June 12, 1959
Original Report

HG Kolsky
personal
Copy

PRODUCT PLANNING

Technical Report

DATE June 12, 1959

NUMBER P-17

TITLE The STRETCH Virtual Memory Concept and Timing Simulation Program

AUTHOR John Cocke* and Harwood G. Kolsky

LOCATION Poughkeepsie

*(Present location: IBM Research, Poughkeepsie)

This report is company confidential

TABLE OF CONTENTS

		Page
I.	Introduction	1
II.	General Description of the System	1-4
III.	Detailed Description of Virtual Memory Operation	5-24
	A. General Conditions to be Considered	5
	B. Definitions	5-9
	C. Logic of the Virtual Memory	9-24
IV.	Description of Timing Simulatinn Program	24-28
	A. General Considerations	24-25
	B. Logic of the Simulator	26-28
V.	Some Results of the Simulation Studies	29-60
	A. General Description	29-32
	B. Test Problems Used	32-33
	C. Results of Simple Parameter Studies	33-48
	D. The Effect of the Half-Microsecond Instruction Memory on STRETCH Performance	48-55
	E. A Study of Branching on Arithmetic Results in STRETCH	56-60
VI	APPENDIX: Details of Timing Simulation Program SIM-2	61-80

TABLE OF FIGURES

	Title	Page
Figure 1	Schematic of STRETCH Computer	2
Figure 2	Contents of a Virtual Memory Level	6
Figure 3	Virtual Memory Interlocks	8
Figure 4	Instruction Fetch Procedure	10
Figure 5	Indexing Procedure	11
Figure 6	Procedure for Placing Instructions into the Virtual Memory	13
Figure 6A	Logical Conditions for Bring Type Operations	14
Figure 6B	Logical Conditions for Store Type Operations	16
Figure 6C	Logical Conditions for Immediate Type Operations	18
Figure 7	Data Fetch Procedure	19
Figure 8	Data Store Procedure	20
Figure 9	Procedure for Placing Data into Virtual Memory	21
Figure 10	Procedure for Removing Instruction from Virtual Memory	23
Figure 11	SIM-2 Simplified Flow Diagram	28
Figure 12	Listing of Simulator Print-Out	30
Figure 13	Listing of Simulator Summary Print-Out	31
Figure 14	Computer Speed vs. no. Levels of Virtual Memory	36
Figure 15	Computer Speed vs. no. of Main Memory Boxes	38
Figure 16	Computer Speed vs. Indexing Arithmetic Time	39
Figure 17	Computer Speed vs. Average Arithmetic Time	40
Figure 18	Computer Speed vs. Instruction Memory Cycle Time	42
Figure 19	Arithmetic Unit Efficiency vs. Average Arithmetic Time	43
Figure 20	Computing Speed vs. I/O Word Rate	45
Figure 21	Computing Speed vs. Number of Memory Units 1	46
Figure 22	Computing Speed vs, Number of Memory Units 2	47

I. INTRODUCTION

Early in the planning of the STRETCH computer it was seen that by using the latest solid state components in sophisticated circuits that it would be possible to increase the speed of floating point arithmetic by almost two orders of magnitude over that in existing computers. However, there seemed to be no possibility of developing on the same time-scale economically feasible large memories with more than a factor of ten or perhaps twenty increase in speed. As a result, the proposed system appeared to be in danger of being seriously memory-access limited.

Moreover, as the speed of the floating point operations increases, a larger and larger percentage of the computer's time is spent on "parasitic operations", i.e., operations whose only function is program control and data selection. It was obvious that a radically new machine organization was necessary in order to capitalize upon the possibilities opened up by the high arithmetic speeds in the presence of relatively slow memories.

At this time, the possibility of a "look-ahead" device was suggested in which an independent indexing arithmetic unit would prepare the effective addresses of instructions and initiate memory references to a multiplicity of memory boxes. The data thus fetched would be held in high speed buffer registers until needed by the arithmetic unit. This device would serve two desirable purposes; (1) some of the parasitic operations would be done in parallel and thus not delay the principal calculations, and (2) several memory boxes could be running simultaneously, giving the effect of higher memory speed.

II. GENERAL DESCRIPTION OF THE SYSTEM

The major logically-independent blocks of the STRETCH computer are shown in Figure 1. Each of the units pictured may be considered as operating asynchronously. That is, each does its tasks as fast as possible independently of the others. In theory, each box could have its own clocking circuits and still operate properly. In practice, for economy's sake they are all timed by the same master oscillator, but this does not destroy their logical independence.

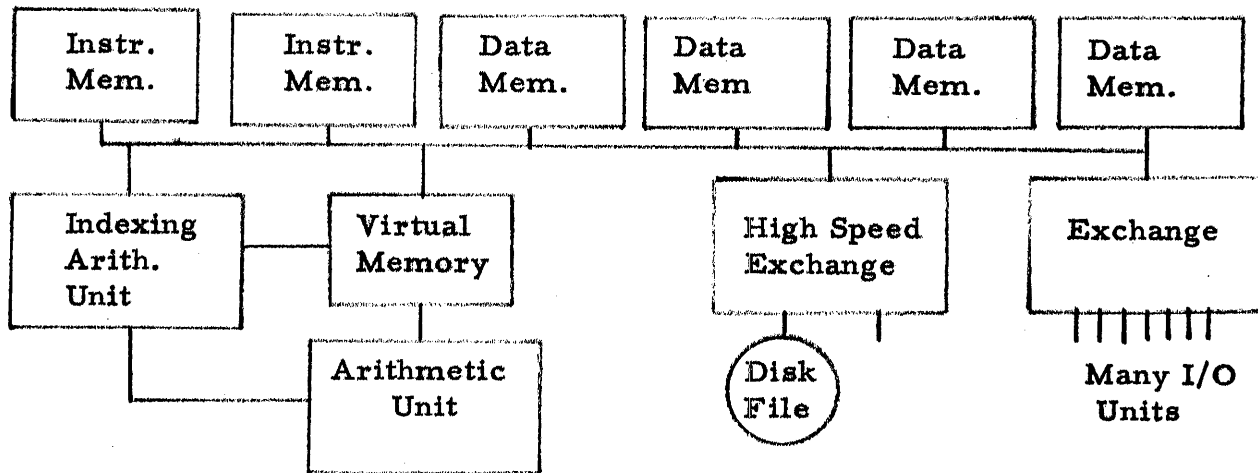
The bus control unit serves as a routing agent between the memories and the various data processing units. If two or more units make a request simultaneously the control unit assigns priorities in the following order: (1) High speed Exchange, (2) Basic Exchange, (3) Virtual Memory, and (4) Indexing Arithmetic Unit.

The Indexing Arithmetic Unit fetches instructions, performs all necessary indexing operations and sends the instructions to be executed to the Virtual Memory.

The Virtual Memory fetches and receives the data required by the instruction and holds this data until the arithmetic unit is ready for it. The Virtual Memory also performs all store operations. It holds the data generated by the Arithmetic Unit or Indexing Arithmetic Unit until the memory to which the data must be sent is available. Thus the Virtual Memory acts not only as a "look-ahead" for instructions to be fed to the arithmetic unit, but also acts as a "look-behind" storage buffer.

FIGURE 1

SCHEMATIC OF SIGMA COMPUTER



The actual design of such a "look-ahead" device posed a number of logical problems, particularly in connection with conditional branches. In colaboration with John Griffith, a device was proposed later named "virtual memory", which answered these logical problems and served as guide for the actual organization of STRETCH.

However, a machine organization of this complexity requires a detailed timing analysis in order to determine the value of adding hardware in the form of the "virtual memory". This is especially true since the sole function of the "virtual memory" is to increase machine speed, by increasing the efficiency of other devices. It was also felt that the timing analysis could not be made on the basis of a few trivial examples (e.g. matrix multiply). Machine performance obtained in this fashion can be extremely deceptive. Since a detailed timing analysis of a computer of this complexity is extremely tedious to carry out by hand, it became clear that if the job were to be done, it would be necessary to simulate the proposed machine on another computer. This prompted us to write the simulation program described below.

With the above general organization in mind, let us discuss some of the logical problems posed by such a system. The first problem is a result of the very concept which enables us to obtain such great benefits from the stored program computer--the ability to treat instructions as data. In a system such as we have proposed there is a large amount of simultaneous operation. For example, the indexing arithmetic unit may be busy preparing an instruction before previous instructions have been completed or even started by the arithmetic unit. One of these previous instructions may modify the instruction which is presently being indexed. The virtual memory must recognize this situation and allow the intervening instructions to be completed before doing the modified instruction.

A similar problem exists with respect to ordinary data. In order to operate several memories simultaneously, it is necessary to start obtaining data from these memories before the preceding operations have been completed. Yet, one of these operations may be a store into one of the data locations. The virtual memory must make provisions to insure that each instruction obtains the most up-to-date data as implied by the order of the program.

One of the novel features of the STRETCH computer is its elaborate interrupt system. Under this system whenever some unexpected occurrence arises, the program will be interrupted and control will pass to a special routine which is designed to take care of the case in question, then return control to the original program. In this situation the virtual memory must have provisions to retain enough information so that when an interrupt occurs we can resume the computation exactly where we left off. It must be able to recognize which of the changes that have been made in advance are not desired and should be obliterated and which are exact solutions that must be restored.

Another special case arises when a conditional branch on arithmetic results occurs. Here we will not know which of the two branches we should have taken until the preceeding instruction is executed. In the case the wrong path has been selected, the virtual memory must be prepared to drop the intermediate results which have been computed and pick up the correct branch in a way very similar to that of an interrupt.

Summing up all these logical problems, we may state that the fundamental rule for the Virtual Memory is that it must make the asynchronous and non-sequential computer give results identical to those which would be obtained by performing the program one instruction at a time in the order in which they are written.

Since our original work on the virtual memory and simulation in 1957-58, a large number of detailed changes have been made in the actual hardware design of STRETCH. These necessitated several modifications in the Simulation program to estimate their effect on the overall system performance. In this report we are omitting many of these changes for expository reasons since our purpose is to describe the virtual memory and timing simulation concepts not to describe the STRETCH hardware exactly. The result is that the system described below imbodies a more general system than that found in the Simulator which in turn is more general than that found in the actual computer.

III. DETAILED DESCRIPTION OF VIRTUAL MEMORY OPERATION

A. General Conditions to be Considered

The conditions which occur in the following situations must be considered in some detail:

1. The fetching of instructions by the Indexing Arithmetic Unit.
2. The indexing of instructions and modification of Index registers.
3. The loading of the Virtual Memory and the setting of its conditions by the IAU.
4. The action of the Virtual Memory in fetching data.
5. The action of the Virtual Memory in storing data.
6. The communication between the Virtual Memory and the main arithmetic unit.
7. Special situations such as conditional branching on arithmetic results, etc.

B. Definitions

Some of the terms we will use are defined as follows:

1. Operations

Operations are considered to be of three types:

(1) Bring or fetch type - All instructions requiring data to be transmitted from external memory to the Virtual Memory

(2) Store Type - Instructions requiring the transmission of data from the Virtual Memory to external memory or index memory.

(Note: We consider all indexing instructions to be of the Store Type, although the store may be to either external memory or index memory.)

(3) Immediate Type - All operations not requiring data transmission.

2. Virtual Memory Quantities

(1) Virtual Memory - A number of Virtual Memory (or look-ahead) levels (numbered 0 to N-1).

(2) Level of Virtual Memory - A collection of registers and control bits. The contents of the jth level is shown in Figure 2.

I_j	OP_j	S_j Bit	B_j Bit	F_j Bit	FA_j	OK_j Bit	C_j Bit	D_j	DA_j

Figure 2 Contents of a Virtual Memory Level

(3) Instruction Address Register (I_j) - Contains the address of the instruction currently in the j th level.

(4) Operation Code Register (OP_j) - Contains the operation to be performed by the arithmetic unit.

(5) Store Bit - (S_j) - A one bit trigger which indicates the level contains a Store type instruction.

(6) Bring Bit - (B_j) - A one bit trigger which indicates the level contains a fetch type instruction for which the data access has not been started.

(7) Forwarding Bit (F_j) - A one bit trigger which indicates that the j th level must transmit data to another level.

(8) Forwarding Address (FA_j) - A register which contains the number of the level to which the data must be sent if F_j is set.

(9) O.K. Bit (OK_j) - A trigger which when set indicates that the correct data for the instruction to be executed is present in the j th Data Field.

(10) Data Field (D_j) - A register which contains the operand data for the instruction.

(11) Data Address (DA_j) - The operand data address (already indexed by the IAU) for D_j.

(12) Compare Bit (C_j) - A trigger which if not set indicates the address in DA_j should not be included in any address comparisons being made.

3. Counters

The Virtual Memory is controlled by a set of counters which count mod (N), where N is the number of Virtual Memory levels.

(1) Counter one (C_1) - Indicates the level into which the next instruction may be placed.

(2) Counter two (C_2) - Indicates the level from which the next bring type instruction may be initiated.

(3) Counter three (C_3) - Indicates the level from which the next store type instruction may be initiated.

(4) Counter four (C_4) - Indicates the level from which the arithmetic unit will get its next operation and data.

