

Chapter 4: Commentary Articles No.2: Kanji Setsuda "Various Arts and Tools for Studying Magic Squares"

Section 7-2: How to Compose The Other Types of Magic Squares of Order 4 by Binary Number System

10. How to Make Self-Complementary MS44 by Binary Decompositions

Can we make any Self-complementary magic square of order 4 by this method of Binary Number System? Though the object is no longer the 'Complete' type of 'Euler Square', can we compose it by combining binary layer units?

I guess our success depends upon how to design and compose the appropriate layer units well before all.

[Basic Position]				[Self-Complementary Conditions]					
rw1	1	2	3	4	1	2	3	4	$n1+n16=1; \rightarrow n16=1-n1;$
rw2	5	6	7	8	5	6	7	8	$n2+n15=1; \rightarrow n15=1-n2;$
rw3	9	10	11	12	8	7	6	5	$n3+n14=1; \rightarrow n14=1-n3;$
rw4	13	14	15	16	4	3	2	1	$n4+n13=1; \rightarrow n13=1-n4;$
									$n5+n12=1; \rightarrow n12=1-n5;$
									$n6+n11=1; \rightarrow n11=1-n6;$
									$n7+n10=1; \rightarrow n10=1-n7;$
									$n8+n9=1; \rightarrow n9=1-n8;$
cl 1	cl 2	cl 3	cl 4	[Basic Conditions]					
$n1+n2+n3+n4=2$... rw1;	$n1+n5+n9+n13=2$... cl 1;	
$n5+n6+n7+n8=2$... rw2;	$n2+n6+n10+n14=2$... cl 2;	
$n9+n10+n11+n12=2$... rw3;	$n3+n7+n11+n15=2$... cl 3;	
$n13+n14+n15+n16=2$... rw4;	$n4+n8+n12+n16=2$... cl 4;	
$n1+n6+n11+n16=2$... pd1;	$n4+n7+n10+n13=2$... pb4;	

Let's assume these conditions above all true and write our new program as follows:

```

/** Reconstruction of Self-Complementary Magic Squares of */
/** Order 4 by "New Euler's Method" with Binary N. System */
/** 'CES4SC.c' built by Kanji Setsuda */
/** on Oct. 22, 2004; Mar. 17, 2006 */
/** Working on MacOSX and Xcode 2.1 */
/**/
#include <stdio.h>
/**/
short cnt;
short u1, u2, u3, u4;
short LSM, PSM;
short nm[17], uflg[17];
short tnm[49][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];
short rw3[2], cl3[2];
short rw4[2], cl4[2], pd4[2], pb4[2];
/**/
void stp11(void), stp12(void), stp13(void);

```

```

void stp14(void), stp15(void), stp16(void);
void stp17(void), stp18(void), stp19(void);
void lurecord(void);
void prlunit(void), cmbcmp(void), recordans(void);
void pr2ans(short x), pr6ans(short x, short y);
/**/
/* Main Program */
int main(){
short n;
printf("\n** Reconstructi on of Sel f-Compl ementary Magi c Squares of Order 4 **\n");
printf("** by the 'New Euler's Method' with Bi nary Number System **\n");
for(n=0;n<17;n++){nm[n]=0;}
for(n=0;n<2;n++){
rw1[n]=0; cl 1[n]=0; pd1[n]=0; pb1[n]=0;
rw2[n]=0; cl 2[n]=0;
rw3[n]=0; cl 3[n]=0;
rw4[n]=0; cl 4[n]=0; pd4[n]=0; pb4[n]=0;
}
cnt=0; LSM=2; PSM=1;
stp11(); /* Make Layer Uni ts */
printf("\n[Li st of Layer Uni ts for Decomposi ti ons by Bi nary System]\n");
prlunit(); /* Pri nt Layer Uni ts */
cnt=0;
cmbcmp(); /* Combi ne and Compose */
pr2ans(12);
pr6ans(13, 48);
printf(" [Count of Sol uti ons = %d] OK!\n", cnt);
return 0;
}
/**/
/* Make Layer Uni ts */
/* Set n1 & n16 */
void stp11(){
short a, b;
for(a=0; a<2; a++){b=PSM-a;
if((rw1[a]<2)&&(cl 1[a]<2)&&(pd1[a]<2)){
if((rw4[b]<2)&&(cl 4[b]<2)&&(pd1[b]<2)){
nm[1]=a; nm[16]=b;
rw1[a]++; cl 1[a]++; pd1[a]++;
rw4[b]++; cl 4[b]++; pd1[b]++;
stp12();
rw1[a]--; cl 1[a]--; pd1[a]--;
rw4[b]--; cl 4[b]--; pd1[b]--;
}}}
}
}
/* Set n2 & n15 */
void stp12(){
short a, b;
for(a=1; a>=0; a--){b=PSM-a;
if((rw1[a]<2)&&(cl 2[a]<2)){
if((rw4[b]<2)&&(cl 3[b]<2)){
nm[2]=a; nm[15]=b;
rw1[a]++; cl 2[a]++;
rw4[b]++; cl 3[b]++;
stp13();
rw1[a]--; cl 2[a]--;
rw4[b]--; cl 3[b]--;
}}}
}
}

```

```

}
}
/* Set n3 & n14 */
void stp13(){
short a,b;
for(a=1; a>=0; a--){b=PSM-a;
if((rw1[a]<2)&&(cl3[a]<2)){
if((rw4[b]<2)&&(cl2[b]<2)){
nm[3]=a; nm[14]=b;
rw1[a]++; cl3[a]++;
rw4[b]++; cl2[b]++;
stp14();
rw1[a]--; cl3[a]--;
rw4[b]--; cl2[b]--;
}}
}
}
/* Set n4 & n13 */
void stp14(){
short a,b;
for(a=0; a<2; a++){b=PSM-a;
if((rw1[a]<2)&&(cl4[a]<2)&&(pb4[a]<2)){
if((rw4[b]<2)&&(cl1[b]<2)&&(pb4[b]<2)){
nm[4]=a; nm[13]=b;
rw1[a]++; cl4[a]++; pb4[a]++;
rw4[b]++; cl1[b]++; pb4[b]++;
stp15();
rw1[a]--; cl4[a]--; pb4[a]--;
rw4[b]--; cl1[b]--; pb4[b]--;
}}
}
}
/* Set n5 & n12 */
void stp15(){
short a,b;
for(a=1; a>=0; a--){b=PSM-a;
if((rw2[a]<2)&&(cl1[a]<2)){
if((rw3[b]<2)&&(cl4[b]<2)){
nm[5]=a; nm[12]=b;
rw2[a]++; cl1[a]++;
rw3[b]++; cl4[b]++;
stp16();
rw2[a]--; cl1[a]--;
rw3[b]--; cl4[b]--;
}}
}
}
/* Set n6 & n11 */
void stp16(){
short a,b;
for(a=0; a<2; a++){b=PSM-a;
if((rw2[a]<2)&&(cl2[a]<2)&&(pd1[a]<2)){
if((rw3[b]<2)&&(cl3[b]<2)&&(pd1[b]<2)){
nm[6]=a; nm[11]=b;
rw2[a]++; cl2[a]++; pd1[a]++;
rw3[b]++; cl3[b]++; pd1[b]++;
stp17();
rw2[a]--; cl2[a]--; pd1[a]--;
rw3[b]--; cl3[b]--; pd1[b]--;
}
}
}
}

```

```

    }}
}
}
/* Set n7 & n10 */
void stp17(){
short a,b;
for(a=1; a>=0; a--){b=PSM-a;
if((rw2[a]<2)&&(cl 3[a]<2)&&(pb4[a]<2)){
if((rw3[b]<2)&&(cl 2[b]<2)&&(pb4[b]<2)){
nm[7]=a; nm[10]=b;
rw2[a]++; cl 3[a]++; pb4[a]++;
rw3[b]++; cl 2[b]++; pb4[b]++;
stp18();
rw2[a]--; cl 3[a]--; pb4[a]--;
rw3[b]--; cl 2[b]--; pb4[b]--;
}}
}
}
/* Set n8 & n9 */
void stp18(){
short a,b;
for(a=0; a<2; a++){b=PSM-a;
if((rw2[a]<2)&&(cl 4[a]<2)){
if((rw3[b]<2)&&(cl 1[b]<2)){
nm[8]=a; nm[9]=b;
rw2[a]++; cl 4[a]++;
rw3[b]++; cl 1[b]++;
lurecord();
rw2[a]--; cl 4[a]--;
rw3[b]--; cl 1[b]--;
}}
}
}
/**/
/* Record Answers to the Table */
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;
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;
}
}
for(t=0; t<cnt; t=t+8){
printf("%9d/%9d/%9d/%9d/%9d/%9d/%9d/%9d/\n", t+1, t+2, t+3, t+4, t+5, t+6, t+7, t+8);
for(l=0; l<4; l++){l4=l*4;
for(m=t; m<(t+8); m++){
printf(" ");
for(n=1; n<5; n++){printf("%2d", tlu[m][l4+n]);}
}
}
}

```

```

        printf("\n");
    }
}
printf(" [Count of Layer Units = %d]\n\n", cnt);
printf("[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 l u, l uh, md, n, d, fc;
    l u=8; l uh=l u/2; md=8;
    for(u1=0; u1<l uh; u1++){
        for(u2=0; u2<l u; u2++){
            if(mtc[u2][u1]==md){
                for(u3=0; u3<l u; u3++){
                    if((mtc[u3][u1]==md)&&(mtc[u3][u2]==md)){
                        for(u4=0; u4<l u; 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 with /D2i */
void pr2ans(short x){
    short t, l, l4, m, n, s, v;
    printf("\n[Li st 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[Li st of Standard Solu ti ons(to be conti nued): Used Uni ts////Ol d
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");
        }
    }
}
}
}
}
/**/

```

This is one of the simplest programs I have ever written. Let me show you the result of my recent experiment, calculating with another advanced program of mine.

