Report Number: WUCS-79-3

1979-06-01

Gauss-Jordan Elimination By VLSI Mech-Connected Processors

Takayuki D. Kimura

It is shown that a mesh-connected n x (n+m) toroidal array of processors can perform Gauss-Jordan elimination without pivoting, on an n x (n+m) matrix, in 4n+m-1 steps, each step involving at most two artithmetic operations for every processor.

GAUSS-JORDAN ELIMINATION
BY VLSI MESH-CONNECTED PROCESSORS

Takayuki Kimura

WUCS-79-3

Department of Computer Science

Washington University

St. Louis, Missouri 63130

June 1979

ABSTRACT

It is shown that a mesh-connected n x (n+m) toroidal array of processors can perform Gauss-Jordan elimination without pivoting, on an n x (n+m) matrix, in 4n+m-1 steps, each step involving at most two arithmetic operations for every processor.

Keywords: mesh-connected multiprocessor system, Gauss-Jordan elimination, communication complexity

CR Categories: 5.14, 6.29

1. Introduction

The purpose of this paper is to show that a mesh-connected $n \times (n+m)$ toroidal array of processors can perform Gauss-Jordan elimination without pivoting, on an $n \times (n+m)$ matrix, in $4n+m-1$ steps, each involving at most two arithmetic operations for every processor. As a corollary, a system of n linear equations can be solved in $4n$ steps by an $n \times (n+1)$ troid of processors, if the process does not involve pivoting. Van Scoy[8], whose result is closest to ours, shows that a matrix inversion by Gaussian elimination with partial pivoting requires $O(n^2)$ steps with $n \times n$ processors, where n is the order of the matrix.

Our result is not better, on surface, than the known result of the following parallel algorithm for Gauss-Jordan elimination, requiring $3n+1$ steps by $(n-1)(n+m)$ processors (Heller[2]).

G-J Algorithm:

```
for j=1 step 1 until n do
    row i := row i - $(a_{ij}/a_{jj})$ row j ;   $(1 \le i \le n,\ i \neq j)$
     $x_{ij} := a_{ij}/a_{ii}$ ;   $(1 \le i \le n,\ n+1 \le j \le n+m)$.
```

In assessing the time complexity of this algorithm to be $3n+1$, it is assumed that the datum a_{jj} is accessible to every processor within a constant time that is independent of n. We assert that the assumption is not practical from our point of view.

Our objective is to minimize the communication complexity rather than the processing complexity (the number of arithmetic operations), and to maximize the degree of distribution of communication activities in an actual execution of an algorithm.

One way of satisfying the assumption made for the G-J algorithm is to provide a private communication path between every pair of processors. This requires $O(n^4)$ connections within the system and $O(n^2)$ communication ports for each processor. Another way is to store all data in a shared memory space to which every processor is connected in such a way that the memory contention problem can be resolved within a fixed amount delay that is independent of the number of processors.

Even though the above kinds of requirements for communication limit the value of an algorithm to theoretical interests, particularly when the number of processors is unbounded, they have been shared by many of known parallel algorithms, for example, those surveyed in Sameh[6] and Heller[2].

Some matrix algorithms have been constructed with conscious efforts to minimize the communication complexity. For example, Kant & Kimura [3] give an algorithm, called KK-algorithm, for solving a system of n linear equations in 2n-1 steps by a hex-connected (four neighbours plus one diagonal) $n \times (n+1)$ array of processors, and Kung & Leiserson[4] show that LU-decomposition (factoring a matrix into lower and upper triangular matrices) of an $n \times n$ matrix can be done in 4n steps, including I/O time, by an $n \times n$ hexagonal 'systolic' array.

The KK-algorithm requires the coefficient matrix to be 'strongly non-singular', that is to say, every square submatrix is non-singular, and Kung & Leiserson's algorithm assumes that the matrix can be factored by Gauss-Jordan elimination without pivoting. Both algorithms, however, can be executed by an array of processors in which every processor has a fixed number of communication ports (6 ports, in fact) that is independent of the size of the matrix.

The importance of a communication complexity in a matrix computation is discussed in Gentleman[1]. The significance of the communication issue in computer science, with the ongoing impact of VLSI technology (Mead & Conway[5]), is expounded in Sutherland & Mead [7].

Even though our algorithm is amenable to SIMD computer architectures such as ILLIAC IV, it is best fit to VLSI architecures by the virtue of its uniformity in the distribution of processing and memory requirements, and its simplicity in the topology of communication requirements. The algorithm requires that each processor have three data registers, three control registers (two bits long each),and arithmetic capabilities. Each processor communicates with immediate four neighbours only. A single step of computation consists of either one division or one addition and one multiplication.

A practical implication of the algorithm is that, for example, assuming 10 μs of division time, 100 × 100 matrix can be inversted every 5 ms by a two-dimensional array of 20,000 chip processors.

The technique of our construction is a two dimensional pipelining.

In the next section, we will define the underlying architecture. The algorithm will be given in Section 3 without a mathematical proof of correctness. In Section 4, we will demonstrate how the algorithm works, step by step, with an example of inverting a 4 × 4 matrix.

2. The Toroid System

We define a toroid system as a two dimensional array of $p \times q$ processors, $\{P(i,j) \mid 1 \leq i \leq p,\ 1 \leq j \leq q\}$, mesh-connected into a toroidal shape as shown in Figure 1. All processors have the same structure and the same capabilities which are independent of p and q; i.e. a fixed number of data registers, control registers and arithmetic capabilities. Contents of the registers represent a state of a processor. Every processor has four neighbours and it synchronizes with them locally, in the sense that the processor can recognize any state-change in the neighbouring processors.

Logically speaking, a toroid system works without any mechanism of global communication (synchronization). We assume, however, for the sake of simplicity, that all processors will change its state at the same time as if there exist a global clock.

The next state of a processor is a function of the current state of the processor itself and the current states of the neighbours. (See Figure 2) The next state function is the same for all the $p \times q$ processors in the system. Consequently, a toroid system can be uniquely identified by (1) the dimension p and q, (2) the registers and their types (the type of values each register can contain), and (3) the next-state function. A computation by a toroid system can be identified by the initial configuration and the final configuration in addition to the above three items.

As an example, a toroid system for matrix multiplication is given in Figure 3. Note that the next-state function is given in the form of a schematic replacement rule.

The actual computation of

$$\begin{bmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{bmatrix} \begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 5 & 8 & 11 \\ 7 & 10 & 13 \\ 9 & 12 & 15 \end{bmatrix}$$

by the toroid system is given in Figure 4. The initialization of x_{ij} and y_{ij} requires matrix transformations $(a_{ij}) \longmapsto (a_{i,i+j-1})$ and $(b_{ij}) \longmapsto (b_{i+j-1})$. There exists a toroid system that can perform exactly the above transformations in linear time. The reader is encouraged to try a construction.

3. The Algorithm

In this section, we will describe a toroid system GJ for Gauss-Jordan elimination without pivoting, working on matrices $A = (a_{ij})$ and $B = (b_{ik})$, where $1 \leq i,j \leq n$, $1 \leq k \leq m \leq n$. The matrix A must be a nonsingular square matrix which can be inverted without pivoting. The matrix B may not be square.

Initially, the toroid contains (A,B) and after $4n+m-1$ steps of transformations, it will contain $(I, A^{-1}B)$, where I is the identity matrix of order n. (Figure 5)

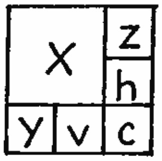
A specification of GJ is as follows:

Dimension: $n \times (n+m)$

Registers: x,y,z: Real

h,v: $\{-1,0,1\}$

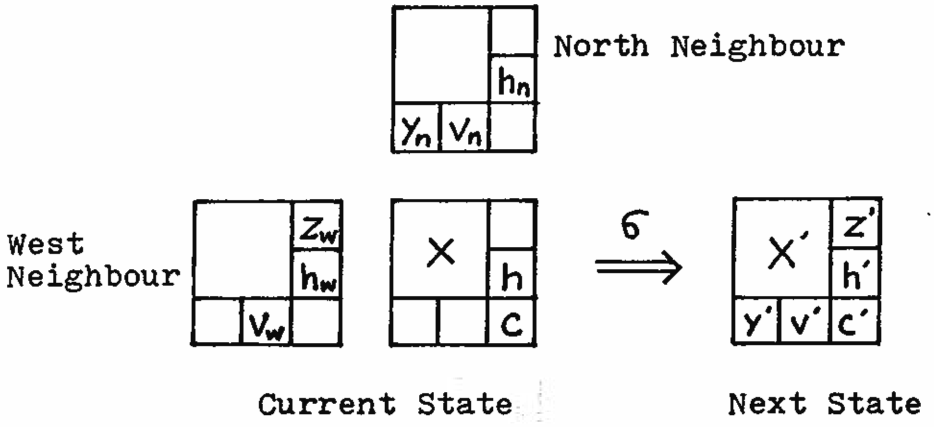
c: $\{-1,0,1,2\}$



Processor

Next-State

Function : $(x',y',z',c',h',v') := \sigma(x,h,c,y_n,h_n,v_n,z_w,h_w,v_w)$



(The exact definition of the next-state function σ is given in Table I, and is represented schematically in Figure 6.)

Initial

Configuration:

$$\begin{cases} x_{ij} = a_{ij} & 1 \le i \le n, \quad 1 \le j \le n, \\ \phantom{x_{ij}} = b_{ik} & 1 \le i \le n, \quad n+k \le j \le n+m. \end{cases}$$

$$y_{ij} = z_{ij} = h_{ij} = v_{ij} = 0 \qquad 1 \le i \le n, \quad 1 \le j \le n+m.$$

$$\begin{cases} c_{ij} = 1 & \text{if} \quad i=j=1, \\ \phantom{c_{ij}} = -1 & \text{if} \quad j=i+n, \\ \phantom{c_{ij}} = 0 & \text{otherwise.} \end{cases}$$

Final

Configuration:

$$\begin{cases} x_{ij} = 1 & 1 \le i,j \le n \quad \text{and} \quad i=j, \\ \phantom{x_{ij}} = 0 & 1 \le i,j \le n \quad \text{and} \quad i \ne j, \\ \phantom{x_{ij}} = \tilde{a}_{ik} & 1 \le i \le n, \quad n+1 \le j=n+k \le n+m, \end{cases}$$

$$\text{where } (\tilde{a}_{ik}) = A^{-1}B,$$

$$y_{ij} = z_{ij} = h_{ij} = v_{ij} = 0 \qquad 1 \le i \le n, \quad 1 \le j \le n+m,$$

$$c_{ij} = 2 \qquad 1 \le i \le n, \quad 1 \le j \le n+m.$$

With respect to Table I, the following should be noted:

(1) The blanks in the left half indicate "don't care" conditions, and in the right half, they indicate that no change occurs on the value.

(2) With a proper initialization, the following conditions always hold: (a) if $h=-1$, then $c=2$, (i.e. -1 does not appear for h),

(b) if $v_n=1$, then $h_w=0$, (Rule(5)),

(c) if $c=-1$ and $h_w, v_n \le 0$, then $h_w=v_n=0$, (Rule(7)),

(d) if $c=0$ and $h=0$, then $h_w+v_n \ge -1$, (i.e. $-1 \le a \le 0$).

(e) if $h=1$, then $h_w=v_n=0$, (Rule(3)).

The next-state function σ is broken down into seven disjoint functions (transformation rules), and for every processor at any moment, there exists exactly one rule applicable to the processor, provided that the system start with a properly initialized configuration. In that case, every computation is deterministic.

The essential part of the algorithm works in the following way. Consider the formula for Gauss-Jordan elimination:

```
for j=1 step 1 until n do  $a_{ik} := a_{ik} - (a_{ij}/a_{jj}) \cdot a_{jk}$ ;
    ($1 \leq i \leq n$ and $i \neq j$,  $1 \leq k \leq n+m$ and $k \neq j$).
```

The GJ toroid implements the computation by pipelining the individual assignment operations over a two dimensional array of processors. Ignoring the pipelining aspect of the algorithm, let us assume that the toroid is ready to start the process of executing the assignment operation $a_{ik} := a_{ik} - (a_{ij}/a_{jj}) \cdot a_{jk}$ at some point of the computation, say at the p-th step of transformation, and also assume that a_{ik}, a_{ij}, a_{jj}, and a_{jk} are now contained in the x registers of the processors P(i,k), P(i,j), P(j,j) and P(j,k), respectively. Further, without loss of generality, we assume that $j<i<k$. Then, Figure 7 shows the data flow for this particular assignment operation.

The processor P(j,j) starts the process by making a_{jj} available to the east neighbour and a signal 1 to the south neighbour at the (p+1)-th step. It also changes its value to 1 at the same time. (Figure 7(ii)) The datum a_{jj} travels eastwards through the z registers of the processors on the j-th row. Similarly, the signal 1 travels southwards through the v registers on the j-th column.

The signal 1 becomes available to P(i,j) at Step (p+i-j), before a_{jj} arrives at P(j,k), (recall that i<k), and P(i,j) makes a_{ij} available to its east neighbour through the register z, at the next step (p+i-j+1). (Figure 7(iii)) At the same time, P(i,j) changes its value to 0 and keeps passing the signal 1 towards south.

When the datum a_{jj} becomes available to P(j,k) at Step (p+k-j), P(j,k) changes its value to a_{jk}/a_{jj}, and makes it available to the south neighbour through the register y, at the next step (p+k-j+1). (Figure 7(iv)) Since the path length between P(j,j) and P(i,k) via P(i,j) is the same as the path via P(j,k), the data a_{jk}/a_{jj} and a_{ij} will become available to P(i,k) at the same time, i.e. at Step (p+i+k-2j). The processor P(i,k) completes the process by executing the assignment operation $a_{ik}:=a_{ik}-(a_{jk}/a_{jj})\cdot a_{ij}$ at the next step (p+i+k-2j+1). (Figure 7(v))

The processor P(j+1,j+1) will initiate a similar process at the (p+3)-th step, i.e. 3 steps behind P(j,j), to implement the assignment operation $a_{ik}:=a_{ik}-(a_{i,j+1}/a_{j+1,j+1})\cdot a_{j+1,k}$. The processor P(i,k) will receive the data $(a_{j+1,k}/a_{j+1,j+1})$ and $a_{i,j+1}$, and execute the above assignment operation at the ((p+3)+k+i-2(j+1)+1) = (p+k+i-2j+2)-th step, i.e. immediately after the execution of $a_{ik}:=a_{ik}-(a_{ij}/a_{jj})\cdot a_{jk}$.

4. Example

In lieu of a mathematical proof for the correctness of the algorithm, we demonstrate an execution of the algorithm for a particular 4×4 matrix, step by step, showing which transformation rule is executed by which processor at which step. The example matrices are: (A,B), where

$$A = \begin{pmatrix} 1 & 2 & 3 & 1 \\ 1 & 5 & 15 & -5 \\ -1 & -1 & 2 & -4 \\ 3 & 4 & 2 & 10 \end{pmatrix} \qquad B = \begin{pmatrix} 12 & 0 & 0 & 0 \\ 0 & 12 & 0 & 0 \\ 0 & 0 & 12 & 0 \\ 0 & 0 & 0 & 12 \end{pmatrix}$$

The expected result is: $(I, A^{-1}B)$, where

$$I = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \qquad A^{-1}B = \begin{pmatrix} 100 & -28 & 60 & 0 \\ -38 & 14 & -42 & -6 \\ 1 & -1 & 9 & 3 \\ -15 & 3 & -3 & 3 \end{pmatrix}$$

The matrix B is so chosen in order to avoid fractions during the computation.

