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

1. 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 Latin Units?

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

```
   [Basic Position]     [Self-Complementary Conditions]
    .-----------.  .-----------.   n1+n16=1; ->   n16=1-n1;
rw1| 1| 2| 3| 4|  | 1| 2| 3| 4|   n2+n15=1; ->   n15=1-n2;
   |--+--+--+--|  |--+--+--+--|   n3+n14=1; ->   n14=1-n3;
rw2| 5| 6| 7| 8|  | 5| 6| 7| 8|   n4+n13=1; ->   n13=1-n4;
   |--+--+--+--|  |--+--+--+--|   n5+n12=1; ->   n12=1-n5;
rw3| 9|10|11|12|  | 8| 7| 6| 5|   n6+n11=1; ->   n11=1-n6;
   |--+--+--+--|  |--+--+--+--|   n7+n10=1; ->   n10=1-n7;
rw4|13|14|15|16|  | 4| 3| 2| 1|   n8+n9=1;  ->    n9=1-n8;
   '-----------'  '-----------'
   cl1 cl2 cl3 cl4   [Basic Conditions]
  n1+n2+n3+n4=2       ... rw1;  n1+n5+n9+n13=2    ... cl1;
  n5+n6+n7+n8=2       ... rw2;  n2+n6+n10+n14=2   ... cl2;
  n9+n10+n11+n12=2    ... rw3;  n3+n7+n11+n15=2   ... cl3;
  n13+n14+n15+n16=2   ... rw4;  n4+n8+n12+n16=2   ... cl4;
  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:

```
/** Euler Squares of Order 4 for the Self-complementary Magic Type **/
/** Composed by the "New Euler's Method" with Binary Number System **/
/** 'CES44SC.c': Dictated by Kanji Setsuda in 2004, 2006, 2010 and **/
/**  Revised on Mar.20, 2011 with MacOSX 10.5.8 and Xcode 3.1.2 **/
/**/
 #include <stdio.h>
/**/
 short cnt, lcnt, tcnt;
 short u1, u2, u3, u4;
 short LSM, CC;
 short nm[17], uflg[17];
/**/
 short bcf[9][2];
 short tlu[9][17];
 short mtc[9][9];
 short ta[49][21];
 short d[2][21];
 short lnsm[2][9];
/**/
 void sstp01(void), sstp02(void), sstp03(void), sstp04(void);
 void sstp05(void), sstp06(void), sstp07(void), sstp08(void);
 void clurecord(void);
 void prlunit(void);
 void cmbcmp(void), recordsol(void);
 void sortsol(void), exc(short x);
```

```
 void chksum(void);
 void prpat2(void);
 void pr2sol(short x), pr6sol(short x, short y);
/**/
int main(){
 short m,n;
  printf("\n** Making the Self-complementary 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;}
  for(m=0;m<9;m++){for(n=0;n<2;n++){bcf[m][n]=0;}}
  lcnt=0; CC=1;
  sstp01();     // Make Latin Units
  printf(" [List of Latin Units with Decompositions by Binary Number System]\n");
  prlunit();    // Print Latin Units
  cnt=0; LSM=34;
  cmbcmp();     // Combine and Compose
  sortsol();    // Sort the Solutions
  printf("\n** Euler Squares of Order 4 for the Self-complementary Magic Type: **\n");
  printf("**  Composed by the 'New Euler's Method' with Binary Number System **\n");
  printf(" [List of Standard Solutions(Sorted): Used Units//Sol.Number#|Sum-Checks/]\n");
  pr2sol(12);
  pr6sol(13,cnt);
  printf(" [Solution Counts = %d]",cnt);
  printf("  [OK!]\n");
  printf("** Calculated and Listed by Kanji Setsuda on Mar.20, 2011 with MAC Xcode 3 **\n");
 return 0;
}
/**/
/** Program Modules for Latin Units of 'Euler Squares' of **/
/**  Order 4 for the Self-complementary Magic Squares 4x4 **/
/** Dictated by Kanji Setsuda on Mar.20, '11 with MacOSX & Xcode 3 **/
/**/
/* Level 1: */
/* Set n1 & n16 */
void sstp01(){
 short a,b;
  for(a=0;a<2;a++){b=CC-a;
    if((bcf[1][a]<2)&&(bcf[5][a]<2)){
      if((bcf[4][b]<2)&&(bcf[8][b]<2)){
        bcf[1][a]++; bcf[5][a]++;
        bcf[4][b]++; bcf[8][b]++;
        nm[1]=a; nm[16]=b;
        sstp02();
        bcf[8][b]--; bcf[4][b]--;
        bcf[5][a]--; bcf[1][a]--;
    }}
  }
}
/**/
/* Set n2 & n15 */
void sstp02(){
 short a,b;
  for(a=1;a>=0;a--){b=CC-a;
    if((bcf[1][a]<2)&&(bcf[6][a]<2)){
      if((bcf[4][b]<2)&&(bcf[7][b]<2)){
        bcf[1][a]++; bcf[6][a]++;
        bcf[4][b]++; bcf[7][b]++;
        nm[2]=a; nm[15]=b;
        sstp03();
        bcf[7][b]--; bcf[4][b]--;
        bcf[6][a]--; bcf[1][a]--;
    }}
  }
}
```

