/3/4/7/[33]$ $1/3/4/8/[??]$ $1/3/5/1/[??]$ $1/3/5/2/[51]$
 2 14 7 11 2 13 7 12 2 13 8 11 2 13 8 11 3 16 5 10 3 16 5 10
 15 3 10 6 15 4 10 5 16 3 10 5 15 4 9 6 14 1 12 7 13 2 11 8
 10 6 15 3 10 5 15 4 9 6 15 4 9 6 15 4 12 7 14 1 12 7 14 1
 7 11 2 14 7 12 2 13 7 12 1 14 8 11 2 13 5 10 3 16 6 9 4 15
 $1/3/5/3/[??]$ $1/3/5/4/[??]$ $1/3/5/5/[??]$ $1/3/5/6/[??]$ $1/3/5/7/[75]$ $1/3/5/8/[??]$
 3 16 6 9 3 15 6 10 4 16 5 9 4 15 5 10 4 15 6 9 4 15 6 9
 14 1 11 8 14 2 11 7 13 1 12 8 13 2 12 7 14 1 12 7 13 2 11 8
 11 8 14 1 11 7 14 2 12 8 13 1 12 7 13 2 11 8 13 2 11 8 13 2
 6 9 3 16 6 10 3 15 5 9 4 16 5 10 4 15 5 10 3 16 6 9 4 15

I mean each of [??] above is the wrong answer we could find no corresponding solution to in the old list of 384 pan-diagonal MS44.

They are all against the definition (3) of 'Complete Euler Squares', since they have some repetitions and drop-offs of a certain value in each solution.

What makes them wrong against the definition (3)?

(1) When you use No.1 unit twice, three times or more often, you must always compose wrong answers.

Any repeating usage of a certain unit always makes your answers wrong.

You should avoid any combinations like $\{1/,1/,1/,1/\}$, $\{2/,1/,1/,2/\}$, $\{2/,3/,3/,4/\}$, ... , or $\{8/,3/,8/,6/\}$.

As you know, you can have such permutations as $8 \times 7 \times 6 \times 5$ in place of $8 \times 8 \times 8 \times 8$, when you avoid these repetitions.

(2) When you use No.1 with No.8, then you must always get wrong answers. When you use No.2 with No.7, you must make another mistake, too. Why?

**** Complementary Units ****

T0:	1/		8/	T1:	2/		7/	T2:	3/		6/
0 1 0 1	1 0 1 0	0 1 0 1	1 0 1 0	0 1 0 1	1 0 1 0	0 1 1 0	1 0 0 1	0 1 1 0	1 0 0 1	0 1 1 0	1 0 0 1
1 0 1 0	0 1 0 1	0 1 0 1	1 0 1 0	0 1 0 1	1 0 1 0	1 0 0 1	0 1 1 0	1 0 0 1	0 1 1 0	1 0 0 1	0 1 1 0
1 0 1 0	0 1 0 1	1 0 1 0	0 1 0 1	1 0 1 0	0 1 0 1	0 1 1 0	1 0 0 1	0 1 1 0	1 0 0 1	1 0 0 1	1 0 0 1
0 1 0 1	1 0 1 0	1 0 1 0	0 1 0 1	1 0 1 0	0 1 0 1	1 0 0 1	0 1 1 0	1 0 0 1	0 1 1 0	1 0 0 1	0 1 1 0
T3:	4/		5/								
0 0 1 1	1 1 0 0										
1 1 0 0	0 0 1 1	if $[A_n + B_n == 1 \ (n=1, 2, 3, 4, \dots, 15, 16)]$,									
0 0 1 1	1 1 0 0	then [Unit Pair(A, B) is 'Complementary'.]									
1 1 0 0	0 0 1 1										

No.1 and No.8 are 'Complementary Units' to each other, since every number in corresponding position is put with the negative value of the other one. '0' is always replaced with '1', and '1' is always replaced with '0'.

(No.2 & No.7), (No.3 & No.6) and (No.4 & No.5) are also Complementary Units.

'Complete Euler Squares' 4×4 always refuse any combination of complementary

units as well as any combination of the same units.

What you have to do before all is to choose either one out of each pair: T0(1, 8), T1(2, 7), T2(3, 6), T3(4, 5), and manage to take the permutation of those 4 units chosen.

You will get such combinations as {1/,2/,3/,4/}, {1/,2/,3/,5/}, {1/,2/,4/,6/}, {1/,2/,5/,6/}, {1/,3/,4/,7/}, ... , {1/,3/,5/,7/}, ... , {2/,3/,4/,8/}, {2/,3/,5/,8/}, {3/,4/,2/,8/}, {3/,2/,8/,5/}, ... as many as 384 (=2x2x2x2x4x3x2x1).

