NASA TECHNICAL MEMORANDUM NASA TM-75322

STRUCTURED PROGRAMMING - PRINCIPLES, NOTATION, PROCEDURE

by

Jost

(NASA-TM-75322) STRUCTURED PROGRAMMING: PRINCIPLES, NOTATION, PROCEDURE (National Aeronautics and Space Administration) 20 p HC A02/MF A01 CSCL 09B N78-24818 Unclas G3/61 21227

Translation of "Strukturierte Programmierung - Prinzip, Notation, Vorgehensweisse", Siemens Aktiengesellschaft, (West Germany), Report 209 gf/J/Nou, November 22, 1974, 23 pp.

NATIONAL AERONAUTICS AND SPACE ADMINISTRATION
WASHINGTON, D, C. 20546 MAY 1978

STANDARD TITLE PAGE

1. Report No. NASA TM-75322	2. Government Accession No.	3. Recipient's Catalog No.
4. Title and Subtitle STRUCTURED PROGRAMMING - PRINCIPLE NOTATION, PROCEDURE		5. Report Date 1978
		6. Performing Organization Code
7. Author(s) Jost		8. Performing Organization Report No.
		10. Work Unit No.
9. Performing Organization Name and Address SCITRAN P. O Box 5456 SANTA BARBARA, CA 93108		11. Contract or Grant No. NASW-2791
		13. Type of Report and Period Covered Translation
12. Sponsoring Agency Name and Address National Aeronautics and Space Administration Washington, D. C. 20546		14. Sponsoring Agency Code

15. Supplementary Notes

Translation of "Strukturierte Programmierung - Prinzip, Notation, Vorgehensweise", Siemens Aktiengesellschaft, (West Germany), Report 209 gf/J/Nou, November 22, 1974, 23 pp.

16. Abstract

ORIGINAL PAGE IS OF POOR QUALITY

17. Key Words (Selected by Author(s))		18. Distribution Statement Unclassified - Unlimited	
19. Security Classif. (of this report) Unclassified	20. Security Classif. (of this page) Unclassified	21. No. of Pages 21	22.

STRUCTURED PROGRAMMING - PRINCIPLE, NOTATION, PROCEDURE

Jost

Siemens Aktiengesellschaft

Text pages/Appendices 15/N ZL anl 4177, pages 1-8

Classification Symbol: 209 gf/J/Nou

Specialty or Main Division: N ZL

Location: Fg ZL VS 3

Mch H

Date: November 22, 1974

Agency: Fg ZL Lab 386

Telephone Number: 47097

Verification: [illegible] i.V. [illegible]

Key Words: 5-10 terms which characterize the content:

Software, structured programming, program structure, systematics, symbols, design, testing.

SUMMARY

Structured programming is known as a method of developing a specially clear programs with low maintenance. At our firm, several steps have been taken for application of this method. This report is intended as an introduction to the principle and the procedure.

The structured programming method brings about transparency of the programs by strict adhesion to the block principle, and by reducing the control to only three basic types. A block is part of one input and one output each. Each program is a block, and its inner design can be represented as a continued block encapsulation. Three control flux types are allowed for structuring a block into new blocks:

- sequence, a sequence of two blocks

- selection: alternatives between two blocks in connection with a condition,

- repetition: repetition of a block in connection with a final criterion.

STRUCTURED PROGRAMMING - PRINCIPLE, NOTATION, PROCEDURE

Jost

Siemens Aktiengesellschaft

Text pages/Appendices 15/N ZL anl 4177, pages 1-8

Classification Symbol: 209 gf/J/Nou

Specialty or Main Division: N ZL

Location: Fg ZL VS 3

Mch H

Date: November 22, 1974

Agency: Fg ZL Lab 386

Telephone Number: 47097

Verification: [illegible] i.V. [illegible]

Key Words: 5-10 terms which characterize the content:

Software, structured programming, program structure, systematics, symbols, design, testing.

ORIGINAL PAGE IS
OF POOR QUALITY

SUMMARY

Structured programming is known as a method of developing a specially clear programs with low maintenance. At our firm, several steps have been taken for application of this method. This report is intended as an introduction to the principle and the procedure.

The structured programming method brings about transparency of the programs by strict adhesion to the block principle, and by reducing the control to only three basic types. A block is part of one input and one output each. Each program is a block, and its inner design can be represented as a continued block encapsulation. Three control flux types are allowed for structuring a block into new blocks:

- sequence, a sequence of two blocks
- selection: alternatives between two blocks in connection with a condition,
- repetition: repetition of a block in connection with a final criterion.

Structured programs are best represented using a notation developed by Nassi and Shneiderman, which gives a clear representation of the block encapsulation. In this report, we will suggest a set of symbols which can be used until binding directives are republished.

Structured programming also allows a new method of procedure for design and testing. Programs can be designed "top down", that is, they can start at the highest program plane and can penetrate to the lowest plane by step-wise refinements. The testing methodology also is adapted to this procedure. First, the highest program plane is tested, and the programs which are not yet finished in the next lower plane are represented by so-called "dummies". They are gradually replaced by the real programs.

CONTENTS

1. INTRODUCTION

In the development of programs, it is customary to represent the program in the form of a flow diagram. By using connectors, it is possible to arbitrarily branch the control flow and to bring it back together again. Later on, during coding, jumps are put in at the points.

The unrestricted use of jumps leads to programs which are meshed in a complex manner, and are therefore characterized by the following features;

- they are difficult to understand
- they are difficult to test
- they are difficult to change
- they are difficult to maintain.

Structured programming removes this difficulty by allowing branches and node points only according to a fixed basic model. Strict adhesion to the structure rules allows the possibility of designing a program "top down", that is, starting in the highest plane. The test method is also based on this principle. The method of proceeding is somewhat contradictory to conventional practice, where a major program is considered for the most part as a collection of a series of elementary components, and the overall function is built up from below in a certain sense (by bottom-up).