For this example, the toroid system has $4 \times (4+4) = 32$ processors, and the computation requires $5 \times 4 - 1 = 19$ steps.

The snapshots of the computation are given as Figure 8.

Step(0): This is the initial configuration. All blank spaces contain 0. The first row and column are reserved to repeat the last row and column for easy perception of applicability of transformation rules. By marking the processor P(1,1) with '(2)', it is shown that Rule(2) is applicable to P(1,1). For the rest of processors, the rules (6) and (7) are applicable. However, since their applications do not change processor states, the other processors are unmarked.

Step(1): The result of applying Rule (2) to P(1,1) is displayed. No change occurs in the rest of the system. Only relevant components for the transformations that change the processor states are displayed. Note that the applicable rule for each processor is unique.

Step(2): Once Rule (1) is applied, no more change will occur to the processor. Since P(1,1) will be in the same state for the rest of the computation, the state of P(1,1) will not be displayed after some point. Similarly for the processors which have executed Rule (1). An application of Rule (6) will change the state of P(2,2), while the other processors executing (6) or (7) will not change the state. That is the reason the state of P(2,2) is explicitly displayed.

Step(3)-(16): Similarly.

Step(17): At this point, the final values for $A^{-1}B$ are obtained in the x registers. The remaining two steps are for the bookkeeping purpose only.

5. Conclusions

We have given a linear time algorithm for Gauss-Jordan elimination without pivoting by a toroid of processors, an algorithm which is relatively easy to explain. The algorithm can be improved by a slight modification, with the same pipelining principle, to show that if m is known to be less than or equal to n, then $n \times n$ processors, rather than $n \times (n+m)$, are sufficient to perform Gauss-Jordan elimination without pivoting in a linear time.

It is still an open problem whether there exists a linear time algorithm for matrix inversion by a mesh-connected multiprocessor system.

References

(1) Gentleman, W. M.,

"Some Complexity Results for Matrix Computations on Parallel Processors,"

JACM 25,1(1978), pp 112-115.

(2) Heller,D.,

"A Survey of Parallel Algorithms in Numerical Linear Algebra,"

SIAM Review 20,4(1978) pp 740-777.

(3) Kant, R. M. and Kimura, T.,

"Decentralized Parallel Algorithms for Matrix Computation,"

Proceedings of the Fifth Annual Symposium on Computer Architecture, April 1978, pp 96-100.

(4) Kung, H. T. and Leiserson, C. E.,

"Systolic Arrays (For VLSI),"

Computer Science Research Review 1977-1978, Carnegie-Mellon University, Pittsburg, pp 37-57.

(5) Mead, C. A. and Conway, L. A.,

Introduction to VLSI Systems, Addison-Wesley, 1979.

(6) Sameh, A. H.,

"Numerical Parallel Algorithms -- A Survey,"

High Speed Computer and Algorithm Organization, Academic Press, 1977, pp 207-228.

(7) Sutherland, I. E. and Mead, C. A.,

"Microelectronics and Computer Science,"

Scientific American 237(1977), pp 210-228.

(8) Van Scoy, F. L.,

"Some Parallel Cellular Matrix Algorithms,"

Proceedings of 1977 ACM Computer Science Conference, February 1977.

Table I: The Next-State Function σ for GJ

Rule-#	c	h	h_w	v_n	h_w+v_n	$h_n v_w$	x'	y'	z'	c'	h'	v'
(1)	2										0	0
(2)	1						1		x	2	1	1
(3)	≤ 0	1						y_n	z_w		0	0
(4)	≤ 0	0	1				x/z_w	x/z_w	z_w		1	
(5)	≤ 0	0	(0)	1			0		x	2		1
(6)	0	0	≤ 0	≤ 0	a	b	$x-y_n z_w$	y_n	z_w	$b+ba-2a$	h_w	v_n
(7)	-1	0	0	0	(0)	b	$x-y_n z_w$	y_n	z_w	$3b-1$	$-b$	$-b$

Note: $-1 \leq a \leq 0 \leq b \leq 1$ and

a	b	$b+ba-2a$
0	0	0
0	1	1
-1	0	2
-1	1	2

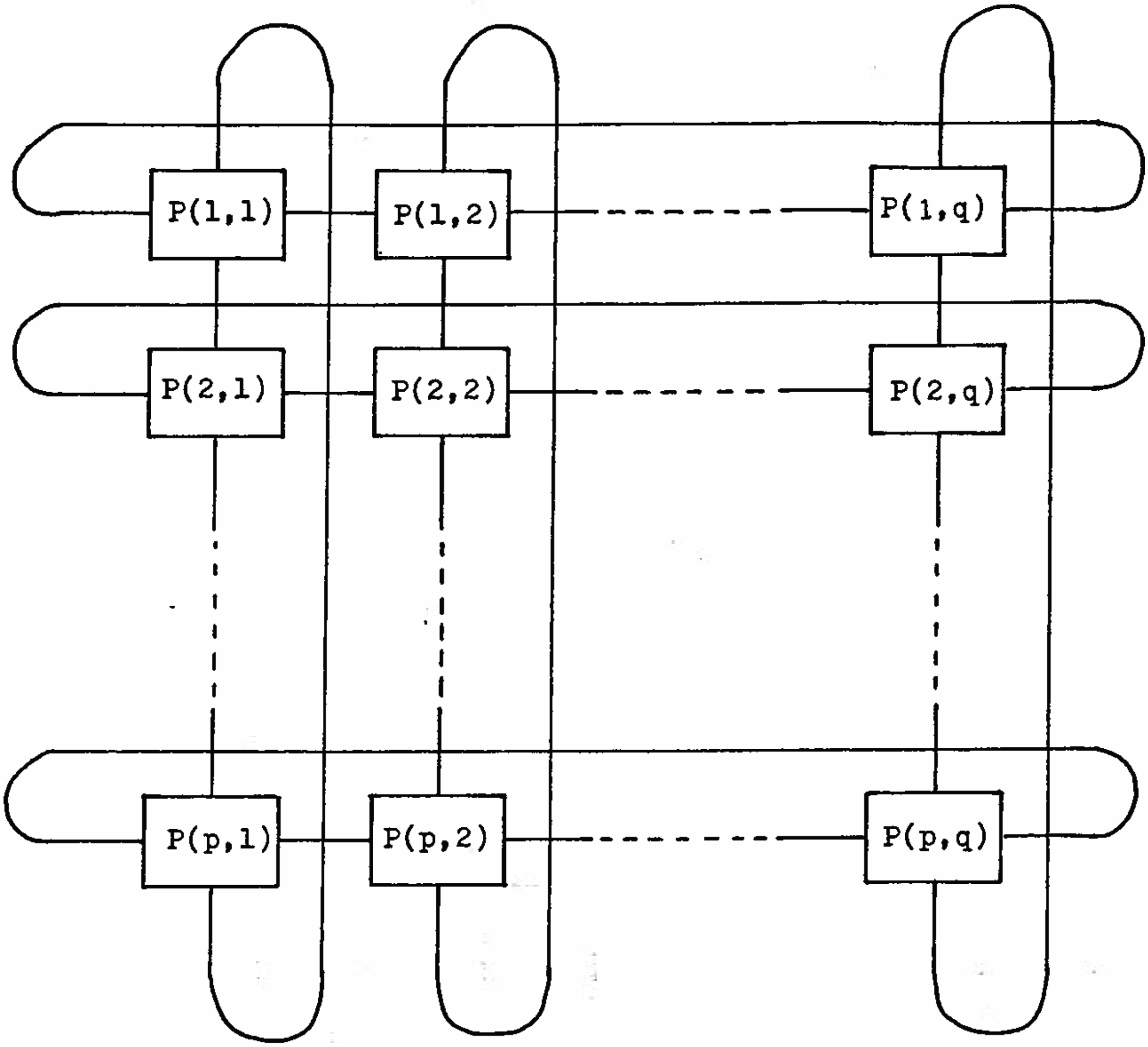
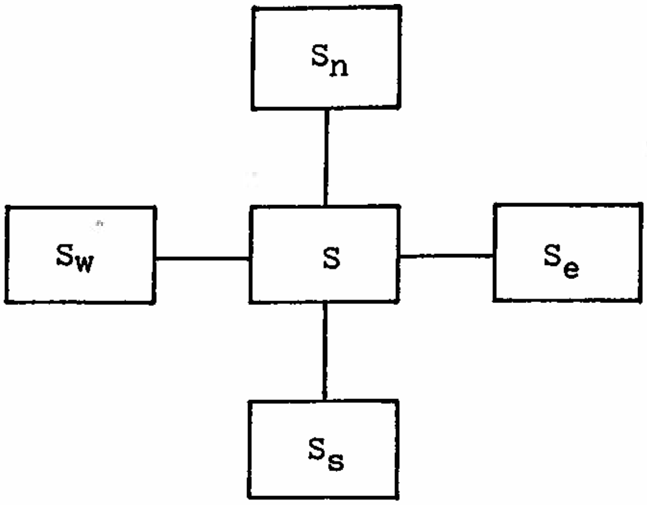


Figure 1: A Toroid System



$$S' = f(S,S_n,S_w,S_s,S_n)$$

where

f : the next-state function

S': the next state of $P(i,j)$

S : the current state of $P(i,j)$

S_n: the current state of $P(i-1,j)$

S_w: the current state of $P(i,j-1)$

S_s: the current state of $P(i+1,j)$

S_e: the current state of $P(i,j+1)$

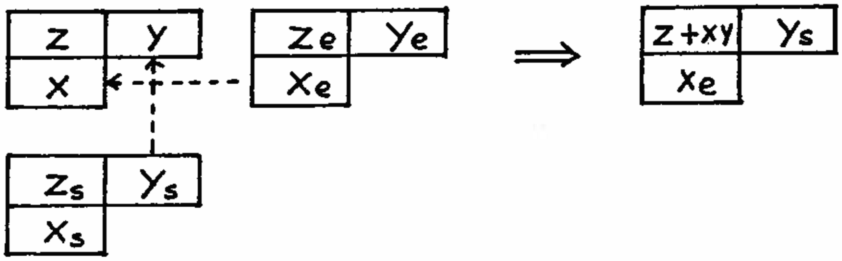
Note that the index computation is always modulo p for i and modulo q for j.

Figure 2: The Next-State Function

Dimension: n × n

Registers: x, y, z : Real

Next-State Function: $(x', y', z') := (x_e, y_s, z+xy)$



$$\text{Initial Configuration:}\quad \left.\begin{aligned} x_{ij} &= a_{i,i+j-1} \\ y_{ij} &= b_{i+j-1,j} \\ z_{ij} &= 0 \end{aligned}\right\} \qquad 1 \leq i, j \leq n$$

$$\text{Final Configuration:}\quad \left.\begin{aligned} x_{ij} &= a_{i,i+j-1} \\ y_{ij} &= b_{i+j-1,j} \\ z_{ij} &= c_{ij} \end{aligned}\right\} \qquad 1 \leq i, j \leq n$$