The next job you have to do is composing each solution with these combinations under the following equation:

$$V_n = A_n * 8 + B_n * 4 + C_n * 2 + D_n * 1 + 1 \quad (n=1, 2, 3, 4, \dots, 15, 16)$$

Next examples shown below are no longer wrong answers. But, they are against the list forming inequality conditions put to make the complete and compact list of 48 'standard solutions':

$n_1 < n_{16}$; $n_1 < n_4$; $n_1 < n_{13}$; and $n_2 > n_5$;

4/	1/	2/	3/	19[145]	4/	2/	1/	3/	20[153]
0011	0101	0101	0110	1 8 10 15	0011	0101	0101	0110	1 8 10 15
1100	1010	0101	1001	14 11 5 4	1100	0101	1010	1001	12 13 3 6
0011	1010	1010	0110	7 2 16 9	0011	1010	1010	0110	7 2 16 9
1100	0101	1010	1001	12 13 3 6	1100	1010	0101	1001	14 11 5 4
4/	1/	3/	2/	21[147]	4/	2/	3/	1/	22[155]
0011	0101	0110	0101	1 8 11 14	0011	0101	0110	0101	1 8 11 14
1100	1010	1001	0101	15 10 5 4	1100	0101	1001	1010	12 13 2 7
0011	1010	0110	1010	6 3 16 9	0011	1010	0110	1010	6 3 16 9
1100	0101	1001	1010	12 13 2 7	1100	1010	1001	0101	15 10 5 4

5/1/2/3/[193]	5/2/1/3/[201]	5/1/3/2/[195]	5/2/3/1/[203]	5/3/1/2/[209]	5/3/2/1/[211]
9 16 2 7	9 16 2 7	9 16 3 6	9 16 3 6	9 16 5 4	9 16 5 4
6 3 13 12	4 5 11 14	7 2 13 12	4 5 10 15	7 2 11 14	6 3 10 15
15 10 8 1	15 10 8 1	14 11 8 1	14 11 8 1	12 13 8 1	12 13 8 1
4 5 11 14	6 3 13 12	4 5 10 15	7 2 13 12	6 3 10 15	7 2 11 14

6/7/8/5/[280]	6/8/7/5/[288]	7/6/8/5/[328]	8/6/7/5/[376]	7/8/6/5/[336]	8/7/6/5/[384]
16 2 7 9	16 2 7 9	16 2 11 5	16 2 11 5	16 2 13 3	16 2 13 3
5 11 14 4	3 13 12 6	9 7 14 4	3 13 8 10	9 7 12 6	5 11 8 10
10 8 1 15	10 8 1 15	6 12 1 15	6 12 1 15	4 14 1 15	4 14 1 15
3 13 12 6	5 11 14 4	3 13 8 10	9 7 14 4	5 11 8 10	9 7 12 6

Apply the test with those inequality conditions to the 384 compositions, and you will surely have the compact list of 48 standard solutions of pan-diagonal MS44.

If you want to test them before composing solutions, you may select unit No.1, 2 or 3 for the highest layer. It is because only these 3 units can pass the test.

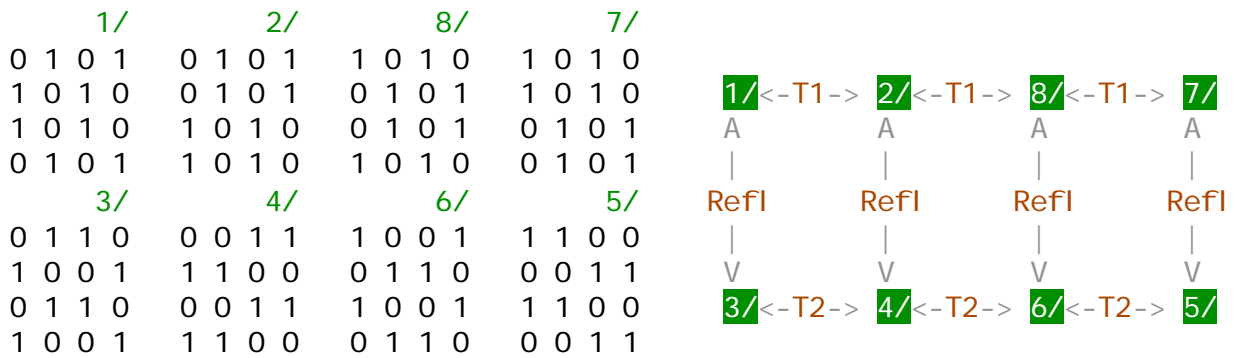
You will have got 144 compositions waiting for the final test of inequality conditions.

