ED 014 115 CG 001 020

COMPUTER CONTROL OF BEHAVIORAL EXPERIMENTS.

BY- SIEGEL, LOUIS
ROCHESTER UNIV., N.Y., ATOMIC ENERGY PROGECT

REPORT NUMBER UR-49-814 PUB DATE 3 SEP 67
EDRS PRICE MF-$0.25 HC-$0.72 16P.

DESCRIPTORS- COMPUTERS, *COMPUTER BASED LABORATORIES, *BEHAVIORAL SCIENCE RESEARCH, *BEHAVIORAL SCIENCES, *EXPERIMENTS, *REINFORCEMENT, SPEECHES,

THE LINC COMPUTER PROVIDES A PARTICULAR SCHEDULE OF REINFORCEMENT FOR BEHAVIORAL EXPERIMENTS BY EXECUTING A SEQUENCE OF COMPUTER OPERATIONS IN CONJUNCTION WITH A SPECIALLY DESIGNED INTERFACE. THE INTERFACE IS THE MEANS OF COMMUNICATION BETWEEN THE EXPERIMENTAL CHAMBER AND THE COMPUTER. THE PROGRAM AND INTERFACE OF AN EXPERIMENT INVOLVING A PIGEON PECKING AT A KEY ARE DISCUSSED. DIAGRAMS PROVIDE ILLUSTRATIVE MATERIAL. (SK)

COMPUTER CONTROL OF BEHAVIORAL EXPERIMENTS*

Louis Siegel

Department of Radiation Biology and Biophysics
University of Rochester
School of Medicine and Dentistry
Rochester, New York

Presented to the American Psychological Association
Washington, D. C.
September 3, 1967

Introduction

I would like to explain today how a computer actually goes about executing a particular schedule of reinforcement, a Fixed Ratio 7. The computer I will refer to, the LINC, was designed and constructed under the direction of Mr. Wesley A. Clark, one of this morning's speakers.

The first slide shows the more obvious components of the experimental setup. They are the experimental chamber, the computer and the investigator. (Slide 1) Now, there are two important components missing from this slide. These are shown in their proper positions in the next slide. (Slide 2) These two components are the interface and the program. The program is the sequence of computer operations that produces the Fixed Ratio schedule. The program is the means of communication between the investigator and the computer. The interface is the necessary hardware that enables the computer

*This paper is based on work performed under contract with the U.S. Atomic Energy Commission at the University of Rochester Atomic Energy Project and has been assigned Report No. UR-49-814.

to operate the feeder and sense the contact opening that occurs during a response. The interface, therefore, is the means of communication between the experimental chamber and the computer. Programming and interfacing are the responsibility of the investigator. Getting the two working individually and together consumes most of his time at the computer.

The Interface

The next slide shows one method of connecting a pigeon key to the computer. (Slide 3) The Schmitt trigger converts a contact opening or closure to a voltage change. The trigger also prevents the contact bounce produced at the key from appearing at the flip-flop intput. That voltage change turns ON the flip-flop, putting it in one of its two stable states. For these flip-flops, the stable states are 0 volts, for OFF, and -3 volts, for ON. It remains in that state until turned OFF by a momentary voltage change from the computer. If the pigeon never makes another key peck, the flip-flop will not change state until it is turned OFF. This memory characteristic is the reason the flip-flop is used. Somewhere in the computer program we will use an instruction that checks to see if sense line 1 is at -3 volts, meaning the flip-flop is ON. If it is, we will know a response has occurred. When this happens, the computer must turn OFF the flip-flop so that the next key peck can turn it ON again. This will be done by an instruction that produces a voltage change or pulse on pulse output line 1. This is an example of program-interface interaction.

Now, a method for interfacing a feeder is shown in the next slide. (Slide 4)

The computer generates a momentary voltage change that is extended in time by the delay circuit. The delay operates the relay and, therefore, the feeder for three seconds. Delays are usually adjustable. Whenever we count seven pecks with our computer program, we will use an instruction that produces a voltage pulse on pulse output line 2. This will send out a momentary voltage change to the delay, which automatically operates the relay and, therefore, the feeder for the required time. This is another example of program-interface interaction. The next slide shows the complete interface. (Slide 5)

When the key is pecked the trigger produces a momentary voltage change, turning ON the flip-flop. This causes the flip-flop's output to be at -3v. When the computer program looks at this flip-flop and senses the -3v, it registers the fact that the key was pecked and then produces a momentary voltage change, turning OFF the flip-flop. The flip-flop is then ready for the next key peck. When the computer program counts seven key pecks, it produces a momentary voltage change on a different wire that operates the delay. The output of the delay is -3v for three seconds. This, then,operates the relay and, therefore, the feeder for three seconds.

In the behavioral laboratory there are usually many experimental chambers, each one needing to be interfaced to the computer. If each chamber has its own separate set of flip-flops, delays, and triggers, interfacing could become rather laborious and expensive. Another approach uses one set of flip-flops, delays and other digital modules that serves many chambers, as shown in the next slide. (Slide 6)

With this method, the chambers, computer pulse and sense lines, and the digital modules are connected to separate pins on a plugboard receiver. The interface for each chamber, or group of chambers, is then wired on plugboards that mate with the receiver. Changing interfaces is merely a matter of changing plugboards. One feature of this approach is that all of the modules (flip-flops, delays, relays, etc.) and all of the computer input-output lines are available for any one experiment. With this method, there is never a permanent connection between the computer and the experimental chambers. The connection is made by plugboards.