**** Reconstruction of Self-Complementary Magic Squares of Order 4 ****
**** by the 'New Euler's Method' with Binary Number System ****

[List of Layer Units for Decompositions by Binary System]

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

[Count of Layer Units = 8]

[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

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

1/	2/	3/	4/	1[1]	2/	1/	3/	4/	2[9]
0110	0110	0101	0011	1 15 14 4	0110	0110	0101	0011	1 15 14 4
1001	0110	1010	1100	12 6 7 9	0110	1001	1010	1100	8 10 11 5
0110	1001	1010	1100	8 10 11 5	1001	0110	1010	1100	12 6 7 9
1001	1001	0101	0011	13 3 2 16	1001	1001	0101	0011	13 3 2 16
1/	3/	2/	4/	3[5]	2/	3/	1/	4/	4[13]
0110	0101	0110	0011	1 15 12 6	0110	0101	0110	0011	1 15 12 6
1001	1010	0110	1100	14 4 7 9	0110	1010	1001	1100	8 10 13 3
0110	1010	1001	1100	8 10 13 3	1001	1010	0110	1100	14 4 7 9
1001	0101	1001	0011	11 5 2 16	1001	0101	1001	0011	11 5 2 16
3/	1/	2/	4/	5[33]	3/	2/	1/	4/	6[37]
0101	0110	0110	0011	1 15 8 10	0101	0110	0110	0011	1 15 8 10
1010	1001	0110	1100	14 4 11 5	1010	0110	1001	1100	12 6 13 3
1010	0110	1001	1100	12 6 13 3	1010	1001	0110	1100	14 4 11 5
0101	1001	1001	0011	7 9 2 16	0101	1001	1001	0011	7 9 2 16
1/	2/	4/	3/	7[3]	2/	1/	4/	3/	8[11]
0110	0110	0011	0101	1 14 15 4	0110	0110	0011	0101	1 14 15 4
1001	0110	1100	1010	12 7 6 9	0110	1001	1100	1010	8 11 10 5
0110	1001	1100	1010	8 11 10 5	1001	0110	1100	1010	12 7 6 9
1001	1001	0011	0101	13 2 3 16	1001	1001	0011	0101	13 2 3 16
2/	3/	4/	1/	9[15]	3/	2/	4/	1/	10[39]
0110	0101	0011	0110	1 14 12 7	0101	0110	0011	0110	1 14 8 11
0110	1010	1100	1001	8 11 13 2	1010	0110	1100	1001	12 7 13 2
1001	1010	1100	0110	15 4 6 9	1010	1001	1100	0110	15 4 10 5
1001	0101	0011	1001	10 5 3 16	0101	1001	0011	1001	6 9 3 16
2/	4/	1/	3/	11[21]	2/	4/	3/	1/	12[23]
0110	0011	0110	0101	1 12 15 6	0110	0011	0101	0110	1 12 14 7
0110	1100	1001	1010	8 13 10 3	0110	1100	1010	1001	8 13 11 2
1001	1100	0110	1010	14 7 4 9	1001	1100	1010	0110	15 6 4 9
1001	0011	1001	0101	11 2 5 16	1001	0011	0101	1001	10 3 5 16

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

1/2/3/5/[2]	2/1/3/5/[10]	1/3/2/5/[6]	2/3/1/5/[14]	3/1/2/5/[34]	3/2/1/5/[38]
2 16 13 3	2 16 13 3	2 16 11 5	2 16 11 5	2 16 7 9	2 16 7 9
11 5 8 10	7 9 12 6	13 3 8 10	7 9 14 4	13 3 12 6	11 5 14 4
7 9 12 6	11 5 8 10	7 9 14 4	13 3 8 10	11 5 14 4	13 3 12 6
14 4 1 15	14 4 1 15	12 6 1 15	12 6 1 15	8 10 1 15	8 10 1 15
1/2/4/6/[4]	2/1/4/6/[12]	2/3/4/8/[16]	3/2/4/8/[40]	2/4/1/6/[22]	2/4/3/8/[24]
2 13 16 3	2 13 16 3	2 13 11 8	2 13 7 12	2 11 16 5	2 11 13 8
11 8 5 10	7 12 9 6	7 12 14 1	11 8 14 1	7 14 9 4	7 14 12 1
7 12 9 6	11 8 5 10	16 3 5 10	16 3 9 6	13 8 3 10	16 5 3 10
14 1 4 15	14 1 4 15	9 6 4 15	5 10 4 15	12 1 6 15	9 4 6 15

We know the 'one-to-one correspondence' between two types is broken at the last half of the solution list. Let me show you some examples above.

Each result above could not be identified with any solution of the old list. Why?

They have got against the list-forming conditions: $n_1 < n_{16}$; $n_1 < n_4$; $n_1 < n_{13}$; and $n_2 > n_5$; so that you could not find any solution in the old list, which was also formed under the same conditions.

This is unavoidable, whenever you take the two sets of 48 'standard solutions'.

What should you do, then?

You should prepare the complete set of 384 'primitive solutions' of that type, before you apply your type converter program. If you want to have the other type of 48 standard solutions, you should select the object solutions afterward out of the converted results under those inequality conditions.

```

/**/
/* Combine and Compose */
void cmbcmp(){
short l u, md, n, d, fc;
l u=8; md=8;
for(u1=0; u1<l u; u1++){
for(u2=0; u2<l u; u2++){
if(mtc[u2][u1]==md){
for(u3=0; u3<l u; u3++){
if((mtc[u3][u1]==md)&&(mtc[u3][u2]==md)){
for(u4=0; u4<l u; u4++){
if((mtc[u4][u1]==md)&&(mtc[u4][u2]==md)&&(mtc[u4][u3]==md)){
for(n=1; n<17; n++){ufl g[n]=0; }
for(n=1; n<17; n++){
d=t l u[u1][n]*8+t l u[u2][n]*4+t l u[u3][n]*2+t l u[u4][n]+1;
nm[n]=d; ufl g[d]++;
}
fc=0;
for(n=1; n<17; n++){if(ufl g[n]==1){fc++; }el se{break; }}
if(fc==16){recordans();}
}}}}}}
}
}
/**/
/* Type Converter: SC into CC */
void tcnvrt(){
short m;
for(m=0; m<tcnt; m++){
sans[m][1]=tans[m][1]; sans[m][2]=tans[m][2]; sans[m][3]=tans[m][4]; sans[m][4]=tans[m][3];
sans[m][5]=tans[m][5]; sans[m][6]=tans[m][6]; sans[m][7]=tans[m][8]; sans[m][8]=tans[m][7];
sans[m][9]=tans[m][13]; sans[m][10]=tans[m][14]; sans[m][11]=tans[m][16]; sans[m][12]=tans[m][15];
sans[m][13]=tans[m][9]; sans[m][14]=tans[m][10]; sans[m][15]=tans[m][12]; sans[m][16]=tans[m][11];
}
}
/**/

```

The next list shows the result of such a type conversion as above.

[List of 384 Primitive Solutions: Used Uni ts//// Sol _Number#SC Ol d_Nmb#CC]

1/	2/	3/	4/	1#S	1#C	1/	2/	3/	5/	2#S	25#C
0110	0110	0101	0011	1 15 14 4	1 15 4 14	0110	0110	0101	1100	2 16 13 3	2 16 3 13
1001	0110	1010	1100	12 6 7 9	12 6 9 7	1001	0110	1010	0011	11 5 8 10	11 5 10 8
0110	1001	1010	1100	8 10 11 5	13 3 16 2	0110	1001	1010	0011	7 9 12 6	14 4 15 1
1001	1001	0101	0011	13 3 2 16	8 10 5 11	1001	1001	0101	1100	14 4 1 15	7 9 6 12

$\begin{array}{r} 2/d0 \\ 1---- 2 \\ 3---+ 4 \\ 9-- -10 \\ 11----12 \end{array}$	$\begin{array}{r} /d1 \\ 5---- 6 \\ 7---+ 8 \\ 13-- -14 \\ 15----16 \end{array}$	$\begin{array}{l} n1+n2+n3+n4=C \\ n1+n2+n9+n10=C \\ n1+n3+n9+n11=C \\ n2+n4+n10+n12=C \\ n3+n4+n11+n12=C \\ n9+n10+n11+n12=C \end{array}$	$\begin{array}{l} n5+n6+n7+n8=C \\ n5+n6+n13+n14=C \\ n5+n7+n13+n15=C \\ n6+n8+n14+n16=C \\ n7+n8+n15+n16=C \\ n13+n14+n15+n16=C \end{array}$
$\begin{array}{r} 3/d0 \\ 1---- 2 \\ 5---+ 6 \\ 9-- -10 \\ 13----14 \end{array}$	$\begin{array}{r} /d1 \\ 3---- 4 \\ 7---+ 8 \\ 11-- -12 \\ 15----16 \end{array}$	$\begin{array}{l} n1+n2+n5+n6=C \\ n1+n2+n9+n10=C \\ n1+n5+n9+n13=C \\ n2+n6+n10+n14=C \\ n5+n6+n13+n14=C \\ n9+n10+n13+n14=C \end{array}$	$\begin{array}{l} n3+n4+n7+n8=C \\ n3+n4+n11+n12=C \\ n3+n7+n11+n15=C \\ n4+n8+n12+n16=C \\ n7+n8+n15+n16=C \\ n11+n12+n15+n16=C \end{array}$
$\begin{array}{r} 4/d0 \\ 1---- 3 \\ 5---+ 7 \\ 9-- -11 \\ 13----15 \end{array}$	$\begin{array}{r} /d1 \\ 2---- 4 \\ 6---+ 8 \\ 10-- -12 \\ 14----16 \end{array}$	$\begin{array}{l} n1+n3+n5+n7=C \\ n1+n3+n9+n11=C \\ n1+n5+n9+n13=C \\ n3+n7+n11+n15=C \\ n5+n7+n13+n15=C \\ n9+n11+n13+n15=C \end{array}$	$\begin{array}{l} n2+n4+n6+n8=C \\ n2+n4+n10+n12=C \\ n2+n6+n10+n14=C \\ n4+n8+n12+n16=C \\ n6+n8+n14+n16=C \\ n10+n12+n14+n16=C \end{array}$

**** The 9 Essential 'Composite Conditions': ****

$n1+n2+n3+n4=C \dots cd1;$ $n1+n2+n5+n6=C \dots cd2;$ $n1+n2+n9+n10=C \dots cd3;$
 $n1+n3+n5+n7=C \dots cd4;$ $n1+n3+n9+n11=C \dots cd5;$ $n1+n5+n9+n13=C \dots cd6;$
 $n2+n4+n6+n8=C \dots cd7;$ $n2+n4+n10+n12=C \dots cd8;$ $n2+n6+n10+n14=C \dots cd9;$

**** 'Self-complementary Conditions': ****

$n1+n16=n2+n15=n3+n14=n4+n13=n5+n12=n6+n11=n7+n10=n8+n9=CC \dots cd10$

**** Developed Forms into 4x4 No. 2 ****

1	2	3	4	1	2	9	10
5	6	7	8	3	4	11	12
9	10	11	12	5	6	13	14
13	14	15	16	7	8	15	16

Let's prepare some developed 'View Forms' of that object and some simultaneous equations listed as above.