6. Summary: How to Compose Pan-diagonal C.E.S. 4x4 by Binary System

(1) At the beginning you have to compose a basic binary unit by hand as follows:

Step 1/	Step 2/	Step 3/	Step 4/
0 - -	0 1 - -	0 1 - 1	0 1 0 1
- 0 - -	1 0 - -	1 0 - -	1 0 1 0
- - 1 -	- - 1 0	- 0 1 0	1 0 1 0
- - - 1	- - 0 1	- - 0 1	0 1 0 1

**** Layer Units for Decompositions by Binary System ****



(2) Make 8 layer units from the first one by such transformation methods as shown above: Rotation of rows one by one (T1), Mirror reflection with respect to the first primary diagonal (Refl), and rotation of columns one by one (T2).

Check the result if all pairs {1/,8/}, {2/,7/}, {3/,6/}, and {4/,5/} are 'complementary units' with each other.

(3) Pick up 4 units to combine and assume them for 4 decomposed layers of each solution by binary number system.

Choose one representative unit out of each pair {1/, 8/}, {2/, 7/}, {3/, 6/}, and {4/, 5/}, and manage permutation of those 4 representatives to make the set of decomposed layers.

You have to make 384 combinations in all: {1/,2/,3/,4/}, {1/,2/,3/,5/}, {1/,2/,4/,6/}, {1/,2/,5/,6/}, {1/,3/,4/,7/}, ... , {1/,3/,5/,7/}, ... , {2/,3/,4/,8/}, {2/,3/,5/,8/}, {3/,4/,2/,8/}, {3/,2/,8/,5/}, ... (384 = 2x2x2x2x4x3x2x1).

If you select only {1,2 or 3} for the highest layer, you don't have to make any more than 144 combinations in all.

(4) Compose each solution by combining 4 binary layers and calculating every corresponding value under the next equation:

$$V_n = A_n * 8 + B_n * 4 + C_n * 2 + D_n * 1 + 1 \quad (n=1, 2, 3, 4, \dots, 15, 16);$$

(5) Apply the test to the result by the following inequality conditions:

$$n_1 < n_{16}; \quad n_1 < n_4; \quad n_1 < n_{13}; \quad \text{and} \quad n_2 > n_5;$$

(6) If you want to apply the test of definition (3) to the result, you may add such a program as:

```

/**/
/* Procedure: Combine and Compose */
void cmbcmp() {
    short n, d, fc;
    /* . . . . */
    for(n=1; n<17; n++){ uflg[n]=0; }
    for(n=1; n<17; n++){
        d=tlu[u1][n]*8+tlu[u2][n]*4+tlu[u3][n]*2+tlu[u4][n]+1;
        nm[n]=d; uflg[d]++;
    }
    fc=0;
    for(n=1; n<17; n++){ if(uflg[n]==1){ fc++; } else{ break; } }
    if(fc==16){
        if((nm[1]<nm[16])&&(nm[1]<nm[4])&&(nm[1]<nm[13])&&(nm[2]>nm[5])){
            nextproc(); }
    }
}
/* . . . . */
}
/**/

```

But, no influence will actually be found, even when you check them by the test of definition (3). The 384 compositions are no longer wrong answers.

One eighth of all must be taken for the 'standard solutions'. The rest seven eighth are only 'imitations' made by simple reflection or rotation of standard solutions.

7. Result of Compositions: 48 Pan-diagonal C.E.S. 4x4

Let me show you my recent result of compositions by Binary Number System.

** Reconstructi on of Pan-di agonal Magi c Squares of Order 4 **
 ** by Bi nary Number System **

[Layer Uni ts for Bi nary Decomposi ti ons]

1/	2/	3/	4/
0 1 0 1	0 1 0 1	0 1 1 0	0 0 1 1
1 0 1 0	0 1 0 1	1 0 0 1	1 1 0 0
1 0 1 0	1 0 1 0	0 1 1 0	0 0 1 1
0 1 0 1	1 0 1 0	1 0 0 1	1 1 0 0
5/	6/	7/	8/
1 1 0 0	1 0 0 1	1 0 1 0	1 0 1 0
0 0 1 1	0 1 1 0	1 0 1 0	0 1 0 1
1 1 0 0	1 0 0 1	0 1 0 1	0 1 0 1
0 0 1 1	0 1 1 0	0 1 0 1	1 0 1 0