The next slide illustrates the sequence of logical operations that, when used in conjunction with the previous interface, define the Fixed Ratio 7 schedule of reinforcement. (Slide 7)

Let us assume here that we are in charge of executing the statements. We begin by setting a counter to zero and turning OFF the flip-flop. We then continue to examine the state of the flip-flop until it is turned ON by a key peck. When this happens we add 1 to the counter and ask if it equals 7. If it does not, we go back to turn OFF the flip-flop and repeat the process. If the counter does equal 7, we operate the feeder and go back to the very beginning.

There is almost a one-to-one correspondence between the sequence of logical operations shown in this slide and the sequence of computer instructions shown in the next slide. (Slide 8)

The Program

Notice, first, that the computer instructions are written as 3-letter mnemonics. These mnemonics represent specific computer operations and inherently have nothing to do with our schedule of reinforcement. It is the order in which they are used and the specific interface they control that enables the instructions to define the Fixed Ratio 7 schedule.

Each instruction, when entered into the co.puter, resides in a separate memory location and is stored in the form of a binary number. Some instructions modify the contents of other memory locations. For example, the first instruction on the slide, SETi16 to -7, puts the number -7, in binary, of course, in memory location 16. It replaces the number previously in location 16 with -7. We shall soon see why -7 is used.

The next instruction, OPR1, merely produces a pulse on pulse line 1. Remember, we have connected pulse line 1 so that it turns OFF the flip-flop. The next instruction, SXL1, checks to see if sense line 1 has -3v on it. Since sense line 1 is connected to the output of the flip-flop, a -3v there indicates that a response has been made. This instruction tells the computer to continue to the next instruction if a response occurred or to go back and check the sense line again if no response occurred.

The next instruction, XSKi16, does two things. First, it adds 1 to memory location 16, which is used to count responses. It then checks to see if memory location 16 contains 0. Now, we started with location 16 set to -7, so adding 1 to it after each response will make it equal to 0 only after seven responses. If it has not reached 0, we go back as shown. If it has reached 0, we continue on to OPR2.

OPR2 produces a pulse on pulse line 2. Since we wired the feeder delay in the interface to pulse line 2, the instruction OPR2 merely operates the feeder. As indicated on the slide, we then go back to the beginning of the program.

Now, computers execute instructions as they are written, one after the other. Jumping back to a previous instruction cannot be implied. The jump must be explicitly stated by an instruction. In this slide, we have implied this jumping back in three places. They are where the decision is NO! for SXL1 and XSKi16 and after the last instruction OPR2.

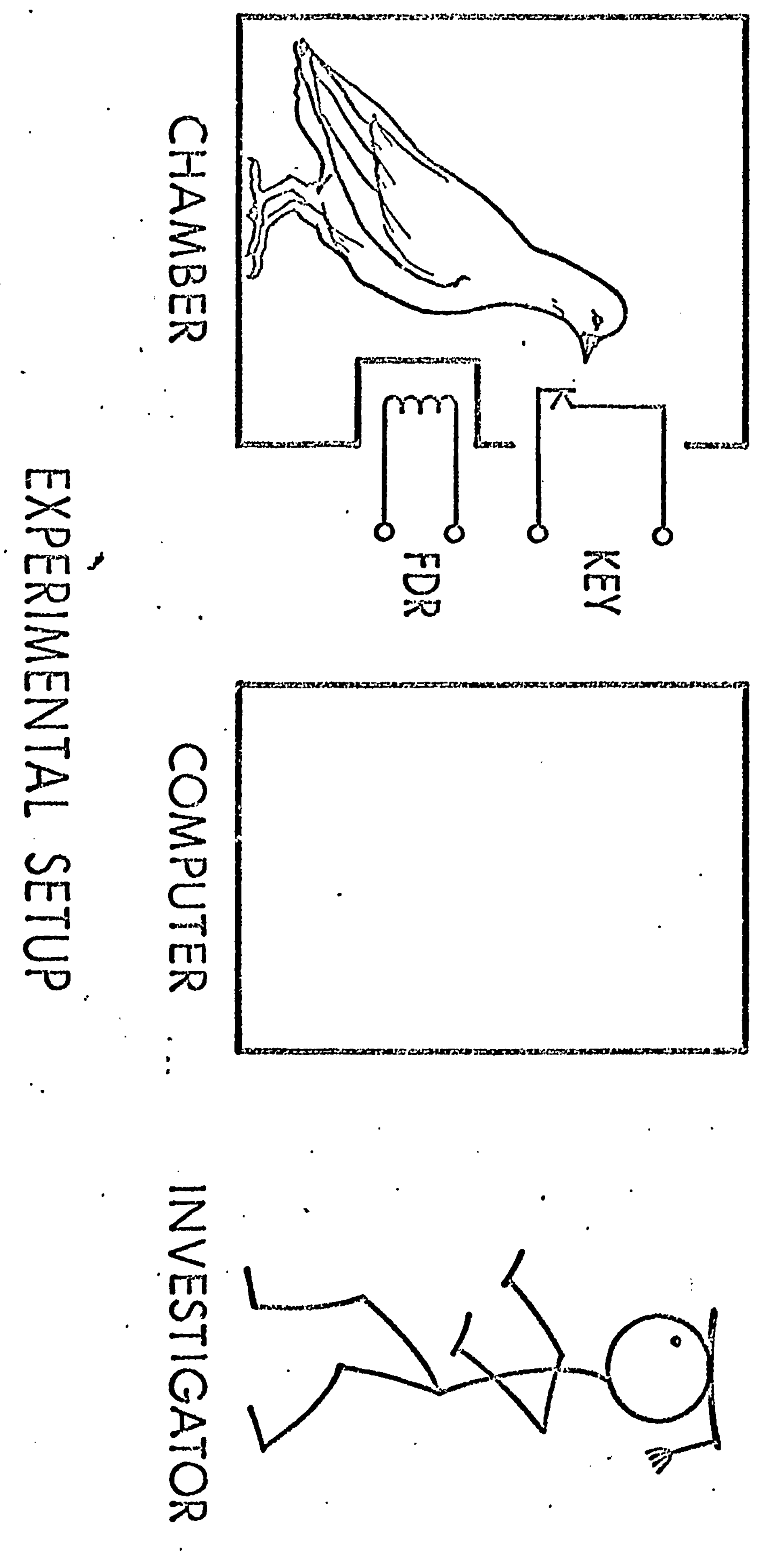
The next slide shows the complete program, specifying the three jumps that were only implied in the last slide. This program is suitable for putting into a computer. (Slide 9)