```
/**/
/* Set n3 & n14 */
void sstp03(){
 short a,b;
  for(a=1;a>=0;a--){b=CC-a;
    if((bcf[1][a]<2)&&(bcf[7][a]<2)){
      if((bcf[4][b]<2)&&(bcf[6][b]<2)){
        bcf[1][a]++; bcf[7][a]++;
        bcf[4][b]++; bcf[6][b]++;
        nm[3]=a; nm[14]=b;
        sstp04();
        bcf[6][b]--; bcf[4][b]--;
        bcf[7][a]--; bcf[1][a]--;
    }}
  }
}
/**/
/* Set n4 & n13 */
void sstp04(){
 short a,b;
  for(a=0;a<2;a++){b=CC-a;
    if((bcf[1][a]<2)&&(bcf[8][a]<2)){
      if((bcf[4][b]<2)&&(bcf[5][b]<2)){
        bcf[1][a]++; bcf[8][a]++;
        bcf[4][b]++; bcf[5][b]++;
        nm[4]=a; nm[13]=b;
        sstp05();
        bcf[5][b]--; bcf[4][b]--;
        bcf[8][a]--; bcf[1][a]--;
    }}
  }
}
/**/
/* Level 2: */
/* Set n5 & n12 */
void sstp05(){
 short a,b;
  for(a=1;a>=0;a--){b=CC-a;
    if((bcf[2][a]<2)&&(bcf[5][a]<2)){
      if((bcf[3][b]<2)&&(bcf[8][b]<2)){
        bcf[2][a]++; bcf[5][a]++;
        bcf[3][b]++; bcf[8][b]++;
        nm[5]=a; nm[12]=b;
        sstp06();
        bcf[8][b]--; bcf[3][b]--;
        bcf[5][a]--; bcf[2][a]--;
    }}
  }
}
/**/
/* Set n6 & n11 */
void sstp06(){
 short a,b;
  for(a=0;a<2;a++){b=CC-a;
    if((bcf[2][a]<2)&&(bcf[6][a]<2)){
      if((bcf[3][b]<2)&&(bcf[7][b]<2)){
        bcf[2][a]++; bcf[6][a]++;
        bcf[3][b]++; bcf[7][b]++;
        nm[6]=a; nm[11]=b;
        sstp07();
        bcf[7][b]--; bcf[3][b]--;
        bcf[6][a]--; bcf[2][a]--;
    }}
  }
```

```
}
/**/
/* Set n7 & n10 */
void sstp07(){
 short a,b;
  for(a=0;a<2;a++){b=CC-a;
    if((bcf[2][a]<2)&&(bcf[7][a]<2)){
      if((bcf[3][b]<2)&&(bcf[6][b]<2)){
        bcf[2][a]++; bcf[7][a]++;
        bcf[3][b]++; bcf[6][b]++;
        nm[7]=a; nm[10]=b;
        sstp08();
        bcf[6][b]--; bcf[3][b]--;
        bcf[7][a]--; bcf[2][a]--;
    }}
  }
}
/**/
/* Set n8 & n9 */
void sstp08(){
 short a,b;
  for(a=1;a>=0;a--){b=CC-a;
    if((bcf[2][a]<2)&&(bcf[8][a]<2)){
      if((bcf[3][b]<2)&&(bcf[5][b]<2)){
        bcf[2][a]++; bcf[8][a]++;
        bcf[3][b]++; bcf[5][b]++;
        nm[8]=a; nm[9]=b;
        clurecord();
        bcf[5][b]--; bcf[3][b]--;
        bcf[8][a]--; bcf[2][a]--;
    }}
  }
}
/**/
/* Save the Latin Units */
void clurecord(){
 short n;
  tlu[lcnt][0]=lcnt+1;
  for(n=1;n<17;n++){tlu[lcnt][n]=nm[n];}
  lcnt++;
}
/**/
/* Make Matching Table and Print the Latin Units */
void prlunit(){
 short t,l,l4,m,n;
  for(m=0;m<lcnt;m++){
    for(n=0;n<lcnt;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<lcnt;t=t+8){
    printf("%8d/%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]);}
        if(m<(t+7)){printf(" ");}
      }
      printf("\n");
    }
  }
  printf(" [Count of Latin Units = %d]\n",lcnt);
```