4. Interlocks

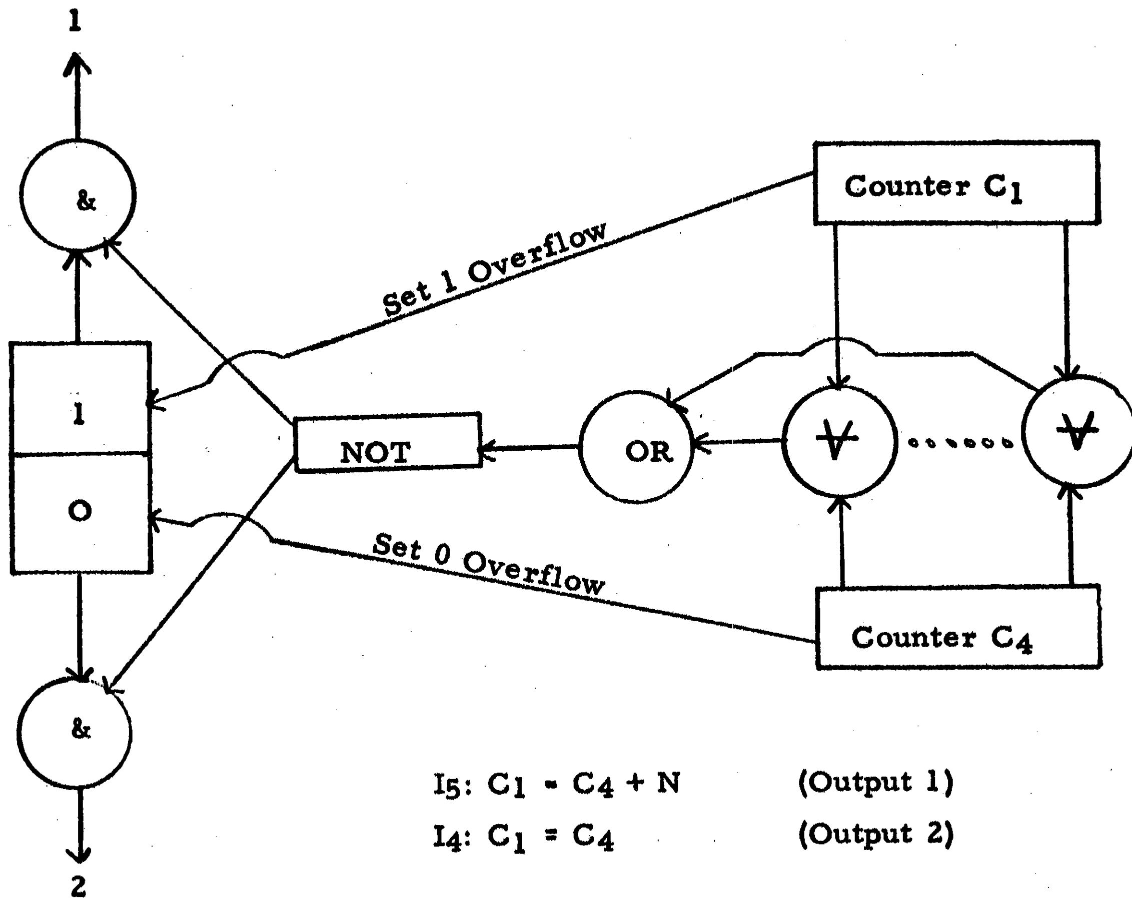
The above counters must be interlocked in the following manner to assure proper sequential operation of the computer (see figure 3):

(1) Interlock one (I1): $C_1 = C_3 + N$ Prevents the IAU from placing the next operation into the level indicated by C_1 because an unexecuted store is still in the level.

(2) Interlock two (I_2): $C_1 = C_3$ Prevents a store from being initiated from the level indicated by C_3 because the store has already been done.

(3) Interlock three (I3): $C_1 = C_2$ Similar to I_2, prevents a fetch from being initiated.

(4) Interlock four(I_4): $C_1 = C_4$ Prevents the arithmetic unit from executing an old instruction.



Interlocks I_4 and I_5 are as shown, the other interlocks are done in a similar manner.

Figure 3. Virtual Memory Interlocks

(5) Interlock five (I_5): $C_1 = C_4 + N$ Prevents the IAU from placing the next instruction into the level indicated by C_1 because the instruction there has not been executed yet.

C. Logic of the Virtual Memory

1. General

There are two basic precepts which must be kept in mind to understand the operation of the Virtual Memory:

(1) The OK bit (O_j) being set in the j th level indicates that the contents of D_j is the correct data called for by DA_j. All operations will be performed only under this condition and logical decisions will be made in such a manner as to make sure this is the case.

(2) Addresses can be compared by the IAU with every DA_j address simultaneously. DA_j is not used for any level which does not have its C_j bit set. If a comparison exists between a new DA_j being placed in the Virtual Memory and an old DA_k, the compare bit C_k is turned off and the address of level j is placed in FA_k. This insures a unique meaning for the comparison. If this were not done, another instruction address DA_e might compare against <u>two</u> levels and thus cause an ambiguity.

2. Instruction Fetch Logic

Figure 4 is a flow diagram of the IAU Instruction Fetch Procedure. The logic is as follows: If the IAU is ready to fetch another instruction, it compares the instruction address with all the DA_j's of Virtual Memory. If there is no comparison, the instruction fetch is initiated. If there is a comparison the IAU must take its instruction from the Virtual Memory provided the OK bit is set, otherwise, it must wait until the OK bit is set.

Note: This procedure prevents the logical difficulty mentioned earlier which would occur if the Virtual Memory contained a store order into the instruction presently being fetched.

For Example:

```
a    STORE Address a+2
a+1  LOAD  M, i
a+2  ADD N, i
a+3  ----
```

The store to a + 2 must be done in sequence or the old value N would be used for the address instead of the quantity being set by a.

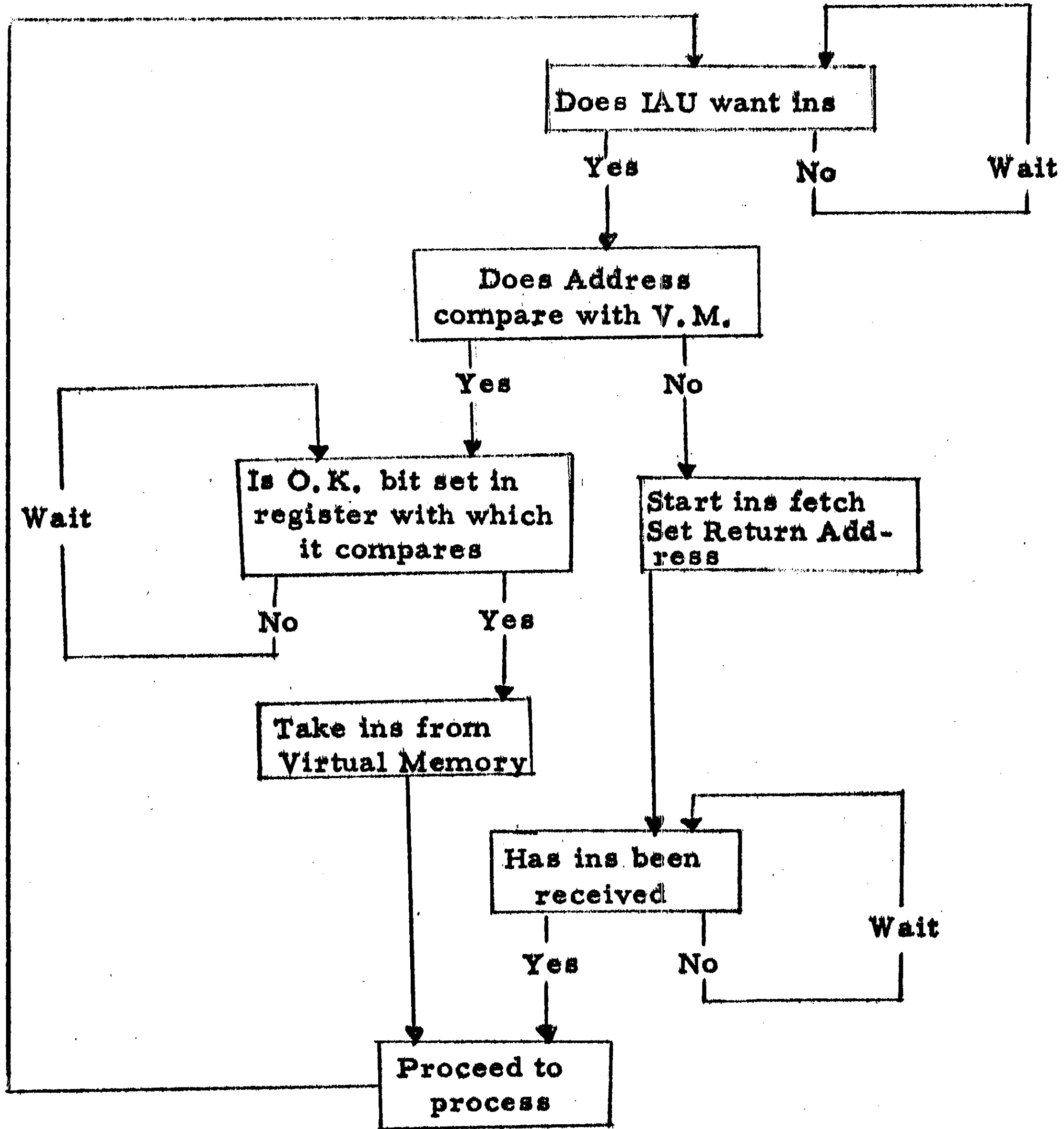


Figure 4, Instruction Fetch Procedure

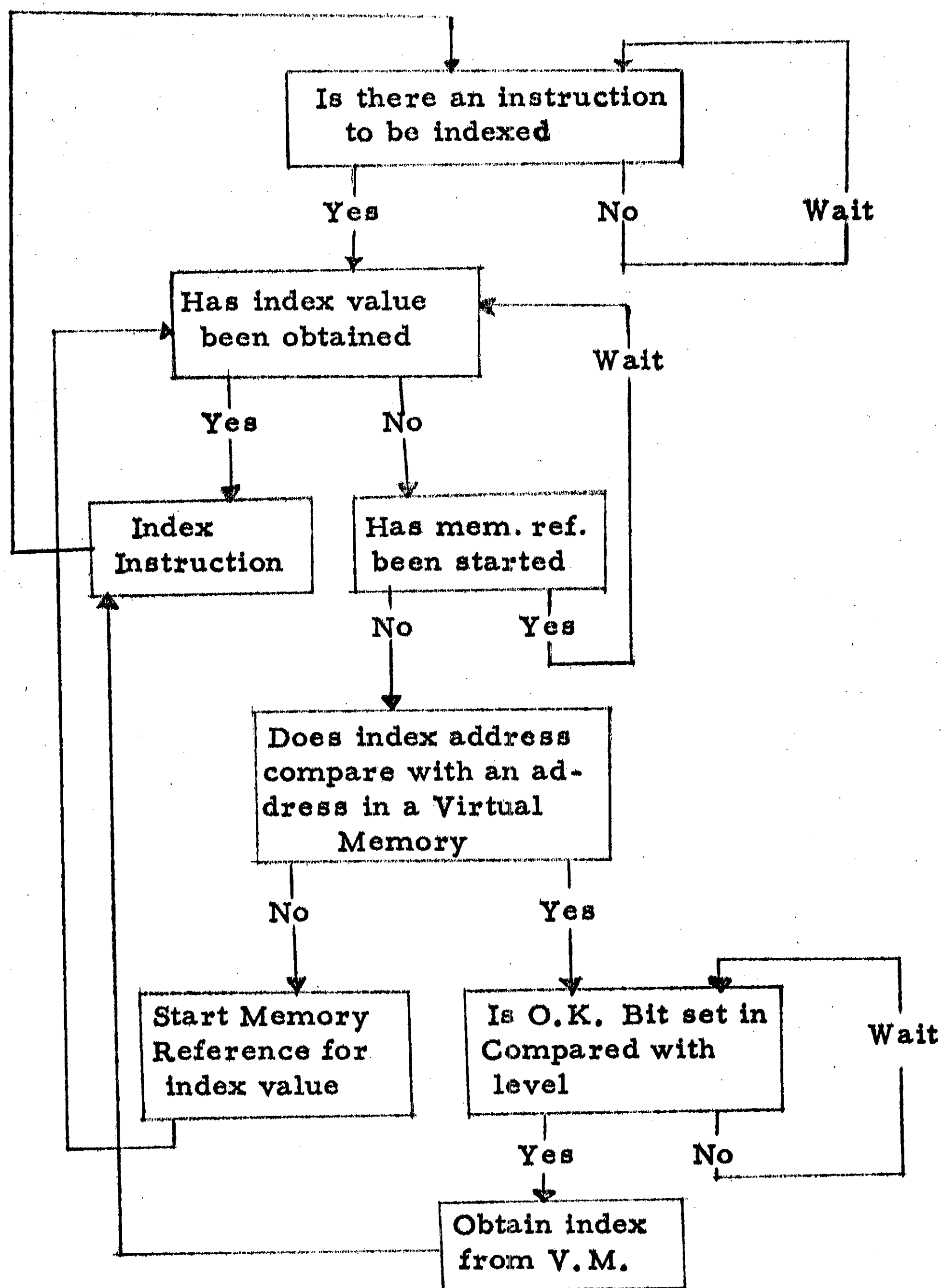


Figure 5, Indexing Procedure

3. Indexing Logic

Figure 5 shows the flow for instruction indexing. After determining that an instruction is ready to be indexed, the IAU tests whether or not the index value is available. If it is, the indexing operation is started, if not the memory reference is started and the IAU waits until the data returns before proceeding. If the index-fetch has not been started, the IAU compares the index address against all the data addresses in Virtual Memory. If none compare, the index value is fetched normally. If one does compare, the index fetch is held up until the OK bit is set for the data. This value from the Virtual Memory is then used for indexing the instruction.

4. Logic of Putting Instructions in the Virtual Memory

(1) Figures 6, 6A, 6B, 6C represent the logical flow for putting instructions into the Virtual Memory. If the indexing arithmetic unit has an instruction prepared for the Virtual Memory, it may transmit the instruction into the Virtual Memory if interlocks one and five do not forbid it. These interlocks prohibit a new instruction from destroying an old one which has not been executed as yet, whether an arithmetic operation (I_5) or an unexecuted store (I_1). The handling of the instructions vary depending on whether they are of the bring type, store type, or immediate type.

(2) The bring type, as described in Figure 6A, proceeds as follows: If the effective data address of the instruction compares with the DA address in some level, the instruction, its op code, and effective data address are loaded into the level marked by C_1. The compare bit for level C_1 is set to one while the compare bit for the compared-with level is set to zero. If the O.K. bit in this compared-with level is set, meaning that the data located there is correct, the data is transmitted directly to the C_1 level and its O.K. bit is also set. If the O.K. bit is not set, we must tag the compared-with level by setting its Forwarding bit and by putting the value of C_1 into its Forwarding address, the bring bit for level C_1 is also set to zero since no further data fetch is required.

If the effective data address does not compare with any Virtual Memory level, the instruction is put directly into level C_1, its O.K. bit is set to zero, and its bring bit is set to one, indicating that a fetch must be started.