Notice that each instruction is assigned a separate memory location (or two). The computer begins by setting Location 16 to -7. Location 16 is used as our response counter. OPR1 turns OFF the flip-flop to prepare for the first response. SXL1 examines sense line 1 and directs the computer to Location 5 if no response has occurred or to Location 6 (skipping over 5) if a response has occurred. This automatic skipping of an instruction is dependent on the occurrence of an external event (i.e.: -3v on sense line 1) and is representative of the powerful skip class instructions available in LINC. With no response, the next instruction is JMP4, which takes us back to SXL1. These two instructions have the effect of waiting for a response. When a response occurs, the next instruction after SXL1 is in Location 6. This instruction, XSKi16, first adds 1 to Location 16 and then checks to see if the resulting number is 0. If it is not, the next instruction is in Location 7, JMP3, which directs the computer to Location 3 for the next instruction. If Location 16 does equal 0, the next instruction, in Location 8, is OPR2, which merely produces a pulse on pulse line 2. Since our interface connects pulse line 2 to the feeder delay, the instruction OPR2 operates the feeder. The next instruction, JMP1, tells the computer the next instruction is in Location 1. At Location 1 the program is repeated.

Summary

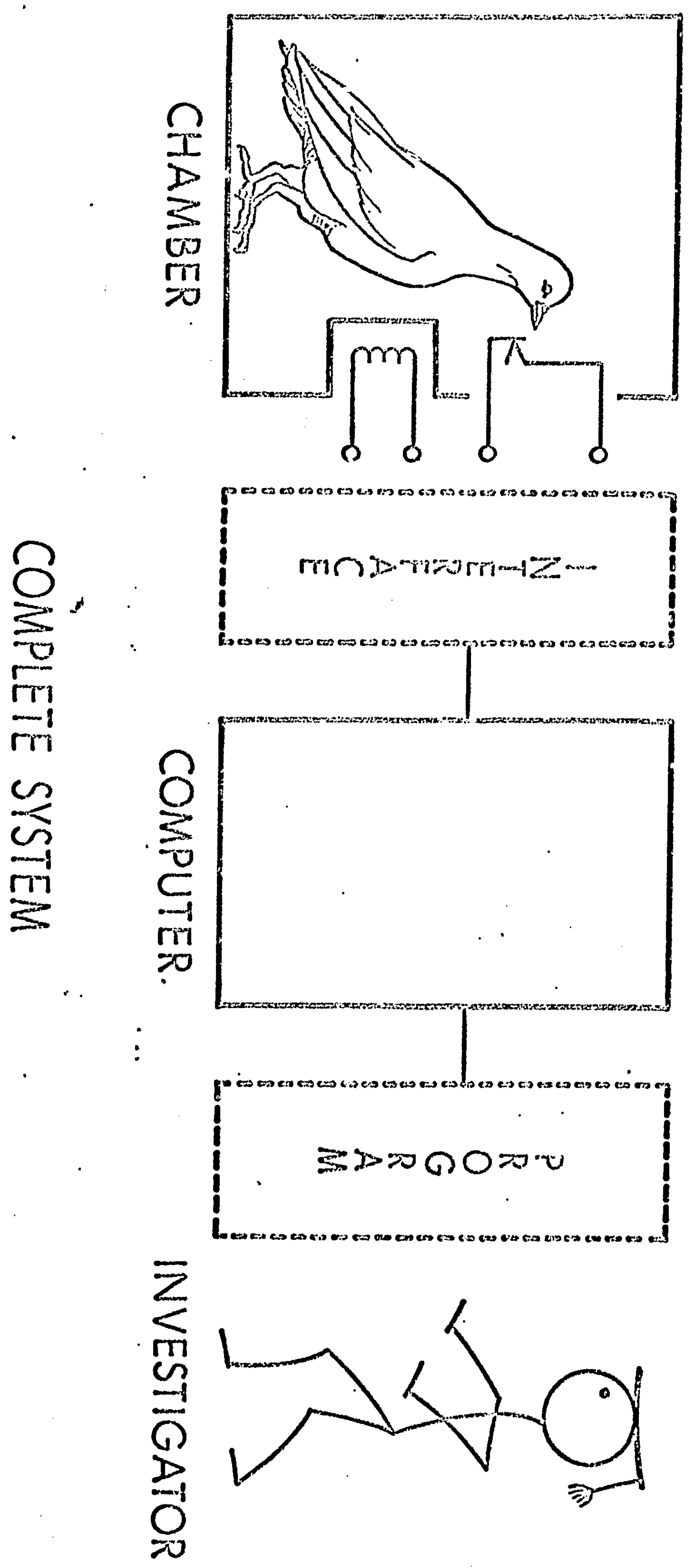
How does a computer actually go about executing a schedule of reinforcement? It does so by executing a sequence of computer operations, in conjunction with a specially-designed interface. Understanding programming and interfacing is the responsibility of the investigator.

