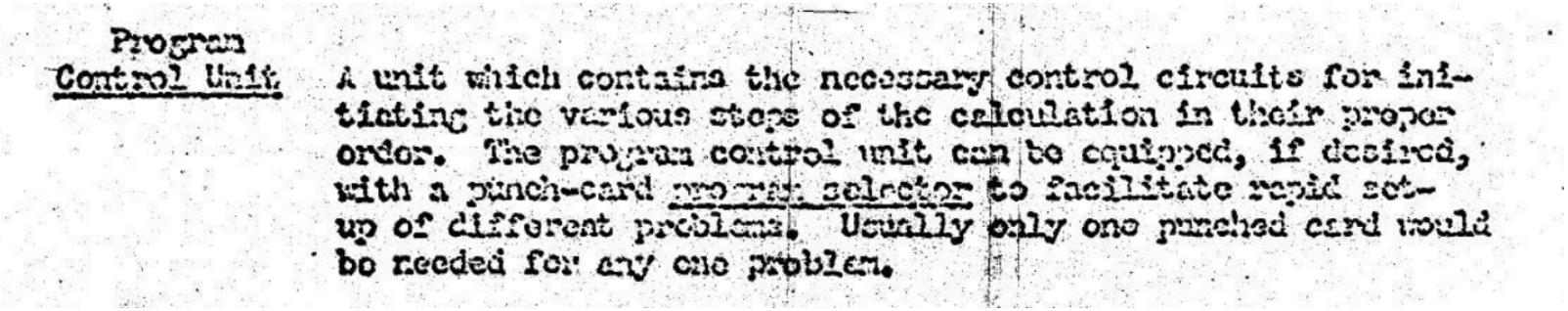
Programs and their graphical representation c. 1946

Mark Priestley
Research Fellow, 2017-8
The National Museum of Computing
Bletchley, UK

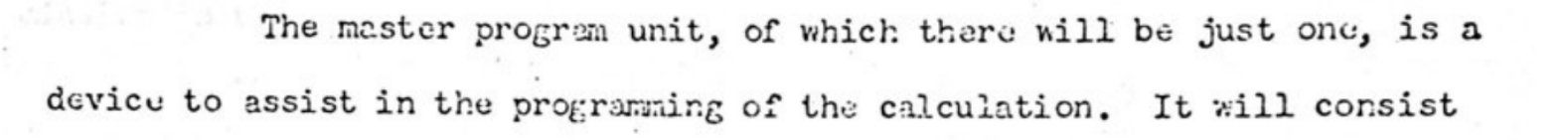
"Program" etc in the ENIAC project:

1943:



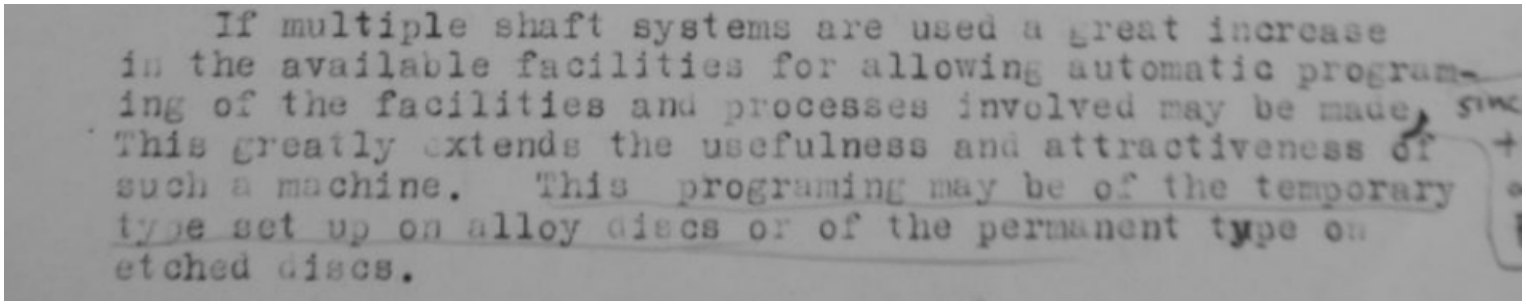
Program Control Unit. A unit which contains the necessary control circuits for initiating the various steps of the calculation in their proper order. The program control unit can be equipped, if desired, with a punch-card program selector to facilitate rapid set-up of different problems. Usually only one punched card would be needed for any one problem.

1943:



The master program unit, of which there will be just one, is a device to assist in the programming of the calculation. It will consist

1944:



If multiple shaft systems are used a great increase in the available facilities for allowing automatic programing of the facilities and processes involved may be made. This greatly extends the usefulness and attractiveness of such a machine. This programing may be of the temporary type set up on alloy discs or of the permanent type on etched discs.

In an extension of its informal meaning (concert programs etc), it refers to the selection and ordering of events, here computational operations of various sorts.

Ambiguous hints of the modern use of “program” – “program of operations”, or coded instructions?

Herman Goldstine, Sep 2, 1944:

remedy this disparity we propose a centralized programming device in which the program routine is stored in a coded form in the same type storage devices suggested above. The other crucial advantage of central programming is that any routine, however complex, can be carried out whereas in the present ENIAC we are limited by the number of switches available on the accumulators.