[Count = 8]

* Li st of Standard Soluti ons: Used Uni ts//// Li st Number Ol d[New] *

1/	2/	3/	4/	1[1]	2/	1/	3/	4/	2[13]
0101	0101	0110	0011	1 15 4 14	0101	0101	0110	0011	1 15 4 14
1010	0101	1001	1100	12 6 9 7	0101	1010	1001	1100	8 10 5 11
1010	1010	0110	0011	13 3 16 2	1010	1010	0110	0011	13 3 16 2
0101	1010	1001	1100	8 10 5 11	1010	0101	1001	1100	12 6 9 7
1/	3/	2/	4/	3[9]	2/	3/	1/	4/	4[21]
0101	0110	0101	0011	1 15 6 12	0101	0110	0101	0011	1 15 6 12
1010	1001	0101	1100	14 4 9 7	0101	1001	1010	1100	8 10 3 13
1010	0110	1010	0011	11 5 16 2	1010	0110	1010	0011	11 5 16 2
0101	1001	1010	1100	8 10 3 13	1010	1001	0101	1100	14 4 9 7
3/	1/	2/	4/	5[37]	3/	2/	1/	4/	6[41]
0110	0101	0101	0011	1 15 10 8	0110	0101	0101	0011	1 15 10 8
1001	1010	0101	1100	14 4 5 11	1001	0101	1010	1100	12 6 3 13
0110	1010	1010	0011	7 9 16 2	0110	1010	1010	0011	7 9 16 2
1001	0101	1010	1100	12 6 3 13	1001	1010	0101	1100	14 4 5 11
1/	2/	4/	3/	7[3]	2/	1/	4/	3/	8[15]
0101	0101	0011	0110	1 14 4 15	0101	0101	0011	0110	1 14 4 15
1010	0101	1100	1001	12 7 9 6	0101	1010	1100	1001	8 11 5 10
1010	1010	0011	0110	13 2 16 3	1010	1010	0011	0110	13 2 16 3
0101	1010	1100	1001	8 11 5 10	1010	0101	1100	1001	12 7 9 6
2/	3/	4/	1/	9[23]	3/	2/	4/	1/	10[43]
0101	0110	0011	0101	1 14 7 12	0110	0101	0011	0101	1 14 11 8
0101	1001	1100	1010	8 11 2 13	1001	0101	1100	1010	12 7 2 13
1010	0110	0011	1010	10 5 16 3	0110	1010	0011	1010	6 9 16 3
1010	1001	1100	0101	15 4 9 6	1001	1010	1100	0101	15 4 5 10
2/	4/	1/	3/	11[29]	2/	4/	3/	1/	12[31]
0101	0011	0101	0110	1 12 6 15	0101	0011	0110	0101	1 12 7 14
0101	1100	1010	1001	8 13 3 10	0101	1100	1001	1010	8 13 2 11
1010	0011	1010	0110	11 2 16 5	1010	0011	0110	1010	10 3 16 5
1010	1100	0101	1001	14 7 9 4	1010	1100	1001	0101	15 6 9 4

Both of them have the common logical equivalence with the same 48 solutions.

(2) It is amazing that we know only 8 layer units can make everything. The way of combination and composition seems to tell us anything like the 'structures.'

(3) It is also amazing that we know how to calculate the total counts of solutions.

$$2 \times 2 \times 2 \times 2 \times 4 \times 3 \times 2 \times 1 \times (1/8) = 48$$

The factor 1/8 is needed when we select the 'standard solutions' under the list-forming conditions: $n_1 < n_{16}$; $n_1 < n_4$; $n_1 < n_{13}$; and $n_2 > n_5$;

(4) What we now know about C.E.S. of order 4 is so similar to what about C.E.S. of order 5 that both of them have the similar equivalence to each other, I consider.

(5) But we don't yet know about the 3 fundamental solutions and the meaning of the equation: $48 = 3 \times 16$ by this method. Another study is necessary to be done.

(6) The next job we have to do is studying about various types of C.E.S. of order 8.

9. Revision

While I was studying Complete Euler Squares of order 8, I got some new knowledge about how to make C.E.S. of any order. I think I have to make it return here and do some revisions for this article about the next two points:

(1) We once made the eight layer units for order 4 by hand, helped by the knowledge we had got before in another study. But how about in the case of higher orders? Can we really do the same thing? Can we exactly know in advance how many layer units we should prepare to reconstruct the complete set of object solutions?