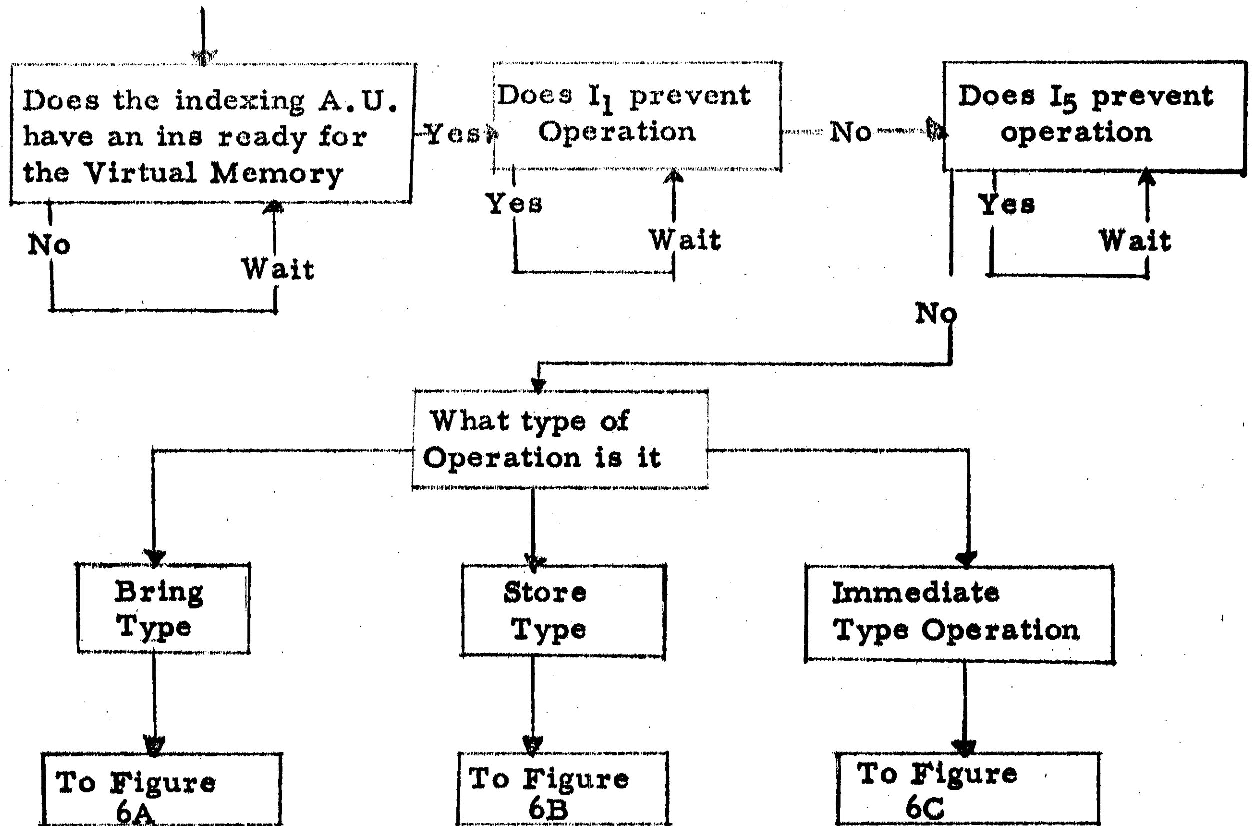


Figure 6, Procedure for placing Instructions into the Virtual Memory

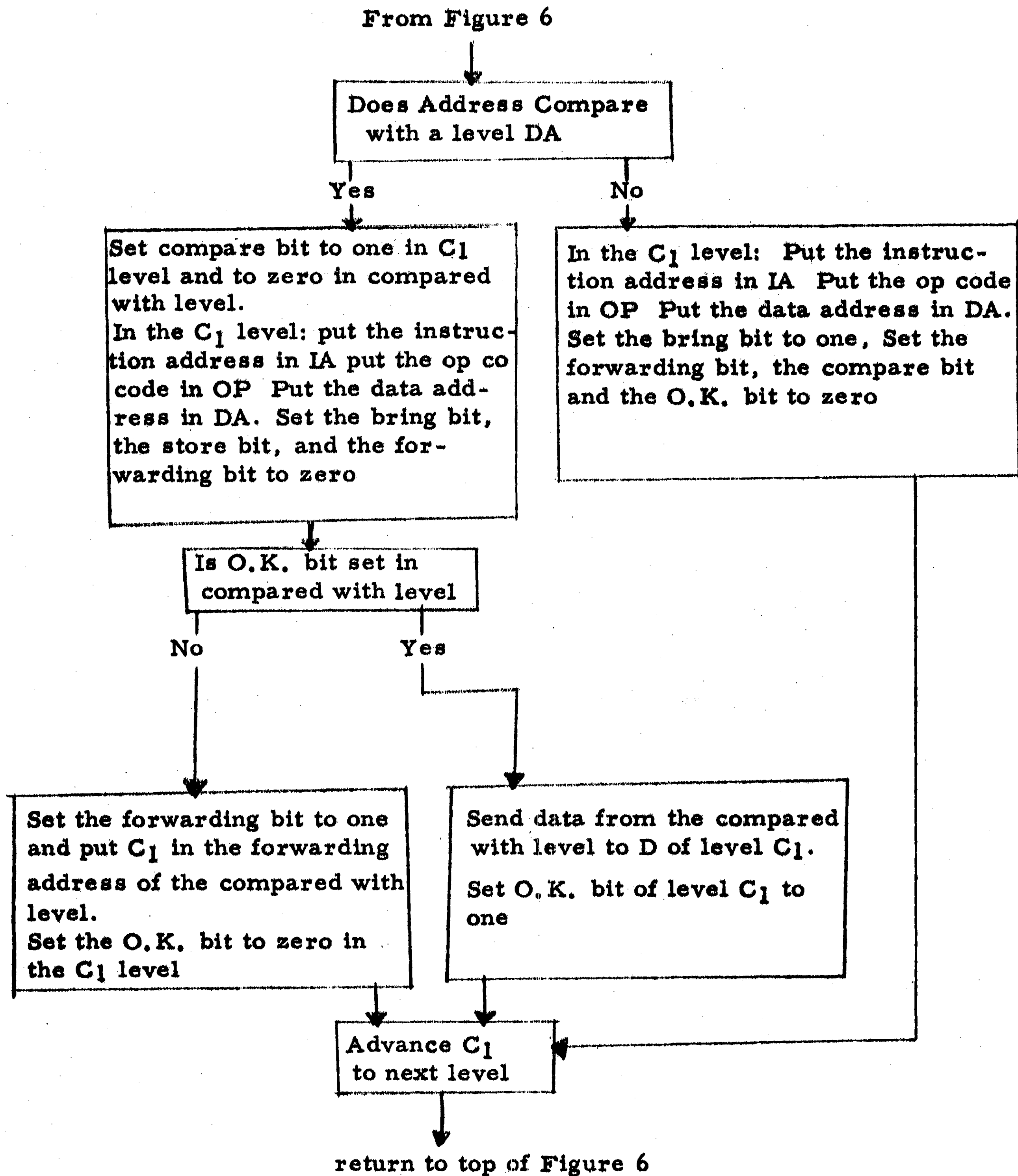


Figure 6A, Logical Conditions for Bring Type Operations

(3) Figure 6B shows the Store type procedure. If the effective address of the instruction does not compare with the DA address in some level, the instruction is placed into the level marked by C_1. The store bit is set to one indicating that a store will be required. The level's bring bit and forwarding bit are set to zero, its compare bit is set to one. If on the other hand the addresses do compare, the same procedure is followed but in addition, the compare bit in the level compared-with is set to zero so that future comparisons will not use it.

The OK bit has not yet been set. It is set to one if the operation is an index store and set to zero if it is an ordinary store. For the ordinary store it is clear that the OK bit should be zero since the data must come from the arithmetic unit after the preceeding instruction is executed.

As was mentioned in the definition on page 5, we treat all indexing instructions as store type and place the new value of the indexed quantitiy into the Virtual Memory. This is done because the Indexing Arithmetic Unit is going ahead of the normal order of instruction execution and an interruption may occur before this indexing instruction should have been done. In this case, the old value of the index is still in the index register. On the other hand the Indexing Arithmetic Unit compares with the Virtual Memory and extracts the most recent value of the index for indexing succeeding instructions. The OK bit is set to one since the appropriate data is in the above level. Both the new and old index values must be carried along to give logically correct conditions in the case of an interrupt.

A situation very similar to interrupt occurs in branches on arithmetic results where the Indexing Arithmetic Unit "guesses" which branch will be taken and proceeds with fetching and processing the instructions on this branch subject to being wiped out if the guess proved to be wrong. (See the discussion on "Wrong way Branches" below.)

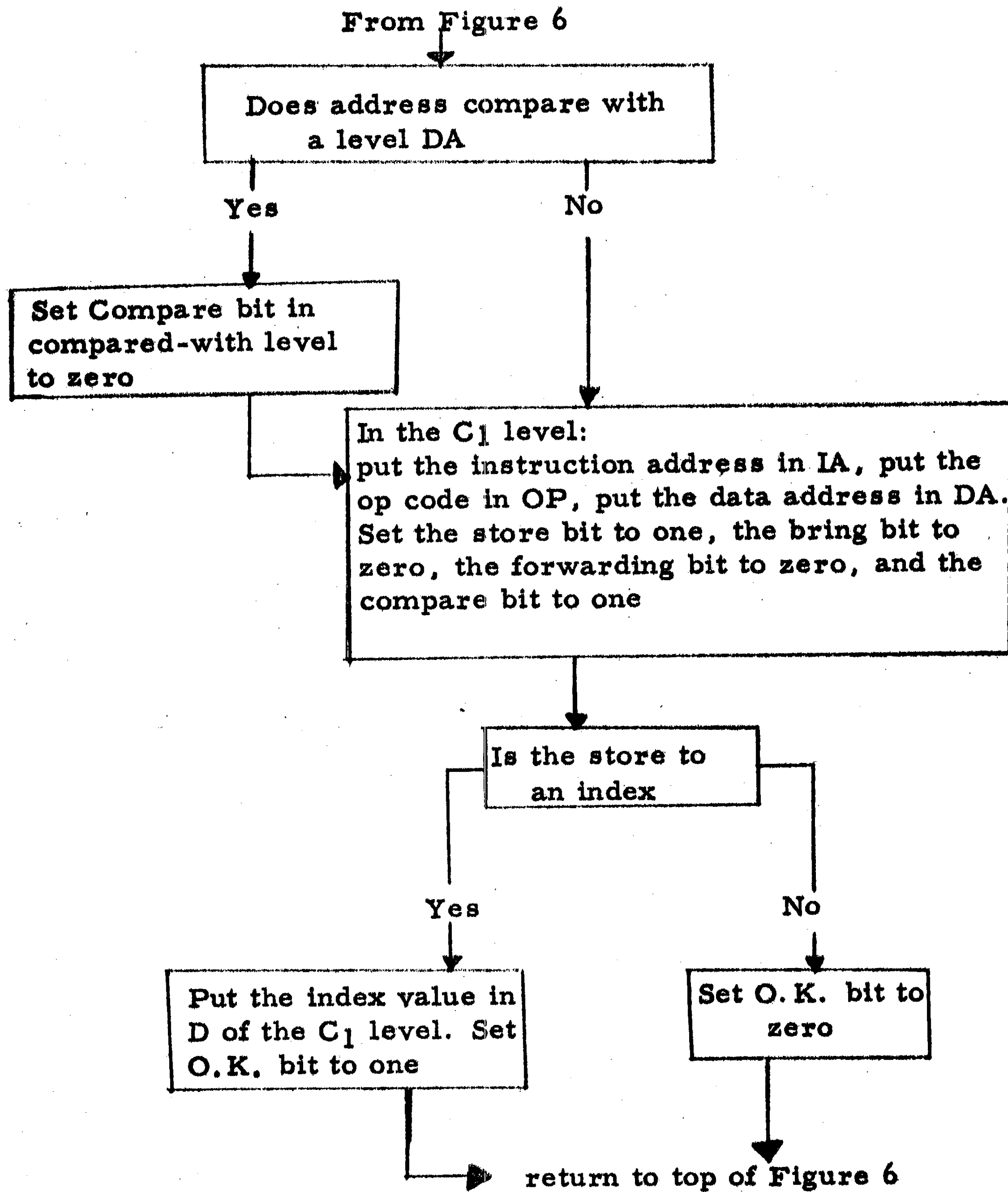


Figure 6B, Logical Conditions for Store Type Operations

(4) Immediate Type instructions are the simplest type because they essentially carry their data with them. Figure 6C shows the logic in this case. The instruction is placed in the Virtual Memory level marked by C_1. The address field of the instruction is placed in the data field of C_1. The OK bit is set to one indicating the data is present. The bring and store bits are both set to zero. The compare bit is set to zero since the DA address field has no meaning for immediate type ops. (The data address of the last instruction which occupied this level still remains in DA so it has no relation to the present D field.)

5. Logic of Data Fetching

See Figure 7: When an instruction of the bring type has been placed in the Virtual Memory, the data required by the instruction in general will not be present (unless a comparison exists as was described above) and thus the data must be obtained from core storage. The fetch cannot be started if interlock I_3 holds which means all the fetches corresponding to the instructions presently in the Virtual Memory have been started. If a fetch is possible, the bring bit at level C_2 indicates whether or not a fetch is necessary. If necessary the fetch may be started if the memory bus and memory unit corresponding to the data address are not already being used. When the fetch is started, the bring bit for level C_2 is set to zero. The counter C_2 is then stepped forward to the next level.

6. Logic of Data Storing

Figure 8 shows the Data Store logic, which is very similar to that for data fetching just described. The only significant difference is that the O.K. bit must be set before the operation can be started.