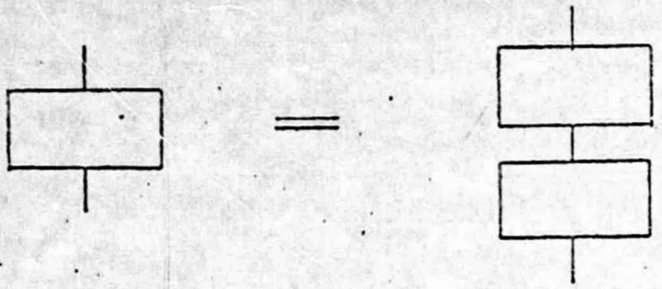
This report gives a short introduction into the nature and method of structured programming. More detailed information is given in the extensive reference list in [1].

2. Principles of Structure

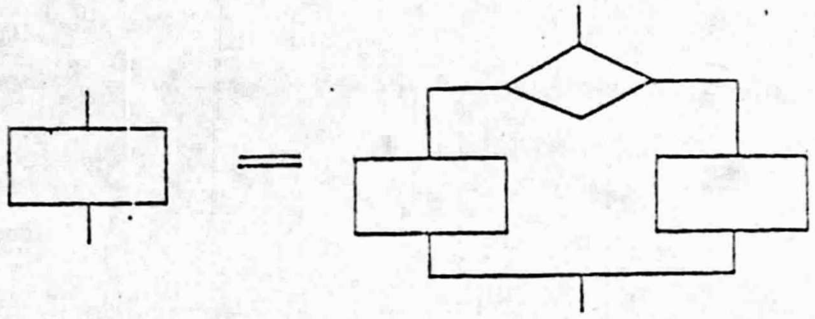
Structured programming is based on the block concept. A block is a program part with one input and one output. The block can only be entered through the input and only can be left through the output. Jumps from the outside into the interior of the block, and jumps from the interior of the block into the other program parts are not allowed. However, subprogram jumps are allowed. The convention is established that the subprogram will return to the calling point. The running of the program occurs as though the subprogram were contained in the block.

Each block can be divided into subblocks, and only three types of control flow are allowed. It can be proven that these structural types are sufficient for all problems.

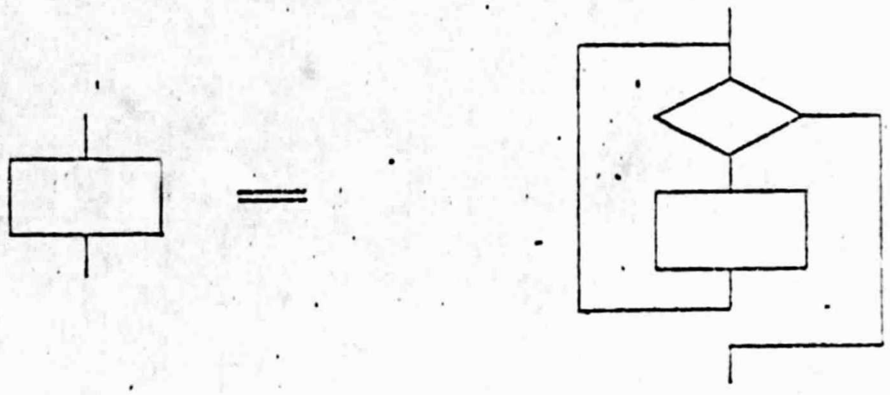
a) Sequence: a block can be divided into a sequence of two blocks.



b) Selection: a block can be divided into two blocks which are executed depending on the condition:



c) Repetition: a block can contain another block, which is executed several times, until a final criterion is reached.



All new blocks are then divided further, according to one of the three structure types.

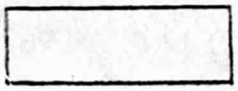
A program built according to these principles is itself a block.

Its inner structure is characterized by a continued block encapsulation. A program of this type is called "well-structured".

3. NOTATION

Well-structured programs, of course, can be represented by flow diagrams. However, a notation of Nassi and Shneiderman is more advantageous, which gives an exact representation of the block encapsulation. This is brought about by drawing the symbols inside one another, and not next to one another as in a flow diagram. In this way, violations of the structuring rules are practically eliminated. The symbols of structured programming are not specified yet in norms. Usually, they are selected so that they can be directly transformed into a statement of the programming language which is used later on. At the present time, only the assembler language is a candidate for communications applications. One has relatively a large amount of freedom in the selection of the symbols, because no specified language constructions have to be considered. We suggest the use of the following symbols; they contain the basic types and are complemented by several variants which are often used. Figures 1-3 show the symbols and their flow diagram equivalents.

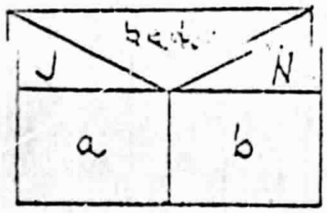
3.1 Block



Each block is represented by a rectangle. The size and side ratio are arbitrary. The block can be embedded in a larger block, and itself can again be decomposed in the blocks.

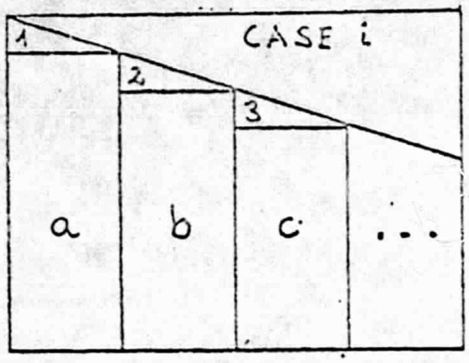
/8

3.2 Branches



ORIGINAL PAGE IS OF POOR QUALITY

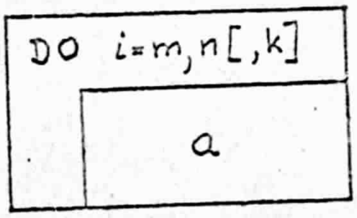
This symbol describes the branches based on the condition bed. If the condition is satisfied, block a is executed, and if it is not satisfied, block b is executed. Each of these blocks can be empty. The sides for J and N can be selected arbitrarily. The symbols can be drawn asymmetrically, for example, when a block is empty, and the other must be subdivided further.