I am afraid you may be surprised to see the 4 view-forms with a lot of simultaneous equations. They only show the concept of our high-dimensional object. We know we need to have only those equations: cd1~cd10 in all at the first definition stage.

I dictated such a program list for our purpose as follows.

```

/** 4-Dimensional Extra-Cubic Magic Form of Order 2 Composed **/
/** by the 'New Euler's Method' with Binary Decompositions **/
/** and Self-Complementary Magic Squares 4x4 Developed **/
/** 'CEulerSX02.c' built by Kanji Setsuda **/
/** on Oct. 10, 2003; Mar. 19, 2006 **/
/** Working on MacOSX and Xcode 2.1 **/
/**/
#include <stdio.h>
/**/
short cnt, cnt2;
short CC;
short u1, u2, u3, u4;
short nm[17], flg[17];
short anm[7][17+5];
short tlu[17][17];

```

```

short mtc[17][17];
/**/
short cd1[2], cd2[2], cd3[2];
short cd4[2], cd5[2], cd6[2];
short cd7[2], cd8[2], cd9[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), ansrecord(void);
void prluni t(void);
void cmbcmp(void);
void prans(void), pr2ans(void);
/**/
/* Main Program */
int main(){
short n;
printf("\n** 4-Dimensional Extra-Cubic Magic Form of Order 2 Composed **\n");
printf("** by the 'New Euler's Method' with Binary Decompositions **\n");
for(n=0; n<17; n++){nm[n]=-1;}
for(n=0; n<2; n++){
cd1[n]=0; cd2[n]=0; cd3[n]=0;
cd4[n]=0; cd5[n]=0; cd6[n]=0;
cd7[n]=0; cd8[n]=0; cd9[n]=0;
}
CC=1; cnt=0;
stp01();
printf("\n[Layer Units of Binary Decompositions]\n");
prluni t();
printf("\n[List of 48 Self-Complementary Magic Squares 4x4 Developed]\n");
cnt=0; cnt2=0;
cmbcmp();
printf("\n [Count = %d] OK!\n", cnt);
return 0;
}
/**/
/* Make Layer Units */
/* Set n1 */
void stp01(){
short a;
for(a=0; a<2; a++){
if((cd1[a]<2)&&(cd2[a]<2)&&(cd3[a]<2)){
if((cd4[a]<2)&&(cd5[a]<2)&&(cd6[a]<2)){
nm[1]=a;
cd1[a]++; cd2[a]++; cd3[a]++;
cd4[a]++; cd5[a]++; cd6[a]++;
stp02();
cd1[a]--; cd2[a]--; cd3[a]--;
cd4[a]--; cd5[a]--; cd6[a]--;}}
}
}
/* Set n2 */
void stp02(){
short a;
for(a=1; a>=0; a--){
if((cd1[a]<2)&&(cd2[a]<2)&&(cd3[a]<2)){
if((cd7[a]<2)&&(cd8[a]<2)&&(cd9[a]<2)){

```

```

        nm[2]=a;
        cd1[a]++; cd2[a]++; cd3[a]++;
        cd7[a]++; cd8[a]++; cd9[a]++;
        stp03();
        cd1[a]--; cd2[a]--; cd3[a]--;
        cd7[a]--; cd8[a]--; cd9[a]--;}}
    }
}
/* Set n3 */
void stp03(){
    short a;
    for(a=1; a>=0; a--){
        if((cd1[a]<2)&&(cd4[a]<2)&&(cd5[a]<2)){
            nm[3]=a;
            cd1[a]++; cd4[a]++; cd5[a]++;
            stp04();
            cd1[a]--; cd4[a]--; cd5[a]--;}
    }
}
/* Set n4 */
void stp04(){
    short a;
    for(a=0; a<2; a++){
        if((cd1[a]<2)&&(cd7[a]<2)&&(cd8[a]<2)){
            nm[4]=a;
            cd1[a]++; cd7[a]++; cd8[a]++;
            stp05();
            cd1[a]--; cd7[a]--; cd8[a]--;}
    }
}
/* Set n5 */
void stp05(){
    short a;
    for(a=1; a>=0; a--){
        if((cd2[a]<2)&&(cd4[a]<2)&&(cd6[a]<2)){
            nm[5]=a;
            cd2[a]++; cd4[a]++; cd6[a]++;
            stp06();
            cd2[a]--; cd4[a]--; cd6[a]--;}
    }
}
/* Set n6 */
void stp06(){
    short a;
    for(a=0; a<2; a++){
        if((cd2[a]<2)&&(cd7[a]<2)&&(cd9[a]<2)){
            nm[6]=a;
            cd2[a]++; cd7[a]++; cd9[a]++;
            stp07();
            cd2[a]--; cd7[a]--; cd9[a]--;}
    }
}
/* Set n7 */
void stp07(){
    short a;
    for(a=1; a>=0; a--){
        if(cd4[a]<2){
            nm[7]=a;
            cd4[a]++;

```

```

        stp08();
        cd4[a]--; }
    }
}
/* Set n8 */
void stp08(){
    short a;
    for(a=0; a<2; a++){
        if(cd7[a]<2){
            nm[8]=a;
            cd7[a]++;
            stp09();
            cd7[a]--; }
    }
}
/* Set n9 & (n8+n9=CC) */
void stp09(){
    short a;
    for(a=1; a>=0; a--){
        if((a+nm[8]==CC)&&(cd3[a]<2)&&(cd5[a]<2)&&(cd6[a]<2)){
            nm[9]=a;
            cd3[a]++; cd5[a]++; cd6[a]++;
            stp10();
            cd3[a]--; cd5[a]--; cd6[a]--; }
    }
}
/* Set n10 & (n7+n10=CC) */
void stp10(){
    short a;
    for(a=0; a<2; a++){
        if((a+nm[7]==CC)&&(cd3[a]<2)&&(cd8[a]<2)&&(cd9[a]<2)){
            nm[10]=a;
            cd3[a]++; cd8[a]++; cd9[a]++;
            stp11();
            cd3[a]--; cd8[a]--; cd9[a]--; }
    }
}
/* Set n11 & (n6+n11=CC) */
void stp11(){
    short a;
    for(a=1; a>=0; a--){
        if((a+nm[6]==CC)&&(cd5[a]<2)){
            nm[11]=a;
            cd5[a]++;
            stp12();
            cd5[a]--; }
    }
}
/* Set n12 & (n5+n12=CC) */
void stp12(){
    short a;
    for(a=0; a<2; a++){
        if((a+nm[5]==CC)&&(cd8[a]<2)){
            nm[12]=a;
            cd8[a]++;
            stp13();
            cd8[a]--; }
    }
}
}

```

```

/* Set n13 & (n4+n13=CC) */
void stp13(){
    short a;
    for(a=1; a>=0; a--){
        if((a+nm[4]==CC)&&(cd6[a]<2)){
            nm[13]=a;
            cd6[a]++;
            stp14();
            cd6[a]--; }
    }
}
/* Set n14 & (n3+n14=CC) */
void stp14(){
    short a;
    for(a=0; a<2; a++){
        if((a+nm[3]==CC)&&(cd9[a]<2)){
            nm[14]=a;
            cd9[a]++;
            stp15();
            cd9[a]--; }
    }
}
/* Set n15 & (n2+n15=CC) */
void stp15(){
    short a;
    for(a=0; a<2; a++){
        if(a+nm[2]==CC){
            nm[15]=a;
            stp16(); }
    }
}
/* Set n16 & (n1+n16=CC) */
void stp16(){
    short a;
    for(a=1; a>=0; a--){
        if(a+nm[1]==CC){
            nm[16]=a;
            ansrecord(); }
    }
}
/**/
/* Record the Answers */
void ansrecord(){
    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 printuni t(){
    short t, l, l4, m, n;
    for(m=0; m<8; m++){
        for(n=0; n<8; n++){
            t=0;
            for(l=1; l<17; l++){if(tlu[m][l]==tlu[n][l]){t++; }}
            mtc[m][n]=t;
        }
    }
}

```

```

for(t=0; t<8; 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 of Layer Units = %d]\n", cnt);
printf("\n[Reference Table for the Best Combination]\n");
printf(" *|");
for(n=0; n<8; n++){printf("%3d", n+1); }
printf("\n");
printf(" ---+-----\n");
for(m=0; m<8; m++){
    printf("%3d|", m+1);
    for(n=0; n<8; 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++){flg[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; flg[d]++;
                                }
                                fc=0;
                                for(n=1; n<17; n++){if(flg[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])){prans(); }
                                }
                            }
                        }
                    }
                }
            }
        }
    }
}
/**/
/* Print the Answers */
void prans(){
    short n;
    cnt++; cnt2++;
    anm[cnt2-1][0]=cnt;
    for(n=1; n<17; n++){anm[cnt2-1][n]=nm[n]; }
    anm[cnt2-1][18]=u1+1; anm[cnt2-1][19]=u2+1;
    anm[cnt2-1][20]=u3+1; anm[cnt2-1][21]=u4+1;
    if(cnt2==2){pr2ans(); cnt2=0; }
}

```

```

/**/
/* Print 2 Answers */
void pr2ans(){
  short l, l4, m, n, s, v;
  for(m=0; m<2; m++){
    printf("%6d/%4d/%4d/%4d/%12d#",
      anm[m][18], anm[m][19], anm[m][20], anm[m][21], anm[m][0]);
    printf("\n");
  }
  for(l=0; l<4; l++){ l4=l*4;
    for(s=0; s<2; s++){
      printf(" ");
      for(m=1; m<5; m++){
        printf(" ");
        for(n=1; n<5; n++){v=tlu[anm[s][m+17]-1][l4+n];
          printf("%d", v);
        }
        printf(" ");
        for(n=1; n<5; n++){printf("%3d", anm[s][l4+n]);
        }
      }
      printf("\n");
    }
  }
}
/**/

```

The following List shows the result of my recent calculations, with another advanced program I dictated afterward. For I wanted to print out everything: (1) list of layer units in the (2x2x2)x2 form and (2) two lists of developed magic squares of order 4: both Self-complementary and Complete.