7. Logic for Placing Data into the Virtual Memory

In Figure 9, we see the logical conditions which must be satisfied by the data returning from Memory addressed to the Virtual Memory. The return address which was supplied when the fetch was started selects the level into which the data will be placed. The O.K. bit is then set to one indicating that the proper data is in the level. The operation is complete at this point unless

From Figure 6

In the C_1 level:
Put the instruction address in IA, put the op code in OP, Put the data address into D (Note this) Set O.K. bit to one Set forwarding bit, the bring bit and and store bit to zero. Set the compare bit to zero (Note)

return to top of Figure 6

Figure 6C, Logical Conditions for Immediate Type Operations

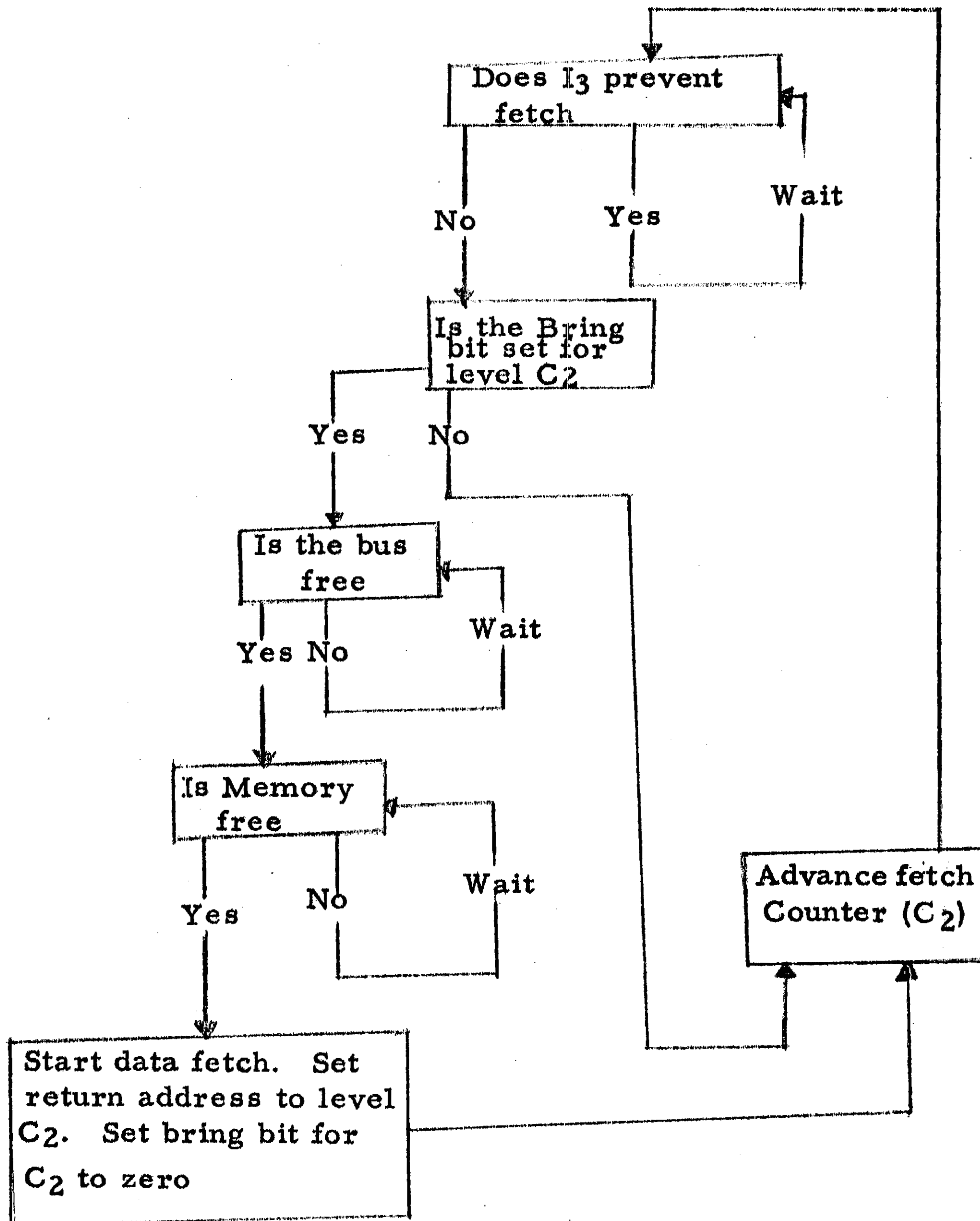


Figure 7, Data Fetch Procedure

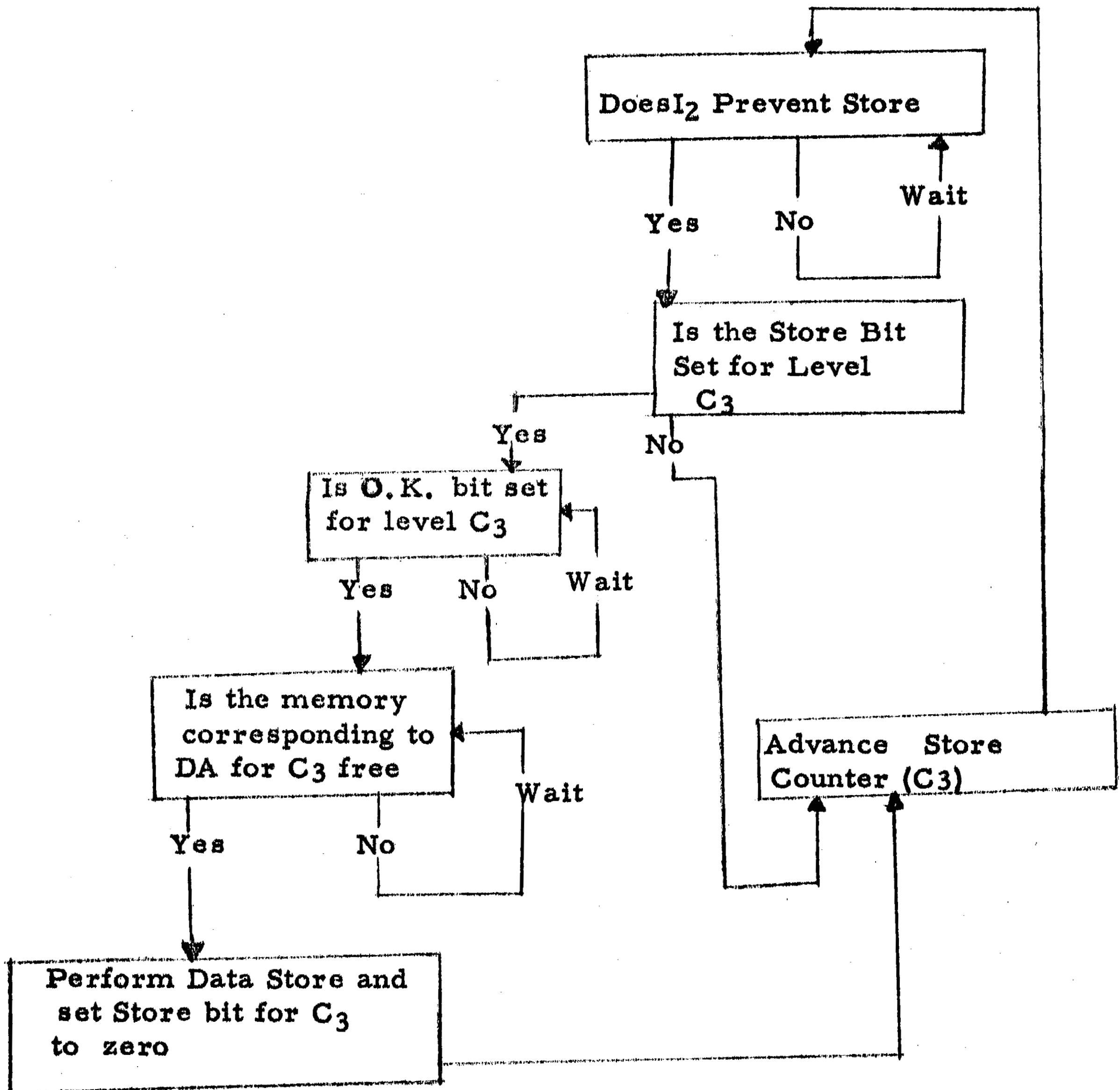


Figure 8, Data Store Procedure

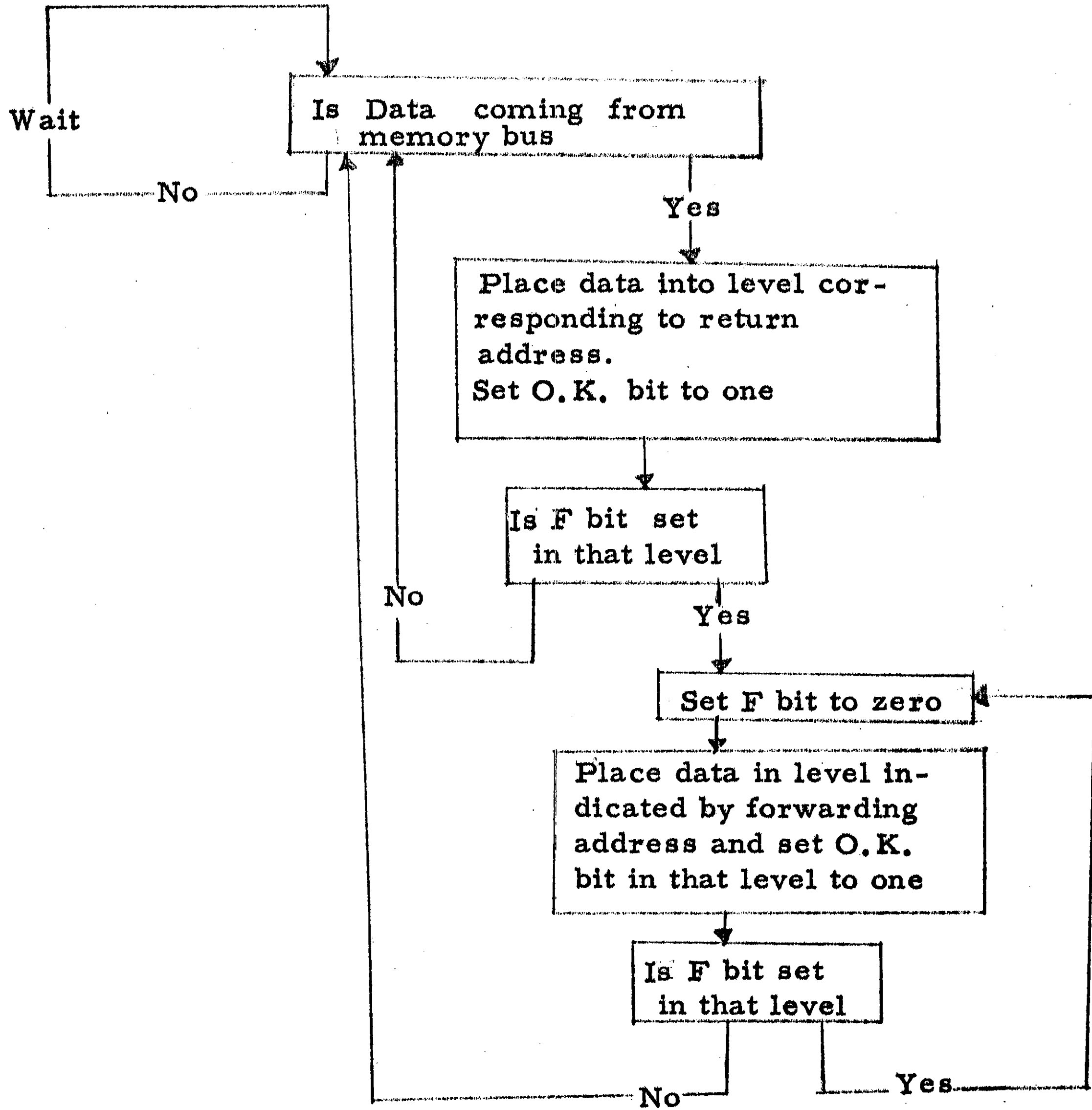


Figure 9, Procedure for Placing Data into Virtual Memory

the Forwarding bit is set. In this case, the data must be forwarded to the level designated by the Forwarding address. This procedure continues from level to level as long as the data continues to arrive into a level whose Forwarding bit is set. This procedure automatically supplies all operands present having identical data addresses with the proper data without additional memory references.

8. Logic of Removing Instructions from the Virtual Memory

Observing Figure 10, we notice that as the arithmetic unit completes an instruction it checks to see if the next instruction in the Virtual Memory is ready to be executed (indicated by interlock I_4). Note: The operation may be an unconditional branch, a conditional branch, or an index type store as well as a normal bring or store type instruction involving the accumulator. Figure 10 shows only the cases which involve the universal accumulator. The index and unconditional branches and the index store operations are merely ignored at this point. They are carried along only to provide the data for recovery in the event an interrupt occurs. The execution of the conditional branches on arithmetic results are described in the next section.

If the next instruction marked by counter C_4 is ready, it is fed into the arithmetic unit. If it is a store type, the data is gated from the accumulator into the data field of level C_4, and the OK bit is set to one. If the Forwarding bit of the level is set, a forwarding procedure in this case is <u>essential</u> for the proper logical operation of the computer, whereas in the bring case it is a time-saver only.

If the instruction is not a store type, the arithmetic unit must hold up until the O.K. bit for the level is set. When the O.K. bit is set, the instruction is gated into the arithmetic unit and executed.

9. Logic of Interrupt Procedure

If for any cause an interrupt (or trap) from a special condition occurs, the instruction which is being executed in the arithmetic unit is completed. However, the next instruction is not executed in spite of the fact all the data preparation for it may have been completed. The address in the IA (instruction address) field will serve as the value to reset the instruction counter if it is desired.