$$\text{where } c_{ij} \triangleq \sum_{k=1}^{n} a_{ik} b_{kj}$$

Figure 3: A Toroid System for Matrix Multiplication

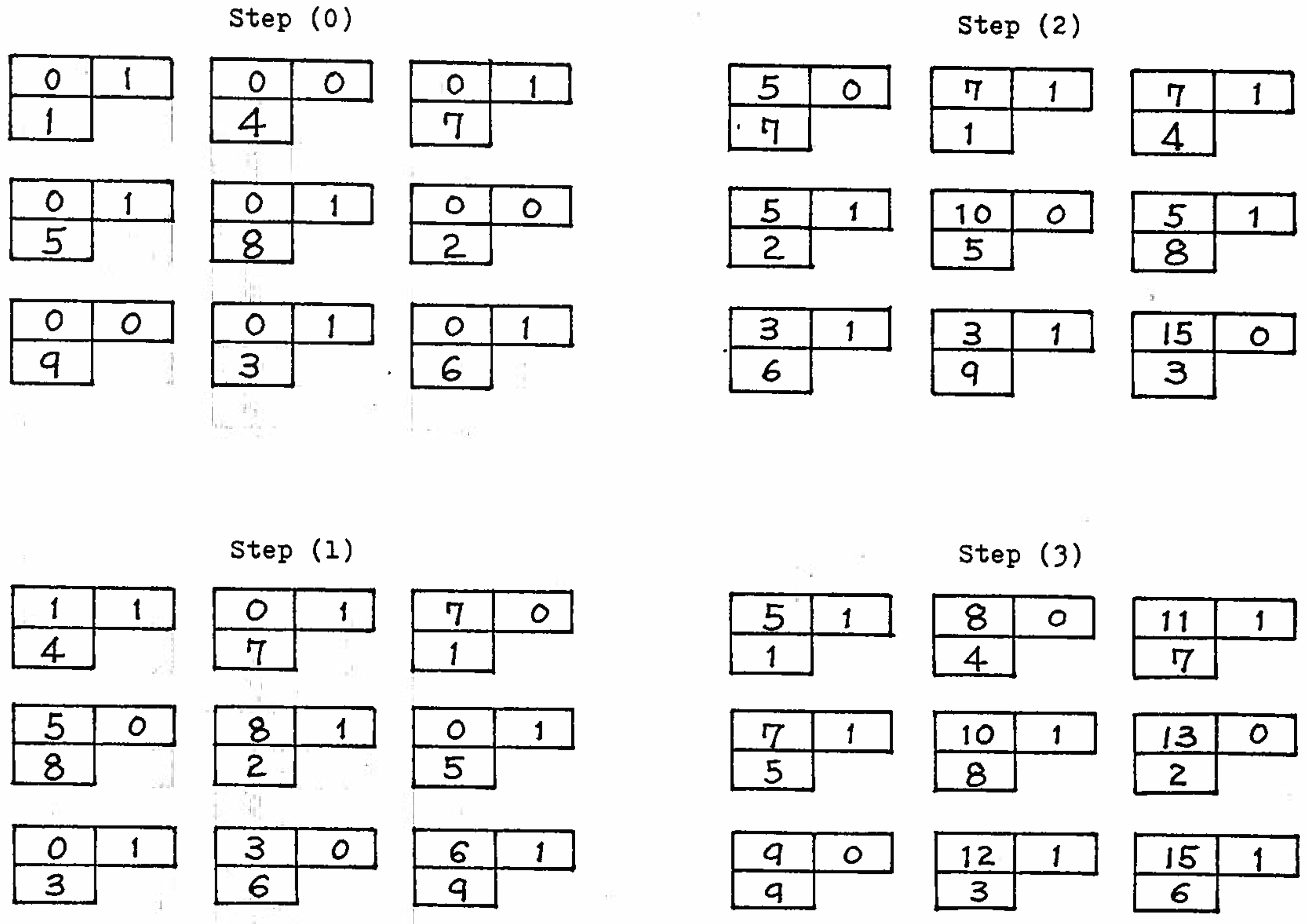


Figure 4: Multiplication of 3 × 3 Matrices

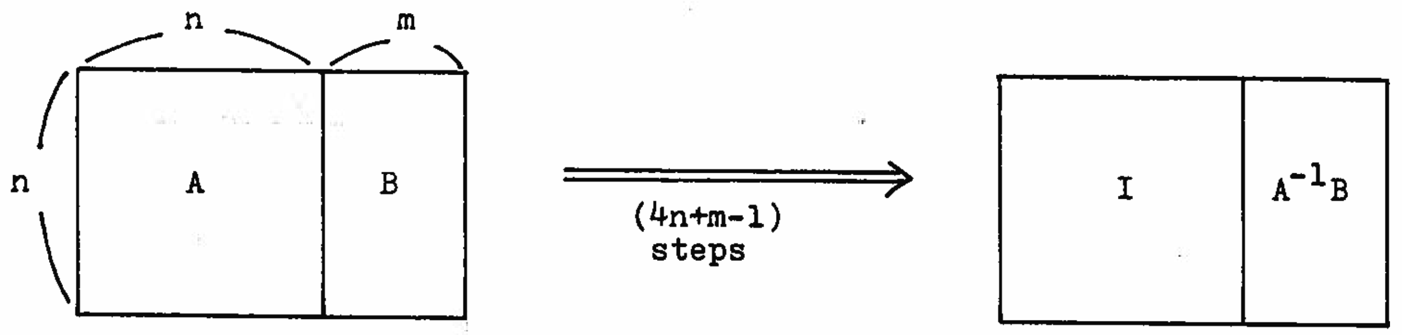


Figure 5: The Process of Gauss-Jordan Elimination by GJ

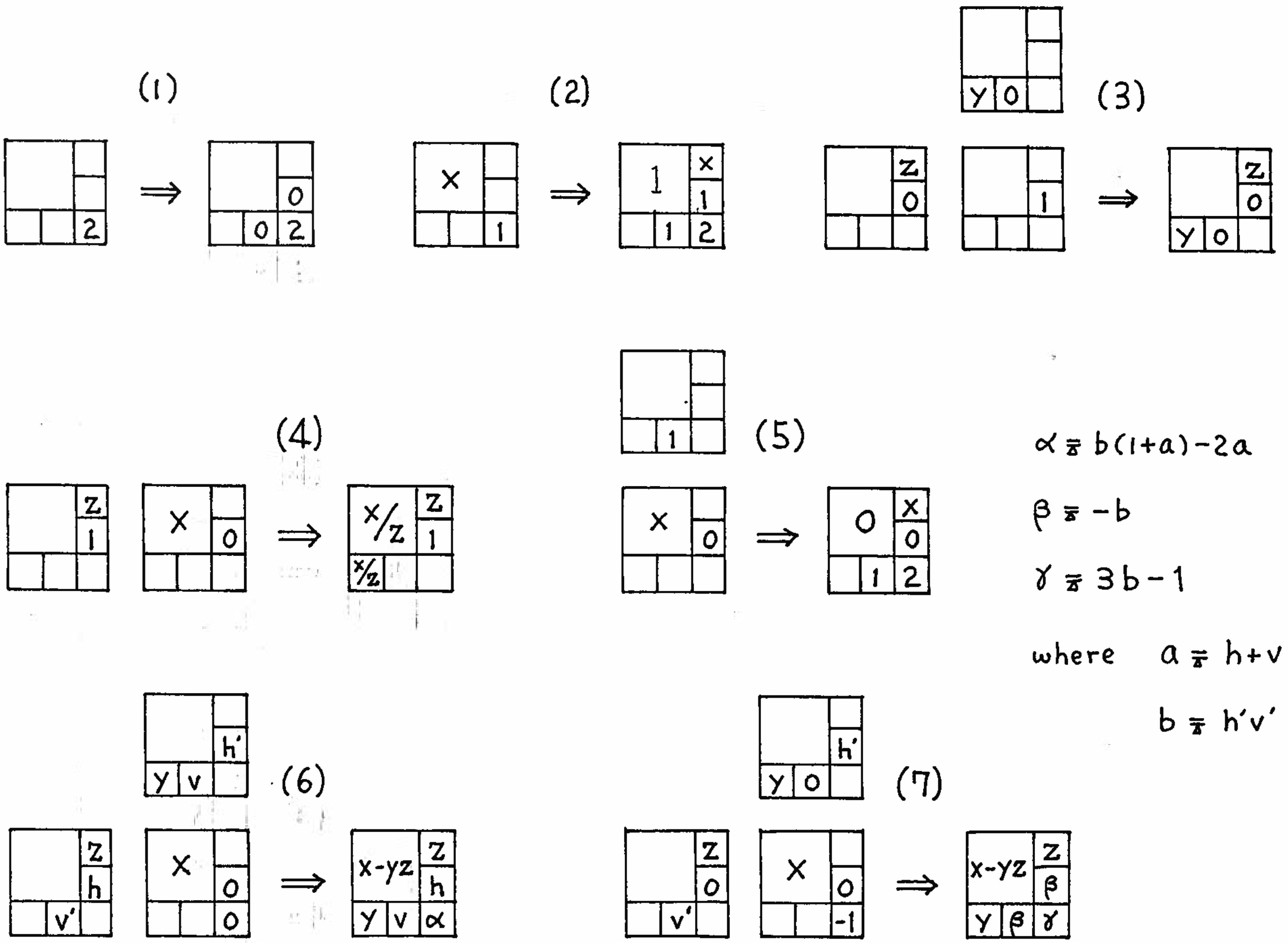


Figure 6: The Schematic Representation of σ for GJ

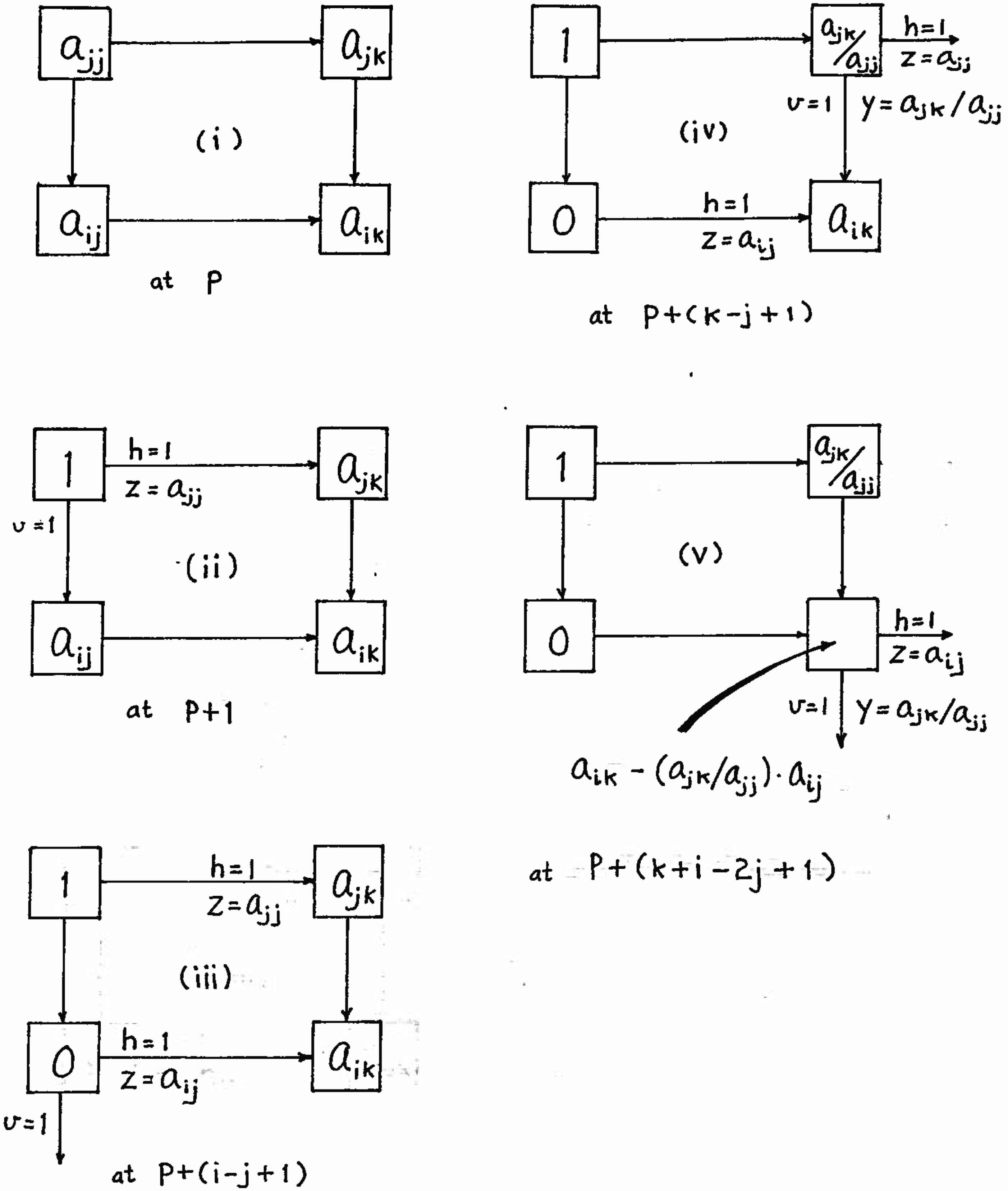
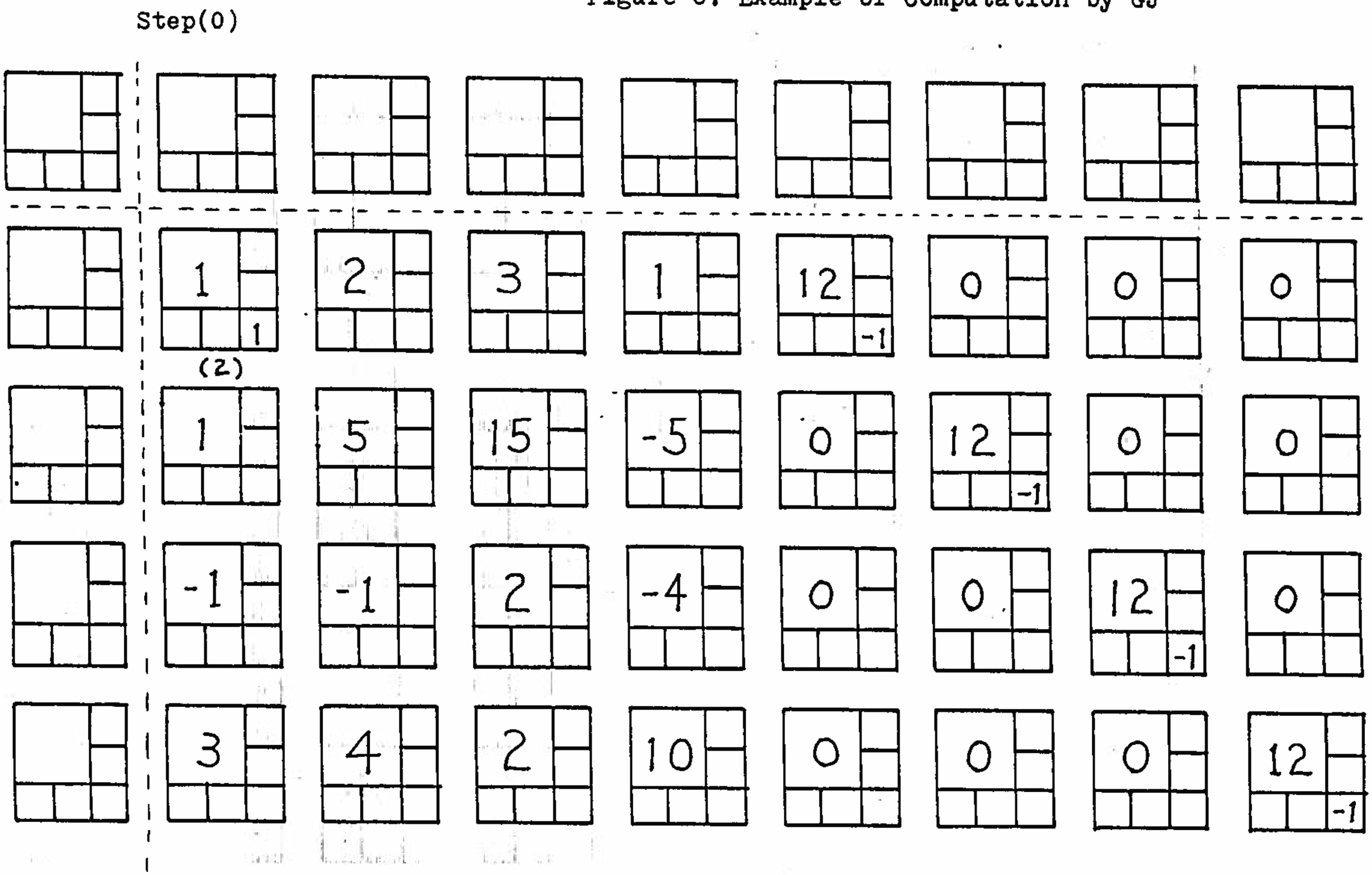
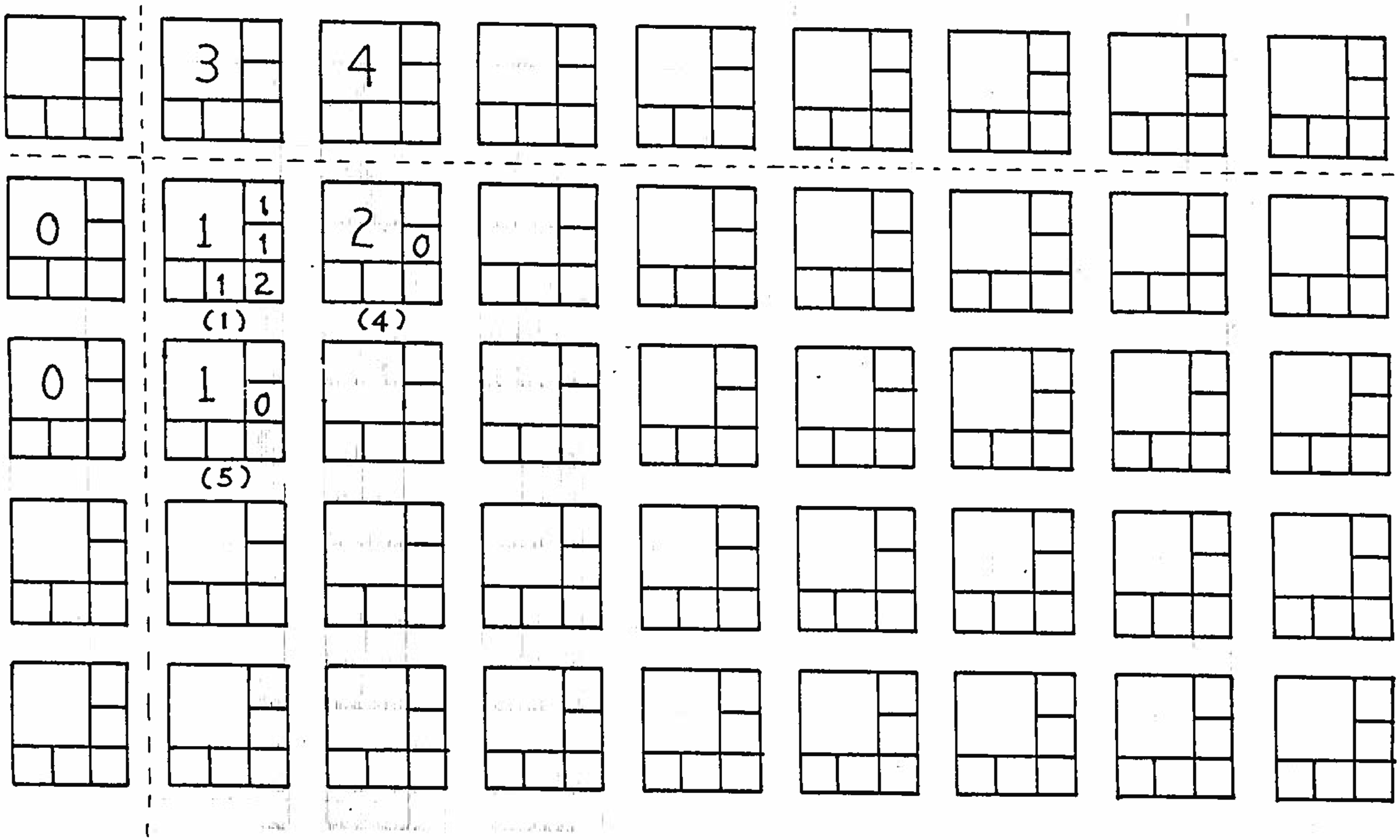