**** 4-Dimensional Extra-Cubic Magic Form of Order 2 Composed ****
**** by the 'New Euler's Method' with Binary Decompositions ****

[Layer Units Developed into (2x2x2)x2 of Binary Decompositions]

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

[Count of Layer Units = 8]

[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

[List of Developed Self-Complementary and Complete MS44 (n1=1)]

1/	2/	3/	4/	1/S	1/C
0-1 1-0	0-1 1-0	0-1 0-1	0-0 1-1	1 15 14 4	1 15 4 14
1-0 0-1	0-1 1-0	1-0 1-0	1-1 0-0	12 6 7 9	12 6 9 7
0 1 1 0	1 0 0 1	1 0 1 0	1 1 0 0	8 10 11 5	13 3 16 2
1-0 0-1	1-0 0-1	0-1 0-1	0-0 1-1	13 3 2 16	8 10 5 11

1/	2/	4/	3/	2/S	3/C
0-1 1-0	0-1 1-0	0-0 1-1	0-1 0-1	1 14 15 4	1 14 4 15
1-0 0-1	0-1 1-0	1-1 0-0	1-0 1-0	12 7 6 9	12 7 9 6
0 1 1 0	1 0 0 1	1 1 0 0	1-0 1-0	8 11 10 5	13 2 16 3
1-0 0-1	1-0 0-1	0-0 1-1	0-1 0-1	13 2 3 16	8 11 5 10

1/	3/	2/	4/	3/S	9/C
0-1 1-0	0-1 0-1	0-1 1-0	0-0 1-1	1 15 12 6	1 15 6 12
1-0 0-1	1-0 1-0	0-1 1-0	1-1 0-0	14 4 7 9	14 4 9 7
0 1 1 0	1 0 1 0	1-0 0-1	1-1 0-0	8 10 13 3	11 5 16 2
1-0 0-1	0-1 0-1	1-0 0-1	0-0 1-1	11 5 2 16	8 10 3 13

1/	3/	4/	2/	4/S	11/C
0-1 1-0	0-1 0-1	0-0 1-1	0-1 1-0	1 14 12 7	1 14 7 12
1-0 0-1	1-0 1-0	1-1 0-0	0-1 1-0	15 4 6 9	15 4 9 6
0 1 1 0	1 0 1 0	1-1 0-0	1-0 0-1	8 11 13 2	10 5 16 3
1-0 0-1	0-1 0-1	0-0 1-1	1-0 0-1	10 5 3 16	8 11 2 13

1/	4/	2/	3/	5/S	17/C
0-1 1-0	0-0 1-1	0-1 1-0	0-1 0-1	1 12 15 6	1 12 6 15
1-0 0-1	1-1 0-0	0-1 1-0	1-0 1-0	14 7 4 9	14 7 9 4
0 1 1 0	1-1 0-0	1-0 0-1	1-0 1-0	8 13 10 3	11 2 16 5
1-0 0-1	0-0 1-1	1-0 0-1	0-1 0-1	11 2 5 16	8 13 3 10

1/	4/	3/	2/	6/S	19/C
0-1 1-0	0-0 1-1	0-1 0-1	0-1 1-0	1 12 14 7	1 12 7 14
1-0 0-1	1-1 0-0	1-0 1-0	0-1 1-0	15 6 4 9	15 6 9 4
0-1 1-0	1-1 0-0	1-0 1-0	1-0 0-1	8 13 11 2	10 3 16 5
1-0 0-1	0-0 1-1	0-1 0-1	1-0 0-1	10 3 5 16	8 13 2 11

2/	1/	3/	4/	7/S	49/C
0-1 1-0	0-1 1-0	0-1 0-1	0-0 1-1	1 15 14 4	1 15 4 14
0-1 1-0	1-0 0-1	1-0 1-0	1-1 0-0	8 10 11 5	8 10 5 11
1-0 0-1	0-1 1-0	1-0 1-0	1-1 0-0	12 6 7 9	13 3 16 2
1-0 0-1	1-0 0-1	0-1 0-1	0-0 1-1	13 3 2 16	12 6 9 7

2/	1/	4/	3/	8/S	51/C
0-1 1-0	0-1 1-0	0-0 1-1	0-1 0-1	1 14 15 4	1 14 4 15
0-1 1-0	1-0 0-1	1-1 0-0	1-0 1-0	8 11 10 5	8 11 5 10
1-0 0-1	0-1 1-0	1-1 0-0	1-0 1-0	12 7 6 9	13 2 16 3
1-0 0-1	1-0 0-1	0-0 1-1	0-1 0-1	13 2 3 16	12 7 9 6

2/	3/	1/	4/	9/S	57/C
0-1 1-0	0-1 0-1	0-1 1-0	0-0 1-1	1 15 12 6	1 15 6 12
0-1 1-0	1-0 1-0	1-0 0-1	1-1 0-0	8 10 13 3	8 10 3 13
1-0 0-1	1-0 1-0	0-1 1-0	1-1 0-0	14 4 7 9	11 5 16 2
1-0 0-1	0-1 0-1	1-0 0-1	0-0 1-1	11 5 2 16	14 4 9 7

2/	3/	4/	1/	10/S	59/C
0-1 1-0	0-1 0-1	0-0 1-1	0-1 1-0	1 14 12 7	1 14 7 12
0-1 1-0	1-0 1-0	1-1 0-0	1-0 0-1	8 11 13 2	8 11 2 13
1-0 0-1	1-0 1-0	1-1 0-0	0-1 1-0	15 4 6 9	10 5 16 3
1-0 0-1	0-1 0-1	0-0 1-1	1-0 0-1	10 5 3 16	15 4 9 6

2/	4/	1/	3/	11/S	65/C
0-1 1-0	0-0 1-1	0-1 1-0	0-1 0-1	1 12 15 6	1 12 6 15
0-1 1-0	1-1 0-0	1-0 0-1	1-0 1-0	8 13 10 3	8 13 3 10
1-0 0-1	1-1 0-0	0-1 1-0	1-0 1-0	14 7 4 9	11 2 16 5
1-0 0-1	0-0 1-1	1-0 0-1	0-1 0-1	11 2 5 16	14 7 9 4

2/	4/	3/	1/	12/S	67/C
0-1 1-0	0-0 1-1	0-1 0-1	0-1 1-0	1 12 14 7	1 12 7 14
0-1 1-0	1-1 0-0	1-0 1-0	1-0 0-1	8 13 11 2	8 13 2 11
1-0 0-1	1-1 0-0	1-0 1-0	0-1 1-0	15 6 4 9	10 3 16 5
1-0 0-1	0-0 1-1	0-1 0-1	1-0 0-1	10 3 5 16	15 6 9 4
3/	1/	2/	4/	13/S	97/C
0-1 0-1	0-1 1-0	0-1 1-0	0-0 1-1	1 15 8 10	1 15 10 8
1-0 1-0	1-0 0-1	0-1 1-0	1-1 0-0	14 4 11 5	14 4 5 11
1-0 1-0	0-1 1-0	1-0 0-1	1-1 0-0	12 6 13 3	7 9 16 2
0-1 0-1	1-0 0-1	1-0 0-1	0-0 1-1	7 9 2 16	12 6 3 13
3/	1/	4/	2/	14/S	99/C
0-1 0-1	0-1 1-0	0-0 1-1	0-1 1-0	1 14 8 11	1 14 11 8
1-0 1-0	1-0 0-1	1-1 0-0	0-1 1-0	15 4 10 5	15 4 5 10
1-0 1-0	0-1 1-0	1-1 0-0	1-0 0-1	12 7 13 2	6 9 16 3
0-1 0-1	1-0 0-1	0-0 1-1	1-0 0-1	6 9 3 16	12 7 2 13
3/	2/	1/	4/	15/S	105/C
0-1 0-1	0-1 1-0	0-1 1-0	0-0 1-1	1 15 8 10	1 15 10 8
1-0 1-0	0-1 1-0	1-0 0-1	1-1 0-0	12 6 13 3	12 6 3 13
1 0 1 0	1 0 0 1	0-1 1-0	1-1 0-0	14 4 11 5	7 9 16 2
0-1 0-1	1-0 0-1	1-0 0-1	0-0 1-1	7 9 2 16	14 4 5 11
3/	2/	4/	1/	16/S	107/C
0-1 0-1	0-1 1-0	0-0 1-1	0-1 1-0	1 14 8 11	1 14 11 8
1-0 1-0	0-1 1-0	1-1 0-0	1-0 0-1	12 7 13 2	12 7 2 13
1 0 1 0	1-0 0-1	1-1 0-0	0-1 1-0	15 4 10 5	6 9 16 3
0-1 0-1	1-0 0-1	0-0 1-1	1-0 0-1	6 9 3 16	15 4 5 10
3/	4/	1/	2/	17/S	113/C
0-1 0-1	0-0 1-1	0-1 1-0	0-1 1-0	1 12 8 13	1 12 13 8
1-0 1-0	1-1 0-0	1-0 0-1	0-1 1-0	15 6 10 3	15 6 3 10
1-0 1-0	1-1 0-0	0-1 1-0	1-0 0-1	14 7 11 2	4 9 16 5
0-1 0-1	0-0 1-1	1-0 0-1	1-0 0-1	4 9 5 16	14 7 2 11
3/	4/	2/	1/	18/S	115/C
0-1 0-1	0-0 1-1	0-1 1-0	0-1 1-0	1 12 8 13	1 12 13 8
1-0 1-0	1-1 0-0	0-1 1-0	1-0 0-1	14 7 11 2	14 7 2 11
1-0 1-0	1-1 0-0	1-0 0-1	0-1 1-0	15 6 10 3	4 9 16 5
0-1 0-1	0-0 1-1	1-0 0-1	1-0 0-1	4 9 5 16	15 6 3 10
4/	1/	2/	3/	19/S	145/C
0-0 1-1	0-1 1-0	0-1 1-0	0-1 0-1	1 8 15 10	1 8 10 15
1-1 0-0	1-0 0-1	0-1 1-0	1-0 1-0	14 11 4 5	14 11 5 4
1-1 0-0	0-1 1-0	1-0 0-1	1-0 1-0	12 13 6 3	7 2 16 9
0-0 1-1	1-0 0-1	1-0 0-1	0-1 0-1	7 2 9 16	12 13 3 6
4/	1/	3/	2/	20/S	147/C
0-0 1-1	0-1 1-0	0-1 0-1	0-1 1-0	1 8 14 11	1 8 11 14
1-1 0-0	1-0 0-1	1-0 1-0	0-1 1-0	15 10 4 5	15 10 5 4
1-1 0-0	0-1 1-0	1-0 1-0	1-0 0-1	12 13 7 2	6 3 16 9
0-0 1-1	1-0 0-1	0-1 0-1	1-0 0-1	6 3 9 16	12 13 2 7
4/	2/	1/	3/	21/S	153/C
0-0 1-1	0-1 1-0	0-1 1-0	0-1 0-1	1 8 15 10	1 8 10 15
1-1 0-0	0-1 1-0	1-0 0-1	1-0 1-0	12 13 6 3	12 13 3 6
1-1 0-0	1-0 0-1	0-1 1-0	1-0 1-0	14 11 4 5	7 2 16 9
0-0 1-1	1-0 0-1	1-0 0-1	0-1 0-1	7 2 9 16	14 11 5 4
4/	2/	3/	1/	22/S	155/C
0-0 1-1	0-1 1-0	0-1 0-1	0-1 1-0	1 8 14 11	1 8 11 14
1-1 0-0	0-1 1-0	1-0 1-0	1-0 0-1	12 13 7 2	12 13 2 7
1-1 0-0	1-0 0-1	1-0 1-0	0-1 1-0	15 10 4 5	6 3 16 9
0-0 1-1	1-0 0-1	0-1 0-1	1-0 0-1	6 3 9 16	15 10 5 4