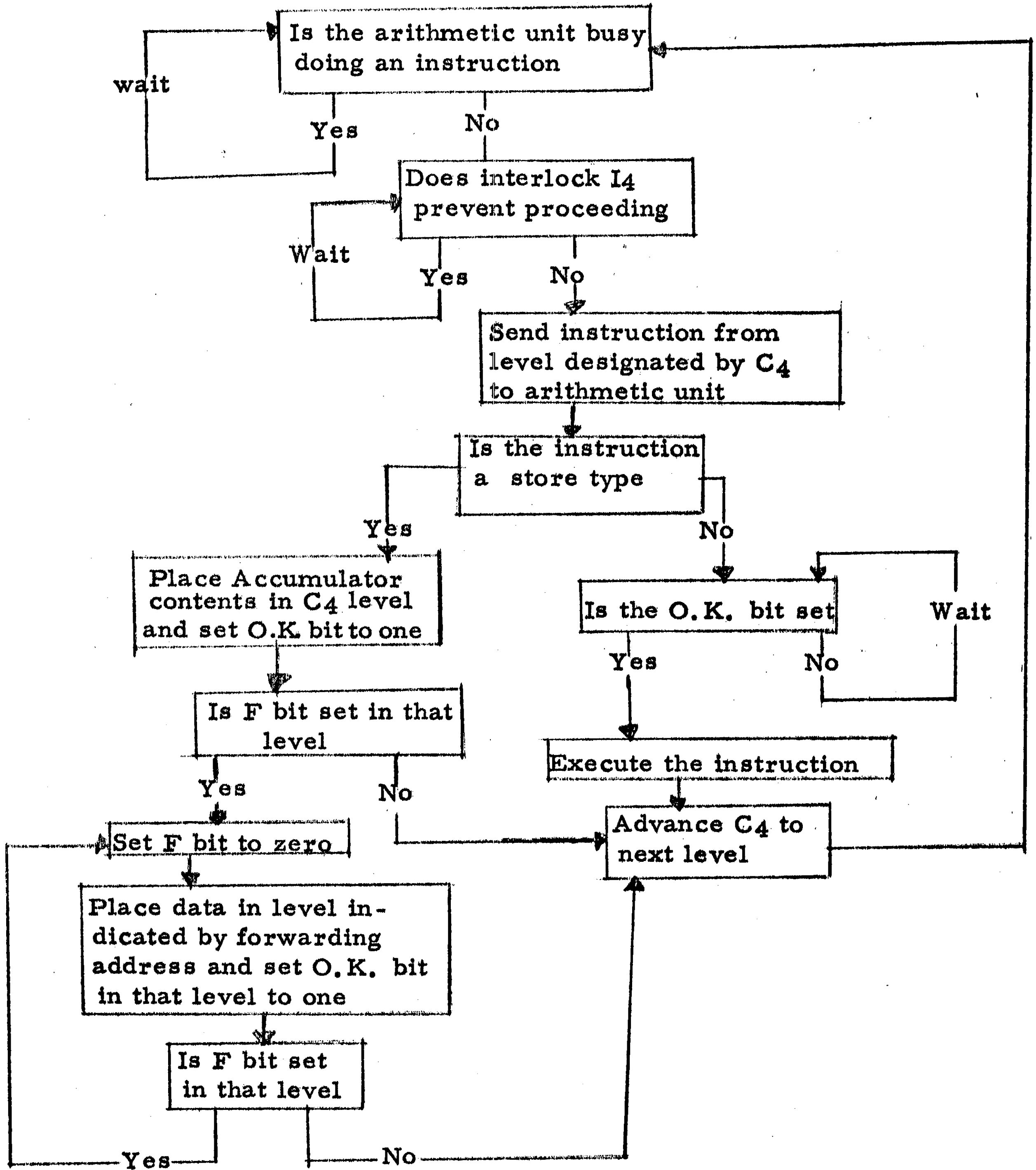


Figure 10, Procedure for Removing Instructions from Virtual Memory

The Vitual Memory is initialized, i. e., set to the starting conditions of an interrupt, with the exception that all store orders which have already received data from the accumulators must be executed first. Note: If the interrupt is of such a nature that the normal flow of instructions is not resumed, the procedure of storing the modified values of the index registers in the Virtual Memory gives logically correct results, i. e., the same as if the interrupt had occurred before the indexing took place.

IV. DESCRIPTION OF TIMING SIMULATION PROGRAM

A. General Considerations

During the logical design of STRETCH it was necessary to prove the value of the Virtual Memory concept and to assist in the selection of optimum values of various system design parameters. Examples of such parameters are: The number of memory boxes, interlace and allocation of memory addresses, and numbers of Virtual Memory levels. Also of interest were trade-off factors for speeds of indexing arithmetic unit, arithmetic unit, memories, etc.

In November 1957 the Timing Simulator (SIM-2) described here was written for the IBM 704. This program attempted to answer such questions quantitatively by simulating the time-wise operation of STRETCH on typical test programs coded in STRETCH language.

The basic logic of the 704 program follows the principles just described in the preceeding section for the Virtual Memory. It should be stressed that the Simulator is a Timing Simulator and does not execute the instructions in an arithmetic sense. It traces the time-wise progress of the instructions through the components of the computer observing all the interlocks and time delays necessary for correct representation of the behavior of the machine.

One of the fundamental concepts in the STRETCH design is that of asynchronous operation of the components. This means that there are a large number of logical steps being executed at any one time in the computer, each of them proceeding at its own rate. To simulate this flow of many parallel continuous operations, we have broken the continuous time variable into finite time steps. The basic time step is taken as 0.1 microsecond in the Simulator.

Several reasons prompted us to select this time interval. Some are relatively simple, such as the desire to have the results come out in microseconds and decimal fractions thereof. Taking a coarse time interval makes a given problem run faster on the Simulator since the running time is almost inversely proportional to the time step being used.

More fundamentally, the "natural" internal time scales of the computer are represented on one hand by the cycle time of the main memories (2 microsec) and on the other hand by the time required for signals to traverse one logical level in the circuits (5 to 20 millimicroseconds). The external time scale of the computer, as given by the I/O devices, is in the order of milliseconds for start up time and tens of microseconds for data flow rates.

Most internal macro logical processes require 0.1 microseconds or more since they usually require at least 10 logical levels. These represent the scale of quantities we wished to study in this simulator. Other scales could have been chosen. For example, one could write a program which followed the operation of every logical "and" and "or" circuit in the computer. (In fact, the authors have written such a program for a small experimental study.) The simulation program is simpler on this scale but the specification of a computer such as STRETCH would be an enormous task-- equivalent to laying out the whole circuit design. Another difficulty, would be that changing a gross parameter such as the multiply time might require the changing of thousands of "and" and "or" blocks in the circuit specification.

By taking 0.1 microsec as our quantum of time, we are automatically setting the scale of the smallest circuit entities which we will consider as being those which accomplish complete functions in a 0.1 microsec or few multiples thereof. Thus by using this philosophy, and considering many of the components of the computer as "black boxes", we greatly simplify the details which must be considered without introducing serious timing inaccuracies.

Our experience has indicated that more information was gained by making a large number of fast parameter studies using different configurations and programs that could have been obtained by a very slow, detailed simulation of a few runs with more precision per run. Even so our time scale is too fine to make serious Input-Output applications studies. These would require a simpler Simulator having at least a factor of 10 coarser basic time interval.

B. Logic of the Simulator

In the asynchronous organization of STRETCH there can be many major components operating at any one time. To achieve this parallel effect in the Simulator we essentially "hold time still" and scan the entire machine representation at each time step. Although every major block of the program is traversed at each time step, if there is no activity required in a given block, only a few tests need be made by the code.

If in this process it is determined that a given logical unit should do an operation, the time interval required for the operation is obtained from a table of constants. The speed of the various logical units can thus be changed parametrically by changing the values in the tables. A constant obtained from the tables is inserted into a memory location called the time counter for that unit. At each time step the program reduces this counter by one until it reaches zero. Thus the fact that the counter is non-zero can be used to indicate that the particular logical unit is busy and not available to service other requests. When the counter is zero the unit can consider a new input.

In addition to the time counters many of the logical blocks contain other conditions or interlocks which effect the operation of the block. These conditions are stored in the program and tested before action is undertaken. As an example, the "O.K. bit" described in the previous section is stored as a "one" or "zero" in a memory location associated with each Virtual Memory level (called LAU6, i in the program, where i is the Virtual Memory level). The "O.K." bit indicates that the data in the level is the correct value for the operation. In the program the "OK bit" is set by storing a one in location LAU6, i.

Each logical unit when it completes its operation may have data available to start another unit. The other unit may be notified that the data is available in two possible ways. Either (1) The subroutine corresponding to the receiving logical unit searches all possible inputs to determine if any of them has data for it, or (2) the sending unit sets logical constants within the receiving unit which indicate that the data is available. For example, the "O.K. bit" is set for a given level by the memory in-bus subroutine. While on the other hand, the arithmetic unit subroutine tests the O.K. bit to determine whether or not data is available for it.

The simplified Flow Diagram in Figure 11, indicates the order in which the subroutines for the various logical units are executed at each time step. Using the types of techniques just described above, the logical subroutines simulate the action of the components of the computer such as the Virtual Memory, arithmetic Unit,etc.

The details of the Simulator are described by Tables 1 through 4 and in the detailed flow diagrams at the end of this report. These flow diagrams can be corrolated in the obvious way with those given in the section which describes the Virtual Memory. The Simulator also contains several other subroutines which do such things as initialize the program at the beginning of a run, set up the timing diagrams and summarize the results of the run. (see discussion in the Result Section).

The STRETCH instructions being simulated are read into the 704 from tape as required. The instructions are put on tape from cards at the beginning of a run. (The input quantities read in for each operation are listed in Table I, column 1.) It is interesting to note that since the Simulator simulates timing only, not the arithmetic or indexing functions, the sequence of instructions to be executed must be furnished as a "string" with all loops unwound. However, to make the computer behave as it actually would, the loops must be furnished with "wrong way" paths given for the cases where the computer would take such paths. Also one must furnish more than enough information along such paths since it is difficult to predict in advance how far the computer will get down the wrong path before it is called back.

Parameters are changed from one run to another by use of control cards. The control cards are set up in such a way that any number of parameters may be changed between runs.

Results are given either as detailed timing charts or as summary listings for each problem. The usual procedure has been to print only summary results while making a series of parameter studies. At the end of each run the new control card or cards are read in, the problem tape is rewound, and the Simulator reruns the problem with the new constants.

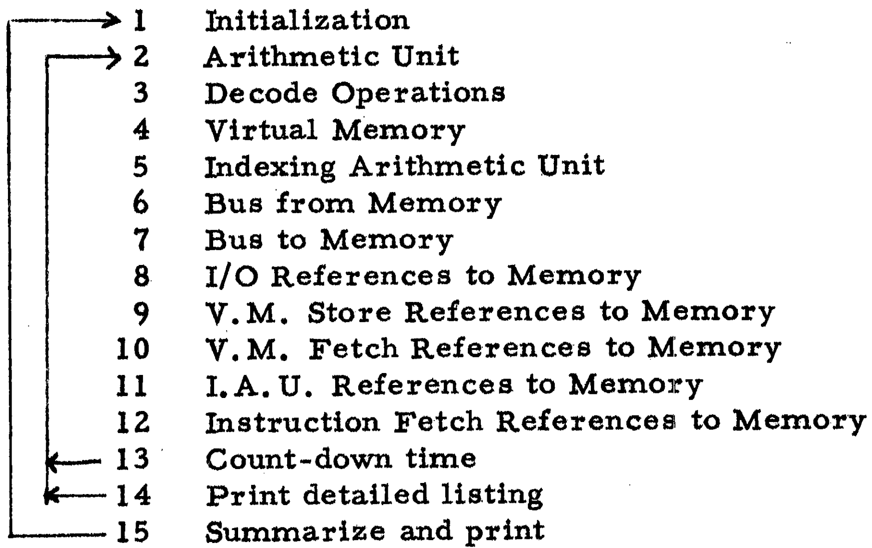


Figure 11. SIM - 2 Simplified Flow Diagram

V. SOME RESULTS OF THE SIMULATION STUDIES

A. General Description

1. Introduction

During 1958 a number of reports were written giving results of runs made with the SIM-2 program. We will not attempt to record here all of the results thus presented because many of them were superseded by later reports or were concerned with specific problems in the design of STRETCH. The results quoted here were chosen for their general interest as parametric studies and are not intended to represent STRETCH as it is actually designed.

2. Output Listings of Simulator

Figures 12 and 13 show examples of the type of output listings given by the Simulator. Figure 12 is a piece of a long timing chart with each line of printing representing 0.1 microsecond of time. The columns represent the various components of the computer. On the left and right are timing counts subdividing each microsecond. On the far right are conflict indicators ("C" on the charts) and waiting indicators, "W" which indicate when interlocks prevent operations from proceeding.

The 2nd column, II, gives the number of the instruction being indexed. The 4th column, AU, gives the number of the instruction using the arithmetic unit. The next four columns represent the instructions using the memory buses. The columns labeled X-,F-, and M- represent the index, fast, and main memories. A string of "X's" in the columns represents the cycle time of the memory. The number indicates the instruction using the memory and number of times which it is repeated gives the read-out time of the memory. The columns L- indicate which instruction is located in the Virtual Memory levels. The other columns are for details in analysis and need not be considered here.

Figure 13 gives an example of a series of summary listings. Each set of numbers represents a total problem run. The quantities listed are given in Tables III and IV. As was mentioned earlier, for most of the runs made in the Simulator studies, only summary runs were made.