```
  printf("\n[Reference Table for the Best Combination]\n");
  printf("  *|");
  for(n=0;n<lcnt;n++){printf("%3d",n+1);}
  printf("\n");
  printf(" --+------------------------\n");
  for(m=0;m<lcnt;m++){
    printf("%3d|",m+1);
    for(n=0;n<lcnt;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])){
                recordsol();}
            }
    }}}}}}
  }
}
/**/
/* Record the Answers to the Table */
void recordsol(){
 short n;
  ta[cnt][0]=cnt+1;
  for(n=1;n<17;n++){ta[cnt][n]=nm[n];}
  ta[cnt][17]=u1; ta[cnt][18]=u2;
  ta[cnt][19]=u3; ta[cnt][20]=u4;
  cnt++;
}
/**/
/* Solution Sorting for the Smart List */
void sortsol(){
 short mx,m,f;
 short d1,d2,d3,d4,d5,d6;
  mx=cnt-1;
  do{f=0;
    for(m=0;m<mx;m++){d1=ta[m][1]; d2=ta[m+1][1];
      if(d1>d2){exc(m); f=1;}
      else{
        d3=(ta[m][2]*17+ta[m][3])*17+ta[m][5];
        d4=(ta[m+1][2]*17+ta[m+1][3])*17+ta[m+1][5];
        if((d1==d2)&&(d3<d4)){exc(m); f=1;}
        else{
          d5=(ta[m][4]*17+ta[m][6])*17+ta[m][9];
          d6=(ta[m+1][4]*17+ta[m+1][6])*17+ta[m+1][9];
```

```
            if((d1==d2)&&(d3==d4)&&(d5>d6)){exc(m); f=1;}
        }}
      }
      mx--;
    }while(f>0);
}
/**/
void exc(short x){
 short n,d;
   for(n=1;n<21;n++){d=ta[x][n]; ta[x][n]=ta[x+1][n]; ta[x+1][n]=d;}
}
/**/
/* Check every sums of rows and columns */
void chksum(){
 short m,m4,n,s;
   for(m=0;m<4;m++){m4=m*4; s=0;
      for(n=1;n<5;n++){s=s+d[0][m4+n];}
      lnsm[0][m]=s; s=0;
      for(n=0;n<4;n++){s=s+d[0][n*4+m+1];}
      lnsm[0][m+4]=s; s=0;
      for(n=1;n<5;n++){s=s+d[1][m4+n];}
      lnsm[1][m]=s; s=0;
      for(n=0;n<4;n++){s=s+d[1][n*4+m+1];}
      lnsm[1][m+4]=s;
   }
}
/**/
/* Print 2 Answers */
void pr2sol(short x){
 short t,l,l4,m,n,s,v;
   for(t=0;t<x;t=t+2){
      for(n=1;n<21;n++){d[0][n]=ta[t][n]; d[1][n]=ta[t+1][n];}
      chksum();
      printf("%5d/%4d/%4d/%4d/%12d#|RW|CL|%6d/%4d/%4d/%4d/%12d#|RW|CL|\n",
        d[0][17]+1,d[0][18]+1,d[0][19]+1,d[0][20]+1,t+1,
        d[1][17]+1,d[1][18]+1,d[1][19]+1,d[1][20]+1,t+2);
    /**/
      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[d[s][m+16]][l4+n];
              printf("%d",v);}
          }
          printf(" ");
          for(n=1;n<5;n++){printf("%3d",d[s][l4+n]);}
          printf("|%2d|%2d|",lnsm[s][l],lnsm[s][l+4]);
          if(s==0){printf(" ");}
        }
        printf("\n");
      }
   }
}
/**/
/* Print 6 Answers */
void pr6sol(short x, short y){
 short t,l,l4,m,n;
   for(t=(x-1);t<y;t=t+6){
      for(m=t;m<(t+6);m++){printf(" ");
        printf("%2d/%d/%d/%d/%2d#",
          ta[m][17]+1,ta[m][18]+1,ta[m][19]+1,ta[m][20]+1,m+1);
        if(m<(t+5)){printf(" ");}
      }
```

```
    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",ta[m][l4+n]);}
        if(m<(t+5)){printf(" ");}
      }
      printf("\n");
    }
  }
}
/**/
```

This is one of the recent programs of mine. Let me show you the result of this experiment calculating by this program.