Figure 7: Data Flow for $a_{ik}=a_{ik}-(a_{ij}/a_{jj})\cdot a_{jk}$

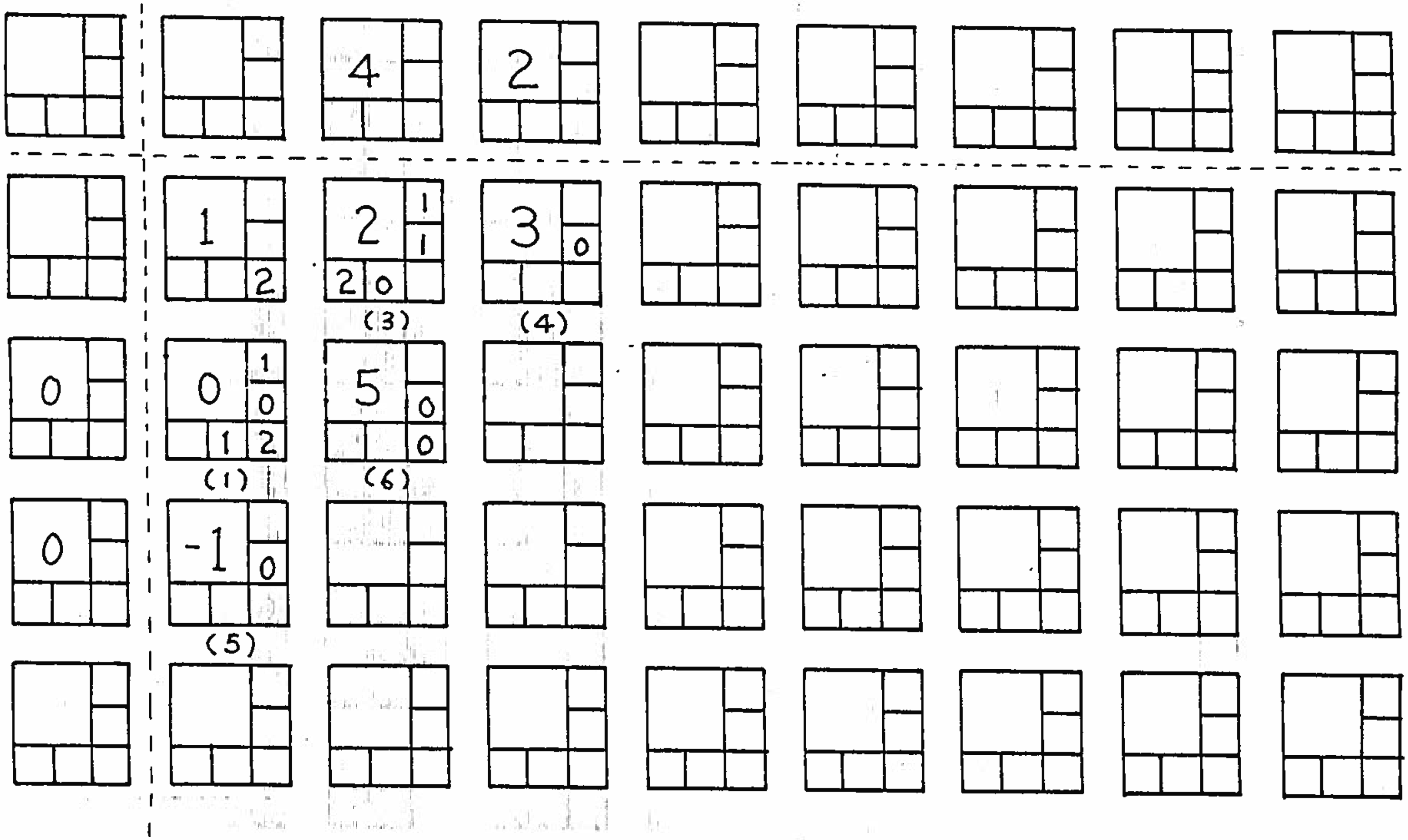
Figure 8: Example of Computation by GJ



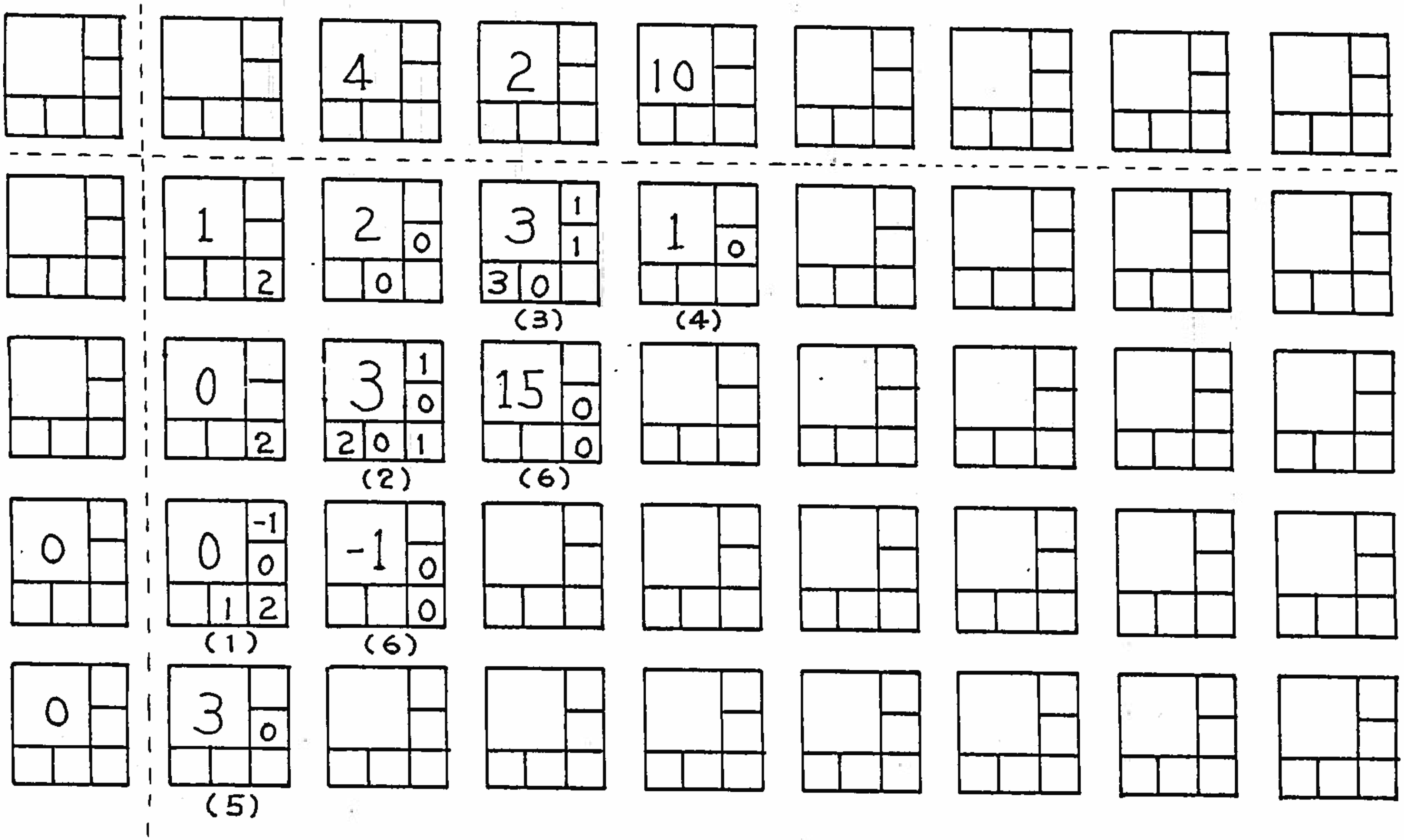
Step(1)



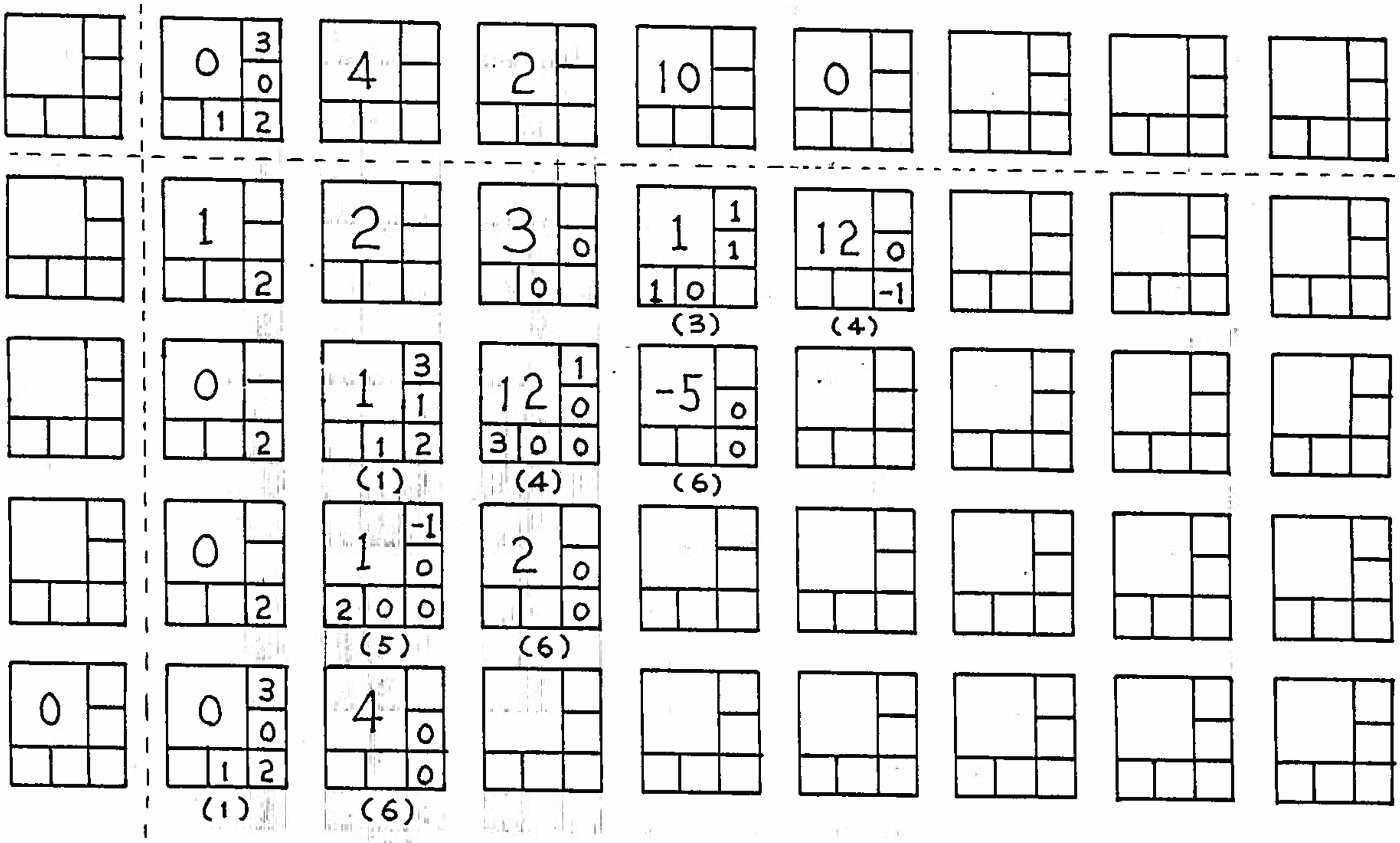
Step(2)



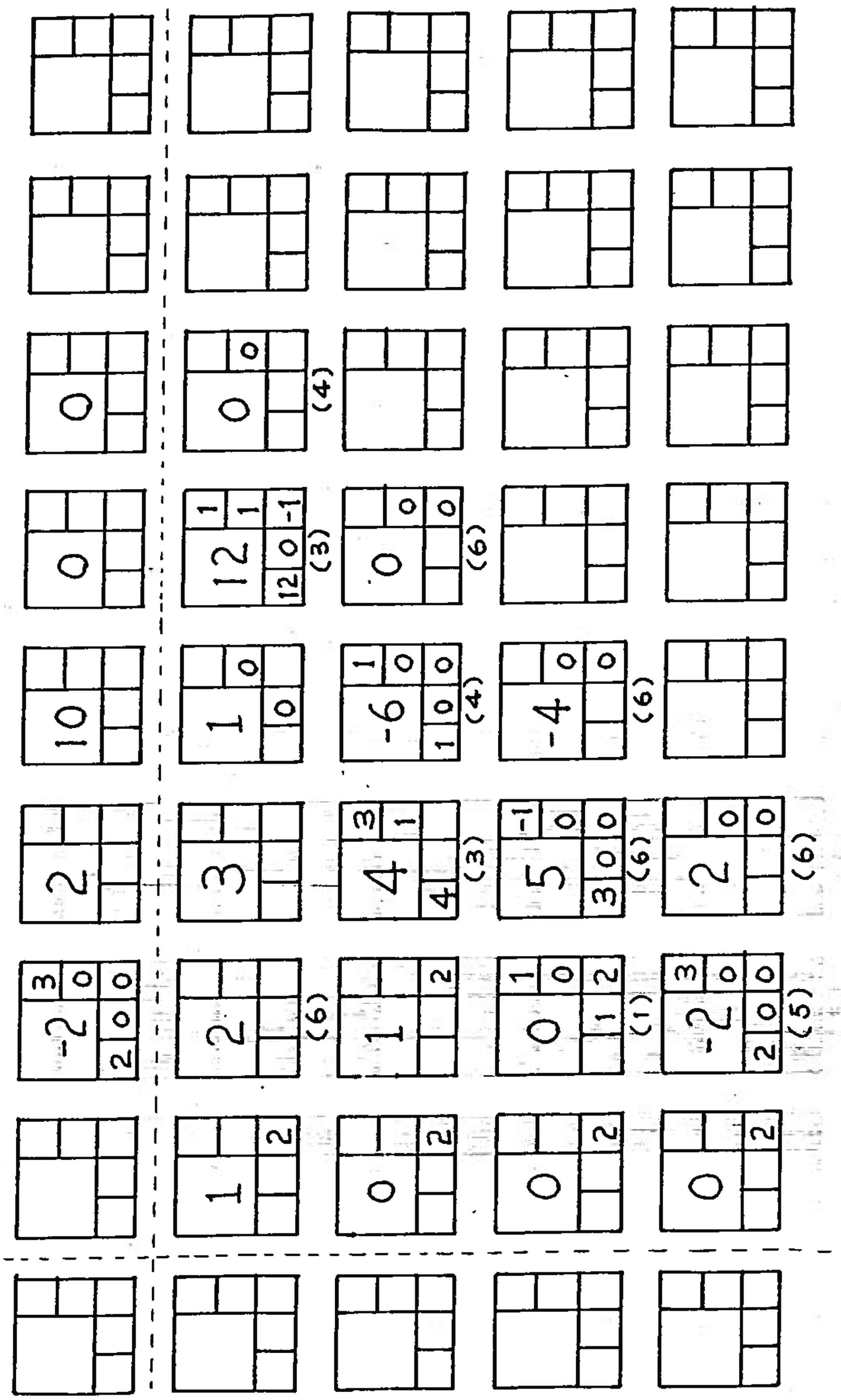
Step(3)