We want to have another way how to make 8 layer units of order 4 here, invented originally by ourselves without consulting with any other knowledge we got before. Whenever we make any type of C.E.S., we want to have an independent, universal method of composition, I mean, any appropriate computer program for the purpose.

(2) We also have to refine our check program to prevent us from making any wrong answers against the definition (3) of C.E.S.

9-1. How can we make these Layer Units below by ourselves?

[Layer Units for Binary Decompositions]

1/	2/	3/	4/
0 1 0 1	0 1 0 1	0 1 1 0	0 0 1 1
1 0 1 0	0 1 0 1	1 0 0 1	1 1 0 0
1 0 1 0	1 0 1 0	0 1 1 0	0 0 1 1
0 1 0 1	1 0 1 0	1 0 0 1	1 1 0 0
5/	6/	7/	8/
1 1 0 0	1 0 0 1	1 0 1 0	1 0 1 0
0 0 1 1	0 1 1 0	1 0 1 0	0 1 0 1
1 1 0 0	1 0 0 1	0 1 0 1	0 1 0 1
0 0 1 1	0 1 1 0	0 1 0 1	1 0 1 0

[Count = 8]

In any unit every row, every column and every pan-diagonal must consist of two '0' and two '1' like {0,0,1,1}, {0,1,0,1}, {0,1,1,0}, {1,0,0,1}, {1,0,1,0} or {1,1,0,0}.

How can we get it and dictate our computer program?

Why don't we make such an ordinary program for magic squares as usual only by Binary System? The next list shows my recent program, the simplest version.

```
/** Reconstruction of Pan-diagonal Magic Squares **/
/** of Order 4 by the "New Euler's Method" **/
/** 'CES44PD.c' dictated by Kanji Setsuda **/
/** on Oct. 22, 2003; Mar. 14, 2006 **/
```

```

/** Working on MacOSX and Xcode 2.1 */
/**/
#include <stdio.h>
short cnt;
short u1, u2, u3, u4;
short nm[17], uflg[17];
short tlu[9][17];
short mtc[9][9];
short tans[49][22];
/**/
short rw1[2], cl1[2], pd1[2], pb1[2];
short rw2[2], cl2[2], pd2[2], pb2[2];
short rw3[2], cl3[2], pd3[2], pb3[2];
short rw4[2], cl4[2], pd4[2], pb4[2];
/**/
void stp01(void), stp02(void), stp03(void);
void stp04(void), stp05(void), stp06(void);
void stp07(void), stp08(void), stp09(void);
void stp10(void), stp11(void), stp12(void);
void stp13(void), stp14(void), stp15(void);
void stp16(void);
void lurecord(void);
void prlunit(void), cmbcmp(void), recordans(void);
void pr2ans(short x), pr6ans(short x, short y);
/**/
int main(){
short n;
printf("\n** Reconstruction of Pan-diagonal Magic Squares of Order 4 **\n");
printf("** by the 'New Euler's Method' with Binary Number System **\n");
for(n=0; n<17; n++){nm[n]=0; uflg[n]=0;}
for(n=0; n<2; n++){
rw1[n]=0; cl1[n]=0; pd1[n]=0; pb1[n]=0;
rw2[n]=0; cl2[n]=0; pd2[n]=0; pb2[n]=0;
rw3[n]=0; cl3[n]=0; pd3[n]=0; pb3[n]=0;
rw4[n]=0; cl4[n]=0; pd4[n]=0; pb4[n]=0;
}
cnt=0;
stp01(); /* Make Layer Units */
prlunit(); /* Print the List of Units */
cnt=0;
cmbcmp(); /* New Euler's Method */
pr2ans(12);
pr6ans(13, cnt);
printf(" [Count of Solutions = %d] OK!\n", cnt);
return 0;
}
/**/
/* Make Layer Units */
/* Set n1 */
void stp01(){
short a;
for(a=0; a<2; a++){
if((rw1[a]<2)&&(cl1[a]<2)&&(pd1[a]<2)&&(pb1[a]<2)){
nm[1]=a;
rw1[a]++; cl1[a]++; pd1[a]++; pb1[a]++;
stp02();
rw1[a]--; cl1[a]--; pd1[a]--; pb1[a]--;
}}
}
}