Figure 12
Listing of Simulator Print-Out

PROJECT 7000 SIMULATOR 2 COCKE + KOLSKY NOV 57

	II	IS	AU	IF	IM	OF	OM	X1	X2	F1	F2	F3	F4	M1	M2	M3	M4	M5	M6	M7	M8	L1	L2	L3	L4	L5	L6	L7	L8	FD	MD	MC		
1	1																													1		2	1	
2	1																													1		1	2	C
3	1																													1			3	C
4	1					1																								3		2	4	
5	1					1																								3		1	5	
6	1										1X																			3			6	
7	1					3					1X																					2	7	
8	1					3					1X																					1	8	
9	1									3X	1X																						9	
10	1			1						3X	X																					2	10	
1	1			1						3X	X																					1	1	
2	1									3X																							2	
3	1			3						X																						2	3	
4	1	1		3						X																						1	4	
5	1	1																															5	
6	1	2																														2	6	
7	1	4																														1	7	
8		1																															8	
9	2	1																							1							2	9	
10	2	2																							1							1	10	
1	2	4	1					X																	1								1	
2		1																							1							2	2	
3	3	1																						2	1					5		1	3	
4	3	2																						2	1					5			4	
5	3	4	2			5		X																2	1							2	5	W
6		1				5																		2	1							1	6	W
7	4	1									5X												3	2	1								7	
8	4	2									5X												3	2	1							2	8	
9	4	2	3					X			5X												3	2	1							1	9	W
10	4	4									5X												3	2	1								10	W
1		1		5							X												3	2	1							2	1	W
2		1		5							X											4	3	2	1					7	4	1	2	
3		1																				4	3	2	1					7	4		3	W
4	5	1				7	4															4	3	2	1							2	4	W
5	5	2				7	4															4	3	2	1							1	5	W
6	5	2								7X				4X								4	3	2	1								6	W
7	5	4								7X				4X								4	3	2	1							2	7	W
8		1								7X				4X								4	3	2	1							1	8	W
9	6	1								7X				4X								4	3	2	5						5		9	W
10	6	2		7						X				4X								4	3	2	5						5	2	10	W
1	6	4		7						X				4X								4	3	2	5						5	1	1	W
2		1												4X								4	3	2	5						5		2	W
3	7	1					5							4X								4	3	6	5					9		2	3	W
4	7	2			4		5							X								4	3	6	5					9		1	4	W
5	7	2			4									X	5X							4	3	6	5					9			5	W
6	7	4				9								X	5X							4	3	6	5							2	6	
7		1	4			9								X	5X							4	3	6	5							1	7	
8	8	1	4								9X			X	5X							4	7	6	5						7		8	
9	8	2									9X			X	5X							4	7	6	5						7	2	9	W
10	8	2						X			9X			X	5X							4	7	6	5						7	1	10	W
1	8	2									9X			X	5X							4	7	6	5						7		1	W
2	8	4		9			7				X			X	5X							4	7	6	5							2	2	W
3		1		9	5		7				X			X	X							4	7	6	5							1	3	W
4		1			5									X	X		7X					8	7	6	5					11			4	CW
5	9	1												X	X		7X					8	7	6	5					11		2	5	C
6	9	2	5												X		7X					8	7	6	5					11		1	6	C
7	9	4	5												X		7X					8	7	6	5					11			7	C
8		1	5			11									X		7X					8	7	6	5							2	8	C
9	10	1	5			11									X		7X					8	7	6	9							1	9	C
10	10	2	5							11X					X		7X					8	7	6	9								10	C
1	10	2	5					X		11X					X		7X					8	7	6	9							2	1	C
2	10	2			7					11X					X		X					8	7	6	9							1	2	CW
3	10	4	6		7			X		11X					X		X					8	7	6	9								3	C
4		1		11						X					X		X					8	7	6	9							2	4	C
5		1	7	11						X							X					8	7	10	9					13		1	5	C
6		1	7														X					8	7	10	9					13			6	C

Figure 12

Listing Of Simulator Print-Out

7	11	1				13							X	8	7	10	9			2	7	CW
8	11	2				13							X	8	7	10	9			1	8	CW
9	11	2						X		13X			X	8	7	10	9				9	CW
10	11	2								13X			X	8	7	10	9			2	10	CW
1	11	4								13X			X	8	7	10	9		8	1	1	W
2		1								13X			X	8	7	10	9		8		2	CW
3	12	1		13			8			X			X	8	11	10	9		10	2	3	W
4	12	2		13			8			X				8	11	10	9		10	1	4	CW
5	12	4											8X	8	11	10	9		10		5	CW
6		1					10						8X	8	11	10	9			2	6	CW
7	13	1					10						8X	8	11	10	9	15		1	7	CW
8	13	1										10X	8X	8	11	10	9	15			8	CW
9	13	1				15						10X	8X	8	11	10	9			2	9	CW
10	13	1				15						10X	8X	8	11	10	9			1	10	CW
1	13	1							15X			10X	8X	8	11	10	9				1	CW
2	13	1							15X			10X	8X	8	11	10	9			2	2	CW
3	13	1			8				15X			10X	X	8	11	10	9			1	3	CW
4	13	1			8				15X			10X	X	8	11	10	9				4	CW
5	13	1		15					X			10X	X	8	11	10	9			2	5	C
6	13	1	8	15	10				X			X	X	12	11	10	9			1	6	C
7	13	2	8		10							X	X	12	11	10	9				7	C
8	13	2	8					X				X	X	12	11	10	9			2	8	C
9	13	2	8									X	X	12	11	10	9			1	9	C
10	13	4	8									X	X	12	11	10	9				10	C
1		1	8									X	X	12	11	10	9			2	1	C
2	14	1										X	X	12	11	10	9		11	1	2	W
3	14	1	9					X				X	X	12	11	10	9		11		3	
4	14	1					11					X	X	12	11	10	13		13	2	4	
5	14	2	10				11					X		12	11	10	13		13	1	5	
6	14	2	10					X				X	11X	12	11	10	13		13		6	
7	14	2					13					X	11X	12	11	10	13			2	7	W
8	14	4					13						11X	12	11	1C	13			1	8	W
9		1									13X		11X	12	11	10	13				9	W
10	15	1									13X		11X	12	11	14	13	17	14	2	10	W
1	15	2									13X		11X	12	11	14	13	17	14	1	1	W
2	15	4									13X		11X	12	11	14	13	17	14		2	W
3		1				17	14				13X		11X	12	11	14	13			2	3	W
4	16	1			11	17	14				13X		X	12	11	14	13			1	4	W
5	16	1			11					17X	13X	14X	X	12	11	14	13				5	W
6	16	1								17X	13X	14X	X	12	11	14	13			2	6	
7	16	1	11		13					17X	X	14X	X	12	15	14	13			1	7	
8	16	2	11		13					17X	X	14X	X	12	15	14	13				8	
9	16	2	11	17				X		X	X	14X	X	12	15	14	13			2	9	
10	16	2	11	17						X	X	14X	X	12	15	14	13			1	10	
1	16	3	11					X			X	14X	X	12	15	14	13				1	
2	16	3	11								X	14X	X	12	15	14	13			2	2	
3	16	4			14						X	X	X	12	15	14	13			1	3	W
4		1	12		14			X			X	X	X	12	15	14	13				4	
5	17	1									X	X	X	16	15	14	13	19		2	5	
6	17	2	13								X	X		16	15	14	13	19		1	6	
7	17	2	13					X			X	X		16	15	14	13	19			7	
8	17	2				19					X	X		16	15	14	13			2	8	
9	17	3	14			19		X				X		16	15	14	13			1	9	
10	17	3	14						19X			X		16	15	14	13				10	
1	17	4	14						19X			X		16	15	14	13			2	1	
2		1	14						19X			X		16	15	14	13			1	2	
3	18	1	14						19X			X		16	15	14	17				3	
4	18	2	14	19					X			X		16	15	14	17			2	4	
5	18	2		19					X					16	15	14	17			1	5	W
6	18	4	15					X						16	15	14	17				6	
7		1												16	15	14	17			2	7	
8	19	1	16					X						16	15	18	17	21	18	1	8	
9	19	2												16	15	18	17	21	18		9	
10	19	2	17			21	18	X						16	15	18	17			2	10	
1	19	4				21	18							16	15	18	17			1	1	W
2		1								21X	18X			16	15	18	17				2	W
3	20	1								21X	18X			16	19	18	17		19	2	3	W
4	20	2								21X	18X			16	19	18	17		19	1	4	W
5	20	4								21X	18X			16	19	18	17		19		5	W
6		1		21			19			X	18X			16	19	18	17			2	6	W
7		1		21			19			X	18X			20	19	18	17	23		1	7	W

Figure 12

Listing of Simulator Print-Out

```
    8      1                                  18X19X                  20 19 18 17             23        8  W
    9 21   1          23                      18X19X                  20 19 18 17                    2  9  W
   10 21   2       18 23                        X19X                  20 19 18 17                    1 10  W
    1 21   2       18             23X           X19X                  20 19 18 17                       1  W
    2 21   4                      23X           X19X                  20 19 18 17                    2  2
    3      1 18                   23X           X19X                  20 19 18 17                    1  3
    4 22   1 18                   23X           X19X                  20 19 18 21                21     4
    5 22   2    23                  X           X19X                  20 19 18 21                21  2  5  W
    6 22   2    23 19         X     X           X  X                  20 19 18 21                21  1  6  W
    7 22   2       19                           X  X                  20 19 18 21                21     7  W
    8 22   4             21                     X  X                  20 19 18 21                    2  8
    9      1 19          21                     X  X                  20 19 18 21                    1  9
   10 23   1 19                                 X  X21X               20 19 22 21             25 22    10
    1 23   2 19                                 X  X21X               20 19 22 21             25 22  2  1
    2 23   4 19                                    X21X               20 19 22 21             25 22  1  2
    3      1 19                                    X21X               20 19 22 21             25 22     3
    4 24   1 19       25 22                        X21X               20 23 22 21                    2  4
    5 24   2          25 22                        X21X               20 23 22 21                    1  5  W
    6 24   2 20               X      25X           X21X22X            20 23 22 21                       6 C
    7 24   2                  X      25X           X21X22X            20 23 22 21                    2  7  W
    8 24   2       21                25X              X22X            20 23 22 21                    1  8  W
    9 24   4       21                25X              X22X            20 23 22 21                       9  W
   10'     1    25                     X              X22X            20 23 22 21                    2 10
    1      1 21 25                     X              X22X            24 23 22 21             27 24  1  1
    2      1 21                                       X22X            24 23 22 21             27 24     2
    3 25   1          27 24                           X22X            24 23 22 21                    2  3  W
    4 25   2       22 27 24                           X  X            24 23 22 21                    1  4  W
    5 25   2       22         X   27X         24X     X  X            24 23 22 21                       5  W
    6 25   2                      27X         24X     X  X            24 23 22 21                    2  6
    7 25   4 22                   27X         24X     X  X            24 23 22 21                    1  7
    8      1 22                   27X         24X     X  X            24 23 22 21                       8
    9 26   1 22 27                  X         24X     X  X            24 23 22 25                25  2  9
   10 26   2 22 27                  X         24X        X            24 23 22 25                25  1 10
    1 26   2 22               X               24X        X            24 23 22 25                25     1
    2 26   2 22          25                   24X        X            24 23 22 25                    2  2
    3 26   4       24    25                     X        X            24 23 22 25                    1  3  W
    4      1 23    24         X                 X   25X  X            24 23 22 25                       4
    5 27   1                                    X   25X  X            24 23 26 25             29 26  2  5
    6 27   2 24                                 X   25X               24 23 26 25             29 26  1  6
    7 27   2 24               X                 X   25X               24 23 26 25             29 26     7
    8 27   2          29 26                     X   25X               24 23 26 25                    2  8  W
    9 27   4          29 26                     X   25X               24 23 26 25                    1  9  W
   10      1                         29X        X26X25X               24 23 26 25                      10  W
    1 28   1                         29X        X26X25X               24 27 26 25                27  2  1  W
    2 28   2       25                29X        X26X  X               24 27 26 25                27  1  2  W
    3 28   4       25                29X        X26X  X               24 27 26 25                27     3  W
    4      1    29       27            X        X26X  X               24 27 26 25                    2  4
    5      1 25 29       27            X         26X  X               28 27 26 25             31     1  5
    6      1 25                                  26X  X27X            28 27 26 25             31        6
    7 29   1 25       31                         26X  X27X            28 27 26 25                    2  7
    8 29   2 25    26 31                           X  X27X            28 27 26 25                    1  8
    9 29   4 25    26             31X              X  X27X            28 27 26 25                       9
   10      1 25                   31X              X  X27X            28 27 26 25                    2 10
    1 30   1                      31X              X  X27X            28 27 26 29                    1  1
    2 30   2 26                   31X              X  X27X            28 27 26 29                       2
    3 30   4 26 31                  X              X  X27X            28 27 26 29                    2  3
    4      1 26 31 27               X              X     X            28 27 26 29                    1  4
    5      1 26    27         X                    X     X            28 27 30 29             33        5
    6 31   1 26                                    X     X            28 27 30 29             33     2  6
    7 31   2 26                                    X     X            28 27 30 29             33     1  7
    8 31   4                                       X     X            28 27 30 29             33        8
    9      1 27       33                           X     X            28 27 30 29                    2  9
   10 32   1 27       33      X                          X            28 31 30 29                    1 10
    1 32   2 27                       33X                X            28 31 30 29                       1
    2 32   4 27                       33X                X            28 31 30 29                    2  2
    3      1 27                       33X                X            28 31 30 29                    1  3
    4      1 27                       33X                X            28 31 30 29             35        4
    5      1    33                      X                X            28 31 30 29             35     2  5
    6      1 28 33                      X                             32 31 30 29             35     1  6
```