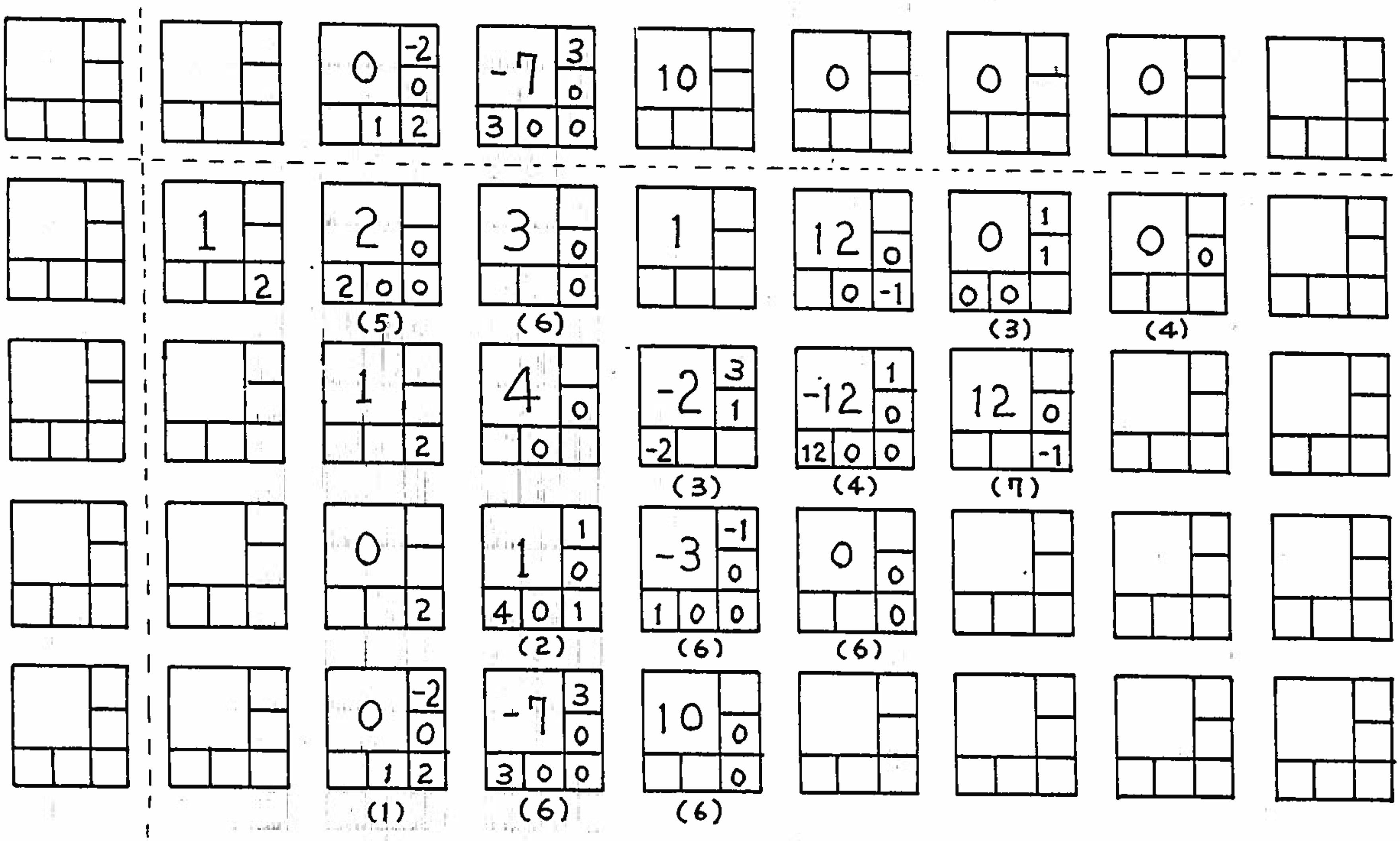
Step(4)


0
3
0
1
2
4
2
10
0
1
2
2
3
0
0
1
1
1
1
0
(3)
12
0
-1
(4)
0
2
1
3
1
1
2
(1)
12
1
0
3
0
0
(4)
-5
0
0
(6)
0
2
1
-1
0
2
0
0
(5)
2
0
0
(6)
0
0
3
0
1
2
(1)
4
0
0
(6)

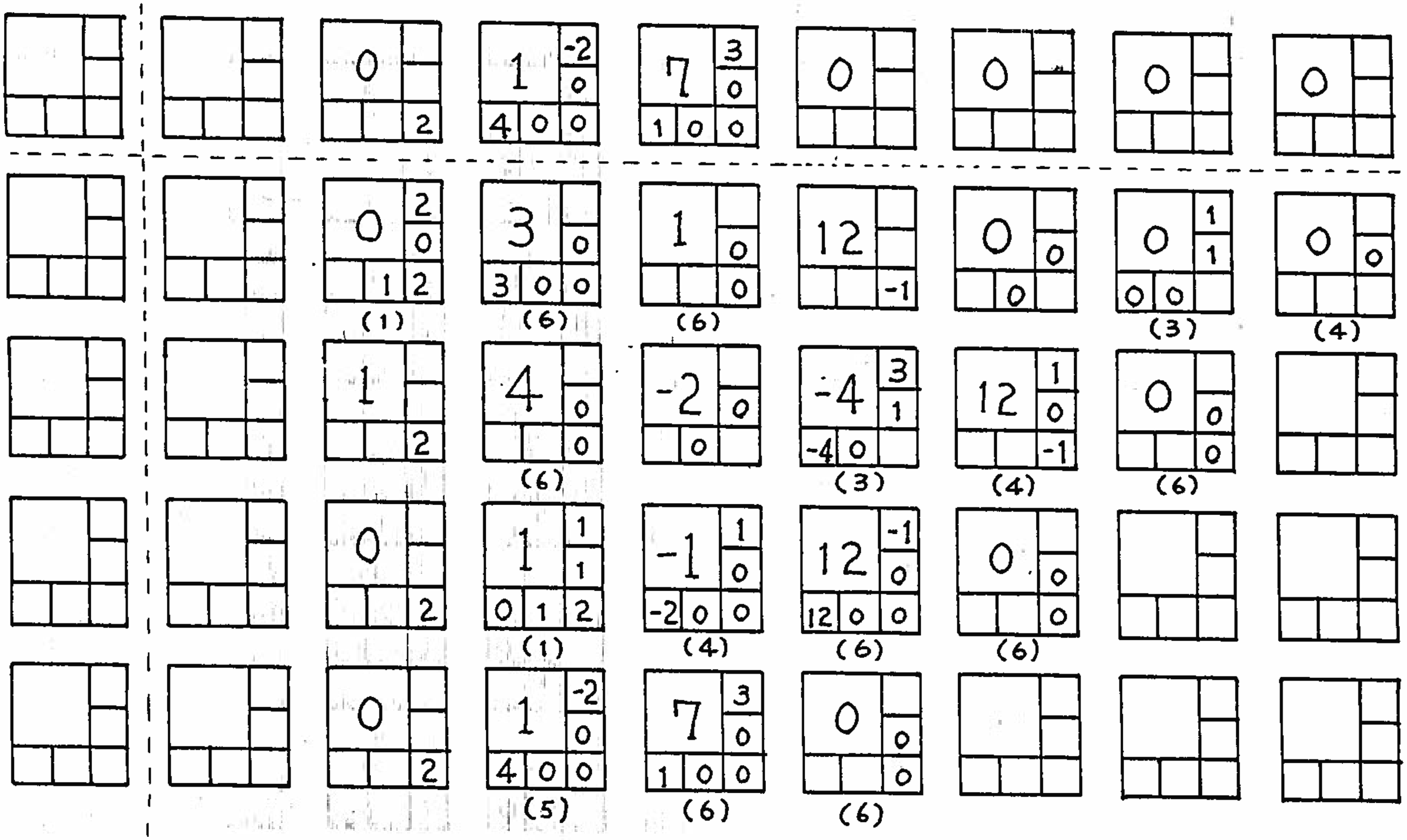
Step(5)



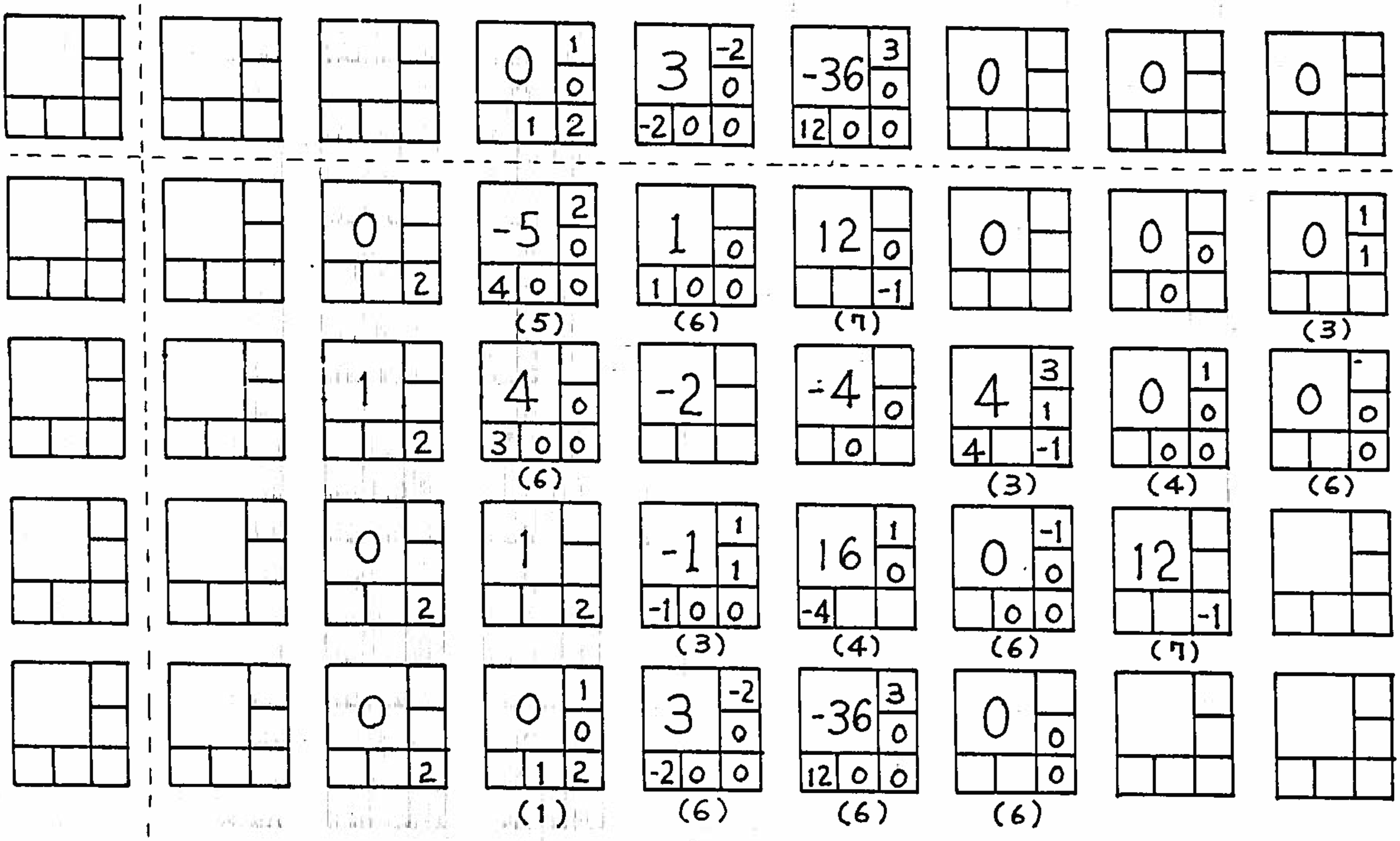
Step(6)


0 -2 0 1 2
-7 3 0 3 0 0
10
0
0
0
1 2
2 0 2 0 0
(5)
3 0 0
(6)
1
12 0 0 -1
0 1 1 0 0
(3)
0 0
(4)
1 2
4 0 0
-2 3 1 -2
(3)
-12 1 0 12 0 0
(4)
12 0 -1
(7)
0 2
1 1 0 4 0 1
(2)
-3 -1 0 1 0 0
(6)
0 0 0
(6)
0 -2 0 1 2
(1)
-7 3 0 3 0 0
(6)
10 0 0
(6)