```

```

/* Set n2 */
void stp02(){
  short a;
  for(a=1; a>=0; a--){
    if((rw1[a]<2)&&(cl 2[a]<2)&&(pd2[a]<2)&&(pb2[a]<2)){
      nm[2]=a;
      rw1[a]++; cl 2[a]++; pd2[a]++; pb2[a]++;
      stp03();
      rw1[a]--; cl 2[a]--; pd2[a]--; pb2[a]--;
    }
  }
}
/* Set n3 */
void stp03(){
  short a;
  for(a=0; a<2; a++){
    if((rw1[a]<2)&&(cl 3[a]<2)&&(pd3[a]<2)&&(pb3[a]<2)){
      nm[3]=a;
      rw1[a]++; cl 3[a]++; pd3[a]++; pb3[a]++;
      stp04();
      rw1[a]--; cl 3[a]--; pd3[a]--; pb3[a]--;
    }
  }
}
/* Set n4 */
void stp04(){
  short a;
  for(a=1; a>=0; a--){
    if((rw1[a]<2)&&(cl 4[a]<2)&&(pd4[a]<2)&&(pb4[a]<2)){
      nm[4]=a;
      rw1[a]++; cl 4[a]++; pd4[a]++; pb4[a]++;
      stp05();
      rw1[a]--; cl 4[a]--; pd4[a]--; pb4[a]--;
    }
  }
}
/* Set n5 */
void stp05(){
  short a;
  for(a=1; a>=0; a--){
    if((rw2[a]<2)&&(cl 1[a]<2)&&(pd4[a]<2)&&(pb2[a]<2)){
      nm[5]=a;
      rw2[a]++; cl 1[a]++; pd4[a]++; pb2[a]++;
      stp06();
      rw2[a]--; cl 1[a]--; pd4[a]--; pb2[a]--;
    }
  }
}
/* Set n6 */
void stp06(){
  short a;
  for(a=0; a<2; a++){
    if((rw2[a]<2)&&(cl 2[a]<2)&&(pd1[a]<2)&&(pb3[a]<2)){
      nm[6]=a;
      rw2[a]++; cl 2[a]++; pd1[a]++; pb3[a]++;
      stp07();
      rw2[a]--; cl 2[a]--; pd1[a]--; pb3[a]--;
    }
  }
}
/* Set n7 */
void stp07(){
  short a;
  for(a=1; a>=0; a--){

```

```

    if((rw2[a]<2)&&(cl 3[a]<2)&&(pd2[a]<2)&&(pb4[a]<2)){
        nm[7]=a;
        rw2[a]++; cl 3[a]++; pd2[a]++; pb4[a]++;
        stp08();
        rw2[a]--; cl 3[a]--; pd2[a]--; pb4[a]--;
    }
}
/* Set n8 */
void stp08(){
    short a;
    for(a=0; a<2; a++){
        if((rw2[a]<2)&&(cl 4[a]<2)&&(pd3[a]<2)&&(pb1[a]<2)){
            nm[8]=a;
            rw2[a]++; cl 4[a]++; pd3[a]++; pb1[a]++;
            stp09();
            rw2[a]--; cl 4[a]--; pd3[a]--; pb1[a]--;
        }
    }
}
/* Set n9 */
void stp09(){
    short a;
    for(a=1; a>=0; a--){
        if((rw3[a]<2)&&(cl 1[a]<2)&&(pd3[a]<2)&&(pb3[a]<2)){
            nm[9]=a;
            rw3[a]++; cl 1[a]++; pd3[a]++; pb3[a]++;
            stp10();
            rw3[a]--; cl 1[a]--; pd3[a]--; pb3[a]--;
        }
    }
}
/* Set n10 */
void stp10(){
    short a;
    for(a=0; a<2; a++){
        if((rw3[a]<2)&&(cl 2[a]<2)&&(pd4[a]<2)&&(pb4[a]<2)){
            nm[10]=a;
            rw3[a]++; cl 2[a]++; pd4[a]++; pb4[a]++;
            stp11();
            rw3[a]--; cl 2[a]--; pd4[a]--; pb4[a]--;
        }
    }
}
/* Set n11 */
void stp11(){
    short a;
    for(a=1; a>=0; a--){
        if((rw3[a]<2)&&(cl 3[a]<2)&&(pd1[a]<2)&&(pb1[a]<2)){
            nm[11]=a;
            rw3[a]++; cl 3[a]++; pd1[a]++; pb1[a]++;
            stp12();
            rw3[a]--; cl 3[a]--; pd1[a]--; pb1[a]--;
        }
    }
}
/* Set n12 */
void stp12(){
    short a;
    for(a=0; a<2; a++){
        if((rw3[a]<2)&&(cl 4[a]<2)&&(pd2[a]<2)&&(pb2[a]<2)){
            nm[12]=a;
            rw3[a]++; cl 4[a]++; pd2[a]++; pb2[a]++;
            stp13();