Herman Goldstine, Oct 3, 1945:

storing program information, whereas the EDVAC will contain a large number of units capable of remembering program instructions. In fact, all the programming information for a given program will be handled inside the EDVAC and the magnetic tapes will be used only as a means for bringing these instructions into the machine before the actual program is started.

Goldstine and von Neumann, 1946:

numbers and what spaces are free for storage purposes. Since he does not know in advance what locations these quantities will have in his completed program, the positions in the first storage box are denoted by 1', 2',...in the second storage

The 1947 *Planning and Coding* reports retreat a bit from the noun “program”. “Coding” appears to be a subtask of “programming”:

There is included in that discussion a tabulation of the orders the machine will be able to obey. In this, the second part of the report, we intend to show how the orders may be used in the actual programming of numerical problems.

Before proceeding to the actual programming of such problems, we consider it desirable to discuss the nature of coding per se and in doing this to lay down a

and “program” is used in its traditional, informal sense:

8.2 Before we start on this program of coding specific problems, however, there remains one technical point that has to be settled first.

In the three reports, “program” only seems to be used once to mean “code”:

12.2 Need for this program. 2

– but this is immediately undercut by the text:

12.3 We call the coded sequence of a problem a *routine*,

Coding is a kind of translation:

precise: This sequence of codes will impose the desired sequence of actions on C by the following mechanism: C scans the sequence of codes, and effects the instructions, which they contain, one by one. If this were just a linear scanning of the coded sequence, the latter remaining throughout the procedure unchanged in form, then matters would be quite simple. Coding a problem for the machine would merely be what its name indicates: Translating a meaningful text (the instructions that govern solving the problem under consideration) from one language (the language of mathematics, in which the planner will have conceived the problem, or rather the numerical procedure by which he has decided to solve the problem) into another language (that one of our code).

But it's more complicated than this:

- Transfers of control, loops, and conditional branching break up the "linear scanning of the coded sequence"
- Program execution can actually change the coded sequence of orders

Our problem is then to find simple, step-by-step methods, by which these difficulties can be overcome. Since coding is not a static process of translation, but rather the technique of providing a dynamic background to control the automatic evolution of a meaning, it has to be viewed as a logical problem and one that represents a new branch of formal logics. We propose to show in the course of this report how this task is mastered.

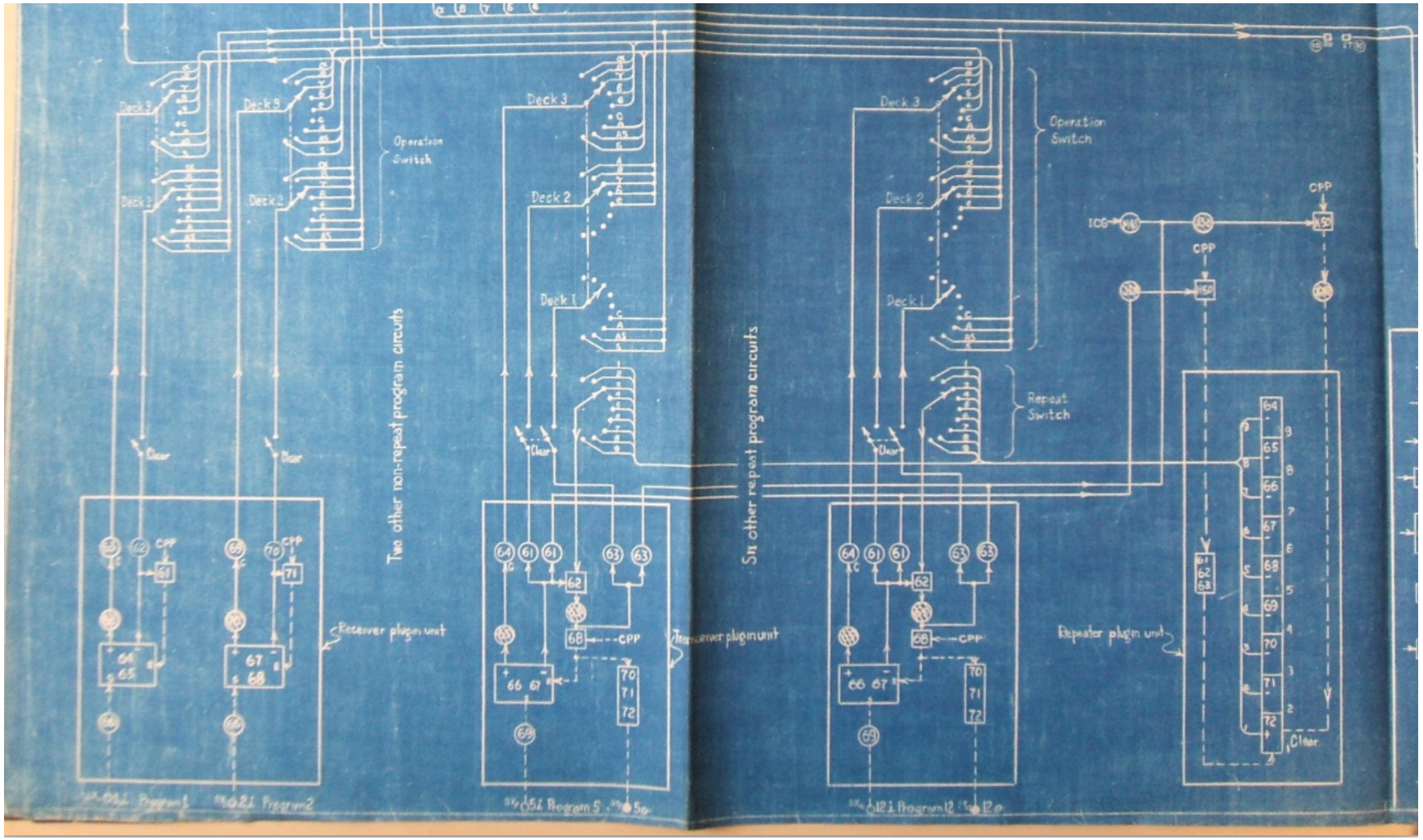
Block Diagrams:

able to obey. In this part of the report we intend to show how these orders may be used in the actual programming of numerical problems.

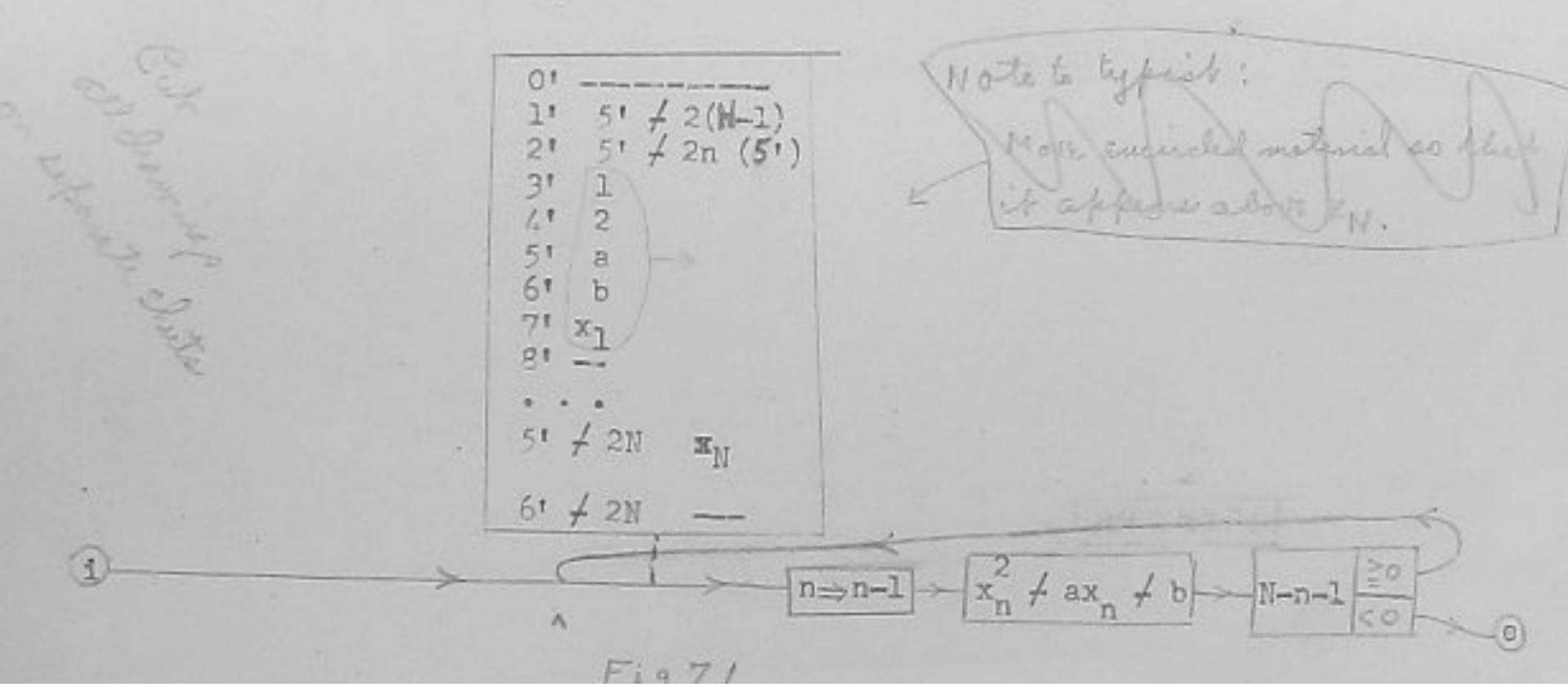
From our limited experience with the coding of numerical problems we have acquired a conviction that this programming is best accomplished with the help of some graphical representation of the problem. We have attempted, in a preliminary way, to standardize upon a graphical notation for a problem in the hope that this symbolism would be sufficiently explicit to make quite clear to a relatively unskilled operator the general outline of the procedure. We further hope that from such a block-diagram the operator will be able with ease to carry out a complete coding of a problem.