```
** Making the Self-complementary Magic Squares of Order 4 **
**  by the 'New Euler's Method' with Binary Number System **
 [List of Latin Units with Decompositions by Binary Number System]
      1/       2/       3/       4/       5/       6/       7/       8/
 0 1 1 0  0 1 1 0  0 1 0 1  0 0 1 1  1 1 0 0  1 0 1 0  1 0 0 1  1 0 0 1
 1 0 0 1  0 1 1 0  1 0 1 0  1 1 0 0  0 0 1 1  0 1 0 1  1 0 0 1  0 1 1 0
 0 1 1 0  1 0 0 1  1 0 1 0  1 1 0 0  0 0 1 1  0 1 0 1  0 1 1 0  1 0 0 1
 1 0 0 1  1 0 0 1  0 1 0 1  0 0 1 1  1 1 0 0  1 0 1 0  0 1 1 0  0 1 1 0
 [Count of Latin 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

** Euler Squares of Order 4 for the Self-complementary Magic Type: **
**  Composed by the 'New Euler's Method' with Binary Number System **
 [List of Standard Solutions(Sorted): Used Units//Sol.Number#|Sum-Checks|]
    1/   2/   3/   4/           1#|RW|CL|     2/   1/   3/   4/           2#|RW|CL|
 0110 0110 0101 0011   1 15 14  4|34|34|   0110 0110 0101 0011   1 15 14  4|34|34|
 1001 0110 1010 1100  12  6  7  9|34|34|   0110 1001 1010 1100   8 10 11  5|34|34|
 0110 1001 1010 1100   8 10 11  5|34|34|   1001 0110 1010 1100  12  6  7  9|34|34|
 1001 1001 0101 0011  13  3  2 16|34|34|   1001 1001 0101 0011  13  3  2 16|34|34|
    1/   3/   2/   4/           3#|RW|CL|     2/   3/   1/   4/           4#|RW|CL|
 0110 0101 0110 0011   1 15 12  6|34|34|   0110 0101 0110 0011   1 15 12  6|34|34|
 1001 1010 0110 1100  14  4  7  9|34|34|   0110 1010 1001 1100   8 10 13  3|34|34|
 0110 1010 1001 1100   8 10 13  3|34|34|   1001 1010 0110 1100  14  4  7  9|34|34|
 1001 0101 1001 0011  11  5  2 16|34|34|   1001 0101 1001 0011  11  5  2 16|34|34|
    3/   1/   2/   4/           5#|RW|CL|     3/   2/   1/   4/           6#|RW|CL|
 0101 0110 0110 0011   1 15  8 10|34|34|   0101 0110 0110 0011   1 15  8 10|34|34|
 1010 1001 0110 1100  14  4 11  5|34|34|   1010 0110 1001 1100  12  6 13  3|34|34|
 1010 0110 1001 1100  12  6 13  3|34|34|   1010 1001 0110 1100  14  4 11  5|34|34|
 0101 1001 1001 0011   7  9  2 16|34|34|   0101 1001 1001 0011   7  9  2 16|34|34|
    1/   2/   4/   3/           7#|RW|CL|     2/   1/   4/   3/           8#|RW|CL|
 0110 0110 0011 0101   1 14 15  4|34|34|   0110 0110 0011 0101   1 14 15  4|34|34|
 1001 0110 1100 1010  12  7  6  9|34|34|   0110 1001 1100 1010   8 11 10  5|34|34|
 0110 1001 1100 1010   8 11 10  5|34|34|   1001 0110 1100 1010  12  7  6  9|34|34|
 1001 1001 0011 0101  13  2  3 16|34|34|   1001 1001 0011 0101  13  2  3 16|34|34|
```

```
  2/   3/   4/   1/          9#|RW|CL|     3/   2/   4/   1/         10#|RW|CL|
0110 0101 0011 0110   1 14 12  7|34|34|  0101 0110 0011 0110   1 14  8 11|34|34|
0110 1010 1100 1001   8 11 13  2|34|34|  1010 0110 1100 1001  12  7 13  2|34|34|
1001 1010 1100 0110  15  4  6  9|34|34|  1010 1001 1100 0110  15  4 10  5|34|34|
1001 0101 0011 1001  10  5  3 16|34|34|  0101 1001 0011 1001   6  9  3 16|34|34|

  2/   4/   1/   3/          11#|RW|CL|     2/   4/   3/   1/         12#|RW|CL|
0110 0011 0110 0101   1 12 15  6|34|34|  0110 0011 0101 0110   1 12 14  7|34|34|
0110 1100 1001 1010   8 13 10  3|34|34|  0110 1100 1010 1001   8 13 11  2|34|34|
1001 1100 0110 1010  14  7  4  9|34|34|  1001 1100 1010 0110  15  6  4  9|34|34|
1001 0011 1001 0101  11  2  5 16|34|34|  1001 0011 0101 1001  10  3  5 16|34|34|