4/	3/	1/	2/	23/S	161/C
0-0 1-1	0-1 0-1	0-1 1-0	0-1 1-0	1 8 12 13	1 8 13 12
1-1 0-0	1-0 1-0	1-0 0-1	0-1 1-0	15 10 6 3	15 10 3 6
1-1 0-0	1-0 1-0	0-1 1-0	1-0 0-1	14 11 7 2	4 5 16 9
0-0 1-1	0-1 0-1	1-0 0-1	1-0 0-1	4 5 9 16	14 11 2 7
4/	3/	2/	1/	24/S	163/C
0-0 1-1	0-1 0-1	0-1 1-0	0-1 1-0	1 8 12 13	1 8 13 12
1-1 0-0	1-0 1-0	0-1 1-0	1-0 0-1	14 11 7 2	14 11 2 7
1 1 0 0	1 0 1 0	1-0 0-1	0-1 1-0	15 10 6 3	4 5 16 9
0-0 1-1	0-1 0-1	1-0 0-1	1-0 0-1	4 5 9 16	15 10 3 6

[Count (n1=1)/Total = 24/384] OK!

All the self-complementary MS44 and complete ones have really been taken out of the ECMF2^4 successfully, as you see. It tells us that we can only make the same thing, whatever different conditions we might assume to them: 'self-complementary', 'pan-diagonal', 'complete', 'composite & complete' or 'Complete Euler Square'.

We are really seeing and touching just the "Miraculous Unity", aren't we?
 Don't you think how effective our ECMF is implying that Unity itself?

And how powerful our New Euler's Method is! You may be surprised as much as I.
 We expect we can make almost everything we want to by this method.

13. Can we make those 880 Standard Magic Squares 4x4 by this Binary Method?

Can we make any "Non-Complete type" of Euler Squares 4x4 such as those 880 standard magic squares 4x4? Let's make some experiments about it now.

I prepared the next figure and basic conditions as follows.

	[Basic Conditions for only Rows and Columns]
----- n1 n2 n3 n4 --+-+----- n5 n6 n7 n8 --+-+----- n9 10 11 12 --+-+----- 13 14 15 16 -----	n1+n2+n3+n4=C ... rw1; n5+n6+n7+n8=C ... rw2; n9+n10+n11+n12=C ... rw3; n13+n14+n15+n16=C ... rw4; n1+n5+n9+n13=C ... cl 1; n2+n6+n10+n14=C ... cl 2; n3+n7+n11+n15=C ... cl 3; n4+n8+n12+n16=C ... cl 4;

I define each of the 4 decomposed layers should have such the same property in common as every row and every column must be made of two '0' and two '1', that means it should accept the Definition (1) of 'Euler Squares'.

But I do not define anything about the two primary diagonals at first, though in the last stage I am going to select only those answers whose primary diagonals n1+n6+n11+n16 and n4+n7+n10+n13 are equal to the same sum C.

The result of my recent calculation is shown below.

** Composing 'Euler Squares' 4x4 Making with **
 ** Layer Units of Binary Decompositions **

[Layer Units of Binary Decompositions]

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

[Composi ti ons: Used Uni ts//// (n1=1)/No#]

1/15/22/28/ 1/ 1# 1 16 13 4 11 6 7 10 8 9 12 5 14 3 2 15	1/15/22/38/ 2/ 2# 1 15 14 4 12 6 7 9 8 10 11 5 13 3 2 16	1/15/24/43/ 3/ 5# 1 15 14 4 11 8 5 10 6 9 12 7 16 2 3 13	1/15/28/22/ 4/ 7# 1 16 13 4 10 7 6 11 8 9 12 5 15 2 3 14	1/15/28/37/ 5/ 8# 1 15 14 4 10 8 5 11 7 9 12 6 16 2 3 13	1/15/37/28/ 6/ 11# 1 14 15 4 11 8 5 10 6 9 12 7 16 3 2 13
1/15/37/38/ 7/ 12# 1 13 16 4 12 8 5 9 6 10 11 7 15 3 2 14	1/15/38/22/ 8/ 15# 1 14 15 4 12 7 6 9 8 11 10 5 13 2 3 16	1/15/38/37/ 9/ 16# 1 13 16 4 12 8 5 9 7 11 10 6 14 2 3 15	1/15/43/24/ 10/ 19# 1 14 15 4 10 8 5 11 7 9 12 6 16 3 2 13	1/22/15/28/ 11/ 24# 1 16 11 6 13 4 7 10 8 9 14 3 12 5 2 15	1/22/15/38/ 12/ 25# 1 15 12 6 14 4 7 9 8 10 13 3 11 5 2 16
1/22/28/15/ 13/ 28# 1 16 10 7 13 4 6 11 8 9 15 2 12 5 3 14	1/22/28/37/ 14/ 29# 1 15 10 8 14 4 5 11 7 9 16 2 12 6 3 13	1/24/15/43/ 15/ 40# 1 15 12 6 13 8 3 10 4 9 14 7 16 2 5 11	1/24/43/15/ 16/ 43# 1 14 12 7 13 8 2 11 4 9 15 6 16 3 5 10	1/25/27/19/ 17/ 47# 1 16 9 8 14 5 4 11 7 10 15 2 12 3 6 13	1/28/15/22/ 18/ 52# 1 16 11 6 10 7 4 13 8 9 14 3 15 2 5 12
1/28/15/37/ 19/ 53# 1 15 12 6 10 8 3 13 7 9 14 4 16 2 5 11	1/28/22/15/ 20/ 56# 1 16 10 7 11 6 4 13 8 9 15 2 14 3 5 12	1/28/22/37/ 21/ 57# 1 15 10 8 12 6 3 13 7 9 16 2 14 4 5 11	1/28/37/15/ 22/ 60# 1 14 12 7 11 8 2 13 6 9 15 4 16 3 5 10	1/28/37/22/ 23/ 61# 1 14 11 8 12 7 2 13 6 9 16 3 15 4 5 10	1/43/15/24/ 24/ 78# 1 12 15 6 10 8 3 13 7 9 14 4 16 5 2 11
1/43/24/15/ 25/ 80# 1 12 14 7 11 8 2 13 6 9 15 4 16 5 3 10	9/11/16/33/ 26/ 84# 1 15 14 4 12 9 6 7 5 8 11 10 16 2 3 13	9/11/19/22/ 27/ 85# 1 16 13 4 12 9 8 5 6 7 10 11 15 2 3 14	9/11/30/33/ 28/ 87# 1 15 14 4 10 11 8 5 7 6 9 12 16 2 3 13	9/11/31/19/ 29/ 88# 1 14 15 4 12 9 6 7 5 8 11 10 16 3 2 13	9/11/33/38/ 30/ 89# 1 13 16 4 12 10 7 5 6 8 9 11 15 3 2 14
9/11/45/19/ 31/ 91# 1 14 15 4 10 11 8 5 7 6 9 12 16 3 2 13	9/16/11/33/ 32/ 92# 1 15 12 6 14 9 4 7 3 8 13 10 16 2 5 11	9/16/30/33/ 33/ 93# 1 15 10 8 14 11 4 5 3 6 13 12 16 2 7 9	9/19/22/28/ 34/ 94# 1 16 9 8 15 10 7 2 4 5 12 13 14 3 6 11	9/19/28/22/ 35/ 95# 1 16 9 8 14 11 6 3 4 5 12 13 15 2 7 10	9/30/11/33/ 36/ 98# 1 15 12 6 10 13 8 3 7 4 9 14 16 2 5 11
9/30/16/33/ 37/ 99# 1 15 10 8 12 13 6 3 5 4 11 14 16 2 7 9	9/30/33/43/ 38/100# 1 13 12 8 11 14 7 2 6 3 10 15 16 4 5 9	9/45/11/19/ 39/102# 1 12 15 6 10 13 8 3 7 4 9 14 16 5 2 11	9/45/19/28/ 40/103# 1 12 13 8 11 14 7 2 6 3 10 15 16 5 4 9	13/ 7/16/41/ 41/104# 1 15 14 4 7 9 6 12 10 8 11 5 16 2 3 13	13/ 7/26/22/ 42/105# 1 16 13 4 6 9 8 11 12 7 10 5 15 2 3 14
13/ 7/30/41/ 43/107# 1 15 14 4 5 11 8 10 12 6 9 7 16 2 3 13	13/ 7/31/26/ 44/108# 1 14 15 4 7 9 6 12 10 8 11 5 16 3 2 13	13/ 7/41/38/ 45/109# 1 13 16 4 6 10 7 11 12 8 9 5 15 3 2 14	13/ 7/45/26/ 46/111# 1 14 15 4 5 11 8 10 12 6 9 7 16 3 2 13	13/16/ 7/41/ 47/112# 1 15 12 6 7 9 4 14 10 8 13 3 16 2 5 11	13/16/30/41/ 48/113# 1 15 10 8 5 11 4 14 12 6 13 3 16 2 7 9
13/16/41/37/ 49/114# 1 13 12 8 6 10 3 15 11 7 14 2 16 4 5 9	13/26/22/24/ 50/116# 1 16 9 8 4 10 7 13 15 5 12 2 14 3 6 11	13/26/24/22/ 51/118# 1 16 9 8 4 11 6 13 14 5 12 3 15 2 7 10	13/30/ 7/41/ 52/124# 1 15 12 6 3 13 8 10 14 4 9 7 16 2 5 11	13/30/16/41/ 53/125# 1 15 10 8 3 13 6 12 14 4 11 5 16 2 7 9	13/30/41/37/ 54/126# 1 13 12 8 2 14 7 11 15 3 10 6 16 4 5 9
13/30/41/38/ 55/127# 1 15 14 4 5 11 8 10 12 6 9 7 16 2 3 13	13/31/ 7/26/ 56/128# 1 14 15 4 7 9 6 12 10 8 11 5 16 3 2 13	13/31/26/24/ 57/129# 1 13 16 4 6 10 7 11 12 8 9 5 15 3 2 14	13/31/45/26/ 58/130# 1 14 15 4 5 11 8 10 12 6 9 7 16 3 2 13	13/41/37/38/ 59/131# 1 15 12 6 7 9 6 12 10 8 13 3 16 2 5 11	13/41/38/37/ 60/132# 1 15 10 8 3 13 6 12 14 4 11 5 16 2 7 9

55/127#	56/130#	57/131#	58/133#	59/134#	60/136#
1 13 12 8	1 12 15 6	1 12 13 8	1 10 15 8	1 9 16 8	1 9 16 8
2 14 7 11	7 9 4 14	6 10 3 15	5 11 4 14	4 12 5 13	4 12 5 13
16 4 9 5	10 8 13 3	11 7 14 2	12 6 13 3	14 6 11 3	15 7 10 2
15 3 6 10	16 5 2 11	16 5 4 9	16 7 2 9	15 7 2 10	14 6 3 11

.....