But perhaps you see a complex technology and wonder if it is necessary for you to understand it all. To you, I would like to say that the computer in the behavioral laboratory is a research tool. And, like many other advanced research tools, it requires that the investigator be able to understand and be competent in the machine's operation. Because the computer in the behavioral laboratory logically and physically stands between the investigator and his experiment, this requirement cannot be overemphasized. It is a vital part of the success of the investigator-computer team.



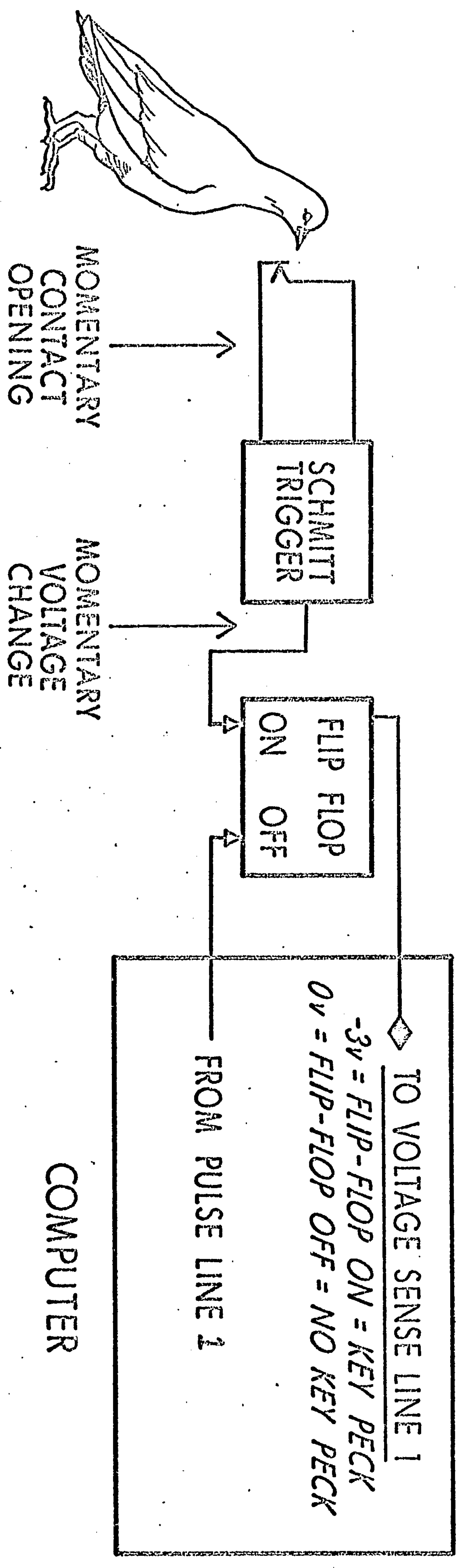
Louis Siegel - "Computer Control of Behavioral Experiments"

Slide 1



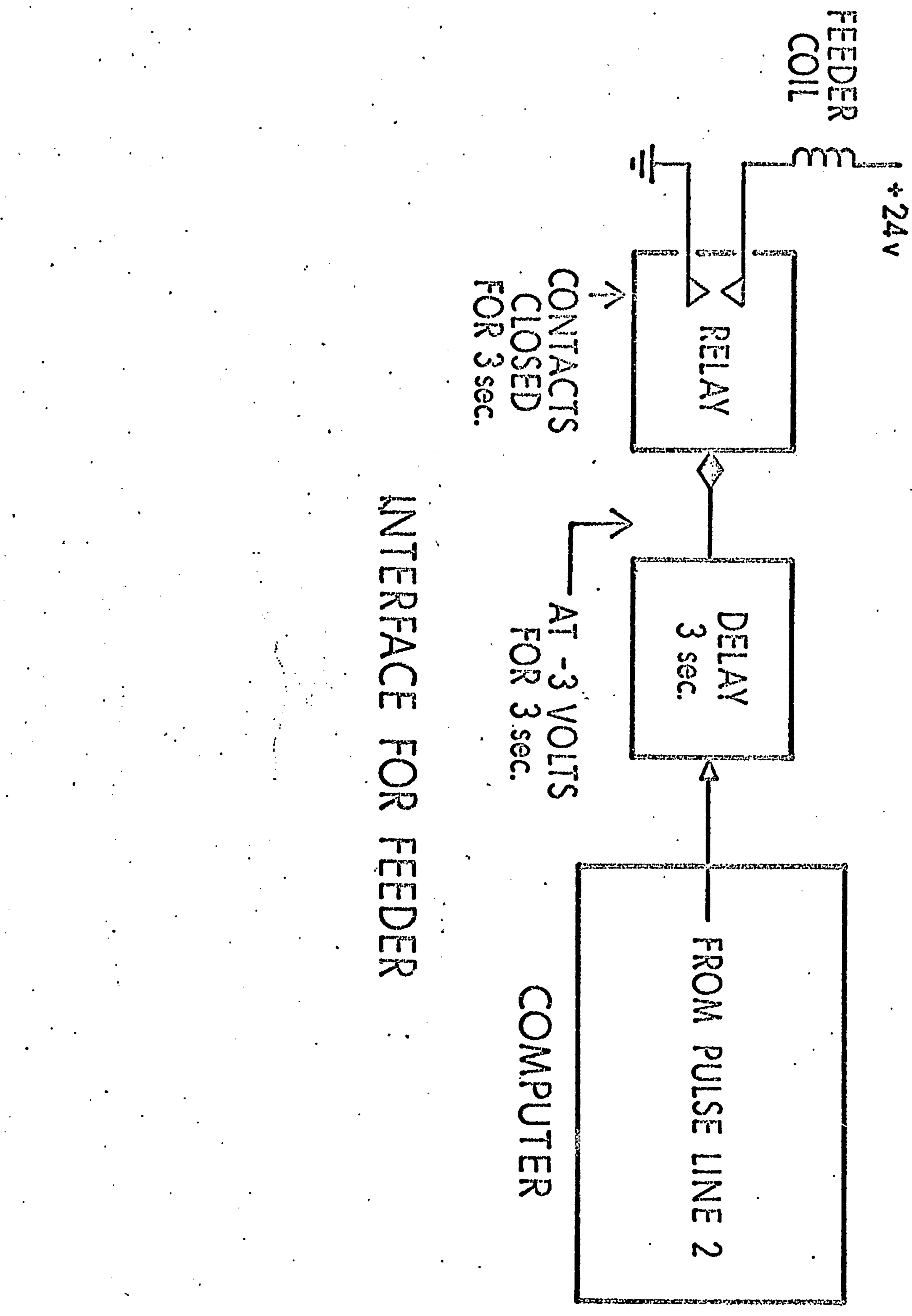
Louis Siegel - "Computer Control of Behavioral Experiments"