1/2/3/5/13#   2/1/3/5/14#   1/3/2/5/15#   2/3/1/5/16#   3/1/2/5/17#   3/2/1/5/18#
 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

1/2/4/6/19#   2/1/4/6/20#   2/3/4/8/21#   3/2/4/8/22#   2/4/1/6/23#   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

1/3/5/2/25#   2/3/5/1/26#   3/1/5/2/27#   3/2/5/1/28#   2/3/8/4/29#   3/2/8/4/30#
 3 16 10  5    3 16 10  5    3 16  6  9    3 16  6  9    3 13 10  8    3 13  6 12
13  2  8 11    6  9 15  4   13  2 12  7   10  5 15  4    6 12 15  1   10  8 15  1
 6  9 15  4   13  2  8 11   10  5 15  4   13  2 12  7   16  2  5 11   16  2  9  7
12  7  1 14   12  7  1 14    8 11  1 14    8 11  1 14    9  7  4 14    5 11  4 14

2/4/6/1/31#   2/4/8/3/32#   1/3/5/7/33#   2/3/5/8/34#   3/1/5/7/35#   3/2/5/8/36#
 3 10 16  5    3 10 13  8    4 15  9  6    4 15  9  6    4 15  5 10    4 15  5 10
 6 15  9  4    6 15 12  1   14  1  7 12    5 10 16  3   14  1 11  8    9  6 16  3
13  8  2 11   16  5  2 11    5 10 16  3   14  1  7 12    9  6 16  3   14  1 11  8
12  1  7 14    9  4  7 14   11  8  2 13   11  8  2 13    7 12  2 13    7 12  2 13

2/3/8/5/37#   3/2/8/5/38#   2/4/6/8/39#   2/4/8/6/40#   3/5/1/2/41#   3/5/2/1/42#
 4 14  9  7    4 14  5 11    4  9 15  6    4  9 14  7    5 16  4  9    5 16  4  9
 5 11 16  2    9  7 16  2    5 16 10  3    5 16 11  2   11  2 14  7   10  3 15  6
15  1  6 12   15  1 10  8   14  7  1 12   15  6  1 12   10  3 15  6   11  2 14  7
10  8  3 13    6 12  3 13   11  2  8 13   10  3  8 13    8 13  1 12    8 13  1 12

2/8/3/4/43#   2/8/4/3/44#   3/5/1/7/45#   3/5/2/8/46#   2/8/3/5/47#   2/8/4/6/48#
 5 11 10  8    5 10 11  8    6 15  3 10    6 15  3 10    6 12  9  7    6  9 12  7
 4 14 15  1    4 15 14  1   12  1 13  8    9  4 16  5    3 13 16  2    3 16 13  2
16  2  3 13   16  3  2 13    9  4 16  5   12  1 13  8   15  1  4 14   15  4  1 14
 9  7  6 12    9  6  7 12    7 14  2 11    7 14  2 11   10  8  5 11   10  5  8 11
[Solution Counts = 48]
 [OK!]
** Calculated and Listed by Kanji Setsuda on Mar.20, 2011 with MacOSX & Xcode 3 **
```

As you see, we have succeeded in getting the complete list of 48 standard solutions of Self-complementary Magic Squares of order 4. You know you can always make a good job, every time you try to do by this binary method, even when you assume nothing for all pan-diagonals.

2. Type Converter Program between Self-Complementary and Complete MS44

You may possibly know that we can make any Self-complementary type of MS44 into Pan-diagonal (or Complete) one by some simple rules of transformation. You can also make it back into the original form by the same rules.

Just exchange 3rd row with 4th one and do the same thing with 3rd column and 4th one at the same time, and you will surely have what you want to.

```
         1#S   1#C                  2#S  25#C                  3#S   7#C
 1 15 14  4    1 15  4 14    2 16 13  3    2 16  3 13    1 14 15  4    1 14  4 15
12  6  7  9   12  6  9  7   11  5  8 10   11  5 10  8   12  7  6  9   12  7  9  6
 8 10 11  5   13  3 16  2    7  9 12  6   14  4 15  1    8 11 10  5   13  2 16  3
13  3  2 16    8 10  5 11   14  4  1 15    7  9  6 12   13  2  3 16    8 11  5 10
```

But you have to pay your special attention to what type of solution sets you should have. Because you cannot transform all 48 'standard' solutions of one type into the other one in one action.