28/38/22/37/	28/38/37/15/	28/38/37/22/	28/39/20/41/	28/39/36/41/	28/39/41/15/
193/742#	194/745#	195/746#	196/761#	197/762#	198/763#
1 11 6 16	1 10 8 15	1 10 7 16	1 11 6 16	1 9 8 16	1 10 8 15
8 14 3 9	7 16 2 9	8 15 2 9	7 13 4 10	7 15 2 10	5 14 4 11
15 5 12 2	14 5 11 4	14 5 12 3	14 2 15 3	14 4 13 3	16 3 13 2
10 4 13 7	12 3 13 6	11 4 13 6	12 8 9 5	12 6 11 5	12 7 9 6
43/ 1/15/24/	43/ 1/24/15/	43/ 9/30/33/	43/15/ 1/24/	43/15/24/ 1/	43/15/25/26/
199/821#	200/823#	201/827#	202/832#	203/834#	204/836#
1 8 15 10	1 8 14 11	1 7 14 12	1 8 15 10	1 8 14 11	1 8 13 12
6 12 3 13	7 12 2 13	6 15 4 9	4 14 5 11	4 15 5 10	3 15 6 10
11 5 14 4	10 5 15 4	11 2 13 8	13 3 12 6	13 2 12 7	16 2 11 5
16 9 2 7	16 9 3 6	16 10 3 5	16 9 2 7	16 9 3 6	14 9 4 7
43/24/ 1/15/	43/24/15/ 1/	43/25/20/26/	43/25/26/15/		
205/852#	206/853#	207/856#	208/857#		
1 8 12 13	1 8 12 13	1 8 9 16	1 8 10 15		
7 14 2 11	6 15 3 10	7 13 4 10	5 14 4 11		
10 3 15 6	11 2 14 7	14 2 15 3	16 3 13 2		
16 9 5 4	16 9 5 4	12 11 6 5	12 9 7 6		

[Count (n1=1)/Total = 208/880] OK!

I was much surprised at the result. They all have the beautiful structure in common, what we call 'Euler Square'. Though all of them don't always have their primary diagonals of the number pattern with two '0' and two '1', and consequently we cannot call them "Complete" Euler Squares at all, but they do always have their rows and columns made of that pattern with two '0' and two '1'.

Some solutions seem to have the same pattern even in their primary diagonals.

How many solutions have the pattern {0, 0, 1, 1} even in their primary diagonals?

The next experiment is made for finding all those objects.

I added the next two equations to the basic definitions and prepared the two flag sets for them: $n1+n6+n11+n16=C \dots pd1$; $n4+n7+n10+n13=C \dots pb4$;

```

/** 'Euler Squares' of Order 4 by The New Euler's Method **/
/** Making with Layer Units of Binary Decompositions **/
/** 'EulerS4B.c' built by Kanji Setsuda **/
/** on Oct. 10, 2003; Mar. 18, 2006 **/
/** Working on MacOSX and Xcode 2.1 **/
/**/
/*
    [Basic Conditions
     for Rows and Columns]
    |-----|
    | n1 | n2 | n3 | n4 | n1+n2+n3+n4=C    ... rw1;
    |-----|
    | n5 | n6 | n7 | n8 | n5+n6+n7+n8=C    ... rw2;
    |-----|
    | n9 | n10| n11| n12| n9+n10+n11+n12=C   ... rw3;
    |-----|
    | n13| n14| n15| n16| n13+n14+n15+n16=C  ... rw4;
    |-----|
    | n9 | 10 | 11 | 12 | n1+n5+n9+n13=C    ... cl 1;
    |-----|
    | n2 | n6 | n10| n14| n2+n6+n10+n14=C   ... cl 2;
    |-----|
    | 13 | 14 | 15 | 16 | n3+n7+n11+n15=C    ... cl 3;
    |-----|
    | n4 | n8 | n12| n16| n4+n8+n12+n16=C   ... cl 4;
    |-----|

```

with Two Primary Diagonals:

```

    n1+n6+n11+n16=C ... pd1;   n4+n7+n10+n13=C ... pb4;
*/
/**/
#include <stdio.h>
/**/
short cnt, cnt2, cnt3;
short nm[17], uflg[17];
short anm[6][17+5];
short mtc[17][17], tlu[17][17];
short u1, u2, u3, u4;
/**/
short rw1[2], cl1[2], pd1[2];
short rw2[2], cl2[2];
short rw3[2], cl3[2];
short rw4[2], cl4[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 ansrecord(void), prlunit(void), cmbcmp(void);
void prans(void);
void anspr(short x);
/**/
int main(){
    short m, n;
    printf("\n** Composing Standard Magic Squares 4x4: 'Euler Type' **\n");
    printf("** Making with Layer Units of Binary Decompositions **\n");
    for(n=0; n<17; n++){nm[n]=0;}
    for(m=0; m<17; m++){for(n=0; n<17; n++){mtc[m][n]=0;}}
    for(n=0; n<2; n++){
        rw1[n]=0; rw2[n]=0; rw3[n]=0; rw4[n]=0;
        cl1[n]=0; cl2[n]=0; cl3[n]=0; cl4[n]=0;
        pd1[n]=0; pb4[n]=0;
    }
    cnt=0;
    stp01(); /* Make Layer Units */
    printf("\n[Layer Units of Binary Decompositions]\n");
    prlunit(); /* Print Layer Units */
    printf("\n[Compositions: Used Units//// (n1=1)/No#]\n");
    cnt=0; cnt2=0; cnt3=0;
    cmbcmp(); /* Combine and Compose */
    if(cnt3>0){anspr(cnt3);}
    printf("\n [Count (n1=1)/Total = %d/%d] OK!\n", cnt2, 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)){
            nm[1]=a;
            rw1[a]++; cl1[a]++; pd1[a]++;
            stp02();
            rw1[a]--; cl1[a]--; pd1[a]--;
        }
    }
}

```

```

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

```

```

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

```

```

        stp13();
        rw3[a]--; cl 4[a]--;
    }}
}
/* Set n13 */
void stp13(){
    short a;
    for(a=1; a>=0; a--){
        if((rw4[a]<2)&&(cl 1[a]<2)&&(pb4[a]<2)){
            nm[13]=a;
            rw4[a]++; cl 1[a]++; pb4[a]++;
            stp14();
            rw4[a]--; cl 1[a]--; pb4[a]--;
        }}
}
/* Set n14 */
void stp14(){
    short a;
    for(a=0; a<2; a++){
        if((rw4[a]<2)&&(cl 2[a]<2)){
            nm[14]=a;
            rw4[a]++; cl 2[a]++;
            stp15();
            rw4[a]--; cl 2[a]--;
        }}
}
/* Set n15 */
void stp15(){
    short a;
    for(a=0; a<2; a++){
        if((rw4[a]<2)&&(cl 3[a]<2)){
            nm[15]=a;
            rw4[a]++; cl 3[a]++;
            stp16();
            rw4[a]--; cl 3[a]--;
        }}
}
/* Set n16 */
void stp16(){
    short a;
    for(a=1; a>=0; a--){
        if((rw4[a]<2)&&(cl 4[a]<2)&&(pd1[a]<2)){
            nm[16]=a;
            rw4[a]++; cl 4[a]++; pd1[a]++;
            ansrecord();
            rw4[a]--; cl 4[a]--; pd1[a]--;
        }}
}
/**/
/* Record the Answers */
void ansrecord(){
    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(){

```



```

/**/
/* Print the Answers */
void prans(){
short n;
cnt++;
if(nm[1]==1){cnt2++;
anm[cnt3][0]=cnt; anm[cnt3][17]=cnt2;
for(n=1;n<17;n++){anm[cnt3][n]=nm[n];}
anm[cnt3][18]=u1+1; anm[cnt3][19]=u2+1;
anm[cnt3][20]=u3+1; anm[cnt3][21]=u4+1;
cnt3++;
if(cnt3==6){anspr(cnt3); cnt3=0;}
}
}
/**/
/* Print Answers */
void anspr(short x){
short l,l4,m,n;
for(m=0;m<x;m++){
printf("%3d/%2d/%2d/%2d/",
anm[m][18], anm[m][19], anm[m][20], anm[m][21]);
if(m<(x-1)){printf(" ");}
}
printf("\n");
for(m=0;m<x;m++){
printf("%8d/%3d#", anm[m][17], anm[m][0]);
if(m<(x-1)){printf(" ");}
}
printf("\n");
for(l=0;l<4;l++){l4=l*4;
for(m=0;m<x;m++){
printf(" ");
for(n=1;n<5;n++){printf("%3d", anm[m][l4+n]);}
if(m<(x-1)){printf(" ");}
}
printf("\n");
}
}
/**/

```

The result of my recent calculation is listed as follows.

```

** Composing Standard Magic Squares 4x4: ' Euler Type' **
** Making with Layer Units of Binary Decompositions **

```

[Layer Units of Binary Decompositions]