Slide 2



INTERFACE FOR PIGEON KEY

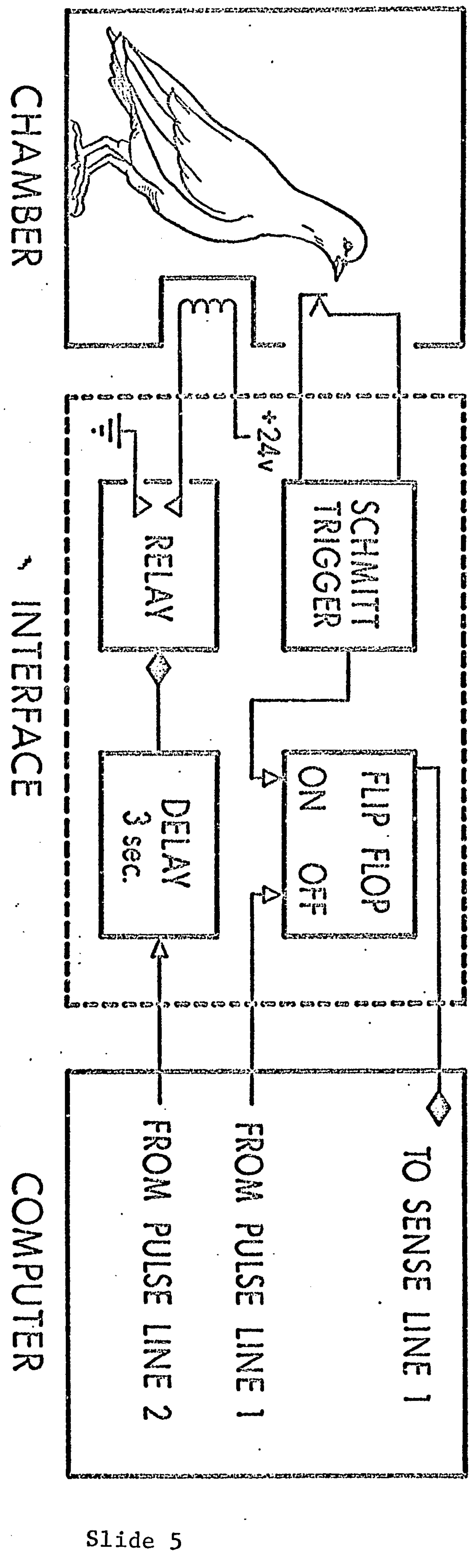
Louis Siegel - "Computer Control of Behavioral Experiments"



INTERFACE FOR FEEDER

Louis Siegel - "Computer Control of Behavioral Experiments"