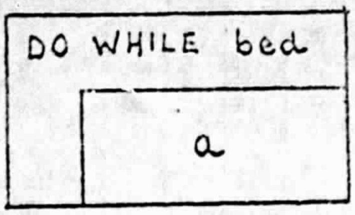
This symbol describes a multiple branch; depending on the value of the variables i, exactly one of the blocks a, b, c, ... is executed.

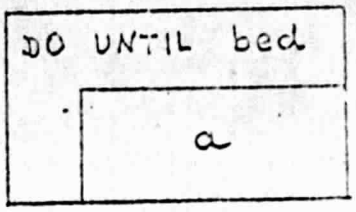
3.3 Loops

/9



This symbol describes a counting loop. Block a is executed several times, and the loop index i runs from an initial value n to a final value n. The increment is k. If k is missing, it is assumed to be 1.





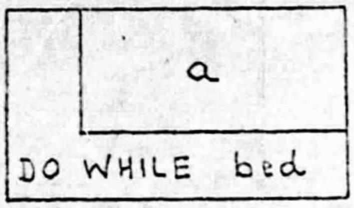
These symbols describe data-dependent loops. The repetition of block a is controlled by the condition bed. It is interrogated before execution of block a.

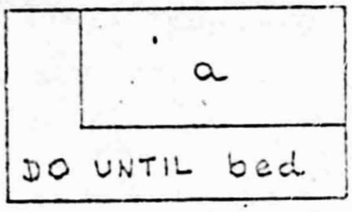
In the WHILE form, bed is a repetition condition. As long as the condition is satisfied, block a is repeated.

In the UNTIL form, bed is an interruption condition. As soon as the condition is satisfied, the loop is terminated.

By negation of this condition, one form can be transformed into the other. /10

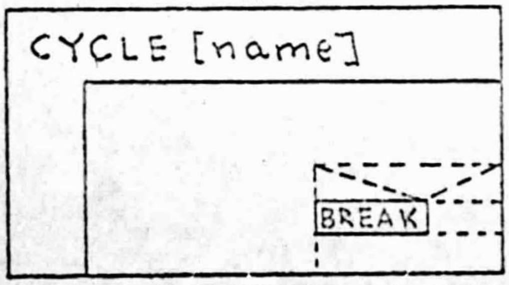
If upon entry into the DO-block in the WHILE form, the condition is not satisfied, but the condition is satisfied in the UNTIL form, then the block a is not executed.





Also these symbols describe data-dependent loops. In contrast to the previous ones, the condition is interrogated after execution of block a. This means that block a is executed at least once.

What was stated above applies for the condition for the WHILE and UNTIL form.



ORIGINAL PAGE IS OF POOR QUALITY

This symbol describes a loop which can be interrupted at an arbitrary point. The interruption point is characterized by the specification BREAK inside of block a. Several interruption points can be present.

If several CYCLE's are encapsulated within one another, each can be characterized by a name. By means of BREAK, one can specify which CYCLE is to be interrupted. If the name is not given in BREAK, then always the innermost CYCLE which encloses BREAK is assumed.

The CYCLE construction can only be used when the other loop constructions are not appropriate.

4. DESIGN PROCEDURE /11

The development of a well-structured program is done in several uniform steps. One starts in the highest program plane, and advances to the lowest plane by continued refinement. First of all, the rough structure of the entire program is outlined on a sheet of paper. Blocks are used for which relatively large program parts are available, but in this first phase, they are not given any additional structure. Each of these blocks has a name, and is detailed on an additional sheet in the next lower plane (Figure 4). This procedure is repeated until only blocks are present which consist of a linear execution plan.

Sheets 5 and 6 show this procedure, for example, of an assembly controller program. Figure 5 shows the rough structure. The program consists of two blocks. The first one is the starting routine, which is executed once during the turn-on procedure. The second is a cycle. It describes the typical endless loop, which is characteristic for real time systems. The body of the CYCLE is divided into a sequence of seven blocks. The second block is a branch, and its NO-block is empty. The third and fifth blocks are counting loops. The seventh block is a data-dependent block. All of the blocks given capital letters represent major program parts, which have to be structured further. The block STIMULUS DETERMINATION is given in detail in Figure 6. It again contains a block called SYMBOL PREPROCESSING, which must be detailed in the next lower plane.

According to the advance from the highest to the lowest pro- /12 gram plane, this method is called the "top-down" design method. One of its advantages is that it decomposes a major complex problem, which cannot be overseen entirely into several planes having a limited complexity. The use of blocks which only later on have to be structured can be interpreted as though in each plane one had available a very high programming language with a corresponding statement. The function of each statement is defined in the next plane, until one comes to the lowest plane which contains the statements of the actual programming language. When the problem is decomposed into several planes, one must make sure that units are created which translate the static structure of the translated program. Each sheet should represent a module which is closed off by itself. All of the modules then will lie next to one another in the working memory, even though according to their functions they are encapsulated within one another. In exceptional cases, it is also permissible to embed the code of a lower plane into the next higher plane. For example, the block SYMBOL PRE_ PROCESSING in Figure 6 will be given on a new sheet. However, the corresponding program can be included in the modulus STIMULUS DETERMINATION. In any case, in a design one must make sure that the sheet limits at the same time also represent module limits.

/13

5. PROCEDURE DURING CODING

The coding of a program in the structured representation is done just as for a flow diagram. Each symbol is converted according to its meaning. In the higher programming languages, such as ALGOL or PL-1, this is especially simple, because a language element corresponds directly with each structure element. In the Assembler language, it is necessary to also program the logic implied by the symbols, unless corresponding macros are created.

When the control flow is divided into simple branches and multiple branches, it is useful to work out blocks next to one another in a fixed sequence, for example, from the left to the right. It then is found advantageous to always put empty blocks into the right branch. Figure 7 shows a segment from a well-structured program, and the resulting structure of the primary code. If one obtains blocks which are only completely structured on the next lower plane, then two possibilities exist, which were indicated in the previous section: the block can then be looked upon as an independent module. It is then necessary to supply and call this module by a code at the proper location. However, one can also drop down to the lower plane and embed the code into the higher plane. Figure 8 shows the two possibilities for a simple example.