```
        21#S  ?#C                  41#S  ?#C
 5 16  4  9    5 16  9  4     3 10 16  5    3 10  5 16
11  2 14  7   11  2  7 14     6 15  9  4    6 15  4  9
10  3 15  6    8 13 12  1    13  8  2 11   12  1 14  7
 8 13  1 12   10  3  6 15    12  1  7 14   13  8 11  2
```

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: n1<n16; n1<n4; n1<n13; and n2>n5; 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 lu,md,n,d,fc;
  lu=8; md=8;
  for(u1=0;u1<lu;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){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 Units//// Sol_Number#SC Old_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
  1/   2/   4/   3/          3#S          7#C    1/   2/   4/   6/          4#S         31#C
0110 0110 0011 0101   1 14 15  4   1 14  4 15  0110 0110 0011 1010   2 13 16  3   2 13  3 16
1001 0110 1100 1010  12  7  6  9  12  7  9  6  1001 0110 1100 0101  11  8  5 10  11  8 10  5
0110 1001 1100 1010   8 11 10  5  13  2 16  3  0110 1001 1100 0101   7 12  9  6  14  1 15  4
1001 1001 0011 0101  13  2  3 16   8 11  5 10  1001 1001 0011 1010  14  1  4 15   7 12  6  9
  1/   2/   5/   3/          5#S         49#C    1/   2/   5/   6/          6#S         73#C
0110 0110 1100 0101   3 16 13  2   3 16  2 13  0110 0110 1100 1010   4 15 14  1   4 15  1 14
1001 0110 0011 1010  10  5  8 11  10  5 11  8  1001 0110 0011 0101   9  6  7 12   9  6 12  7
0110 1001 0011 1010   6  9 12  7  15  4 14  1  0110 1001 0011 0101   5 10 11  8  16  3 13  2
1001 1001 1100 0101  15  4  1 14   6  9  7 12  1001 1001 1100 1010  16  3  2 13   5 10  8 11
  1/   2/   6/   4/          7#S         55#C    1/   2/   6/   5/          8#S         79#C
0110 0110 1010 0011   3 13 16  2   3 13  2 16  0110 0110 1010 1100   4 14 15  1   4 14  1 15
1001 0110 0101 1100  10  8  5 11  10  8 11  5  1001 0110 0101 0011   9  7  6 12   9  7 12  6
0110 1001 0101 1100   6 12  9  7  15  1 14  4  0110 1001 0101 0011   5 11 10  8  16  2 13  3
1001 1001 1010 0011  15  1  4 14   6 12  7  9  1001 1001 1010 1100  16  2  3 13   5 11  8 10
  1/   3/   2/   4/          9#S          3#C    1/   3/   2/   5/         10#S         27#C
0110 0101 0110 0011   1 15 12  6   1 15  6 12  0110 0101 0110 1100   2 16 11  5   2 16  5 11
1001 1010 0110 1100  14  4  7  9  14  4  9  7  1001 1010 0110 0011  13  3  8 10  13  3 10  8
0110 1010 1001 1100   8 10 13  3  11  5 16  2  0110 1010 1001 0011   7  9 14  4  12  6 15  1
1001 0101 1001 0011  11  5  2 16   8 10  3 13  1001 0101 1001 1100  12  6  1 15   7  9  4 14
  1/   3/   4/   2/         11#S          9#C    1/   3/   4/   7/         12#S         33#C
0110 0101 0011 0110   1 14 12  7   1 14  7 12  0110 0101 0011 1001   2 13 11  8   2 13  8 11
1001 1010 1100 0110  15  4  6  9  15  4  9  6  1001 1010 1100 1001  16  3  5 10  16  3 10  5
0110 1010 1100 1001   8 11 13  2  10  5 16  3  0110 1010 1100 0110   7 12 14  1   9  6 15  4
1001 0101 0011 1001  10  5  3 16   8 11  2 13  1001 0101 0011 0110   9  6  4 15   7 12  1 14
  1/   3/   5/   2/         13#S         51#C    1/   3/   5/   7/         14#S         75#C
0110 0101 1100 0110   3 16 10  5   3 16  5 10  0110 0101 1100 1001   4 15  9  6   4 15  6  9
1001 1010 0011 0110  13  2  8 11  13  2 11  8  1001 1010 0011 1001  14  1  7 12  14  1 12  7
0110 1010 0011 1001   6  9 15  4  12  7 14  1  0110 1010 0011 0110   5 10 16  3  11  8 13  2
1001 0101 1100 1001  12  7  1 14   6  9  4 15  1001 0101 1100 0110  11  8  2 13   5 10  3 16
  1/   3/   7/   4/         15#S         57#C    1/   3/   7/   5/         16#S         81#C
0110 0101 1001 0011   3 13 10  8   3 13  8 10  0110 0101 1001 1100   4 14  9  7   4 14  7  9
1001 1010 1001 1100  16  2  5 11  16  2 11  5  1001 1010 1001 0011  15  1  6 12  15  1 12  6
0110 1010 0110 1100   6 12 15  1   9  7 14  4  0110 1010 0110 0011   5 11 16  2  10  8 13  3
1001 0101 0110 0011   9  7  4 14   6 12  1 15  1001 0101 0110 1100  10  8  3 13   5 11  2 16
  1/   4/   2/   3/         17#S         13#C    1/   4/   2/   6/         18#S         37#C
0110 0011 0110 0101   1 12 15  6   1 12  6 15  0110 0011 0110 1010   2 11 16  5   2 11  5 16
1001 1100 0110 1010  14  7  4  9  14  7  9  4  1001 1100 0110 0101  13  8  3 10  13  8 10  3
0110 1100 1001 1010   8 13 10  3  11  2 16  5  0110 1100 1001 0101   7 14  9  4  12  1 15  6
1001 0011 1001 0101  11  2  5 16   8 13  3 10  1001 0011 1001 1010  12  1  6 15   7 14  4  9
  1/   4/   3/   2/         19#S         15#C    1/   4/   3/   7/         20#S         39#C
0110 0011 0101 0110   1 12 14  7   1 12  7 14  0110 0011 0101 1001   2 11 13  8   2 11  8 13
1001 1100 1010 0110  15  6  4  9  15  6  9  4  1001 1100 1010 1001  16  5  3 10  16  5 10  3
0110 1100 1010 1001   8 13 11  2  10  3 16  5  0110 1100 1010 0110   7 14 12  1   9  4 15  6
1001 0011 0101 1001  10  3  5 16   8 13  2 11  1001 0011 0101 0110   9  4  6 15   7 14  1 12
```

```
  1/   4/   6/   2/         21#S         61#C    1/   4/   6/   7/         22#S         85#C
0110 0011 1010 0110   3 10 16  5   3 10  5 16  0110 0011 1010 1001   4  9 15  6   4  9  6 15
1001 1100 0101 0110  13  8  2 11  13  8 11  2  1001 1100 0101 1001  14  7  1 12  14  7 12  1
0110 1100 0101 1001   6 15  9  4  12  1 14  7  0110 1100 0101 0110   5 16 10  3  11  2 13  8
1001 0011 1010 1001  12  1  7 14   6 15  4  9  1001 0011 1010 0110  11  2  8 13   5 16  3 10
  1/   4/   7/   3/         23#S         63#C    1/   4/   7/   6/         24#S         87#C
0110 0011 1001 0101   3 10 13  8   3 10  8 13  0110 0011 1001 1010   4  9 14  7   4  9  7 14
1001 1100 1001 1010  16  5  2 11  16  5 11  2  1001 1100 1001 0101  15  6  1 12  15  6 12  1
0110 1100 0110 1010   6 15 12  1   9  4 14  7  0110 1100 0110 0101   5 16 11  2  10  3 13  8
1001 0011 0110 0101   9  4  7 14   6 15  1 12  1001 0011 0110 1010  10  3  8 13   5 16  2 11
```

```
* Monitor List of Solution Correspondences *
  ??: 0
   1: 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
  41: 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
  81: 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 121: 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 161: 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 201: 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 241: 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 281: 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 321: 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 361: 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
[OK!]
```