Goldstine and von Neumann, Draft report (1946)

Part of the block diagram of ENIAC's accumulators:



Blocks hide complexity and directed lines show connections



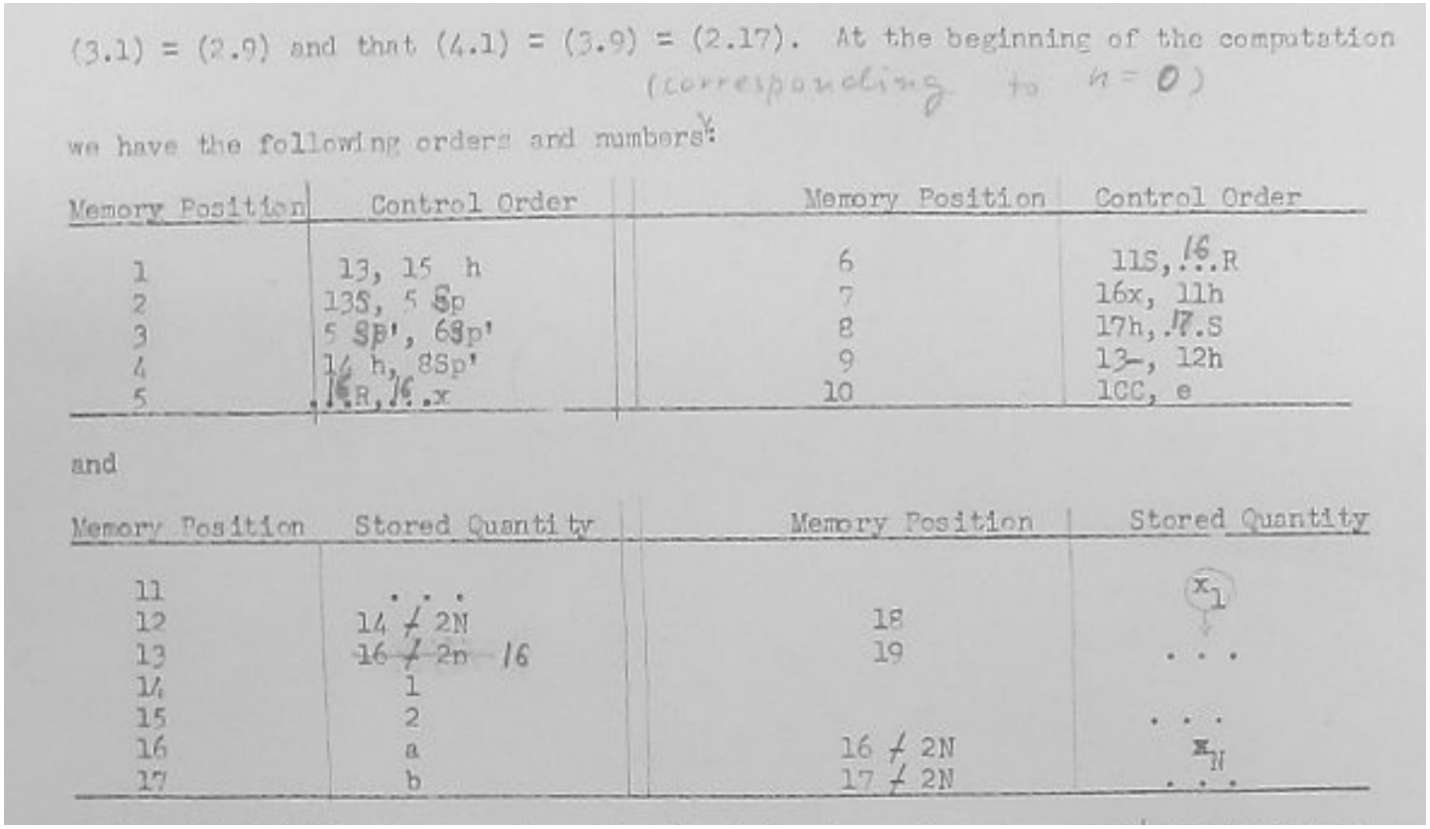
A block diagram is a *picture of a routine in memory*.

The blocks correspond to areas of physical memory:

- Storage boxes correspond to number-words.
- Control boxes – substitution, arithmetic, conditions – correspond to sequences of order-words.

The directed line segments indicate the possible paths that control can take.

After coding, we have:



(3.1) = (2.9) and that (4.1) = (3.9) = (2.17). At the beginning of the computation (corresponding to $n = 0$)

we have the following orders and numbers:

Memory Position	Control Order	Memory Position	Control Order
1	13, 15 h	6	11S, 16.R
2	13S, 5 Sp	7	16x, 11h
3	5 Sp', 6Sp'	8	17h, 17.S
4	14 h, 8Sp'	9	13–, 12h
5	16.R, 16.x	10	1CC, e

and

Memory Position	Stored Quantity	Memory Position	Stored Quantity
11	. . .		x_1
12	14 + 2N	18	
13	~~16 + 2n~~ 16	19	. . .
14	1		
15	2		. . .
16	a	16 + 2N	x_N
17	b	17 + 2N	. . .

What's the "program"? All this? Or just the order words in locations 1–10? The orders in their initial form, or their evolving contents, however expressed?

Obviously, the initial contents of words 1 – 17+2N are important: they mark the endpoint of human labour (ideally), where agency passes from coder to machine.

What do the block diagrams buy us?

On a superficial level, a nice visual representation of the program.