/14

6. PROCEDURE FOR TESTING

The top-down method of procedure during the design and coding also leads to a corresponding test methodology. Testing starts as soon as the rough structure of the program has been specified, and coded, in the highest plane. In order for the entire system to be able to run, the non-structured blocks are replaced by so-called "dummies". These are program parts which do not yet satisfy the function

ORIGINAL PAGE IS
OF POOR QUALITY

and which later on will be required at this point, and which at this point only formally satisfy the requirements at the interface. Using these dummies, it is possible to test the uppermost program planes. If no errors are found, then the second plane is tested. The dummies are replaced gradually by the real blocks, and sometimes new dummies occur for blocks of the third stage. This process is continued until all of the real blocks are available, and have been tested.

The "top-down" procedure is exactly opposite to the previous testing methodology, where one goes from a component test through a composite test up to the total test. The advantage of our method is that one can start testing very early, and each component is tested in the true program surroundings, and not in an artificial test surrounding. The dummies do represent an additional effort. However, because they are used, test environments for conventional components and composite tests are not required.

[illegible]

REFERENCES

1. Sorgenfrei, "Structured Programming, a Method of Developing Error-Free Programs", 209a/Sor/Ch of May 2, 1974.

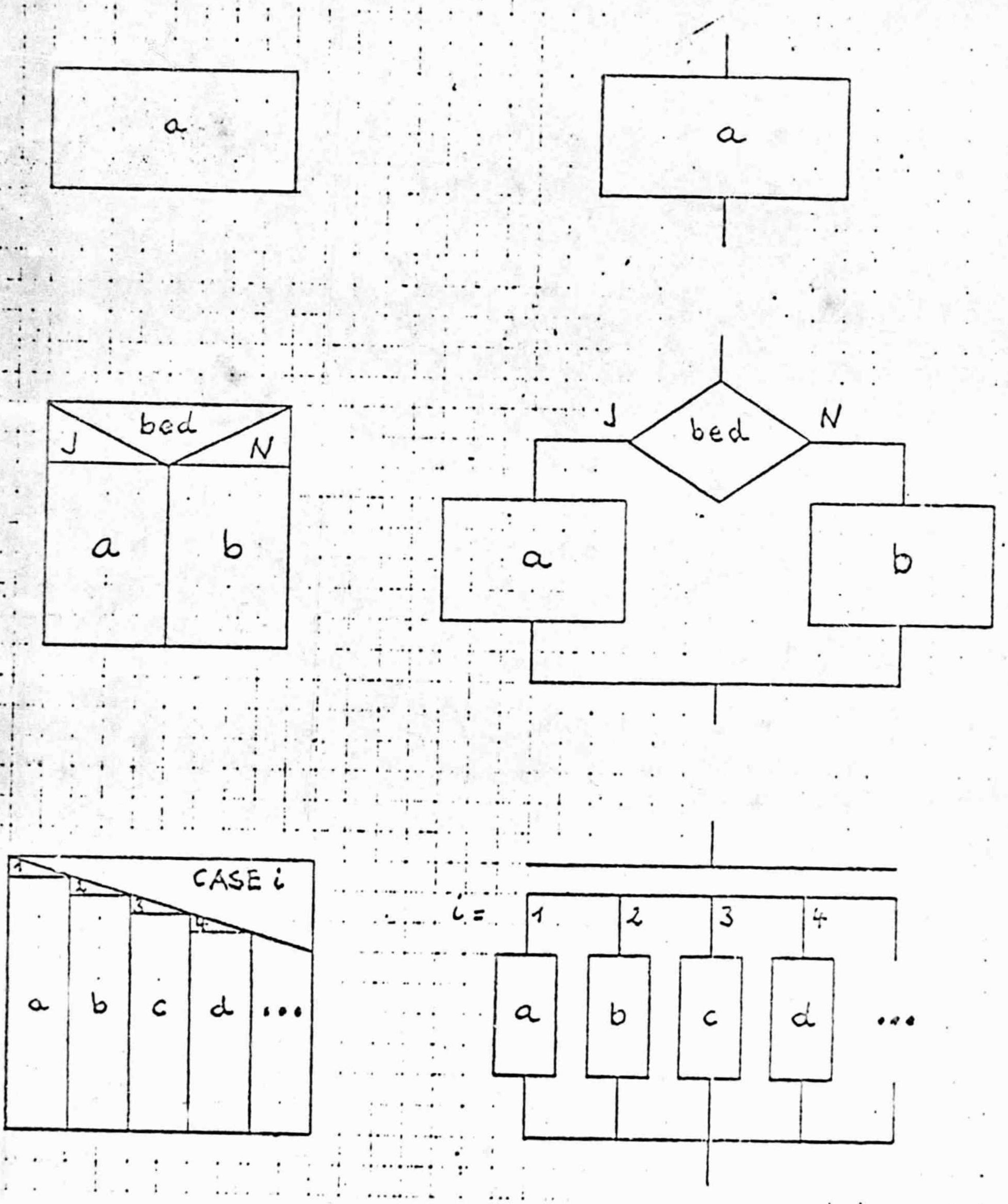
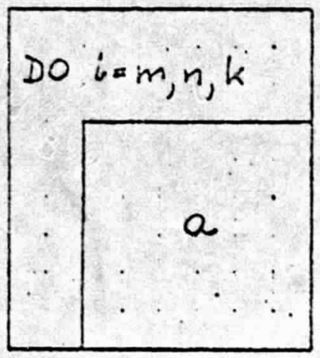
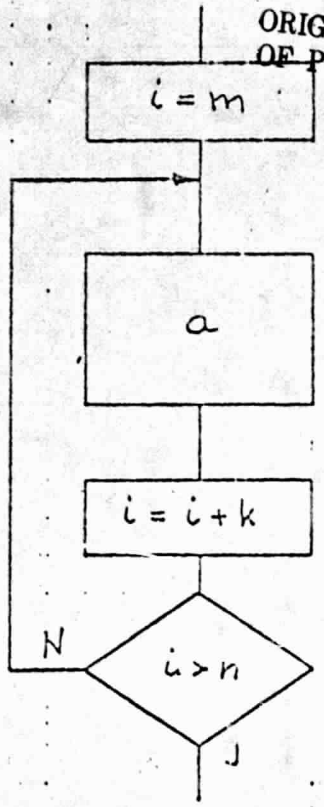
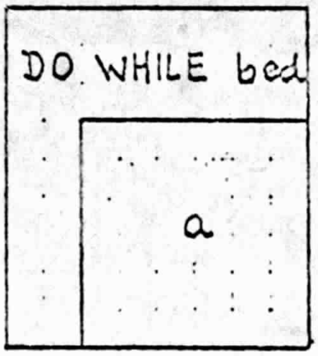