Figure 12

Listing of Simulator Print-Out

```
 7     1 28                                    32 31 30 29   35     7
 8 33  1           35                          32 31 30 29        2  8  W
 9 33  2 29        35   X                      32 31 30 29        1  9
10 33  4                  35X                  32 31 30 29          10
 1     1 30               35X                  32 31 30 29        2  1
 2 34  1 30             X 35X                  32 31 30 33        1  2
 3 34  2                  35X                  32 31 30 33           3
 4 34  4 31  35             X                  32 31 30 33        2  4
 5     1 31  35             X                  32 31 30 33        1  5
 6     1 31             X                      32 31 34 33   37      6
 7 35  1 31                                    32 31 34 33   37   2  7
 8 35  2 31                                    32 31 34 33   37   1  8
 9 35  4 31                                    32 31 34 33   37      9
10     1 31        37                          32 31 34 33        2 10
 1 36  1 31        37                          32 35 34 33        1  1
 2 36  2 31                  37X               32 35 34 33           2
 3 36  4 31                  37X               32 35 34 33        2  3
 4     1 31                  37X               32 35 34 33        1  4
 5     1 31                  37X               32 35 34 33   39      5
 6     1     37                X               32 35 34 33   39   2  6  W
 7     1 32  37         X      X               32 35 34 33   39   1  7
 8     1                X                      36 35 34 33   39      8
 9 37  1 33        39                          36 35 34 33        2  9
10 37  2 33        39                          36 35 34 33        1 10
 1 37  4                  39X                  36 35 34 33           1
 2     1 34               39X                  36 35 34 33        2  2
 3 38  1 34               39X                  36 35 34 37        1  3
 4 38  2 34               39X                  36 35 34 37           4
 5 38  4 34  39             X                  36 35 34 37        2  5
 6     1 34  39             X                  36 35 34 37        1  6
 7     1 34             X                      36 35 38 37   41      7
 8 39  1 34                                    36 35 38 37   41   2  8
 9 39  2 34                                    36 35 38 37   41   1  9
```

```
 8 61  2                               55X     57 60 59 58   63      8
 9 61  4 59        63   X              55X     57 60 59 58        2  9
10     1           63                    X     57 60 59 58        1 10
 1 62  1 60             X 63X            X     61 60 59 58           1
 2 62  2                  63X            X     61 60 59 58        2  2
 3 62  2 61               63X            X     61 60 59 58        1  3  W
 4 62  4                  63X            X     61 60 59 58           4  W
 5     1     63             X            X     61 60 59 58        2  5  W
 6     1     63         X   X            X     61 60 59 62   98   1  6
 7     1                                 X     61 60 59 62   98      7
 8 63  1 62        98                    X     61 60 59 62        2  8  W
 9 63  2           98                    X     61 60 59 62        1  9  W
10 63  2                     98X         X     61 60 59 62          10  W
 1 63  4                     98X         X     61 60 59 62        2  1  W
 2     1                     98X               61 60 59 62        1  2  W
 3 64  1                X    98X               61 60 63 62           3
 4 64  2     98                X               61 60 63 62        2  4
 5 64  2 63  98         X      X               61 60 63 62        1  5  W
 6 64  2                                       61 60 63 62           6  W
 7 64  4                                       61 60 63 62        2  7  W
 8     1                                       61 60 63 62        1  8  W
 9 98  1                X                      61 64 63 62           9
10 98  2                                       61 64 63 62        2 10
 1 98  4 64             X                      61 64 63 62        1  1  W
 2     1                                       61 64 63 62           2  W
 3     1                                       98 64 63 62        2  3
 4     1                                       98 64 63 62        1  4
 5     1 98                                    98 64 63 62           5
```

```
PROJECT 7000 SIMULATOR 2  COCKE + KOLSKY  NOV 57

LA FM MM IR IS IA ID    MB FB FD    DT X1 X2 MD F1 F2 F3 M1 M2 M3 15 14 13 12 11 10  9  8  7  6  5  4  3  2  1     XM
 4  2  4  2  2  2  2     2  2  2  2           1  2  4  4  6  8 17 20 50 18 20 18 12  8  6  4  2  1 10  5  4  2  1
TT     AU     IAU   ADLA   LAF    WI     WM   DLA 8    7     6     5     4     3     2     1     0    IBF   IBM   OBF   OB
1665   454    349   3310   535    28     266                              535   335   64    36    29   144   76    144   8
IS 4   3      2     1      M12    M11    M10   M9     M8    M7    M6    M5    M4    M3    M2    M1   MMC   FMC   MBC   FB
141    9      250   590    7                          132   276   276   180               212   219  191         12
```

Figure 13

Listing of Simulator Summary Print-Out

```
PROJECT 7000 SIMULATOR +2  COCKE + KOLSKY  NOV 57

LA FM MM     IS IA ID    MB FB FD     DT X1 X2 MD F1 F2 F3 M1 M2 M3 15 14 13 12 11 10  9  8  7  6  5  4  3  2  1
 4  2  4  2   3  3  2     2  2  2         2  8  2  4  4  6  8 17 20 50 27 20 30 17  8 10  4  6  1 10  5  4  2  1     291
 TT    AU    IAU  ADLA   LAF    WI     WM  DLA 8     7      6      5      4      3      2      1      0    IRF    IRM    ORF    ORM
2296   6 3   250  2929   439    50     154                                  439    241    179    89     50    1 4    55     1 4    62
 IS 4   3     2     1    M12    M11    M10   M9     M8     M7     M6     M5     M4     M3     M2     M1    MMC    FMC    MRC    FRC
  1 2   34   375   486                               95     2      2     130                         154    159    82            1

LA FM MM     IS IA ID    MB FB FD     DT X1 X2 MD F1 F2 F3 M1 M2 M3 15 14 13 12 11 10  9  8  7  6  5  4  3  2  1
 4  2  4  2   4  4  2     2  2  2         2  8  2  4  4  6  8 17 20 50 27 20 30 17  8 10  4  6  1 10  5  4  2  1     268
 TT    AU    IAU  ADLA   LAF    WI     WM  DLA 8     7      6      5      4      3      2      1      0    IRF    IRM    ORF    ORM
2362   586   289  2784   4 2    70     160                                  4 2    226    195    1 4    70    1 1    54     1 1    60
 IS 4   3     2     1    M12    M11    M10   M9     M8     M7     M6     M5     M4     M3     M2     M1    MMC    FMC    MRC    FRC
  99    44   4 3   452                               93    194    194    127                         149    154    68            1

LA FM MM     IS IA ID    MB FB FD     DT X1 X2 MD F1 F2 F3 M1 M2 M3 15 14 13 12 11 10  9  8  7  6  5  4  3  2  1
 4  2  4  2   4  4  2     2  2  2         2  8  2  4  4  6  8 17 20 50  1 20 30  1  8  1  4  6  1 10  5  4  2  1     453
 TT    AU    IAU  ADLA   LAF    WI     WM  DLA 8     7      6      5      4      3      2      1      0    IRF    IRM    ORF    ORM
2076   143   328  1833   163    239    414                                  163    191    2 9    187    248   115    61     115    69
 IS 4   3     2     1    M12    M11    M10   M9     M8     M7     M6     M5     M4     M3     M2     M1    MMC    FMC    MRC    FRC
 113    38   482   365                               1 5   221    221    144                         170    176    91            2      5

LA FM MM     IS IA ID    MB FB FD     DT X1 X2 MD F1 F2 F3 M1 M2 M3 15 14 13 12 11 10  9  8  7  6  5  4  3  2  1
 4  2  4  2   4  4  2     2  2  2         2  8  2  4  4  6  8 17 20 50  9 20 30  6  8  3  4  6  1 10  5  4  2  1     433
 TT    AU    IAU  ADLA   LAF    WI     WM  DLA 8     7      6      5      4      3      2      1      0    IRF    IRM    ORF    ORM
2073   287   329  1883   172    198    311                                  172    185    236    163    241   115    61     115    69
 IS 4   3     2     1    M12    M11    M10   M9     M8     M7     M6     M5     M4     M3     M2     M1    MMC    FMC    MRC    FRC
 113    42   462   381                               1 6   221    221    144                         170    176    91            2      5

LA FM MM     IS IA ID    MB FB FD     DT X1 X2 MD F1 F2 F3 M1 M2 M3 15 14 13 12 11 10  9  8  7  6  5  4  3  2  1
 4  2  4  2   4  4  2     2  2  2         2  8  2  4  4  6  8 17 20 50 27 20 30 18  8  9  4  6  1 10  5  4  2  1     258
 TT    AU    IAU  ADLA   LAF    WI     WM  DLA 8     7      6      5      4      3      2      1      0    IRF    IRM    ORF    ORM
2371   575   288  2768   397    73     169                                  397    226    196    1 5    73    1 1    53     1 1    60
 IS 4   3     2     1    M12    M11    M10   M9     M8     M7     M6     M5     M4     M3     M2     M1    MMC    FMC    MRC    FRC
  99    37   4 7   456                               92    194    194    126                         149    154    69            1

LA FM MM     IS IA ID    MB FB FD     DT X1 X2 MD F1 F2 F3 M1 M2 M3 15 14 13 12 11 10  9  8  7  6  5  4  3  2  1
 4  2  4  2   4  4  2     2  2  2         2  8  2  4  4  6  8 17 20 50 36 20 30 24  8 12  4  6  1 10  5  4  2  1     235
 TT    AU    IAU  ADLA   LAF    WI     WM  DLA 8     7      6      5      4      3      2      1      0    IRF    IRM    ORF    ORM
2647   660   258  3020   5 6    49     125                                  5 6    189    170    83     49    90     48     90     54
 IS 4   3     2     1    M12    M11    M10   M9     M8     M7     M6     M5     M4     M3     M2     M1    MMC    FMC    MRC    FRC
  89    24   366   519                               83    173    173    113                         133    138    64            1

LA FM MM     IS IA ID    MB FB FD     DT X1 X2 MD F1 F2 F3 M1 M2 M3 15 14 13 12 11 10  9  8  7  6  5  4  3  2  1
 4  2  4  2   4  4  2     2  2  2         2  8  2  4  4  6  8 17 20 50 45 20 30 30  8 15  4  6  1 10  5  4  2  1     216
 TT    AU    IAU  ADLA   LAF    WI     WM  DLA 8     7      6      5      4      3      2      1      0    IRF    IRM    ORF    ORM
2998   711   227  3147   566    47     95                                   566    178    140    67     47    80     42     80     48
 IS 4   3     2     1    M12    M11    M10   M9     M8     M7     M6     M5     M4     M3     M2     M1    MMC    FMC    MRC    FRC
  78    45   310   565                               73    153    153    1                           118    122    52            1
```

The detailed timing charts for most problems would be about 50 feet long for each run. Since over a 1000 cases have been run, it is clear that only a few cases could be printed in full detail. Nevertheless, the detailed timing charts were essential for two reasons: (1) Debugging the program with all its hundreds of conditional branches would have been a staggering task without the detailed listings, and (2) determining the causes of some of the anamolous summary results required that one examine the listings in detail. Also the authors found that studying the listings enabled them to get a "feel" for the flow of information which was necessary to locate bottlenecks in the processing speed.

B. Test Problems Used.

Five of the test problems used most frequently are described below. Other test problems were used for specific studies but since the results were similar for all problems of a given type, we gradually discontinued using them. The following were originally selected as being typical of different classes of problems. A brief description of each is repeated here for completeness.

1. Mesh Problem - Part of an hydrodynamics problem from Los Alamos. It contains a more or less "average" mixture of instructions for scientific problems: 85% Floating Point instructions, 14% index modification instructions, and 1% VFL. It is usually arithmetic unit limited.

2. Monte Carlo Branching Problem - Part of an actual Monte Carlo neutron diffusion code. It represents a chain of logical decisions with very little arithmetic in between. It contains 47% Floating Point, 15% index modification instructions, and 36% branches of the indicator and unconditional types. It is largely instruction-access limited.

3. Reactor Problem - The inner loop of a neutron diffusion problem from Westinghouse. It consists of 90% Floating Point arithmetic (39% of which are multiplys) and 10% index modification instructions. It is almost entirely arithmetic unit limited.

4. Computer Test Problem - The evaluation of a polynominal using computed indices. It was prepared by I. Ziller to compare various computers. It has 71% Floating Point, 10% index modification, 6% VFL and 13% indicator branches. It is usually arithmetic unit limited but not for all configurations.

5. Simultaneous Equations - The inner loop of a matrix inversion routine 67% Floating Point and 33% index modification. Arithmetic and logic are about equally important. It is limited both by arithmetic and instruction-access speeds.

C. Results of Simple Parameter Studies

1. General

When the Simulator Program was first completed in late 1957, we undertook a series of studies in which the main parameters describing the STRETCH system were varied one or two at a time in order to get a measure for the importance of different effects. During this phase we spent much time studying the detailed print outs described above to determine the exact cause of some of the anomolous effects.

After this we began to specialize the studies towards answering specific questions in the STRETCH design and made more use of the Summary listings. Two of these studies are described in the following sections. In the present section the major part of the material is taken from the first parameter studies. The graphs reproduced below are in terms of an arbitrary speed scale in which one of the first problems studied (The Mesh Calc.) was taken as 100.

The table below summarizes the major effects studied. The individual items are discussed in the following subsections.

Examples of STRETCH Timing Simulator Results

Description	Mesh Calc.		Monte Carlo	
	Speed	% Change	Speed	% Change
1. Standard Design	100	0	45.	0
2. A.U. Times Doubled	73	-27%	43.	- 4%
3. I.A.U. Times Doubled	67.	-33%	26	-42%
4. Both AU and IAU doubled	60.	-40%	24.	-46%
5. 2.0 us Instr. Memory	98.	- 2%	35.	-22%
6. Combining Instr. and Data in 4 MM	82.	-16%	32	-29%
7. Combining Instr. and Data in 6 MM	86.	-14%	33	-27%
8. 2 Levels of Virtual Memory	89.	-11%	38	-15%
9. 6 Levels of Virtual Memory	106.	+ 6%	46.	+ 3%

2. Standard Values of Parameters

The combination of constants which was taken as the standard reference values for the original parameter studies is as follows:

a. Machine Components:

1. Levels of Virtual Memory	4
2. Number of Instruction Memories	2
3. Number of Main (data) Memories	4

b. Computer Speeds:

1. Indexing Time*	0.6 usec
2. Arithmetic Unit Times	
Floating Add	0.6 usec
Floating Multiply	1.2 usec
Floating Divide	1.8 usec
Fetch	0.2 usec
usual 6-6-3-1 average	0.64 usec

*This is total time to index one order, includes instruction decoding, index fetch, index addition, and storing modified address.

c. Memory Speeds:

1. Fast (Instr.) Memory Times	
Read out time	0.4 usec
End Signal Time	0.4 usec
Memory cycle time*	0.6 usec

*(The actual effective cycle time is 0.9 usec, since the bus clocking permitted successive references to the same memory box only in multiples of 0.3 usec and the memory box must be free at the time of the reference not just finishing.)

2. Main (Data) Memory Times	
Read out time	0.8 usec
End Signal Time	1.7 usec
Memory cycle time*	2.0 usec

*(The effective cycle is 2.1 us for same reason as above).

3. Index Core Memory Times	
Read out time	0.4 usec
Memory cycle time	0.8 usec

The index cores are assumed tied directly to the IAU, so these figures include bus times.

4. Bus Speeds
   a. Buses to and from Instruction and Data memories 0.2 usec slot (either read or write) available every 0.3 usec.
   b. Decode and switching time in central control unit 0.2 usec to 0.4 usec (depending on bus slots available.)

Note: A separate bus system to instruction and Data memories is assumed, but not necessary.

In addition there is usually a 0.1 usec delay between the completion of any function and the beginning of the next one by the unit, or in the transfer from one register to another.

3. <u>Speed vs Number of Levels of Virtual Memory</u>

Figure 14 shows the effect on computer performance of varying the number of levels of Virtual Memory. Curves for the Monte Carlo and Mesh Calculations with two sets of arithmetic and indexing arithmetic speeds are shown. The AU times given are the 6-6-3-1 averages mentioned above.