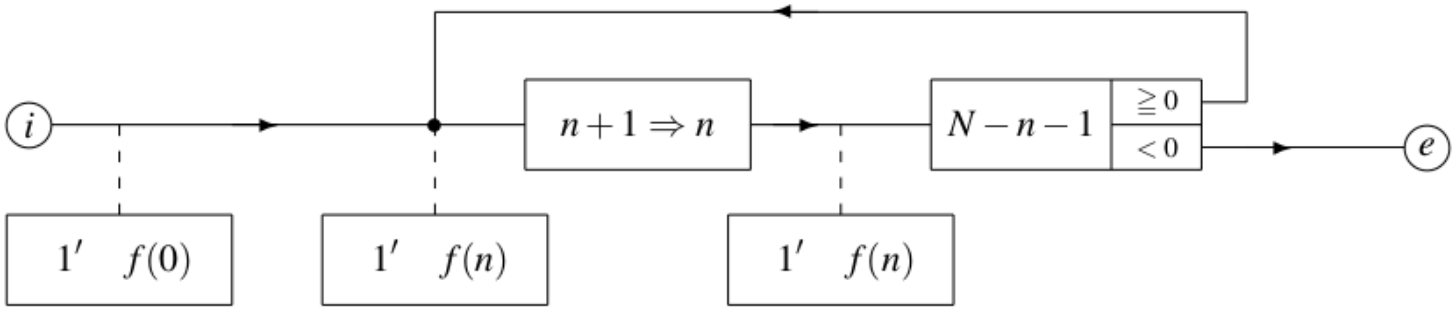
But the real problem is *change*: as a routine executes, both data and instructions will, in general, be modified.

> Thus the relation of the coded instruction sequence to the mathematically conceived procedure of (numerical) solution is not a statical one, that of a translation, but highly dynamical: A coded order stands not simply for its present contents at its present location, but more fully for any succession of passages of C through it, in connection with any succession of modified contents to be found by C there, all of this being determined by all other orders of the sequence (in conjunction with the one now under consideration) This entire, potentially very involved, interplay of interactions evolves successively while C runs through the operations controlled and directed by these continuously changing instructions.

Block diagrams are an attempt to extend mathematical language to express the kinds of changes that take place when a program is running.

The logico-mathematical notation of substitution is used to express change.

Simple loops controlled by a "inductive variable" $0 \leq n < N$ are the test case.

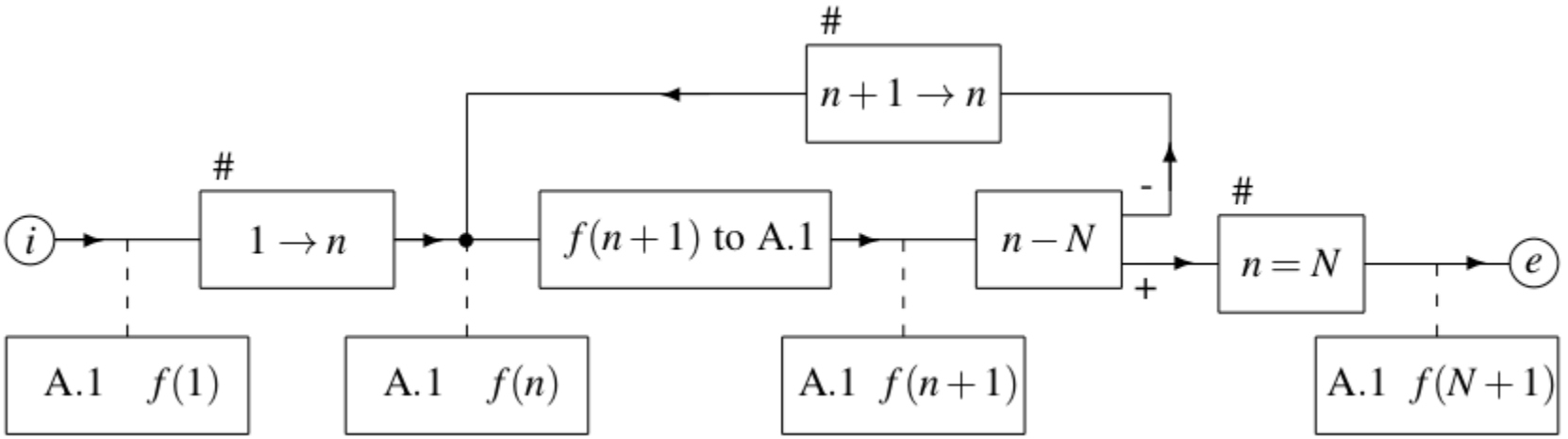


The substitution box stands for the actual code that updates the value of any storage locations that depend on n.

But this is not reflected in the storage boxes. And note that the third box can't read "1' f(n+1)" because it would then be inconsistent with the second one when the loop iterates.

Also, the relationship between the first two storage boxes is not clear.