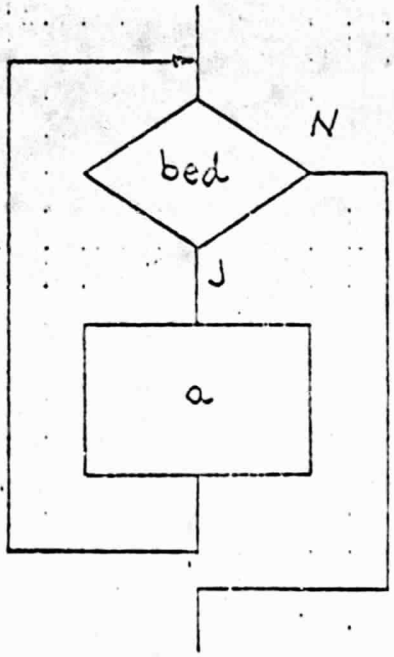
Figure 1: Symbols of Structured Programming and Flow Diagram Equivalents.

ORIGINAL PAGE IS
OF POOR QUALITY



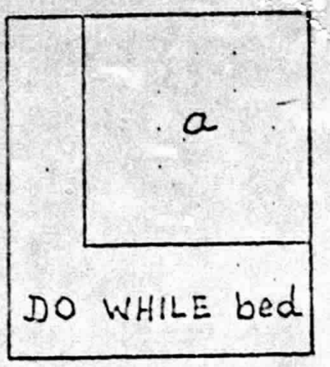


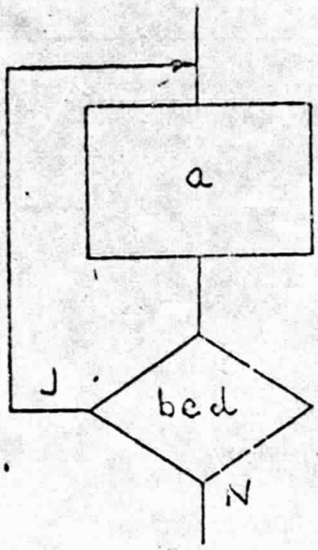




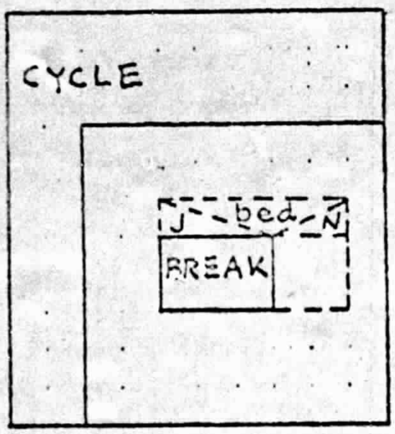
J and N must be exchanged for the UNTIL form.

Figure 2: Symbols of structured programming and flow diagram equivalents (continuation).





J and N must be exchanged
for the UNTIL form



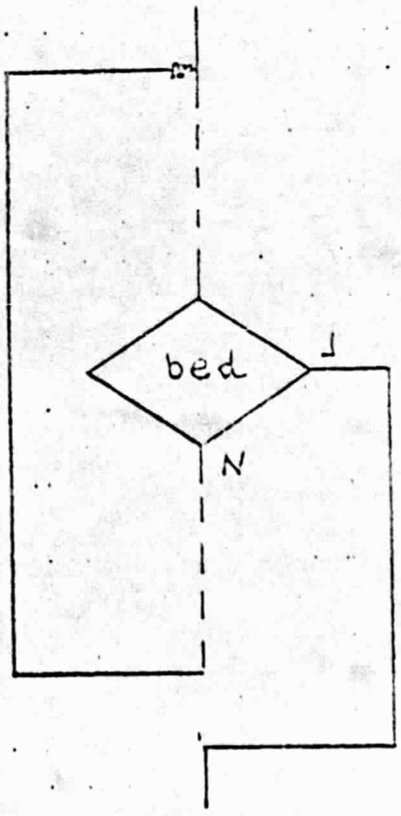


Figure 3: Symbols of structured programming and flow diagram equivalents (continuation)