```

```

        rw3[a]--; cl 4[a]--; pd2[a]--; pb2[a]--;
    }}
}
/* Set n13 */
void stp13(){
    short a;
    for(a=0; a<2; a++){
        if((rw4[a]<2)&&(cl 1[a]<2)&&(pd2[a]<2)&&(pb4[a]<2)){
            nm[13]=a;
            rw4[a]++; cl 1[a]++; pd2[a]++; pb4[a]++;
            stp14();
            rw4[a]--; cl 1[a]--; pd2[a]--; pb4[a]--;
        }}
}
/* Set n14 */
void stp14(){
    short a;
    for(a=1; a>=0; a--){
        if((rw4[a]<2)&&(cl 2[a]<2)&&(pd3[a]<2)&&(pb1[a]<2)){
            nm[14]=a;
            rw4[a]++; cl 2[a]++; pd3[a]++; pb1[a]++;
            stp15();
            rw4[a]--; cl 2[a]--; pd3[a]--; pb1[a]--;
        }}
}
/* Set n15 */
void stp15(){
    short a;
    for(a=0; a<2; a++){
        if((rw4[a]<2)&&(cl 3[a]<2)&&(pd4[a]<2)&&(pb2[a]<2)){
            nm[15]=a;
            rw4[a]++; cl 3[a]++; pd4[a]++; pb2[a]++;
            stp16();
            rw4[a]--; cl 3[a]--; pd4[a]--; pb2[a]--;
        }}
}
/* Set n16 */
void stp16(){
    short a;
    for(a=1; a>=0; a--){
        if((rw4[a]<2)&&(cl 4[a]<2)&&(pd1[a]<2)&&(pb3[a]<2)){
            nm[16]=a;
            rw4[a]++; cl 4[a]++; pd1[a]++; pb3[a]++;
            lurecord();
            rw4[a]--; cl 4[a]--; pd1[a]--; pb3[a]--;
        }}
}
/**/
/* Record the Layer Units */
void lurecord(){
    short n;
    cnt++;
    tlu[cnt-1][0]=cnt;
    for(n=1; n<17; n++){tlu[cnt-1][n]=nm[n];}
}
/**/
/* Make Matching Table and Print the Layer Units */
void prluni t(){
    short t, l, l4, m, n;

```

```

printf("\n [Layer Units for Binary Decompositions]\n");
for(t=0; t<cnt; t=t+4){
    printf("%9d/%9d/%9d/%9d/\n", t+1, t+2, t+3, t+4);
    for(l=0; l<4; l++){l4=l*4;
        for(m=t; m<(t+4); m++){
            printf(" ");
            for(n=1; n<5; n++){printf("%2d", tlu[m][l4+n]); }
        }
        printf("\n");
    }
}
printf(" [Count = %2d]\n", cnt);
for(m=0; m<cnt; m++){
    for(n=0; n<cnt; n++){
        t=0;
        for(l=1; l<17; l++){if(tlu[m][l]==tlu[n][l]){t++; }}
        mtc[m][n]=t;
    }
}
printf("\n [Reference Table for the Best Combination]\n");
printf(" *|");
for(n=0; n<cnt; n++){printf("%3d", n+1); }
printf("\n");
printf(" ---+-----\n");
for(m=0; m<cnt; m++){
    printf("%3d|", m+1);
    for(n=0; n<cnt; n++){t=mtc[m][n];
        if(t>=0){printf("%3d", t); }else{printf(" -"); }}
    printf("\n");
}
}
/**/
/* Combine and Compose */
void cmbcmp(){
short lu, luh, md, n, d, fc;
lu=8; luh=lu/2; md=8;
for(u1=0; u1<luh; u1++){
    for(u2=0; u2<lu; u2++){
        if(mtc[u2][u1]==md){
            for(u3=0; u3<lu; u3++){
                if((mtc[u3][u1]==md)&&(mtc[u3][u2]==md)){
                    for(u4=0; u4<lu; u4++){
                        if((mtc[u4][u1]==md)&&(mtc[u4][u2]==md)&&(mtc[u4][u3]==md)){
                            for(n=1; n<17; n++){uflg[n]=0; }
                            for(n=1; n<17; n++){
                                d=tlu[u1][n]*8+tlu[u2][n]*4+tlu[u3][n]*2+tlu[u4][n]+1;
                                nm[n]=d; uflg[d]++;
                            }
                            fc=0;
                            for(n=1; n<17; n++){if(uflg[n]==1){fc++; }else{break; }}
                            if(fc==16){
                                if((nm[1]<nm[16])&&(nm[1]<nm[4])&&(nm[1]<nm[13])&&(nm[2]>nm[5])){
                                    recordans(); }
                            }
                        }
                    }
                }
            }
        }
    }
}
}
/**/
/* Record the Answers */