To “reconcile the storage boxes”, substitution is (in the 1947 notation, now called “flow diagrams”) promoted to be a purely symbolic operation.



cf. 7.4 and the detailed discussion in 7.7, 7.8.1 When the value of a bound variable changes (this is effected by the substitution boxes, cf. the last part of 7.61, this should also cause indirectly a change of those variable storage items in whose expression this variable occurs. However, we prefer to treat these variable value changes merely as changes in notation which do not entail any actual change in the relevant variable storage items. On the contrary, their function is to establish agreement between preceding and following expressions (occupying identical storage positions), which differ as expressions, but whose difference is removed by the substitution in question. (For

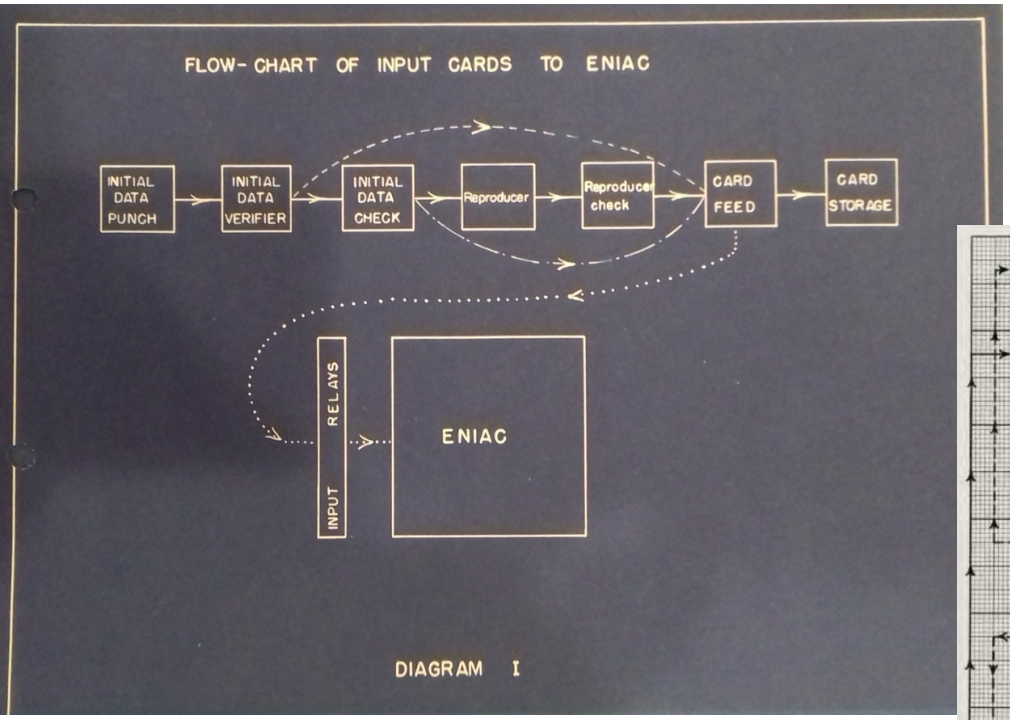
Substitution boxes no longer stand for code. Flow diagrams coordinate the two separate languages – mathematics and code – in one notation.

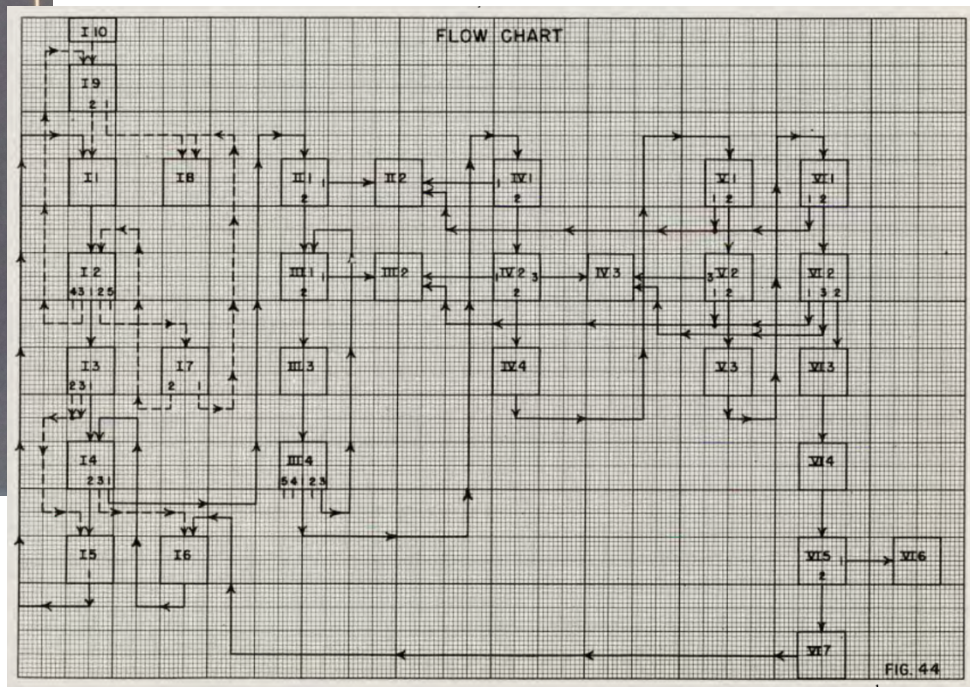
The assertion box “reconciles” the loop variable n with the parameter N.

Why change the terminology from “block diagram” to “flow diagram”?

Block diagrams are straight-forward abstract representations of something physical – electronic circuits or coded words in memory. The new notation is more complex.