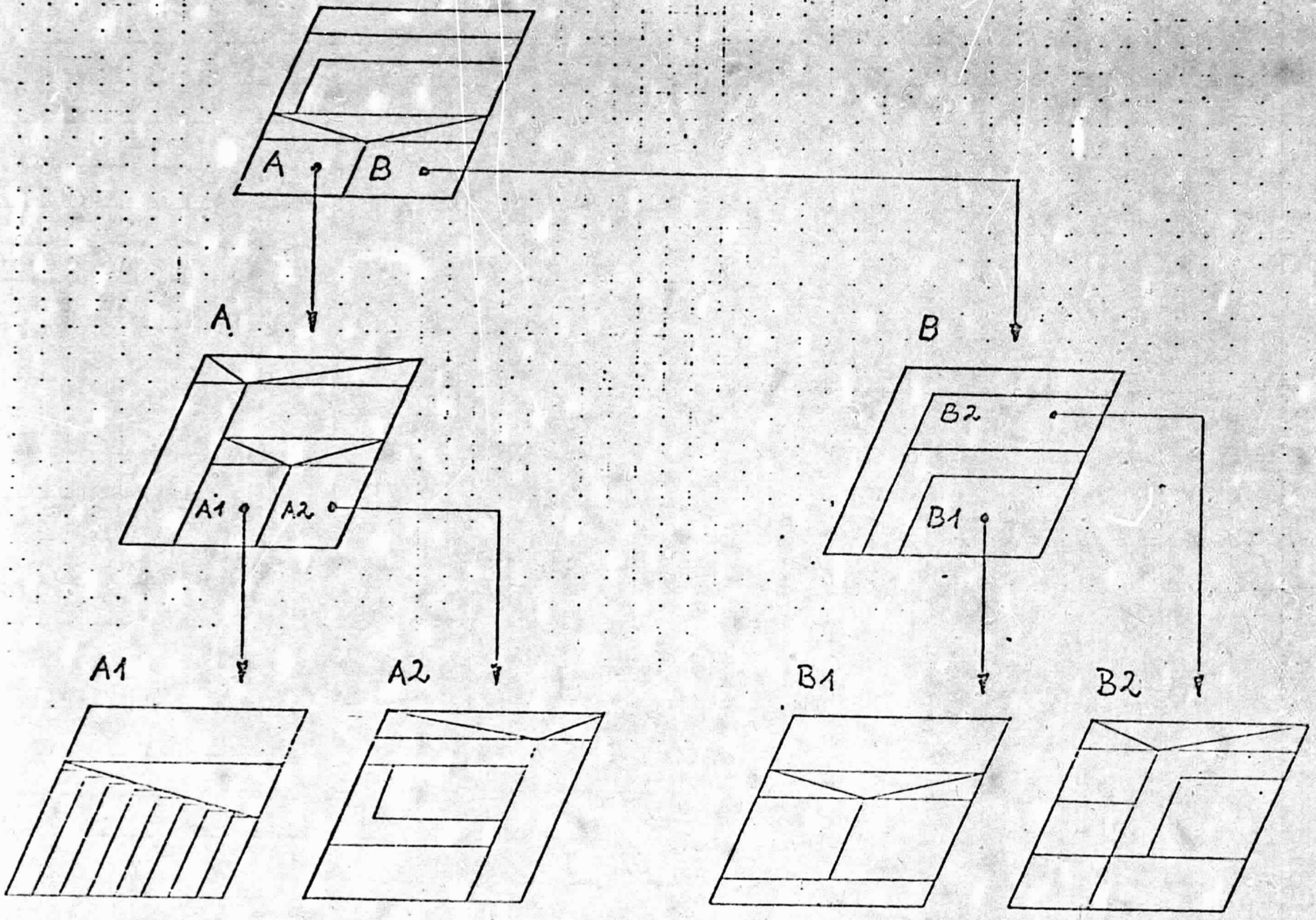


Figure 4: Top-down program design.

Assembly Controller Program

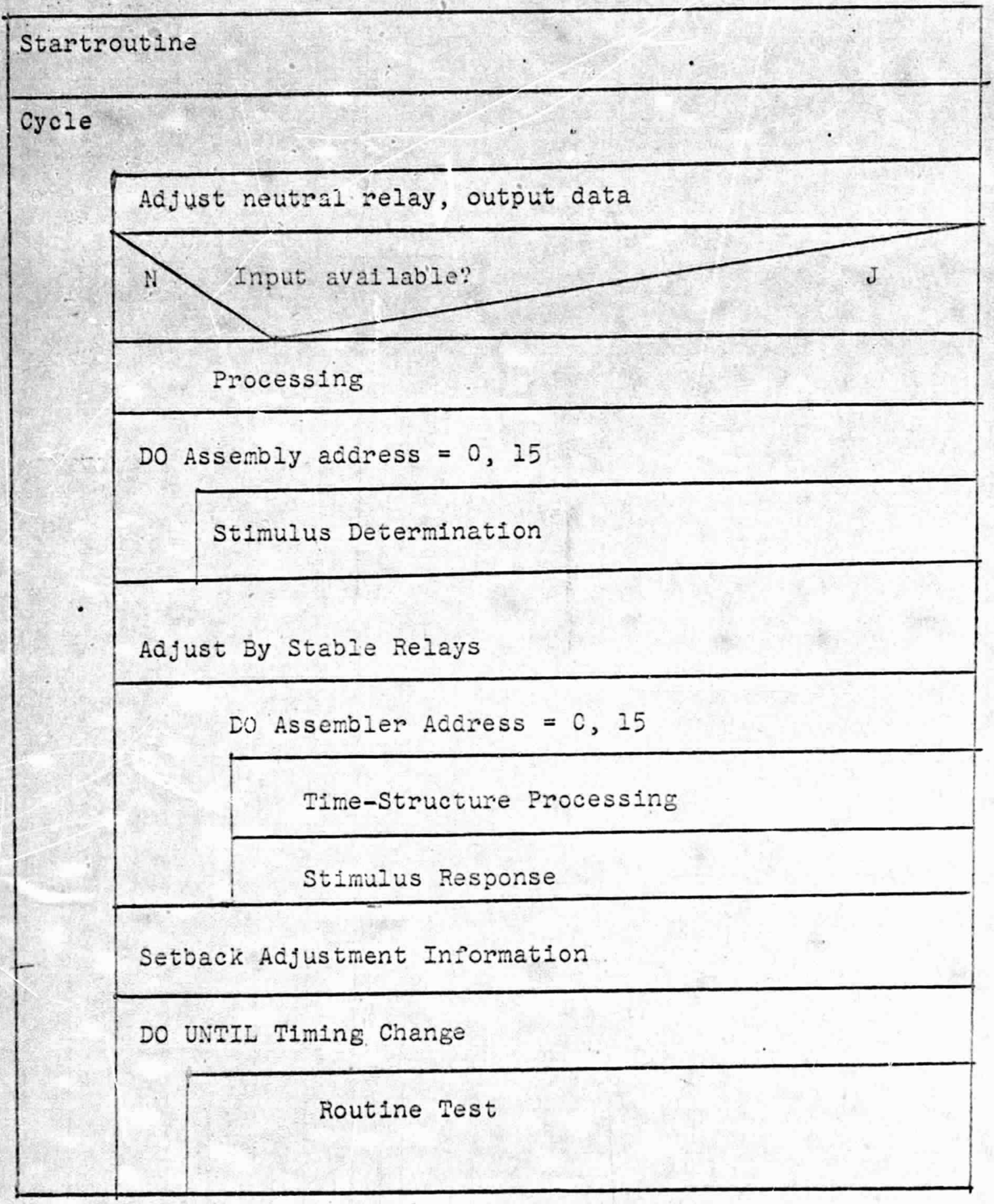

Startroutine
Cycle
Adjust neutral relay, output data
N
Input available?
J
Processing
DO Assembly address = 0, 15
Stimulus Determination
Adjust By Stable Relays
DO Assembler Address = 0, 15
Time-Structure Processing
Stimulus Response
Setback Adjustment Information
DO UNTIL Timing Change
Routine Test