Let's compare the next two sets of basic Latin Units of those types to analyze.

```
[Latin Units of Binary Decompositions for Complete MS44]
     1/C       2/C       3/C       4/C       5/C       6/C       7/C       8/C
 0 1 0 1   0 1 0 1   0 1 1 0   0 0 1 1   1 1 0 0   1 0 0 1   1 0 1 0   1 0 1 0
 1 0 1 0   0 1 0 1   1 0 0 1   1 1 0 0   0 0 1 1   0 1 1 0   1 0 1 0   0 1 0 1
 1 0 1 0   1 0 1 0   0 1 1 0   0 0 1 1   1 1 0 0   1 0 0 1   0 1 0 1   0 1 0 1
 0 1 0 1   1 0 1 0   1 0 0 1   1 1 0 0   0 0 1 1   0 1 1 0   0 1 0 1   1 0 1 0

[Latin Units of Binary Decompositions for Self-complementary MS44]
     1/S       2/S       3/S       4/S       5/S       6/S       7/S       8/S
 0 1 1 0   0 1 1 0   0 1 0 1   0 0 1 1   1 1 0 0   1 0 1 0   1 0 0 1   1 0 0 1
 1 0 0 1   0 1 1 0   1 0 1 0   1 1 0 0   0 0 1 1   0 1 0 1   1 0 0 1   0 1 1 0
 0 1 1 0   1 0 0 1   1 0 1 0   1 1 0 0   0 0 1 1   0 1 0 1   0 1 1 0   1 0 0 1
 1 0 0 1   1 0 0 1   0 1 0 1   0 0 1 1   1 1 0 0   1 0 1 0   0 1 1 0   0 1 1 0
```

Do you notice that in each pair of (1/C, 1/S), (2/C, 2/S), (3/C, 3/S), (4/C, 4/S), ..., (8/C, 8/S) the 3rd row/column and 4th row/column are really exchanged with each other?