The metaphor of “flow” already has some currency in ENIAC circles:

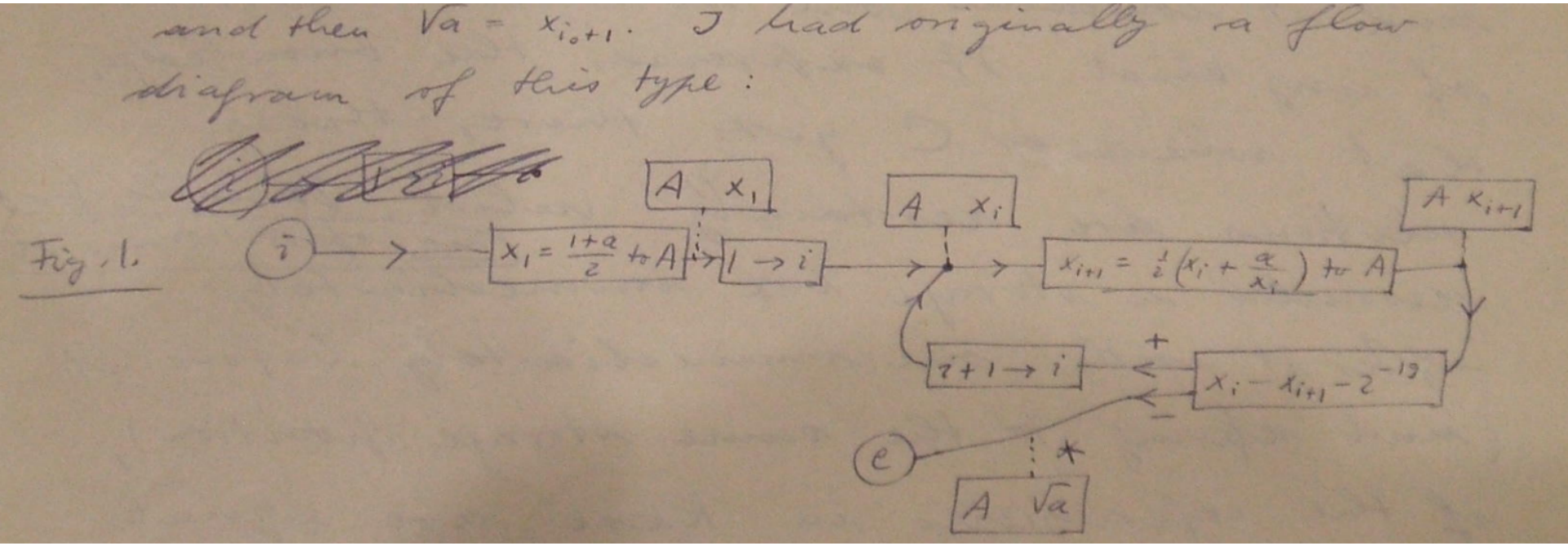




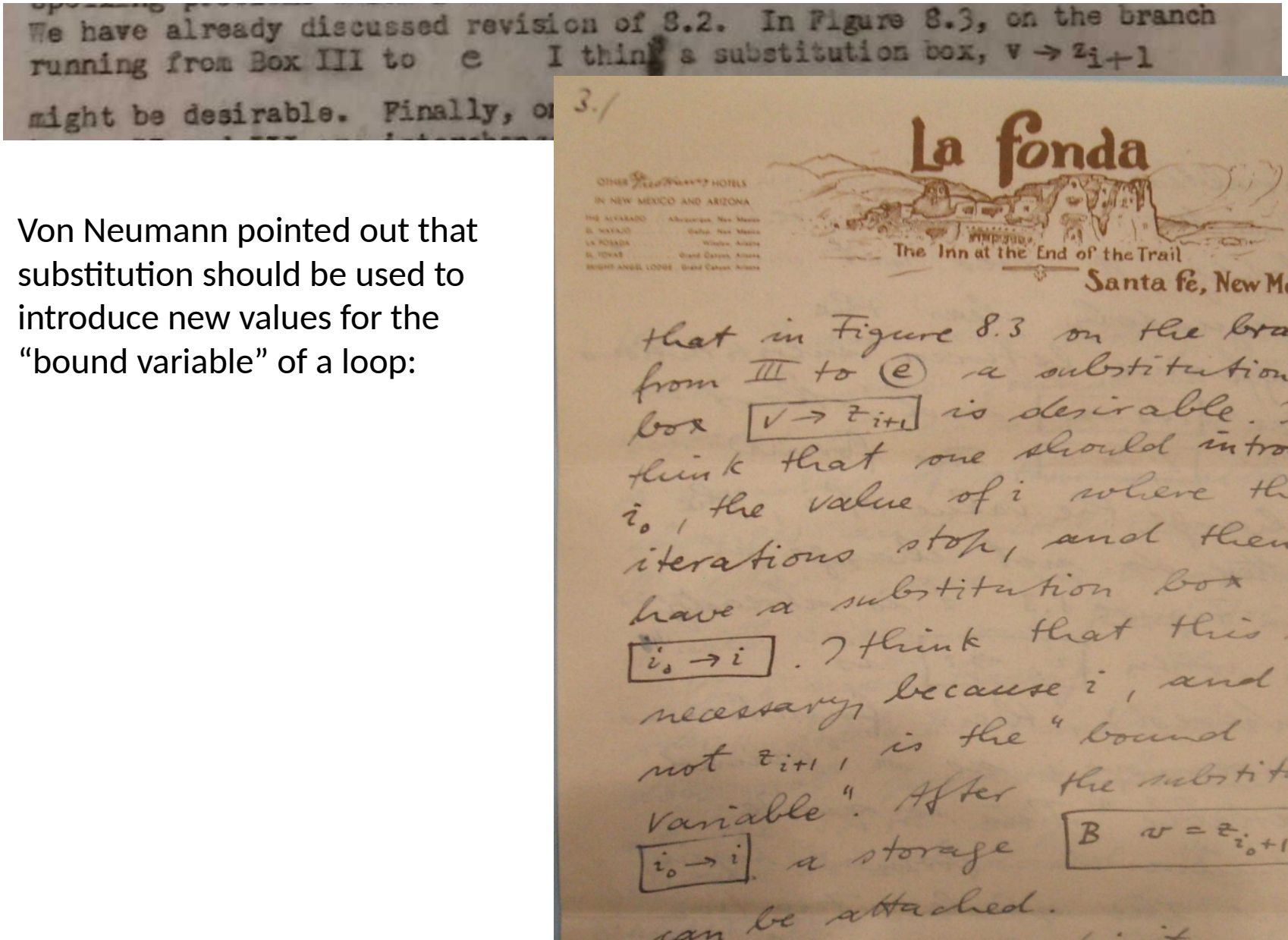
Assertion boxes were defined rather late on, to help “model change”.