Figure 5: Assembler Controller Program - Rough Structure

STIMULUS DETERMINATION

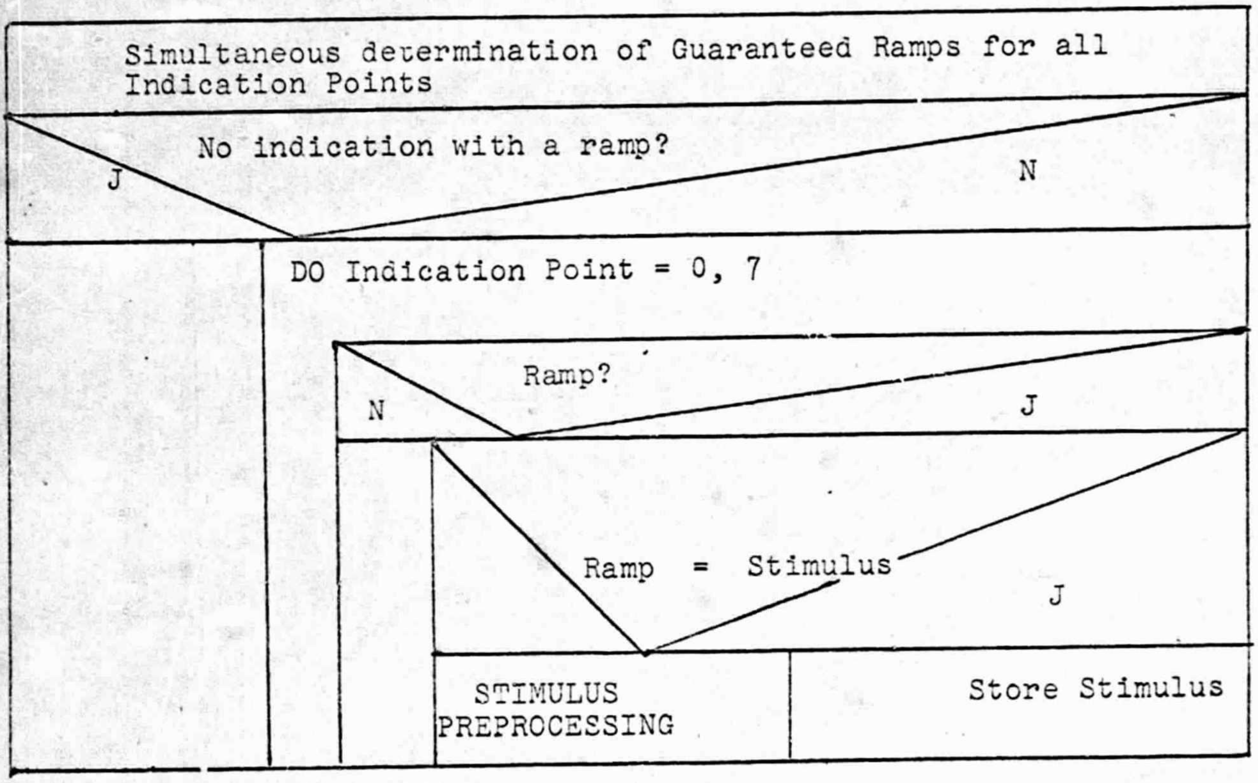
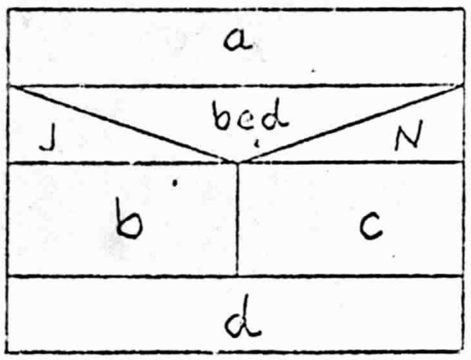

Simultaneous determination of Guaranteed Ramps for all Indication Points
No indication with a ramp?
J
N
DO Indication Point = 0, 7
Ramp?
N
J
Ramp = Stimulus
J
STIMULUS PREPROCESSING
Store Stimulus