1/	2/	3/	4/	5/	6/	7/	8/
0 1 1 0	0 1 1 0	0 1 0 1	0 1 0 1	0 1 0 1	0 0 1 1	0 0 1 1	0 0 1 1
1 0 0 1	0 1 1 0	1 0 1 0	1 1 0 0	0 1 0 1	1 1 0 0	1 1 0 0	0 1 0 1
0 1 1 0	1 0 0 1	1 0 1 0	0 0 1 1	1 0 1 0	0 0 1 1	1 1 0 0	1 0 1 0
1 0 0 1	1 0 0 1	0 1 0 1	1 0 1 0	1 0 1 0	1 1 0 0	0 0 1 1	1 1 0 0
9/	10/	11/	12/	13/	14/	15/	16/
1 1 0 0	1 1 0 0	1 1 0 0	1 0 1 0	1 0 1 0	1 0 1 0	1 0 0 1	1 0 0 1
1 0 1 0	0 0 1 1	0 0 1 1	1 0 1 0	0 0 1 1	0 1 0 1	1 0 0 1	0 1 1 0
0 1 0 1	0 0 1 1	1 1 0 0	0 1 0 1	1 1 0 0	0 1 0 1	0 1 1 0	1 0 0 1
0 0 1 1	1 1 0 0	0 0 1 1	0 1 0 1	0 1 0 1	1 0 1 0	0 1 1 0	0 1 1 0

[Count of Layer Units = 16]

[Reference Table for the Best Combination]

*	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	16	8	8	8	8	8	8	8	8	8	8	8	8	8	8	0
2	8	16	8	8	8	8	8	8	8	8	8	8	8	8	0	8
3	8	8	16	8	8	8	8	8	8	8	8	8	8	0	8	8
4	8	8	8	16	12	12	8	8	8	8	4	4	0	8	8	8
5	8	8	8	12	16	8	8	12	4	8	8	0	4	8	8	8
6	8	8	8	12	8	16	8	12	4	8	0	8	4	8	8	8
7	8	8	8	8	8	8	16	8	8	0	8	8	8	8	8	8
8	8	8	8	8	12	12	8	16	0	8	4	4	8	8	8	8
9	8	8	8	8	4	4	8	0	16	8	12	12	8	8	8	8
10	8	8	8	8	8	8	0	8	8	16	8	8	8	8	8	8
11	8	8	8	4	8	0	8	4	12	8	16	8	12	8	8	8
12	8	8	8	4	0	8	8	4	12	8	8	16	12	8	8	8
13	8	8	8	0	4	4	8	8	8	8	12	12	16	8	8	8
14	8	8	0	8	8	8	8	8	8	8	8	8	8	16	8	8
15	8	0	8	8	8	8	8	8	8	8	8	8	8	8	16	8
16	0	8	8	8	8	8	8	8	8	8	8	8	8	8	8	16

[Compositions: Used Units//// (n1=1)/No#]

1/ 2/ 3/ 5/	1/ 2/ 3/ 7/	1/ 2/ 4/ 8/	1/ 2/ 5/ 3/	1/ 2/ 5/ 6/	1/ 2/ 6/ 5/
1/ 1#	2/ 2#	3/ 5#	4/ 7#	5/ 8#	6/ 11#
1 16 13 4	1 15 14 4	1 15 14 4	1 16 13 4	1 15 14 4	1 14 15 4
11 6 7 10	12 6 7 9	11 8 5 10	10 7 6 11	10 8 5 11	11 8 5 10
8 9 12 5	8 10 11 5	6 9 12 7	8 9 12 5	7 9 12 6	6 9 12 7
14 3 2 15	13 3 2 16	16 2 3 13	15 2 3 14	16 2 3 13	16 3 2 13
1/ 2/ 6/ 7/	1/ 2/ 7/ 3/	1/ 2/ 7/ 6/	1/ 2/ 8/ 4/	1/ 3/ 2/ 5/	1/ 3/ 2/ 7/
7/ 12#	8/ 15#	9/ 16#	10/ 19#	11/ 21#	12/ 22#
1 13 16 4	1 14 15 4	1 13 16 4	1 14 15 4	1 16 11 6	1 15 12 6
12 8 5 9	12 7 6 9	12 8 5 9	10 8 5 11	13 4 7 10	14 4 7 9
6 10 11 7	8 11 10 5	7 11 10 6	7 9 12 6	8 9 14 3	8 10 13 3
15 3 2 14	13 2 3 16	14 2 3 15	16 3 2 13	12 5 2 15	11 5 2 16
1/ 3/ 5/ 2/	1/ 3/ 5/ 6/	1/ 4/ 2/ 8/	1/ 4/ 8/ 2/	1/ 5/ 2/ 3/	1/ 5/ 2/ 6/
13/ 25#	14/ 26#	15/ 37#	16/ 39#	17/ 41#	18/ 42#
1 16 10 7	1 15 10 8	1 15 12 6	1 14 12 7	1 16 11 6	1 15 12 6
13 4 6 11	14 4 5 11	13 8 3 10	13 8 2 11	10 7 4 13	10 8 3 13
8 9 15 2	7 9 16 2	4 9 14 7	4 9 15 6	8 9 14 3	7 9 14 4
12 5 3 14	12 6 3 13	16 2 5 11	16 3 5 10	15 2 5 12	16 2 5 11
1/ 5/ 3/ 2/	1/ 5/ 3/ 6/	1/ 5/ 6/ 2/	1/ 5/ 6/ 3/	1/ 8/ 2/ 4/	1/ 8/ 4/ 2/
19/ 45#	20/ 46#	21/ 49#	22/ 50#	23/ 65#	24/ 67#
1 16 10 7	1 15 10 8	1 14 12 7	1 14 11 8	1 12 15 6	1 12 14 7
11 6 4 13	12 6 3 13	11 8 2 13	12 7 2 13	10 8 3 13	11 8 2 13
8 9 15 2	7 9 16 2	6 9 15 4	6 9 16 3	7 9 14 4	6 9 15 4
14 3 5 12	14 4 5 11	16 3 5 10	15 4 5 10	16 5 2 11	16 5 3 10
2/ 1/ 3/ 5/	2/ 1/ 3/ 7/	2/ 1/ 4/ 8/	2/ 1/ 5/ 3/	2/ 1/ 5/ 6/	2/ 1/ 6/ 5/
25/ 69#	26/ 70#	27/ 73#	28/ 75#	29/ 76#	30/ 79#
1 16 13 4	1 15 14 4	1 15 14 4	1 16 13 4	1 15 14 4	1 14 15 4
7 10 11 6	8 10 11 5	7 12 9 6	6 11 10 7	6 12 9 7	7 12 9 6
12 5 8 9	12 6 7 9	10 5 8 11	12 5 8 9	11 5 8 10	10 5 8 11
14 3 2 15	13 3 2 16	16 2 3 13	15 2 3 14	16 2 3 13	16 3 2 13
2/ 1/ 6/ 7/	2/ 1/ 7/ 3/	2/ 1/ 7/ 6/	2/ 1/ 8/ 4/	2/ 3/ 1/ 5/	2/ 3/ 1/ 7/
31/ 80#	32/ 83#	33/ 84#	34/ 87#	35/ 89#	36/ 90#
1 13 16 4	1 14 15 4	1 13 16 4	1 14 15 4	1 16 11 6	1 15 12 6
8 12 9 5	8 11 10 5	8 12 9 5	6 12 9 7	7 10 13 4	8 10 13 3
10 6 7 11	12 7 6 9	11 7 6 10	11 5 8 10	14 3 8 9	14 4 7 9
15 3 2 14	13 2 3 16	14 2 3 15	16 3 2 13	12 5 2 15	11 5 2 16