When you use the type converter program above, you can always transform 1/C into 1/S, 2/C into 2/S, 3/C into 3/S, ... and 8/C into 8/S. Of course, you can also make them back like 1/S into 1/C, 2/S into 2/C, ... and 8/S into 8/C.

It is certain there is the 'one-to-one correspondence' between the two types even in their essential structures. They are just two faces of the same thing. How impressive!

It is also impressive to me that there is only one solution set among various types of Pan-Diagonal MS44, whether it appears as 'Pan-Diagonal', 'Complete', "Composite & Complete" or "Complete Euler Square".

On top of that the solution set of Self-Complementary MS44 has the 'one-to-one correspondence' with each one of Pan-Diagonal type.

All seem to be the same. I would like to call this as "Miraculous Unity of Order 4".

In the world of order 8, everything gets its own unique form and individual solution

set, after developing itself and differentiating from one another out of this Unity.

3. Making 4-Dimensional ECMF of Order2 developed into MS of Order 4

I would like to examine this new way of composing Euler Squares by binary system, trying an experiment to build the 4-Dimensional Extra-Cubic Magic Forms of order 2.

I would also like to make it into 2 types of magic squares of order 4 by dimensional down-converting. Can we really make them only with '0' and '1'?

*** Definition Diagrams and Equations: ***

```
 1/d0          /d1           | n1+n2+n3+n4=C     | n9+n10+n11+n12=C
  1---- 2       9----10      | n1+n2+n5+n6=C     | n9+n10+n13+n14=C
  |  3--+- 4    | 11--+-12   | n1+n3+n5+n7=C     | n9+n11+n13+n15=C
  5--|- 6  |   13--|-14  |   | n2+n4+n6+n8=C     | n10+n12+n14+n16=C
     7---- 8      15----16   | n3+n4+n7+n8=C     | n11+n12+n15+n16=C
                             | n5+n6+n7+n8=C     | n13+n14+n15+n16=C
 2/d0          /d1           | n1+n2+n3+n4=C     | n5+n6+n7+n8=C
  1---- 2       5---- 6      | n1+n2+n9+n10=C    | n5+n6+n13+n14=C
  |  3--+- 4    |  7--+- 8   | n1+n3+n9+n11=C    | n5+n7+n13+n15=C
  9--|-10  |   13--|-14  |   | n2+n4+n10+n12=C   | n6+n8+n14+n16=C
    11----12      15----16   | n3+n4+n11+n12=C   | n7+n8+n15+n16=C
                             | n9+n10+n11+n12=C  | n13+n14+n15+n16=C
 3/d0          /d1           | n1+n2+n5+n6=C     | n3+n4+n7+n8=C
  1---- 2       3---- 4      | n1+n2+n9+n10=C    | n3+n4+n11+n12=C
  |  5--+- 6    |  7--+- 8   | n1+n5+n9+n13=C    | n3+n7+n11+n15=C
  9--|-10  |   11--|-12  |   | n2+n6+n10+n14=C   | n4+n8+n12+n16=C
    13----14      15----16   | n5+n6+n13+n14=C   | n7+n8+n15+n16=C
                             | n9+n10+n13+n14=C  | n11+n12+n15+n16=C
 4/d0          /d1           | n1+n3+n5+n7=C     | n2+n4+n6+n8=C
  1---- 3       2---- 4      | n1+n3+n9+n11=C    | n2+n4+n10+n12=C
  |  5--+- 7    |  6--+- 8   | n1+n5+n9+n13=C    | n2+n6+n10+n14=C
  9--|-11  |   10--|-12  |   | n3+n7+n11+n15=C   | n4+n8+n12+n16=C
    13----15      14----16   | n5+n7+n13+n15=C   | n6+n8+n14+n16=C
                             | n9+n11+n13+n15=C  | n10+n12+n14+n16=C
```

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

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

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

```
n1+n16=n2+n15=n3+n14=n4+n13=n5+n12=n6+n11=n7+n10=n8+n9=CC  ...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 **/
/** 'CEulerSXQ2.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 prlunit(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[Latin Units of Binary Decompositions]\n");
  prlunit();
  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 Latin 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 Latin Units */
void prlunit(){
 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 Latin 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 Latin 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  **
 [Latin 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 Latin 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.

(Written first in August-October, 2003; Revised on March 20, 2006;
and on March 21, 2011 by Kanji Setsuda with MacOSX and Xcode 3.1)

E-Mail Address <jag12001@nifty.com>