Figure 6: Detail of the STIMULUS DETERMINATION Block



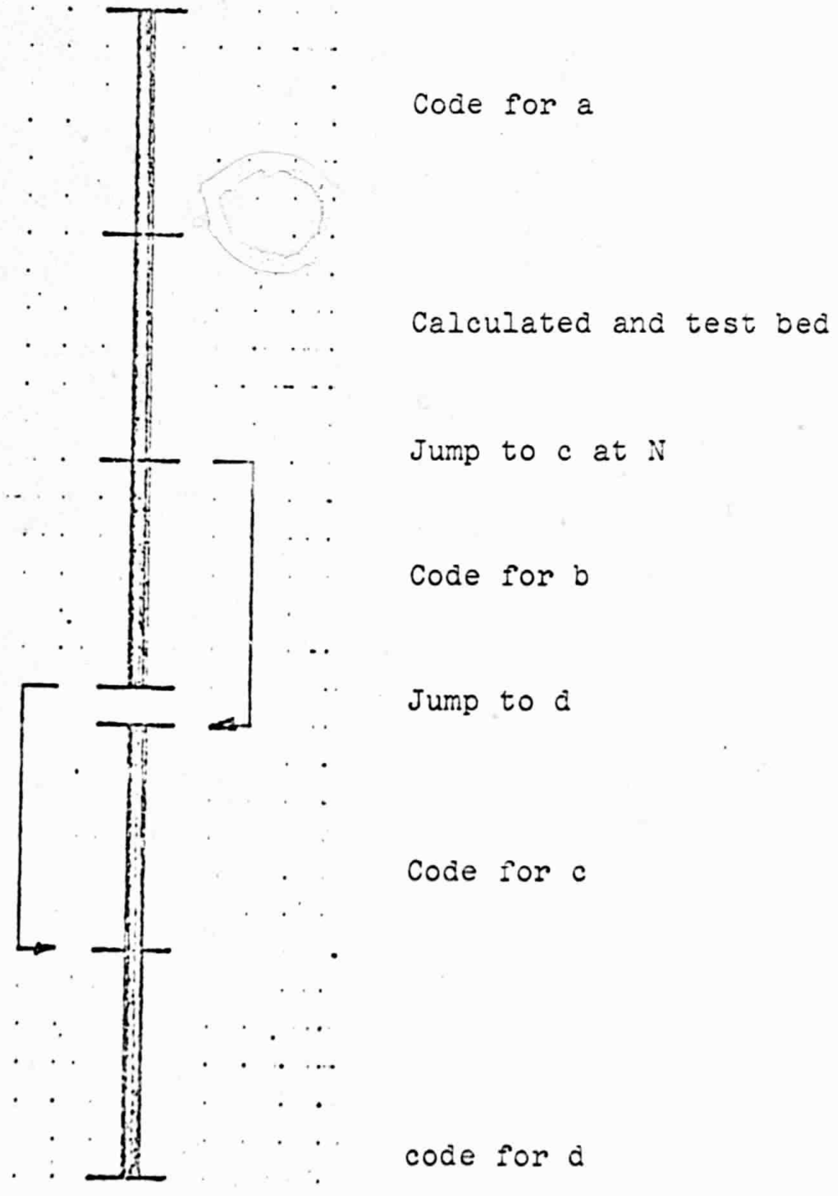


Figure 7: Conversion of structured representation into program code

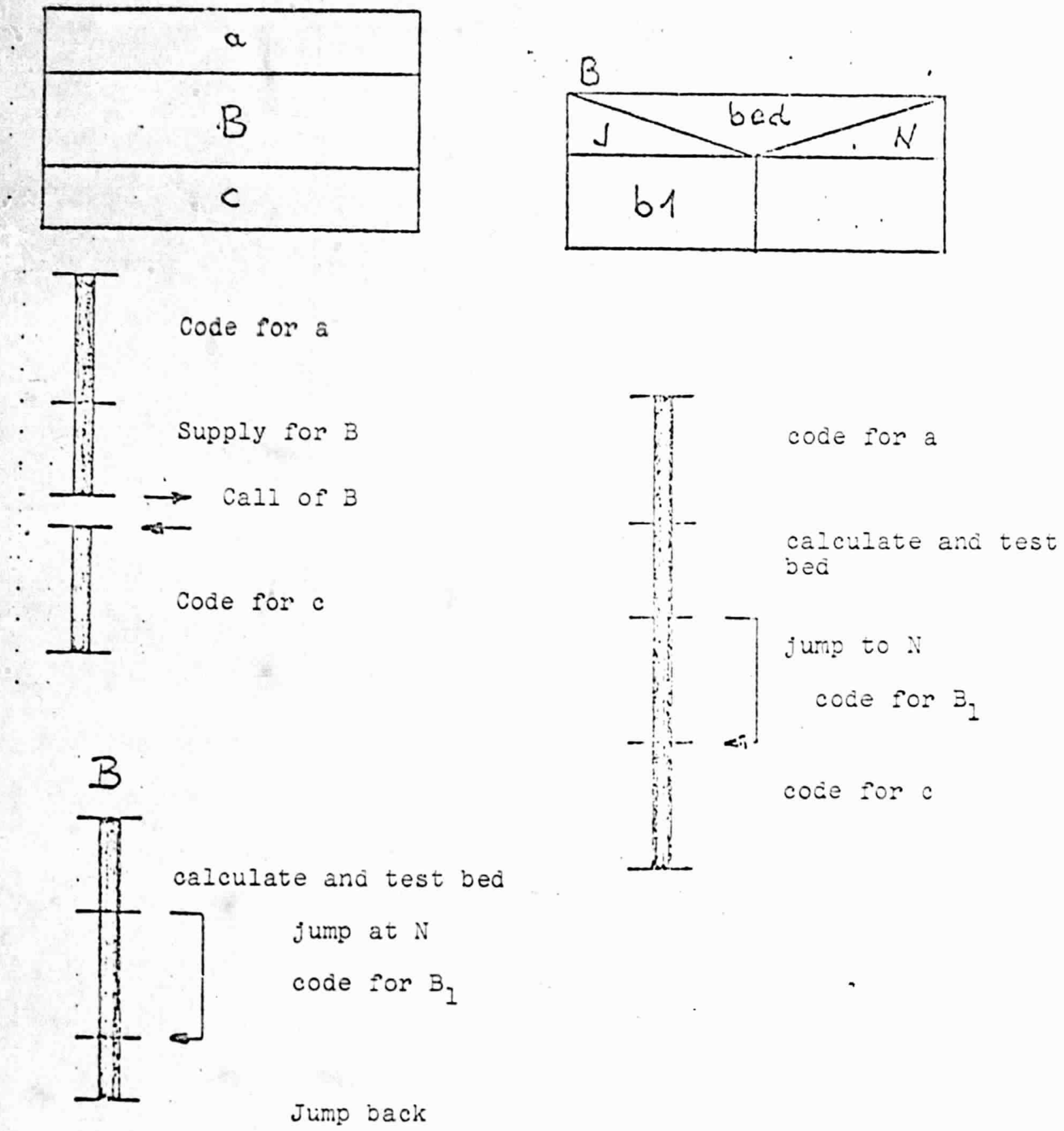


Figure 8: Handling of subordinate blocks

a) independent module

b) embedded