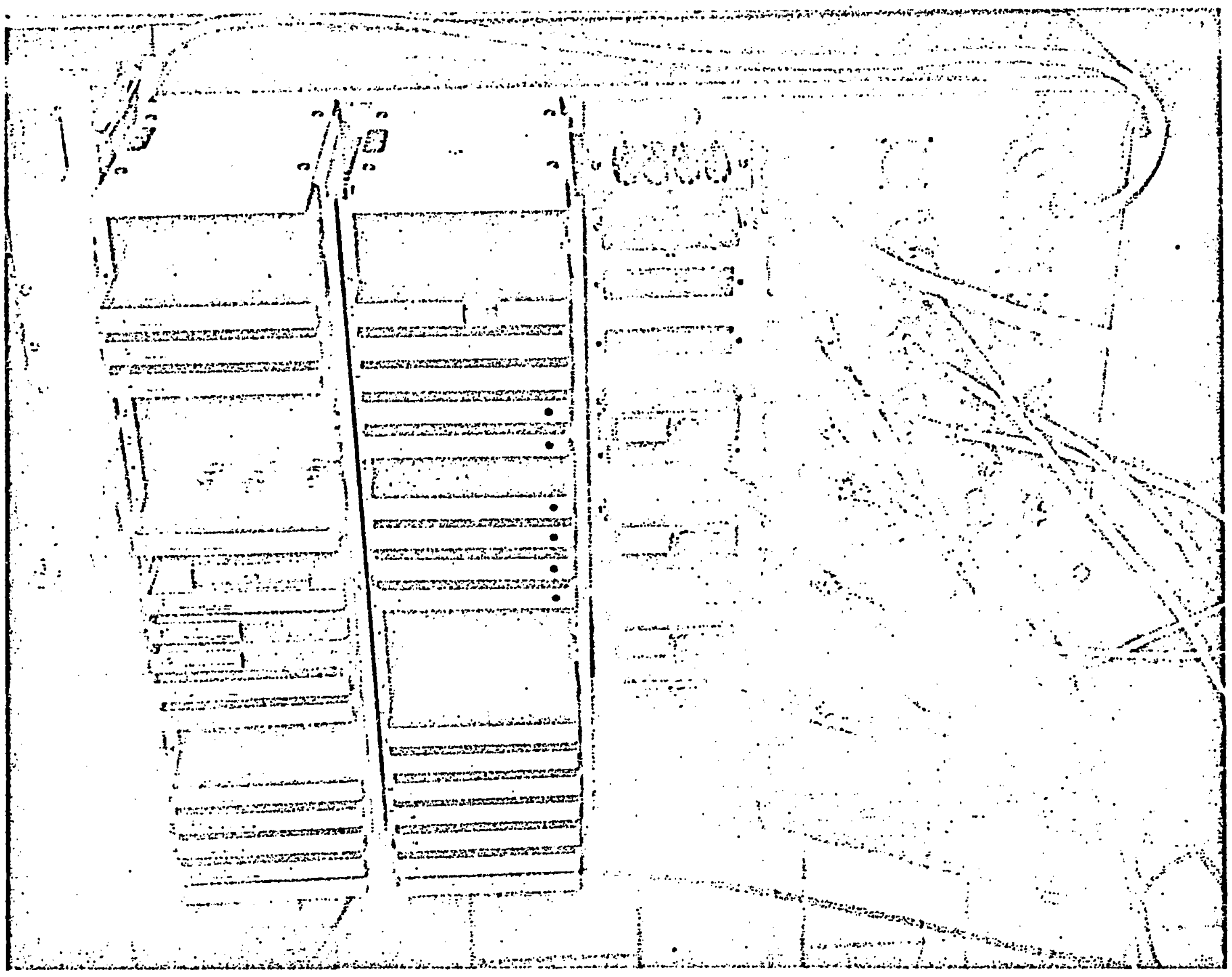
Slide 4

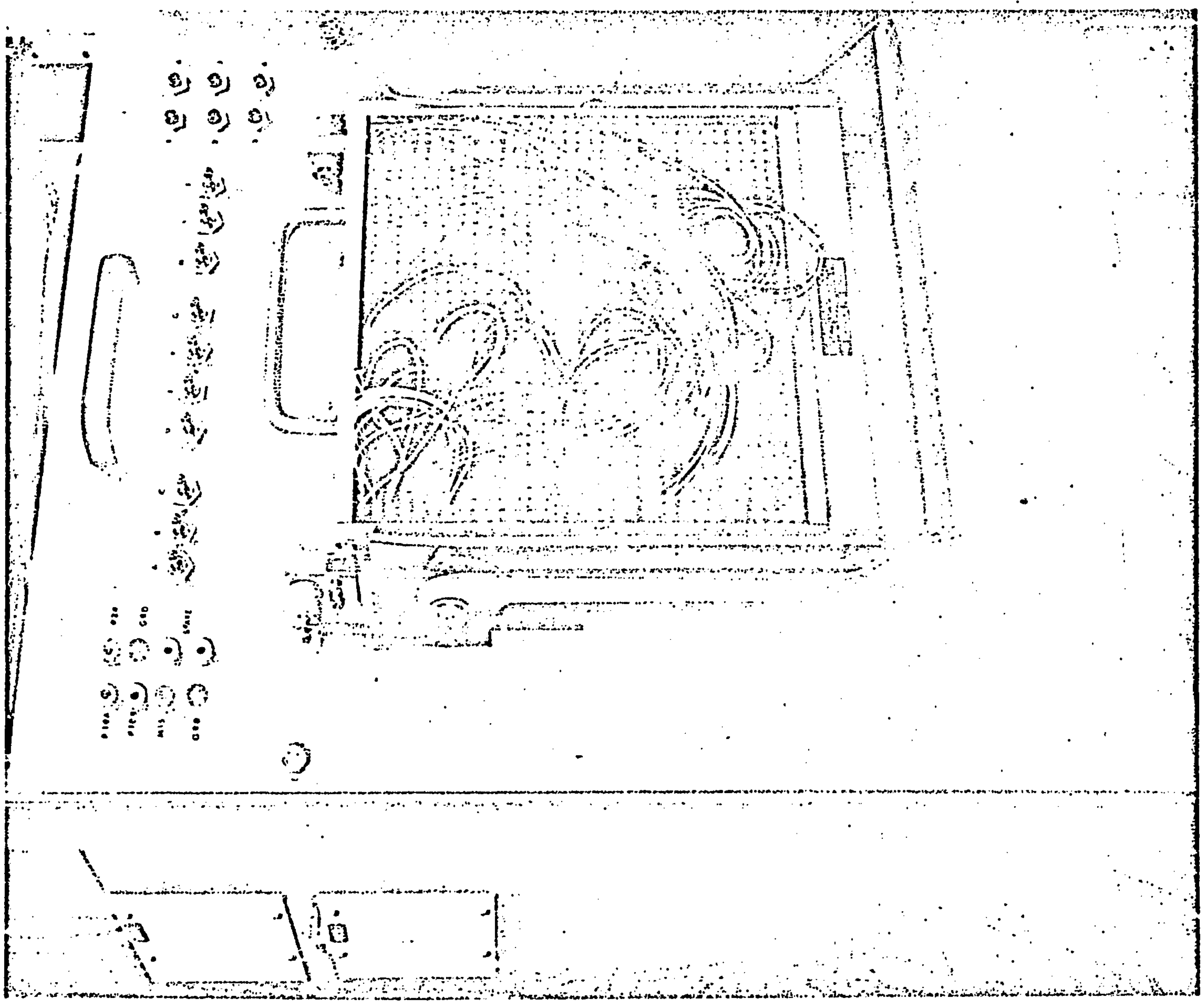


COMPLETE INTERFACE FOR FR7