```

```

void recordans(){
    short n;
    tans[cnt][0]=cnt+1;
    for(n=1;n<17;n++){tans[cnt][n]=nm[n];}
    tans[cnt][18]=u1; tans[cnt][19]=u2;
    tans[cnt][20]=u3; tans[cnt][21]=u4;
    cnt++;
}
/**/
/* Print 2 Answers */
void pr2ans(short x){
    short t, l, l4, m, n, s, v;
    printf("\n* List of Standard Solutions: Used Units///// Solution Number# *\n");
    for(t=0; t<x; t=t+2){
        for(m=t; m<(t+2); m++){
            printf("%6d/%4d/%4d/%4d/%12d#",
                tans[m][18]+1, tans[m][19]+1, tans[m][20]+1, tans[m][21]+1, tans[m][0]); }
        printf("\n");
        for(l=0; l<4; l++){l4=l*4;
            for(s=t; s<(t+2); s++){
                printf(" ");
                for(m=1; m<5; m++){
                    printf(" ");
                    for(n=1; n<5; n++){v=tlu[tans[s][m+17]][l4+n];
                        printf("%d", v); }
                }
                printf(" ");
                for(n=1; n<5; n++){printf("%3d", tans[s][l4+n]); }
            }
        }
        printf("\n");
    }
}
/**/
/* Print 6 Answers */
void pr6ans(short x, short y){
    short t, l, l4, m, n;
    printf("\n* List of Standard Solutions(to be continued): Used Units///// Number# *\n");
    for(t=(x-1); t<y; t=t+6){
        for(m=t; m<(t+6); m++){
            printf("%3d/%d/%d/%d/%3d#",
                tans[m][18]+1, tans[m][19]+1, tans[m][20]+1, tans[m][21]+1, tans[m][0]); }
        printf("\n");
        for(l=0; l<4; l++){l4=l*4;
            for(m=t; m<(t+6); m++){
                printf(" ");
                for(n=1; n<5; n++){printf("%3d", tans[m][l4+n]); }
                printf(" ");
            }
        }
        printf("\n");
    }
}
/**/

```

9-2. How can we improve our check program for the purpose of eliminating any wrong answers?

I would like to propose you an idea to put such a reference table as shown below on the memory array mtc[9][9] and to use it wisely before you combine any units to

compose the object.

[Reference Table for the Best Combination]

*	1	2	3	4	5	6	7	8
1	16	8	8	8	8	8	8	0
2	8	16	8	8	8	8	0	8
3	8	8	16	8	8	0	8	8
4	8	8	8	16	0	8	8	8
5	8	8	8	0	16	8	8	8
6	8	8	0	8	8	16	8	8
7	8	0	8	8	8	8	16	8
8	0	8	8	8	8	8	8	16

This table shows how similar each unit is to the others. I took every two units and compared them to count up if the values of corresponding positions are the same.

Every value of (u1,u1), (u2,u2), (u3,u3), ... (u8,u8) is 16. It means all are the same. As you see, it is natural because I compared the one with itself.

Every value of (u1,u8), (u2,u7), (u3,u6), ... , (u7,u2), (u8,u1) is 0. It means all are different. It tells us that these unit pairs are 'Complementary'. Every position of these two complementary units takes its value according to the next equation.

$$V_n + W_n = 1 \quad (n=1,2,3,4,\dots,15,16)$$

We now know, whenever you take these unit pairs, either the same or the different, for the 4 layers of binary decompositions and combine to compose any object, you will surely be wrong with your result, against the definition (3) of C.E.S.

How can you stop it?

At the stage when you combine any two units, you should consult with this reference table above in advance.

If you take any unit pair whose count is 8 in this table, you will surely have a correct answer by combining them to compose.

I dictated such a new check program as listed above, making the table at `prl unit()` procedure, and using it at `cmbrmp()` as: `if(mtc[u1][u2]==8){nextstep();};`

By this improvement we could make our program run faster.

The result of this program is just the same with the one before, I would like to skip listing it again here and jump over to our next experiment.

(Written in English first on July 20, 2003; Calculated with MWCW_for_Mac;
Revised and recalculated on March 14, 2006 with MacOSX and Xcode 2.1)

Kanji Setsuda: E-Mail Address <jag12001@ni fty. ne. jp>