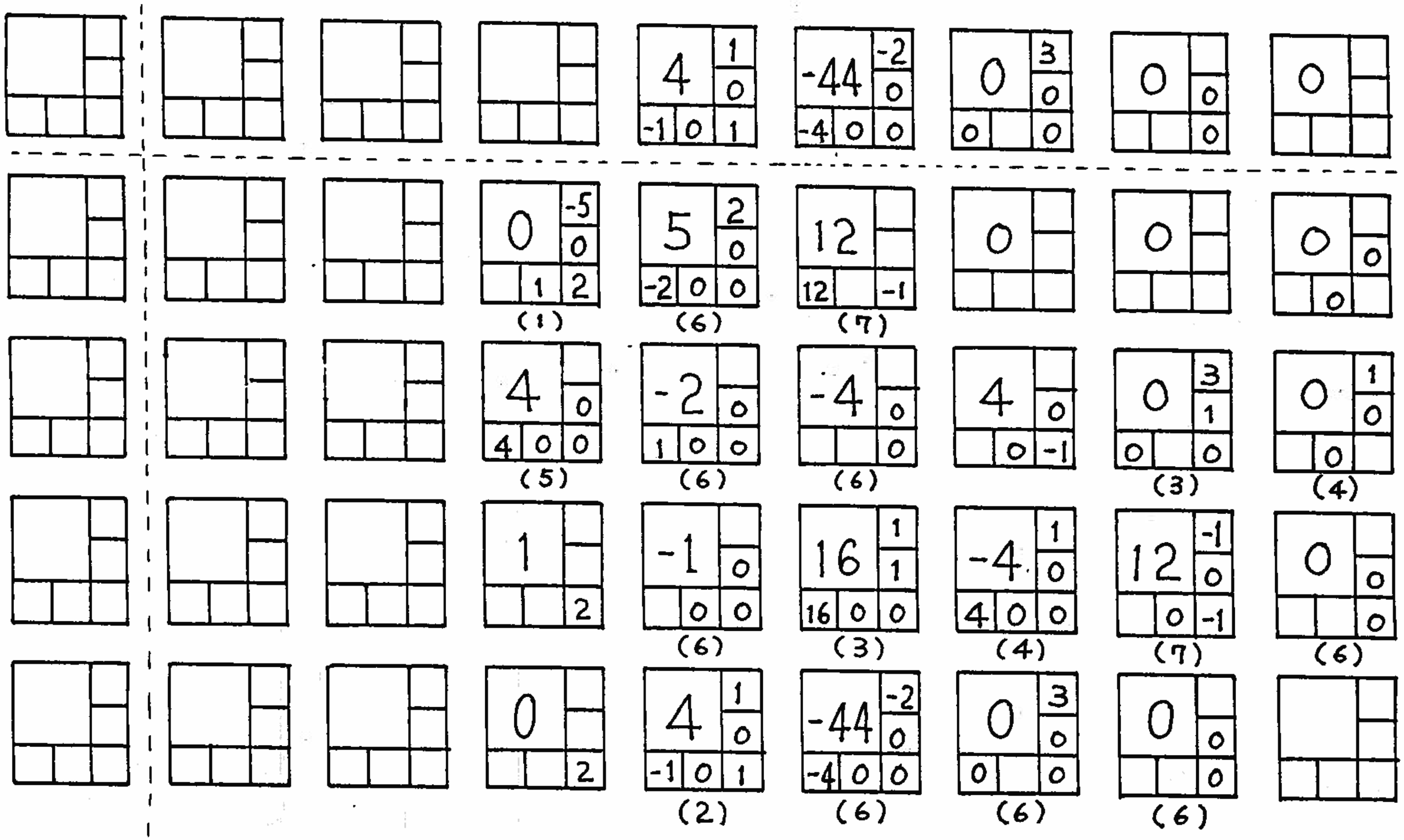
Step(7)



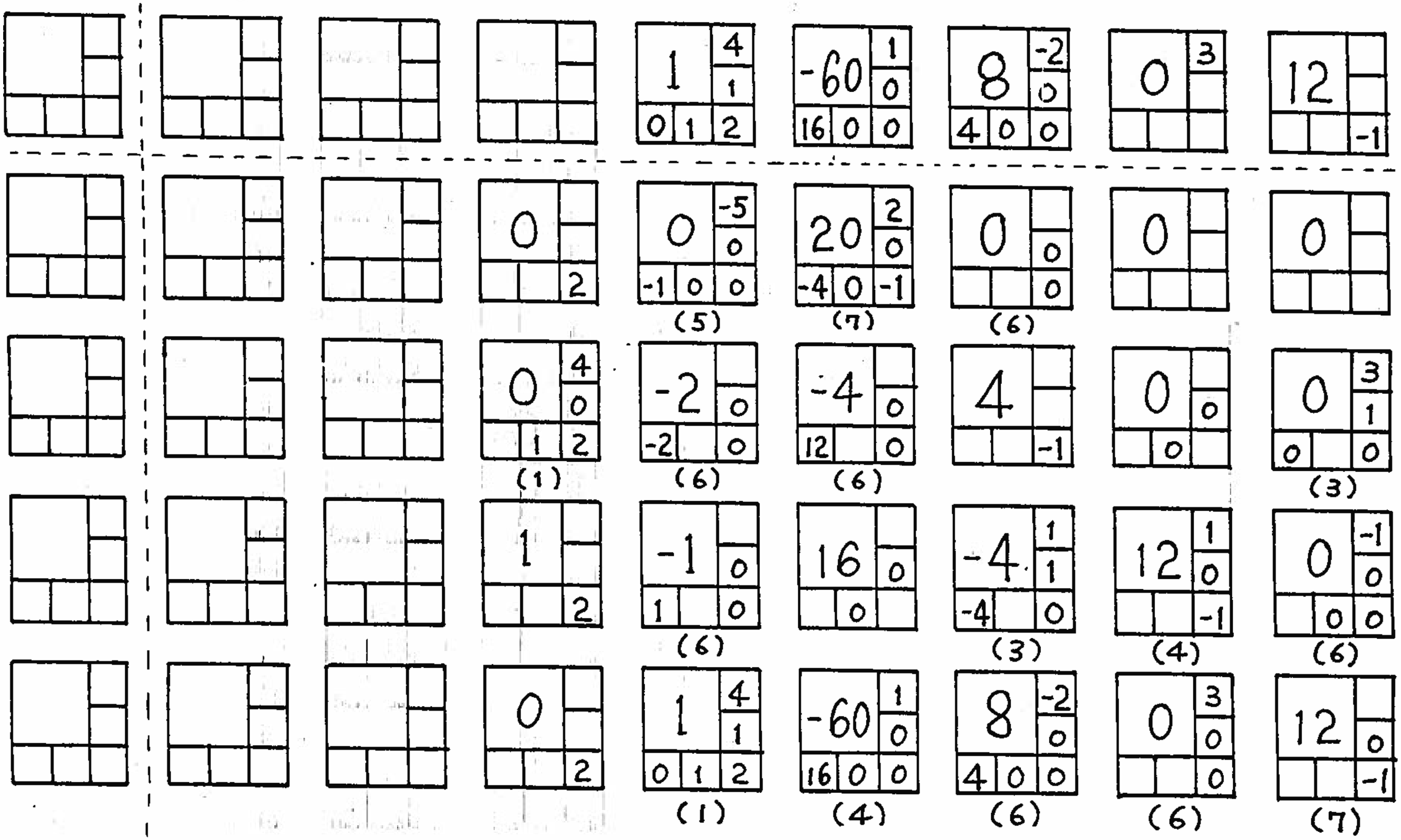
Step(8)


0
1
0
1
2
3
-2
0
-2
0
0
-36
3
0
12
0
0
0
0
0
0
2
-5
2
0
4
0
0
(5)
1
0
1
0
0
(6)
12
0
-1
(7)
0
0
0
0
0
1
1
(3)
1
2
4
0
3
0
0
(6)
-2
-4
0
0
4
3
1
4
-1
(3)
0
1
0
0
0
(4)
0
0
0
(6)
0
2
1
2
-1
1
1
-1
0
0
(3)
16
1
0
-4
(4)
0
-1
0
0
0
(6)
12
-1
(7)
0
2
0
1
0
1
2
(1)
3
-2
0
-2
0
0
(6)
-36
3
0
12
0
0
(6)
0
0
0
(6)

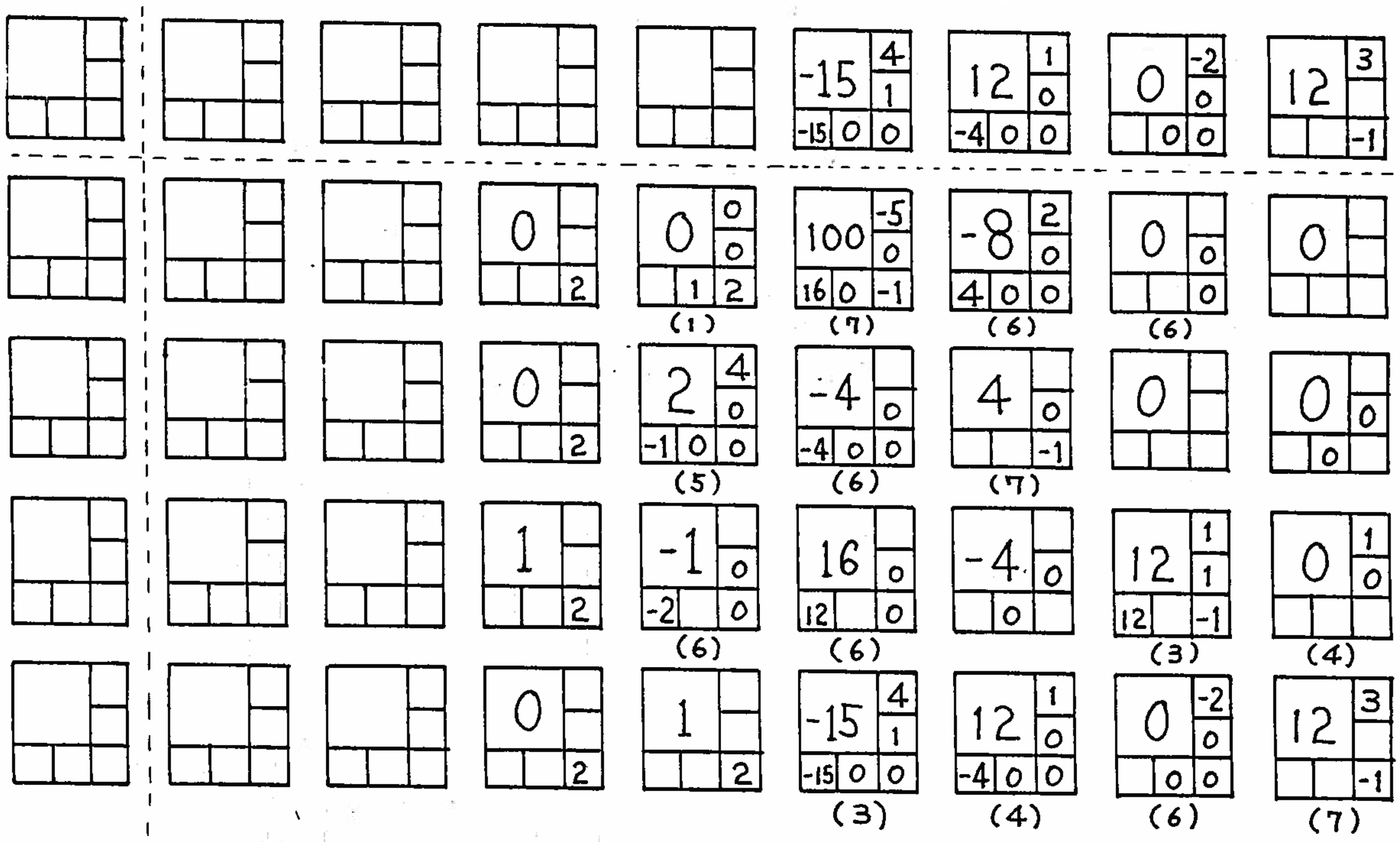
Step(9)


4 1 0 -1 0 1
-44 -2 0 -4 0 0
0 3 0 0 0
0 0 0
0
0 -5 0 1 2
(1)
5 2 0 -2 0 0
(6)
12 12 -1
(7)
0
0
0 0 0
4 0 4 0 0
(5)
-2 0 1 0 0
(6)
-4 0 0
(6)
4 0 0 -1
0 3 1 0 0
(3)
0 1 0 0
(4)
1 2
-1 0 0 0
(6)
16 1 1 16 0 0
(3)
-4 1 0 4 0 0
(4)
12 -1 0 0 -1
(7)
0 0 0
(6)
0 2
4 1 0 -1 0 1
(2)
-44 -2 0 -4 0 0
(6)
0 3 0 0 0
(6)
0 0 0
(6)

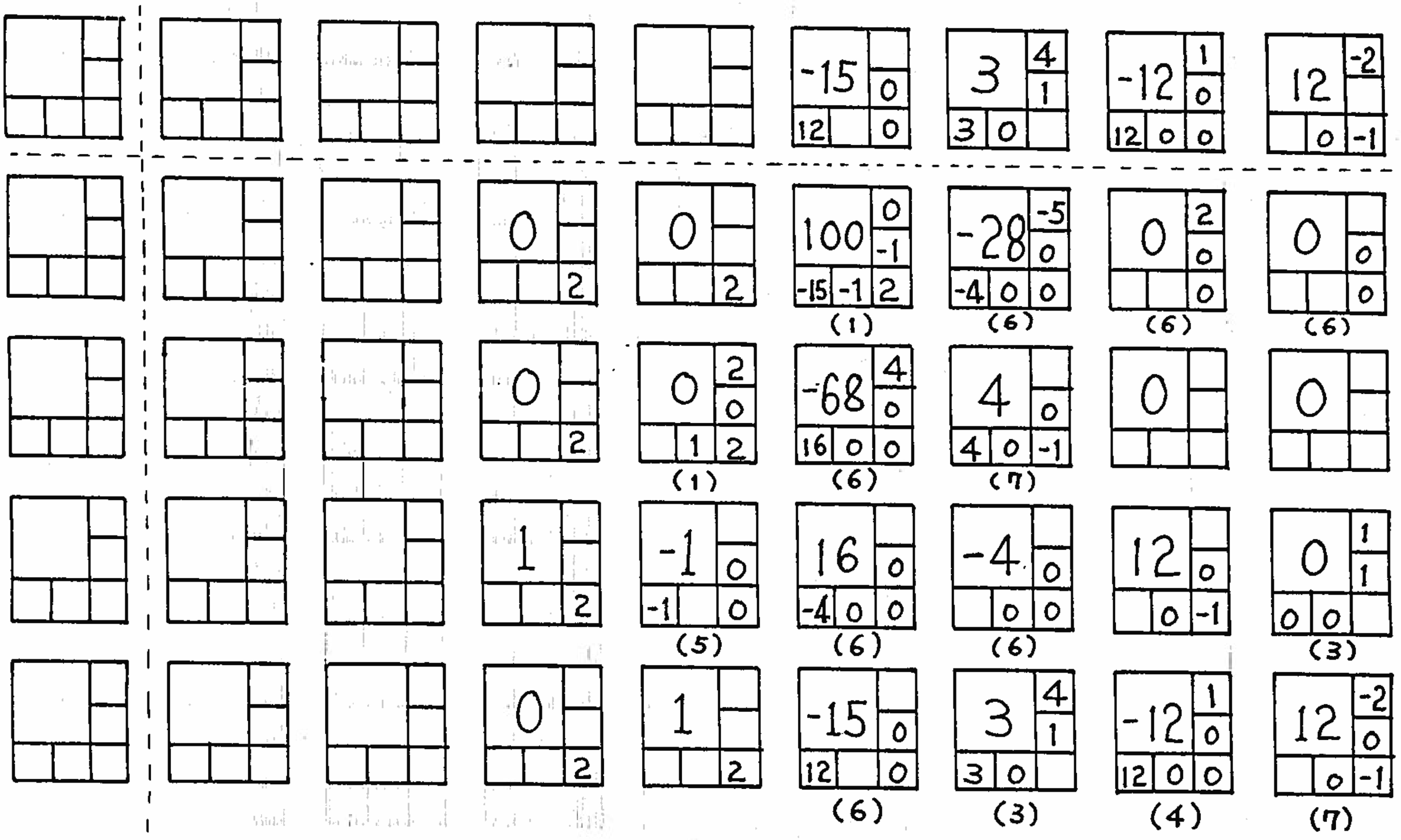
Step(10)