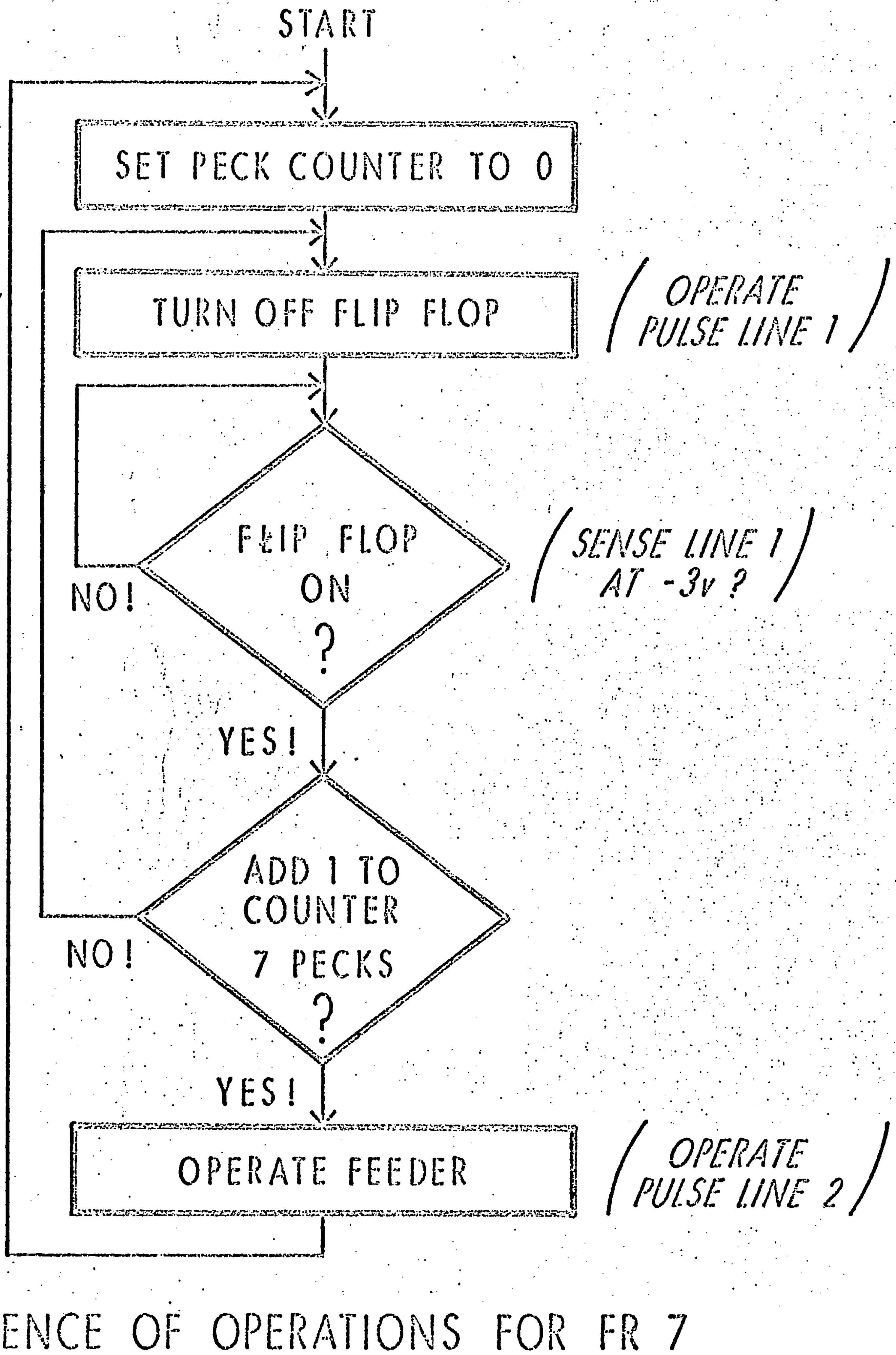
Louis Siegel - "Computer Control of Behavioral Experiments"