2/ 3/ 5/ 1/ 37/ 93# 1 16 10 7 6 11 13 4 15 2 8 9 12 5 3 14	2/ 3/ 5/ 7/ 38/ 94# 1 15 10 8 6 12 13 3 16 2 7 9 11 5 4 14	2/ 3/ 7/ 1/ 39/ 97# 1 14 12 7 8 11 13 2 15 4 6 9 10 5 3 16	2/ 3/ 7/ 5/ 40/ 98# 1 14 11 8 7 12 13 2 16 3 6 9 10 5 4 15	2/ 4/ 1/ 8/ 41/113# 1 15 12 6 7 14 9 4 10 3 8 13 16 2 5 11	2/ 4/ 8/ 1/ 42/115# 1 14 12 7 6 15 9 4 11 2 8 13 16 3 5 10
2/ 5/ 1/ 3/ 43/121# 1 16 11 6 4 13 10 7 14 3 8 9 15 2 5 12	2/ 5/ 1/ 6/ 44/122# 1 15 12 6 4 14 9 7 13 3 8 10 16 2 5 11	2/ 5/ 3/ 1/ 45/125# 1 16 10 7 4 13 11 6 15 2 8 9 14 3 5 12	2/ 5/ 3/ 7/ 46/126# 1 15 10 8 4 14 11 5 16 2 7 9 13 3 6 12	2/ 5/ 6/ 1/ 47/129# 1 14 12 7 4 15 9 6 13 2 8 11 16 3 5 10	2/ 5/ 6/ 7/ 48/130# 1 13 12 8 4 16 9 5 14 2 7 11 15 3 6 10
2/ 5/ 7/ 3/ 49/133# 1 14 11 8 4 15 10 5 16 3 6 9 13 2 7 12	2/ 5/ 7/ 6/ 50/134# 1 13 12 8 4 16 9 5 15 3 6 10 14 2 7 11	2/ 6/ 1/ 5/ 51/153# 1 12 15 6 7 14 9 4 10 3 8 13 16 5 2 11	2/ 6/ 1/ 7/ 52/154# 1 11 16 6 8 14 9 3 10 4 7 13 15 5 2 12	2/ 6/ 5/ 1/ 53/157# 1 12 14 7 6 15 9 4 11 2 8 13 16 5 3 10	2/ 6/ 5/ 7/ 54/158# 1 11 14 8 6 16 9 3 12 2 7 13 15 5 4 10
2/ 6/ 7/ 1/ 55/161# 1 10 16 7 8 15 9 2 11 4 6 13 14 5 3 12	2/ 6/ 7/ 5/ 56/162# 1 10 15 8 7 16 9 2 12 3 6 13 14 5 4 11	2/ 7/ 1/ 3/ 57/177# 1 12 15 6 8 13 10 3 14 7 4 9 11 2 5 16	2/ 7/ 1/ 6/ 58/178# 1 11 16 6 8 14 9 3 13 7 4 10 12 2 5 15	2/ 7/ 3/ 1/ 59/181# 1 12 14 7 8 13 11 2 15 6 4 9 10 3 5 16	2/ 7/ 3/ 5/ 60/182# 1 12 13 8 7 14 11 2 16 5 4 9 10 3 6 15
2/ 7/ 5/ 3/ 61/185# 1 12 13 8 6 15 10 3 16 5 4 9 11 2 7 14	2/ 7/ 5/ 6/ 62/186# 1 11 14 8 6 16 9 3 15 5 4 10 12 2 7 13	2/ 7/ 6/ 1/ 63/189# 1 10 16 7 8 15 9 2 13 6 4 11 12 3 5 14	2/ 7/ 6/ 5/ 64/190# 1 10 15 8 7 16 9 2 14 5 4 11 12 3 6 13	2/ 8/ 1/ 4/ 65/209# 1 12 15 6 4 14 9 7 13 3 8 10 16 5 2 11	2/ 8/ 4/ 1/ 66/211# 1 12 14 7 4 15 9 6 13 2 8 11 16 5 3 10
3/ 1/ 2/ 5/ 67/237# 1 16 7 10 13 4 11 6 12 5 14 3 8 9 2 15	3/ 1/ 2/ 7/ 68/238# 1 15 8 10 14 4 11 5 12 6 13 3 7 9 2 16	3/ 1/ 5/ 2/ 69/241# 1 16 6 11 13 4 10 7 12 5 15 2 8 9 3 14	3/ 1/ 5/ 6/ 70/242# 1 15 6 12 14 4 9 7 11 5 16 2 8 10 3 13	3/ 2/ 1/ 5/ 71/253# 1 16 7 10 11 6 13 4 14 3 12 5 8 9 2 15	3/ 2/ 1/ 7/ 72/254# 1 15 8 10 12 6 13 3 14 4 11 5 7 9 2 16
3/ 2/ 5/ 1/ 73/257# 1 16 6 11 10 7 13 4 15 2 12 5 8 9 3 14	3/ 2/ 5/ 7/ 74/258# 1 15 6 12 10 8 13 3 16 2 11 5 7 9 4 14	3/ 2/ 7/ 1/ 75/261# 1 14 8 11 12 7 13 2 15 4 10 5 6 9 3 16	3/ 2/ 7/ 5/ 76/262# 1 14 7 12 11 8 13 2 16 3 10 5 6 9 4 15	3/ 5/ 1/ 2/ 77/281# 1 16 4 13 11 6 10 7 14 3 15 2 8 9 5 12	3/ 5/ 1/ 6/ 78/282# 1 15 4 14 12 6 9 7 13 3 16 2 8 10 5 11
3/ 5/ 2/ 1/ 79/285# 1 16 4 13 10 7 11 6 15 2 14 3 8 9 5 12	3/ 5/ 2/ 7/ 80/286# 1 15 4 14 10 8 11 5 16 2 13 3 7 9 6 12	3/ 5/ 6/ 1/ 81/289# 1 14 4 15 12 7 9 6 13 2 16 3 8 11 5 10	3/ 5/ 6/ 7/ 82/290# 1 13 4 16 12 8 9 5 14 2 15 3 7 11 6 10	3/ 5/ 7/ 2/ 83/293# 1 14 4 15 11 8 10 5 16 3 13 2 6 9 7 12	3/ 5/ 7/ 6/ 84/294# 1 13 4 16 12 8 9 5 15 3 14 2 6 10 7 11
4/ 1/ 2/ 8/ 85/337# 1 15 8 10 13 12 3 6 4 5 14 11 16 2 9 7	4/ 1/ 8/ 2/ 86/339# 1 14 8 11 13 12 2 7 4 5 15 10 16 3 9 6	4/ 2/ 1/ 8/ 87/341# 1 15 8 10 11 14 5 4 6 3 12 13 16 2 9 7	4/ 2/ 8/ 1/ 88/343# 1 14 8 11 10 15 5 4 7 2 12 13 16 3 9 6	4/ 8/ 1/ 2/ 89/353# 1 12 8 13 11 14 2 7 6 3 15 10 16 5 9 4	4/ 8/ 2/ 1/ 90/354# 1 12 8 13 10 15 3 6 7 2 14 11 16 5 9 4
5/ 1/ 2/ 3/ 91/361# 1 16 10 7 6 11 13 4 15 2 8 9 12 5 3 14	5/ 1/ 2/ 6/ 92/362# 1 15 10 8 6 12 13 3 16 2 7 9 11 5 4 14	5/ 1/ 3/ 2/ 93/363# 1 14 12 7 8 11 13 2 15 4 6 9 10 5 3 16	5/ 1/ 3/ 6/ 94/364# 1 14 11 8 7 12 13 2 16 3 6 9 10 5 4 15	5/ 1/ 6/ 2/ 95/365# 1 15 12 6 7 14 9 4 10 3 8 13 16 2 5 11	5/ 1/ 6/ 7/ 96/366# 1 14 12 7 6 15 9 4 11 2 8 13 16 3 5 10

91/361#	92/362#	93/365#	94/366#	95/369#	96/370#
1 16 7 10	1 15 8 10	1 16 6 11	1 15 6 12	1 14 8 11	1 14 7 12
6 11 4 13	6 12 3 13	7 10 4 13	8 10 3 13	7 12 2 13	8 11 2 13
12 5 14 3	11 5 14 4	12 5 15 2	11 5 16 2	10 5 15 4	10 5 16 3
15 2 9 8	16 2 9 7	14 3 9 8	14 4 9 7	16 3 9 6	15 4 9 6
5/ 2/ 1/ 3/	5/ 2/ 1/ 6/	5/ 2/ 3/ 1/	5/ 2/ 3/ 7/	5/ 2/ 6/ 1/	5/ 2/ 6/ 7/
97/385#	98/386#	99/389#	100/390#	101/393#	102/394#
1 16 7 10	1 15 8 10	1 16 6 11	1 15 6 12	1 14 8 11	1 13 8 12
4 13 6 11	4 14 5 11	4 13 7 10	4 14 7 9	4 15 5 10	4 16 5 9
14 3 12 5	13 3 12 6	15 2 12 5	16 2 11 5	13 2 12 7	14 2 11 7
15 2 9 8	16 2 9 7	14 3 9 8	13 3 10 8	16 3 9 6	15 3 10 6
5/ 2/ 7/ 3/	5/ 2/ 7/ 6/	5/ 3/ 1/ 2/	5/ 3/ 1/ 6/	5/ 3/ 2/ 1/	5/ 3/ 2/ 7/
103/397#	104/398#	105/417#	106/418#	107/421#	108/422#
1 14 7 12	1 13 8 12	1 16 4 13	1 15 4 14	1 16 4 13	1 15 4 14
4 15 6 9	4 16 5 9	7 10 6 11	8 10 5 11	6 11 7 10	6 12 7 9
16 3 10 5	15 3 10 6	14 3 15 2	13 3 16 2	15 2 14 3	16 2 13 3
13 2 11 8	14 2 11 7	12 5 9 8	12 6 9 7	12 5 9 8	11 5 10 8
5/ 3/ 6/ 1/	5/ 3/ 6/ 7/	5/ 3/ 7/ 2/	5/ 3/ 7/ 6/	5/ 6/ 1/ 2/	5/ 6/ 1/ 3/
109/425#	110/426#	111/429#	112/430#	113/449#	114/450#
1 14 4 15	1 13 4 16	1 14 4 15	1 13 4 16	1 12 8 13	1 12 7 14
8 11 5 10	8 12 5 9	7 12 6 9	8 12 5 9	7 14 2 11	8 13 2 11
13 2 16 3	14 2 15 3	16 3 13 2	15 3 14 2	10 3 15 6	10 3 16 5
12 7 9 6	11 7 10 6	10 5 11 8	10 6 11 7	16 5 9 4	15 6 9 4
5/ 6/ 2/ 1/	5/ 6/ 2/ 7/	5/ 6/ 3/ 1/	5/ 6/ 3/ 7/	5/ 6/ 7/ 2/	5/ 6/ 7/ 3/
115/453#	116/454#	117/457#	118/458#	119/461#	120/462#
1 12 8 13	1 11 8 14	1 12 6 15	1 11 6 16	1 10 8 15	1 10 7 16
6 15 3 10	6 16 3 9	8 13 3 10	8 14 3 9	7 16 2 9	8 15 2 9
11 2 14 7	12 2 13 7	11 2 16 5	12 2 15 5	12 3 13 6	12 3 14 5
16 5 9 4	15 5 10 4	14 7 9 4	13 7 10 4	14 5 11 4	13 6 11 4
5/ 7/ 2/ 3/	5/ 7/ 2/ 6/	5/ 7/ 3/ 2/	5/ 7/ 3/ 6/	5/ 7/ 6/ 2/	5/ 7/ 6/ 3/
121/465#	122/466#	123/469#	124/470#	125/473#	126/474#
1 12 7 14	1 11 8 14	1 12 6 15	1 11 6 16	1 10 8 15	1 10 7 16
6 15 4 9	6 16 3 9	7 14 4 9	8 14 3 9	7 16 2 9	8 15 2 9
16 5 10 3	15 5 10 4	16 5 11 2	15 5 12 2	14 5 11 4	14 5 12 3
11 2 13 8	12 2 13 7	10 3 13 8	10 4 13 7	12 3 13 6	11 4 13 6
8/ 1/ 2/ 4/	8/ 1/ 4/ 2/	8/ 2/ 1/ 4/	8/ 2/ 4/ 1/	8/ 4/ 1/ 2/	8/ 4/ 2/ 1/
127/505#	128/507#	129/509#	130/511#	131/521#	132/522#
1 8 15 10	1 8 14 11	1 8 15 10	1 8 14 11	1 8 12 13	1 8 12 13
6 12 3 13	7 12 2 13	4 14 5 11	4 15 5 10	7 14 2 11	6 15 3 10
11 5 14 4	10 5 15 4	13 3 12 6	13 2 12 7	10 3 15 6	11 2 14 7
16 9 2 7	16 9 3 6	16 9 2 7	16 9 3 6	16 9 5 4	16 9 5 4

[Count (n1=1)/Total = 132/528] OK!

I was terribly surprised at the result again. 528 solutions among 880 proved to be the "Euler Squares" even in their primary diagonals!

It is really amazing. These 528 solutions should make the core of all 880 solutions.

I would recommend you to analyze the decomposed layers of them directly and find anything beautiful about their structures and relations among solutions.

You have now come to the entrance to the fundamental study of magic squares.

(First version: Written on August-October, 2003 with MWCW"C" _for_Mac;
Revised on March 20, 2006 by Kanji Setsuda with MacOSX and Xcode 2.1)

E-Mail Address <jag12001@ni fty. ne. j p>