1 4 1 0 1 2
-60 1 0 16 0 0
8 -2 0 4 0 0
0 3
12 -1
0 2
0 -5 0 -1 0 0 (5)
20 2 0 -4 0 -1 (7)
0 0 0 (6)
0
0
0 4 0 1 2 (1)
-2 0 -2 0 (6)
-4 0 12 0 (6)
4 -1
0 0 0
0 3 1 0 0 (3)
1 2
-1 0 1 0 (6)
16 0 0
-4 1 1 -4 0 (3)
12 1 0 -1 (4)
0 -1 0 0 0 (6)
0 2
1 4 1 0 1 2 (1)
-60 1 0 16 0 0 (4)
8 -2 0 4 0 0 (6)
0 3 0 0 (6)
12 0 -1 (7)

Step(11)

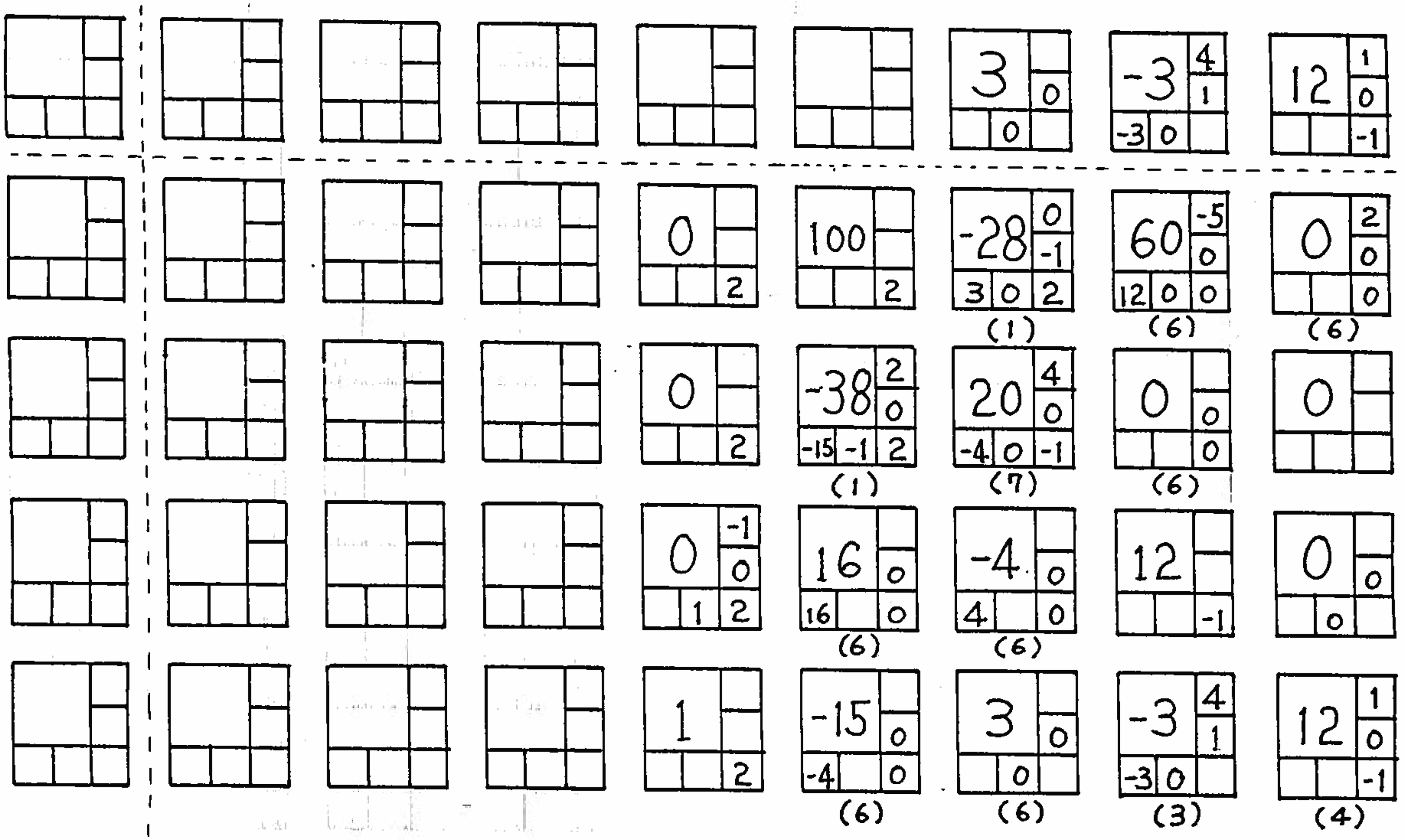

-15 4 1 -15 0 0
12 1 0 -4 0 0
0 -2 0 0 0
12 3 -1
0 2
0 0 0 1 2
(1)
100 -5 0 16 0 -1
(7)
-8 2 0 4 0 0
(6)
0 0 0
(6)
0
0 2
2 4 0 -1 0 0
(5)
-4 0 -4 0 0
(6)
4 0 -1
(7)
0
0 0 0
1 2
-1 0 -2 0
(6)
16 0 12 0
(6)
-4 0 0
12 1 1 12 -1
(3)
0 1 0
(4)
0 2
1 2
-15 4 1 -15 0 0
(3)
12 1 0 -4 0 0
(4)
0 -2 0 0 0
(6)
12 3 -1
(7)

Step(12)

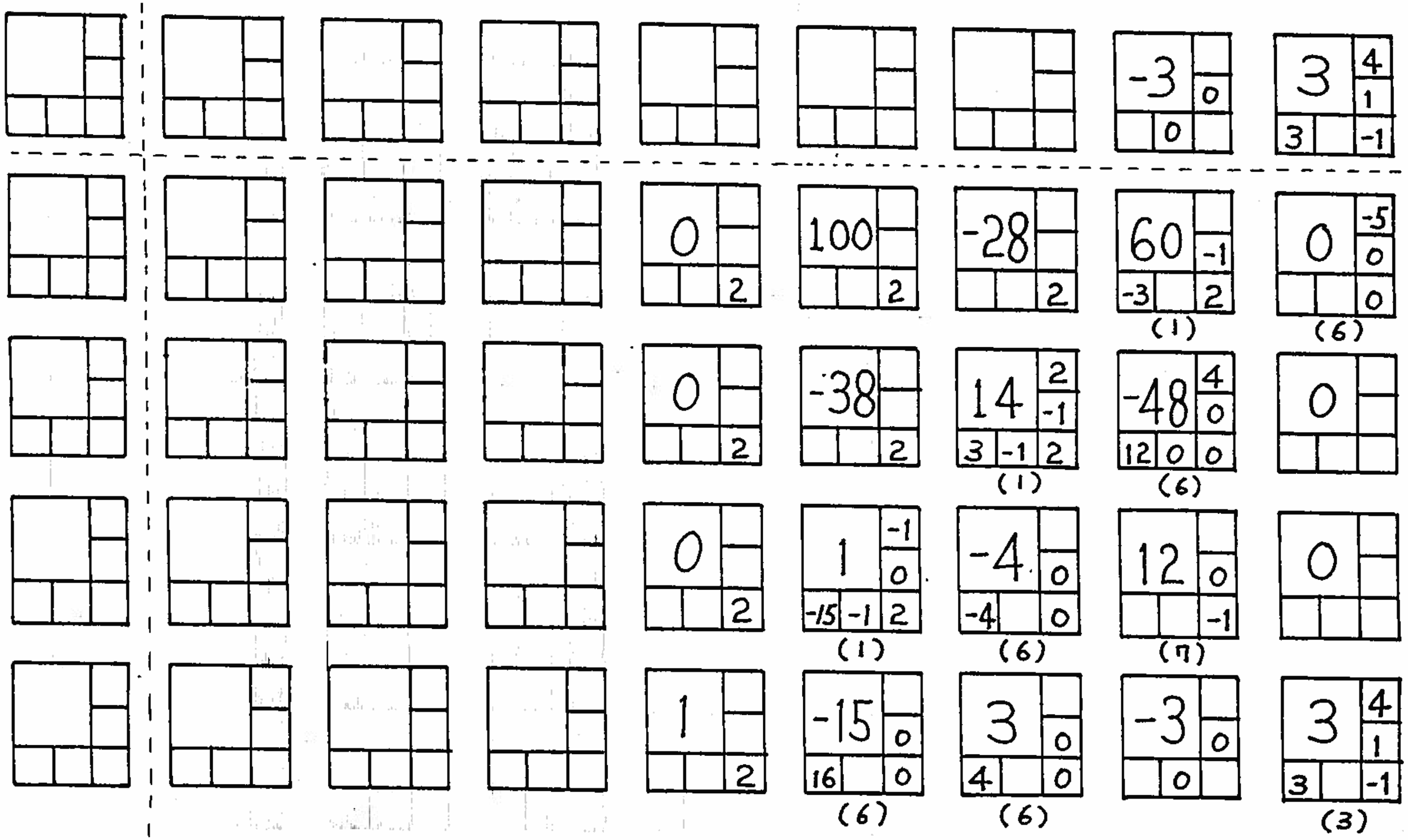


-15 0 12 0
3 4 1 3 0
-12 1 0 12 0 0
12 -2 0 -1
0 2
0 2
100 0 -1 -15 -1 2
(1)
-28 -5 0 -4 0 0
(6)
0 2 0 0
(6)
0 0 0
(6)
0 2
0 2 0 1 2
(1)
-68 4 0 16 0 0
(6)
4 0 4 0 -1
(7)
0
0
1 2
-1 0 -1 0
(5)
16 0 -4 0 0
(6)
-4 0 0 0
(6)
12 0 0 -1
0 1 1 0 0
(3)
0 2
1 2
-15 0 12 0
(6)
3 4 1 3 0
(3)
-12 1 0 12 0 0
(4)
12 -2 0 0 -1
(7)

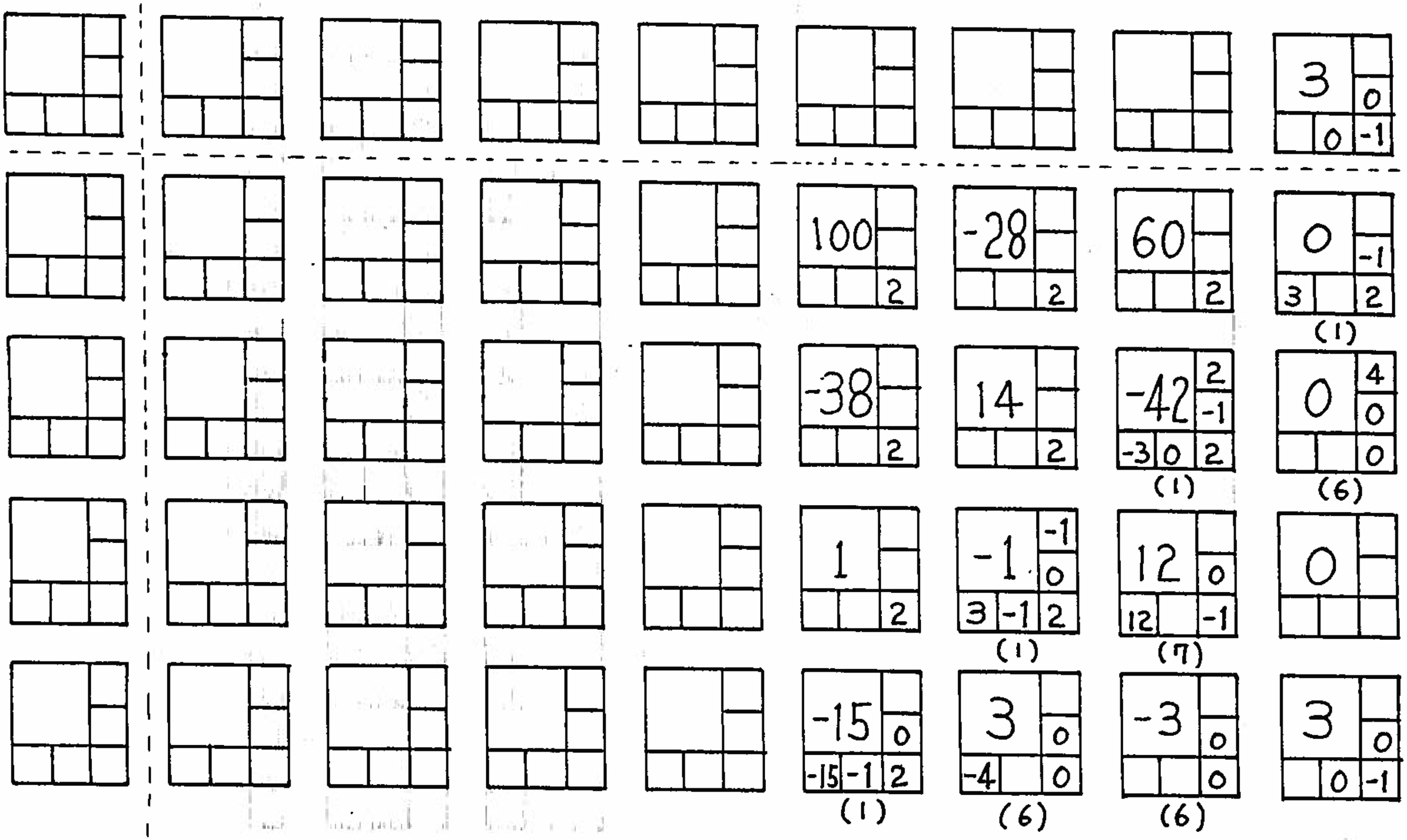
Step(13)