Slide 5





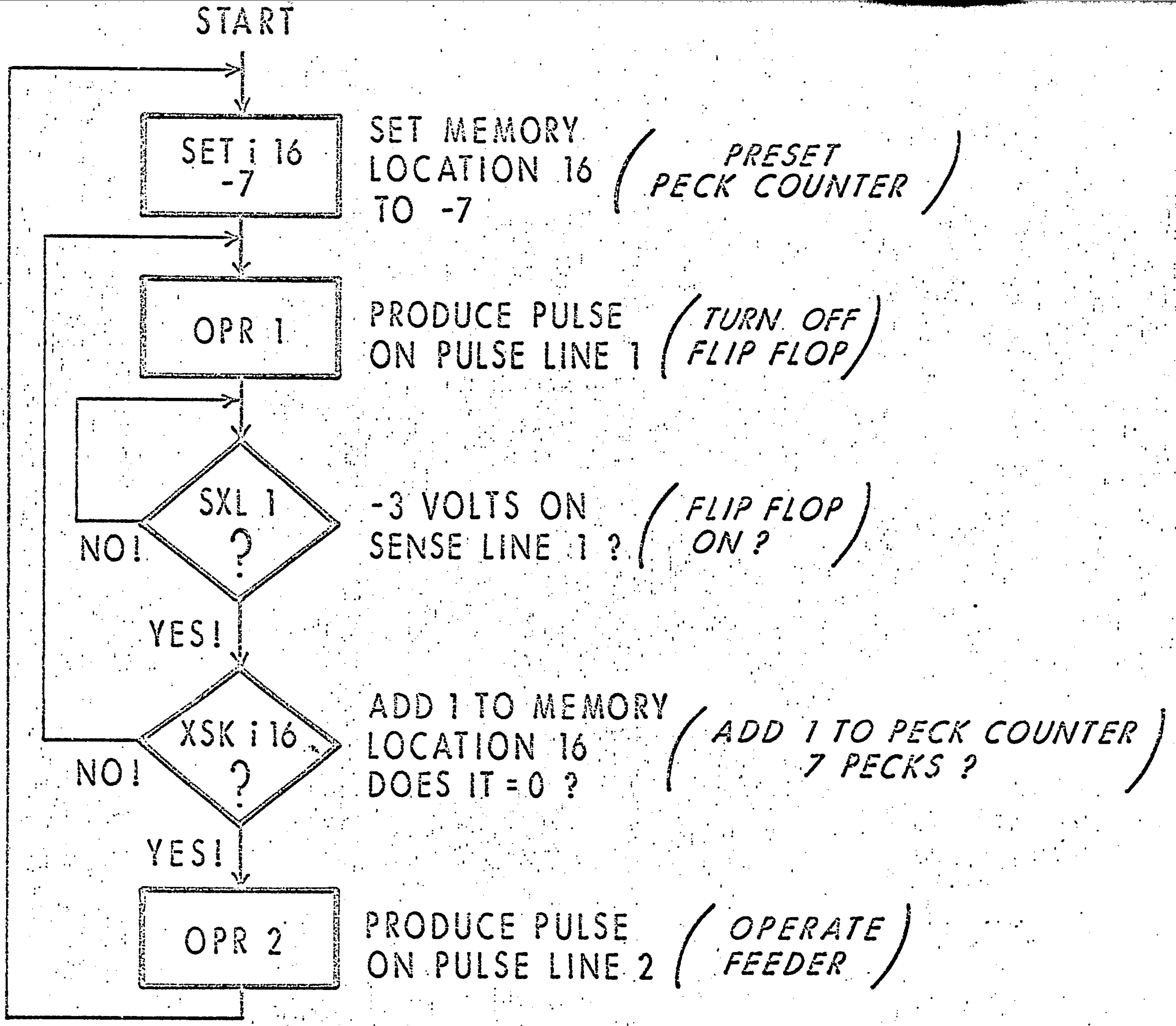
Louis Siegel - "Computer Control of Behavioral Experiments" Slide 6



SEQUENCE OF OPERATIONS FOR FR 7

Louis Siegel - "Computer Control of Behavioral Experiments"

Slide 7


START
SET i 16 -7
SET MEMORY LOCATION 16 TO -7
(PRESET PECK COUNTER)
OPR 1
PRODUCE PULSE ON PULSE LINE 1
(TURN OFF FLIP FLOP)
SXL 1 ?
NO!
-3 VOLTS ON SENSE LINE 1 ?
(FLIP FLOP ON ?)
YES!
XSK i 16 ?
NO!
ADD 1 TO MEMORY LOCATION 16 DOES IT = 0 ?
(ADD 1 TO PECK COUNTER 7 PECKS ?)
YES!
OPR 2
PRODUCE PULSE ON PULSE LINE 2
(OPERATE FEEDER)

COMPUTER INSTRUCTIONS FOR FR 7

MEMORY LOCATION	COMPUTER INSTRUCTION	EFFECT
1	SET i 16	PUTS -7 IN LOCATION 16
2	-7	
3	OPR 1	PULSE TURNS OFF FLIP FLOP
4	SXL 1	FLIP FLOP ON ?
5	JMP 4	NO ! GO BACK TO LOCATION 4
6	XSK i 16	YES ! ADD 1 TO LOCATION 16 DOES IT EQUAL 0 ?
7	JMP 3	NO ! GO BACK TO LOCATION 3
8	OPR 2	YES ! PULSE OPERATES FEEDER
9	JMP 1	GO BACK TO LOCATION 1
•		
•		
•		
16	-7	USED AS PECK COUNTER

COMPUTER PROGRAM FOR FR 7