February/March 1947 correspondence between Goldstine and von Neumann.

Problem: how to document the result of a loop that calculates a square root iteratively?



Goldstine had suggested a substitution box:


We have already discussed revision of 8.2. In Figure 8.3, on the branch running from Box III to e I think a substitution box, $v \to z_{i+1}$ might be desirable. Finally, o

3./

that in Figure 8.3 on the branch from III to (e) a substitution box $v \to z_{i+1}$ is desirable. I think that one should introduce i_0, the value of i where the iterations stop, and then have a substitution box $i_0 \to i$. I think that this is necessary because i, and not z_{i+1}, is the "bound variable". After the substitution $i_0 \to i$ a storage $B \quad v = z_{i_0+1}$ can be attached.

Von Neumann pointed out that substitution should be used to introduce new values for the “bound variable” of a loop:

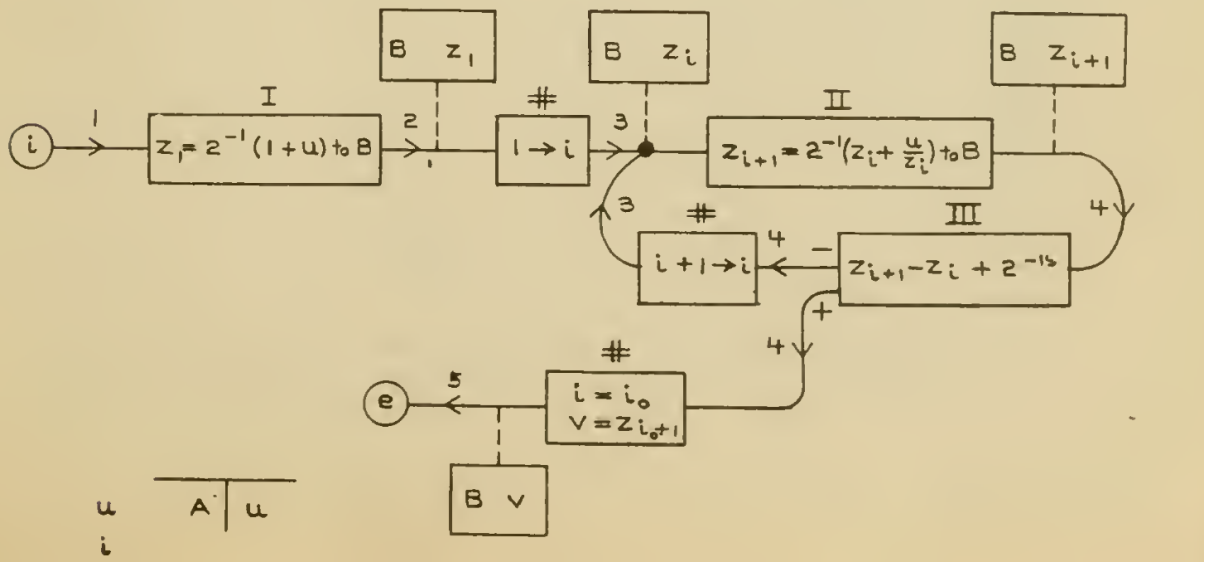
But the following day, von Neumann writes again:

considering, that $\sqrt{a} = x_{i_0}$. I must have been feeble minded when I wrote this: $i_0 \to i$ will reconcile $A \sqrt{a}$ with a succeeding $A x_i$, but not with a preceding one.

Let us introduce a new type of box, called assertion box. It can be inserted

In the published report, the assertion box is used like this:

FIGURE 8.3



By way of conclusions:

1. Circa 1946, the word “program” had some limited currency to refer to coded symbols (on paper? in memory?).

2. But what’s its referent? The code *changes* as the program runs, so what the coder produces is just the starting point, not a fixed text controlling the computer.

3. For Goldstine and von Neumann, the problem was how to find this initial code (order and number words).

4. Flow diagrams define the sequence of operations that the machine will perform. If anything deserves to be called a *program* here, it’s the flow diagram (eg Fig 8.3).

5. Given a flow diagram, it’s meant to be easy to write the (initial) code.

6. Flow diagrams encompass both mathematical language and code. If they are boundary objects, the boundary runs right through the heart of the notation.