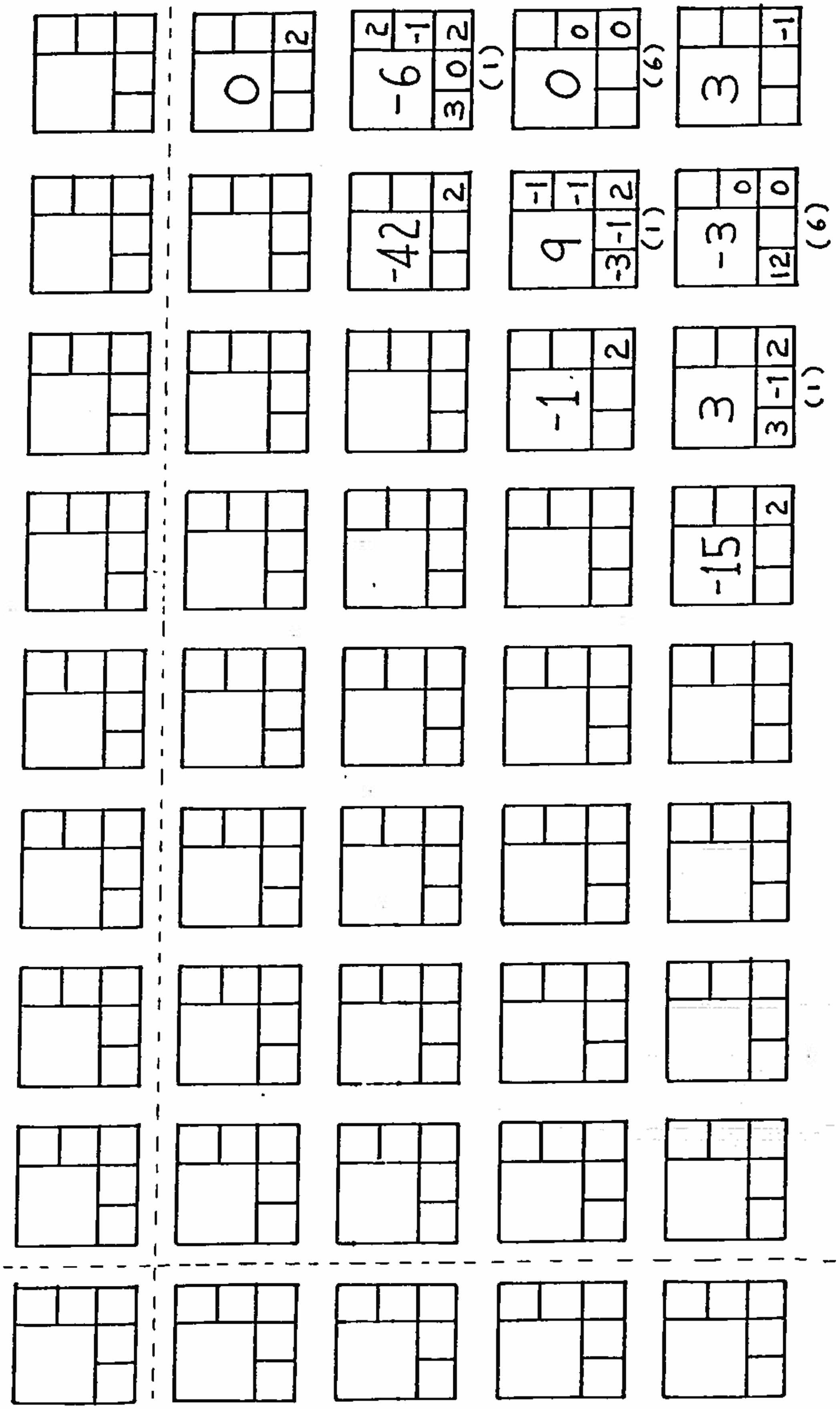

3
0
0
-3
4
1
-3
0
12
1
0
-1
0
2
100
2
-28
0
-1
3
0
2
(1)
60
-5
0
12
0
0
(6)
0
2
0
0
(6)
0
2
-38
2
0
-15
-1
2
(1)
20
4
0
-4
0
-1
(7)
0
0
0
(6)
0
0
-1
0
1
2
16
0
16
0
(6)
-4
0
4
0
(6)
12
-1
0
0
0
1
2
-15
0
-4
0
(6)
3
0
0
(6)
-3
4
1
-3
0
(3)
12
1
0
-1
(4)

Step(14)


-3
0
0
3
4
1
3
-1
0
2
100
2
-28
2
60
-1
-3
2
(1)
0
-5
0
0
(6)
0
2
-38
2
14
2
-1
3
-1
2
(1)
-48
4
0
12
0
0
(6)
0
0
2
1
-1
0
-15
-1
2
(1)
-4
0
-4
0
(6)
12
0
-1
(7)
0
1
2
-15
0
16
0
(6)
3
0
4
0
(6)
-3
0
0
3
4
1
3
-1
(3)

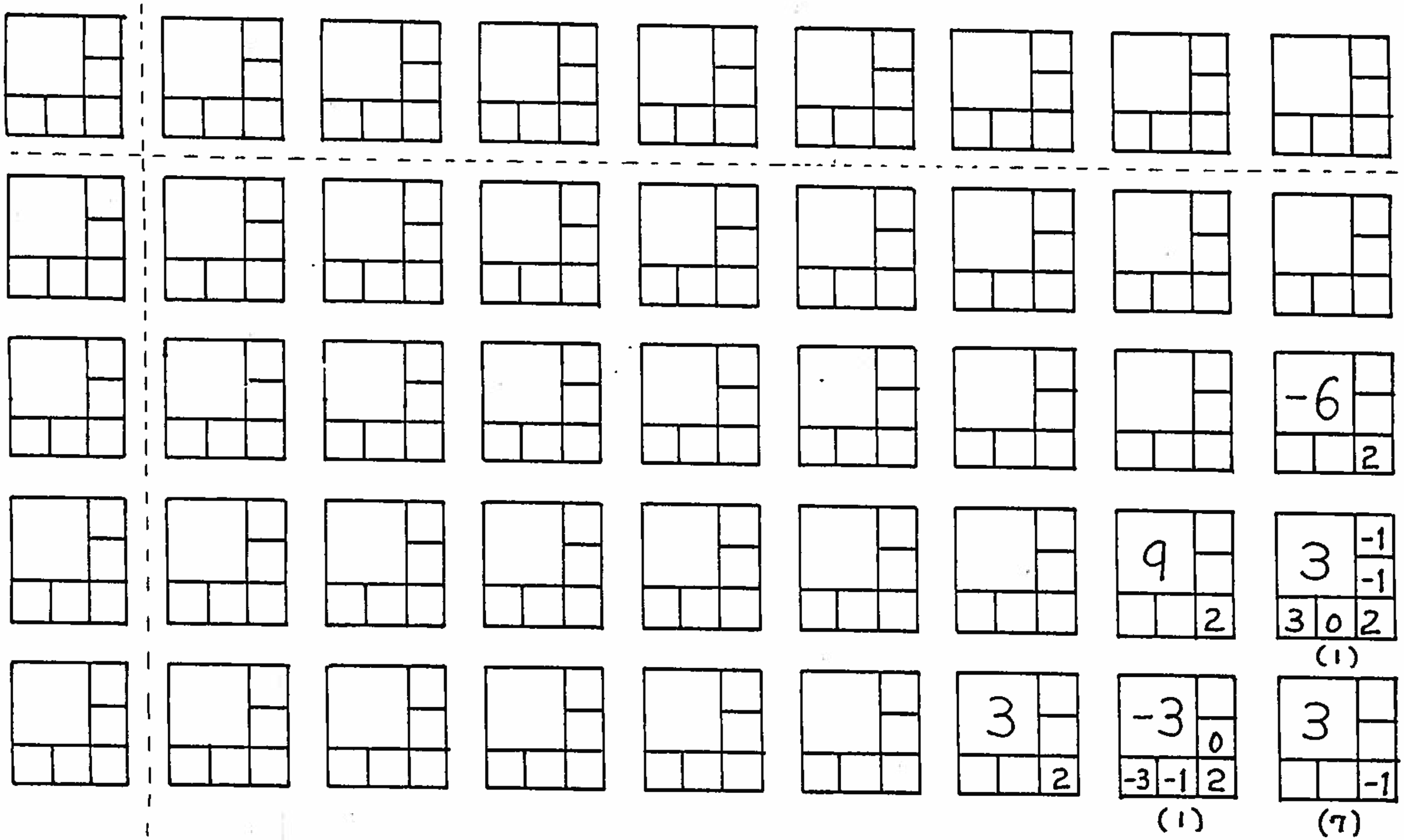
Step(15)


3
0
0
-1
100
2
-28
2
60
2
0
-1
3
2
(1)
-38
2
14
2
-42
2
-1
-3
0
2
(1)
0
4
0
0
(6)
1
2
-1
-1
0
3
-1
2
(1)
12
0
12
-1
(7)
0
-15
0
-15
-1
2
(1)
3
0
-4
0
(6)
-3
0
0
(6)
3
0
0
-1

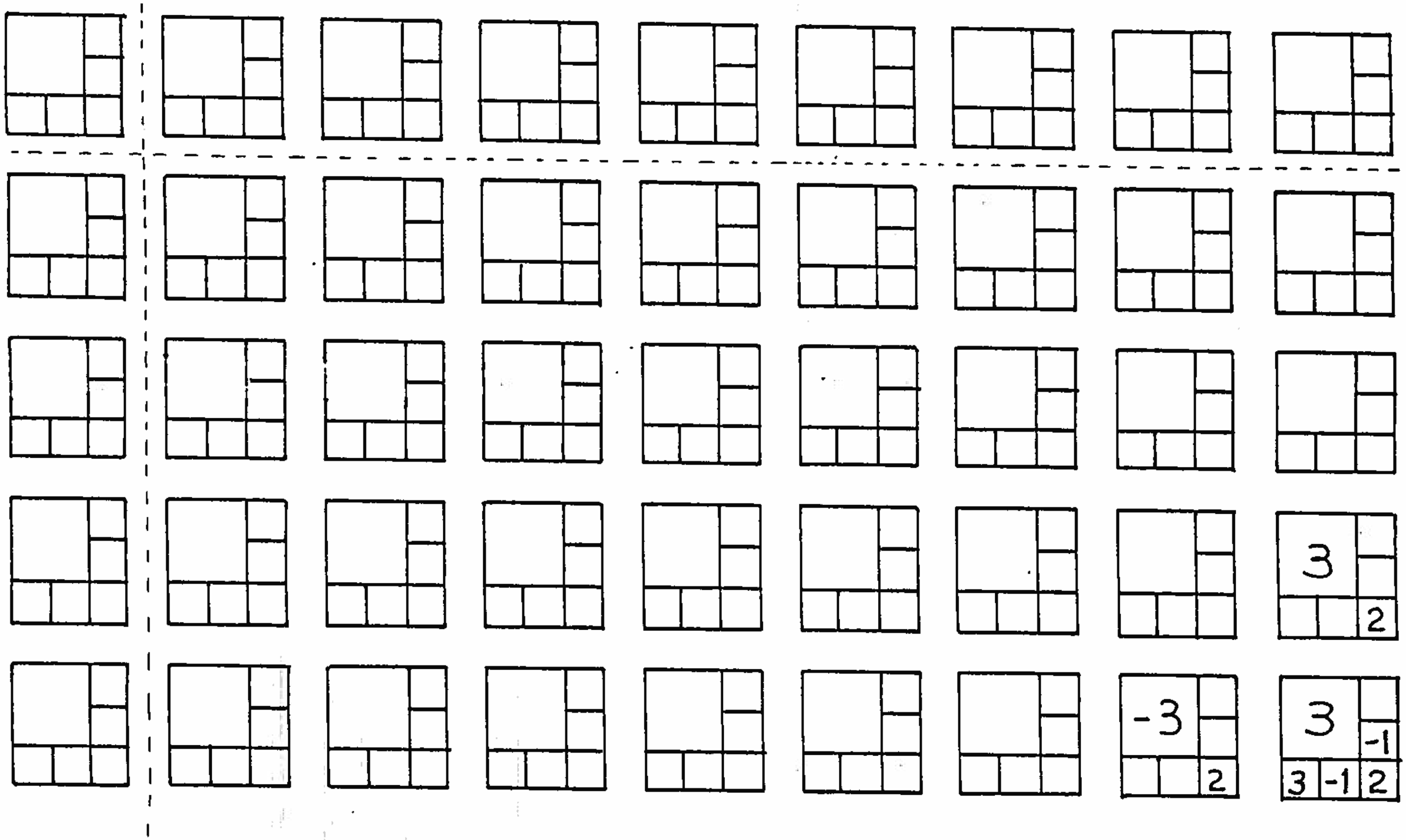


Step(16)

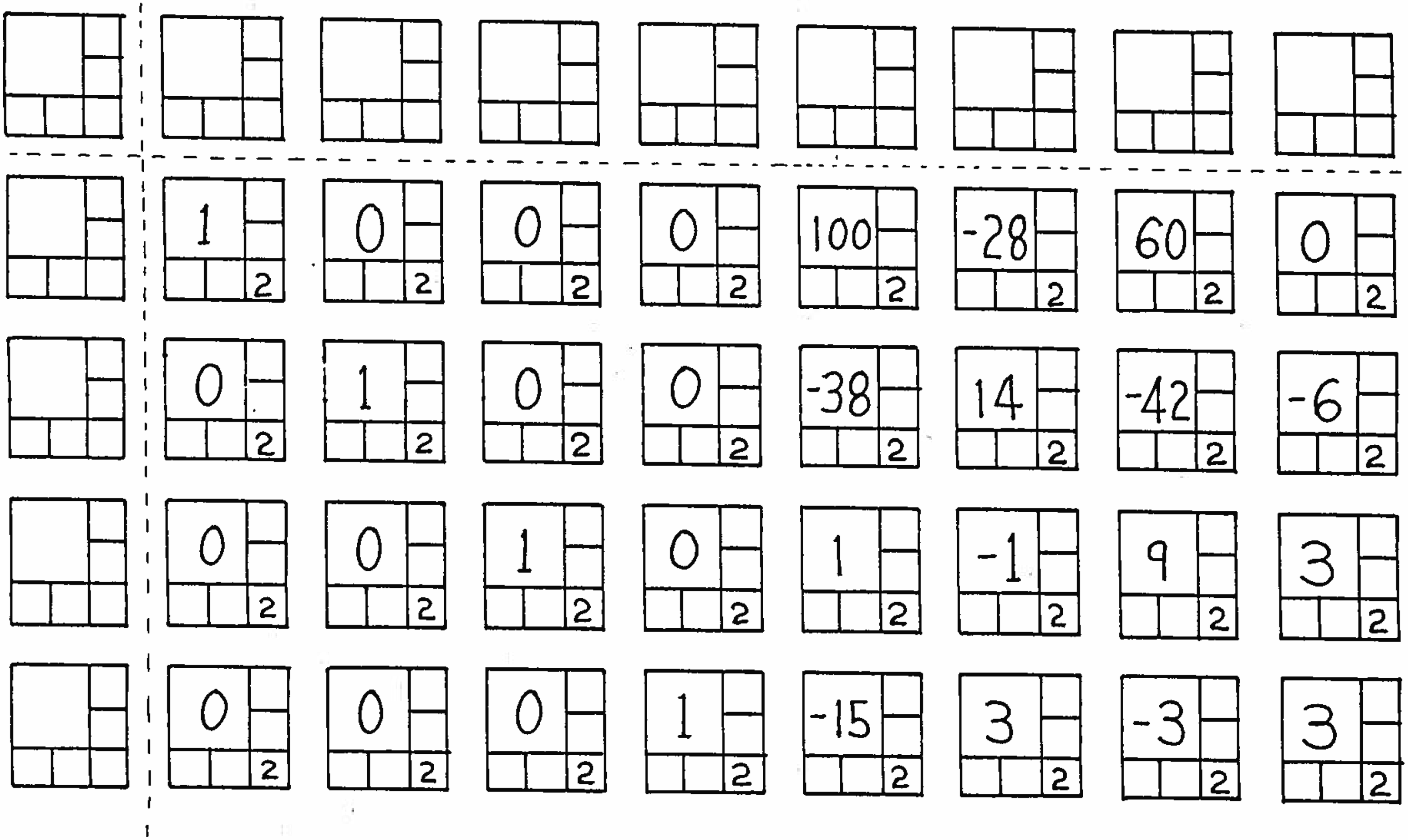
Step(17)



Step(18)



Step(19)



	1 2	0 2	0 2	0 2	100 2	-28 2	60 2	0 2
	0 2	1 2	0 2	0 2	-38 2	14 2	-42 2	-6 2
	0 2	0 2	1 2	0 2	1 2	-1 2	9 2	3 2
	0 2	0 2	0 2	1 2	-15 2	3 2	-3 2	3 2