A number of interesting results are apparent from these curves:

(1) There is a tremendous gain to be had in going to the Vitual Memory organization. The point for "0 levels" means that the arithmetic unit is tied directly to the instruction preparation unit, although simple Indexing-Execution overlap is still possible.

(2) The gain in performance goes up very rapidly for the first two levels then rises more slowly for the rest of the range.

(3) A large number of levels does the Monte Carlo problem less good than the Mesh problem because constant branching in the former spoils the flow of instructions. Notice that the curve for the Monte Carlo problem actually <u>decreases</u> slightly beyond six levels. This phenomenom is a result of memory conflicts caused by extraneous memory references started by the computer running ahead on the wrong-way paths of branches.

(4) The computer performance on a given problem is clearly less for slower arithmetic speeds. However, it is important to note that the <u>sensitivity</u> of the performance is also less for slower arithmetic speeds. The Virtual Memory improves the performance in either case, but it is not a substitute for a fast arithmetic unit.

Figure 14

SIGMA COMPUTER SPEED

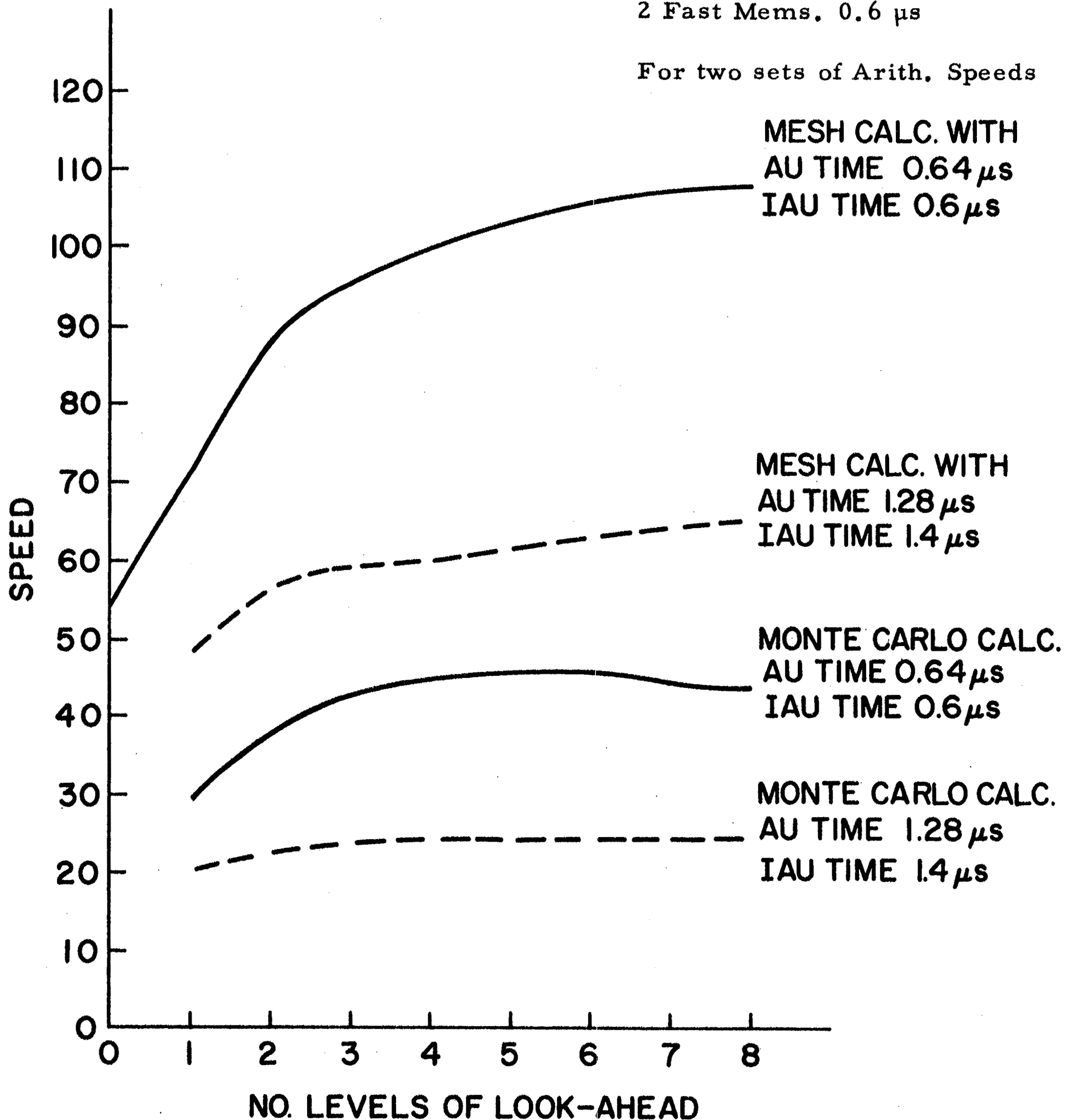
vs. No. of levels of

Look-Ahead Registers

4 Main Mems. 2.0 μs

2 Fast Mems. 0.6 μs

For two sets of Arith. Speeds



4. Speed vs Number of Main Memory Units

Figure 15 shows how internal computer performance varies with the total number of memory units for a particular problem. The entire calculation is assumed to be contained in memory for all cases. The speed gain from overlapping memories is quite apparent from the graphs.

The speed differential between having and not having instructions separated from data arises from delays in instruction fetches caused by the memory units being busy with data. The size of this effect varies from problem to problem, being less pronounced for problems which are arithmetic limited and more for logical problems.

Since memory units for STRETCH are attachable only in pairs after the first and are interlaced only in powers of two, some of the points on the graph do not represent physically attainable combinations, e.g., 5 memories all interlaced. (The simulation program has no such restrictions.)

The "X's" on the graph show the effect of replacing the 0.6 usec instruction memories by a pair of 2.0 usec memories. The resulting performance change is small for the Mesh Problem, which is arithmetic limited, but large for the instruction-fetch limited Monte Carlo problem.

5. Speed vs Arithmetic Unit and Indexing Arithmetic Unit Times

Although everyone realizes the importance of arithmetic speed on overall computer performance, it was not until the Simulator results became available that the true importance of the indexing arithmetic speeds was recognized. Figures 16 and 17 show a two parameter family of curves giving the computer speed as a function of the AU and IAU times.

Figure 17 in which the arithmetic time is the abscissa shows an interesting "saturation" effect where the computer performance is independent of AU speed below some critical value. Thus it makes no sense to strain AU speeds if the IAU is not improved to match. The curves in Figure 16 show the same effect i.e., the IAU speed serves as a "ceiling" on performance beyond which the AU speed cannot pass.

Figure 15

SIGMA COMPUTER SPEED

vs. Number of Main

Memory Boxes

4 levels LA

0.6 μs IAU time

0.64 μs AU time

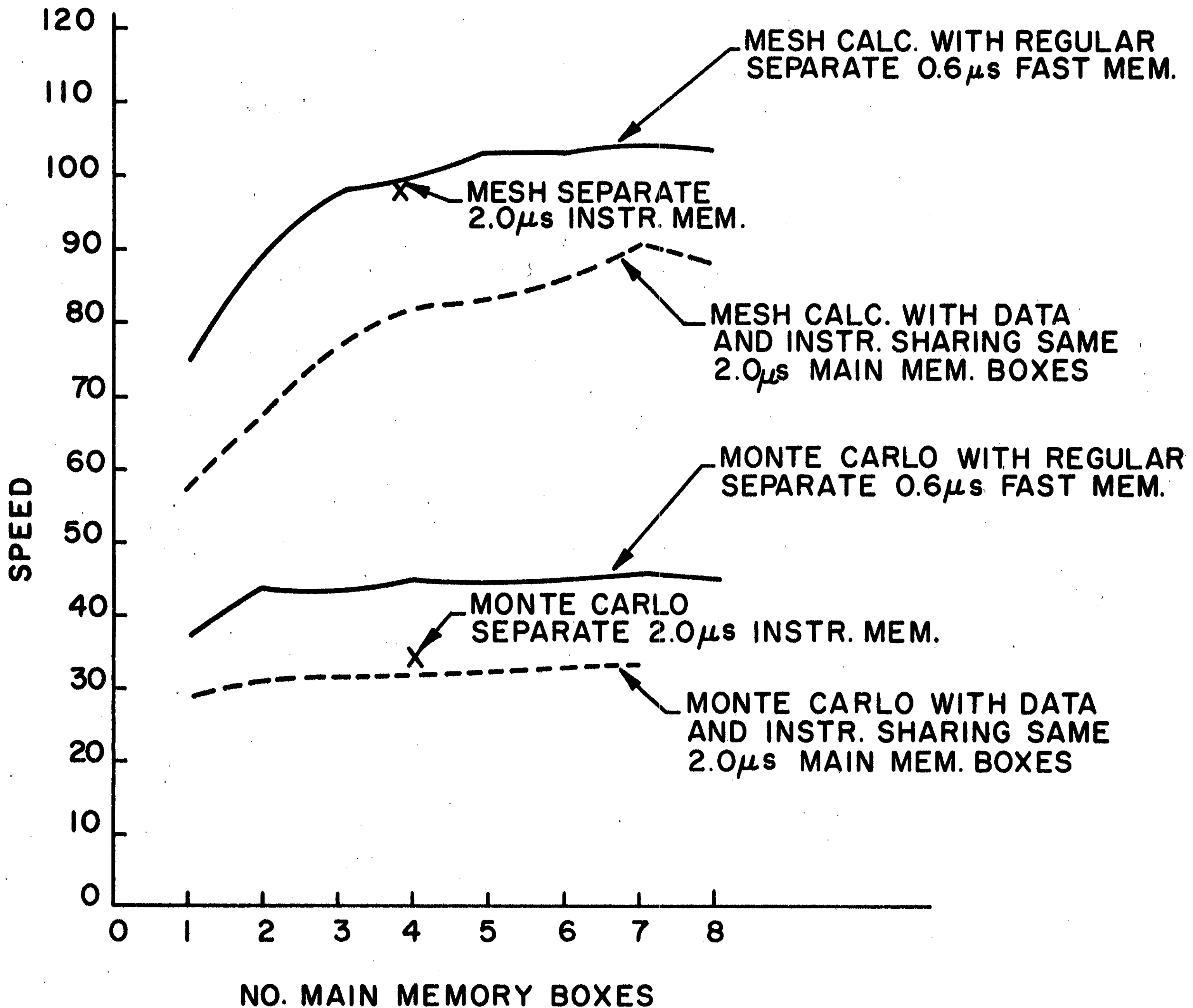
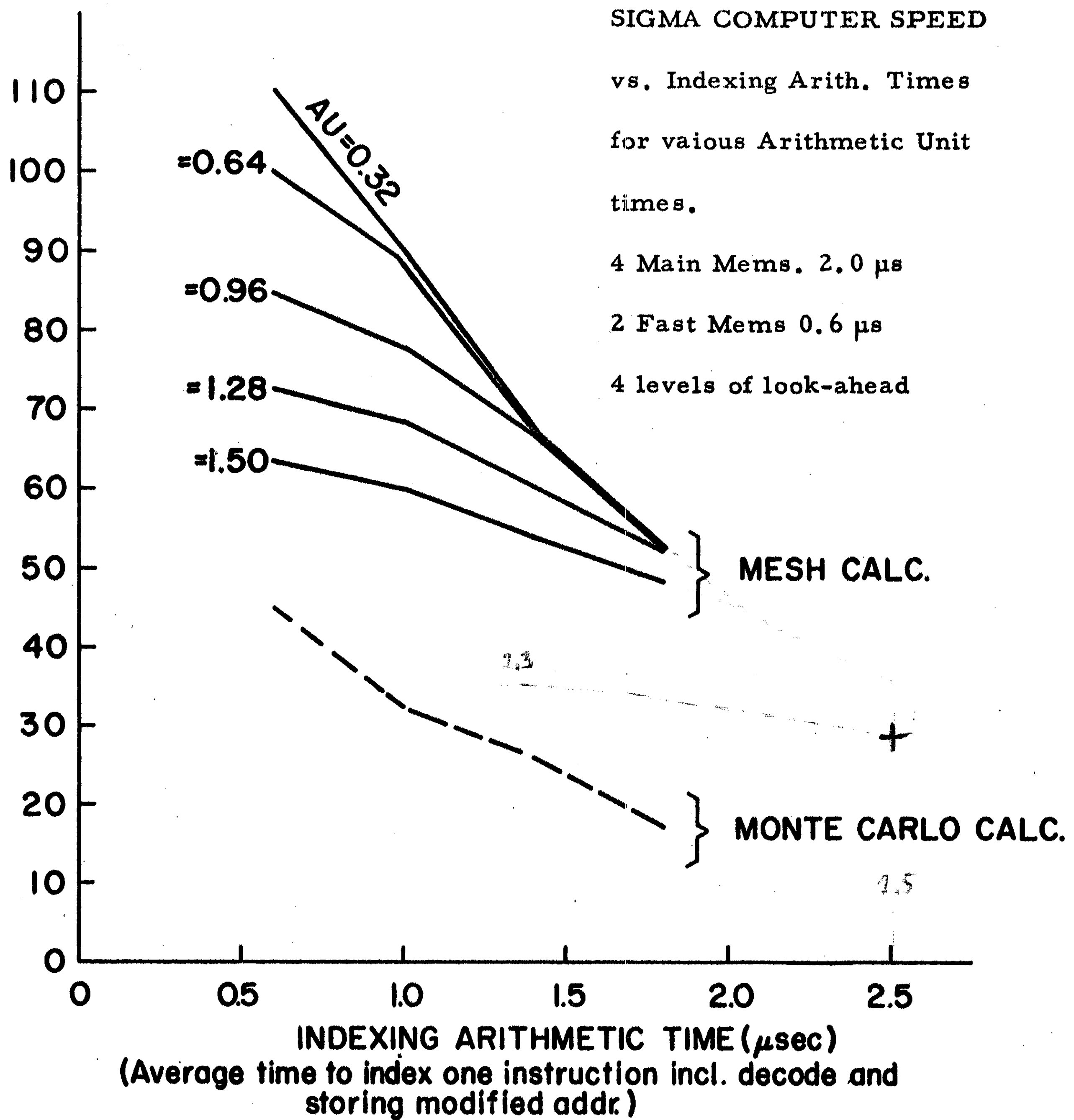
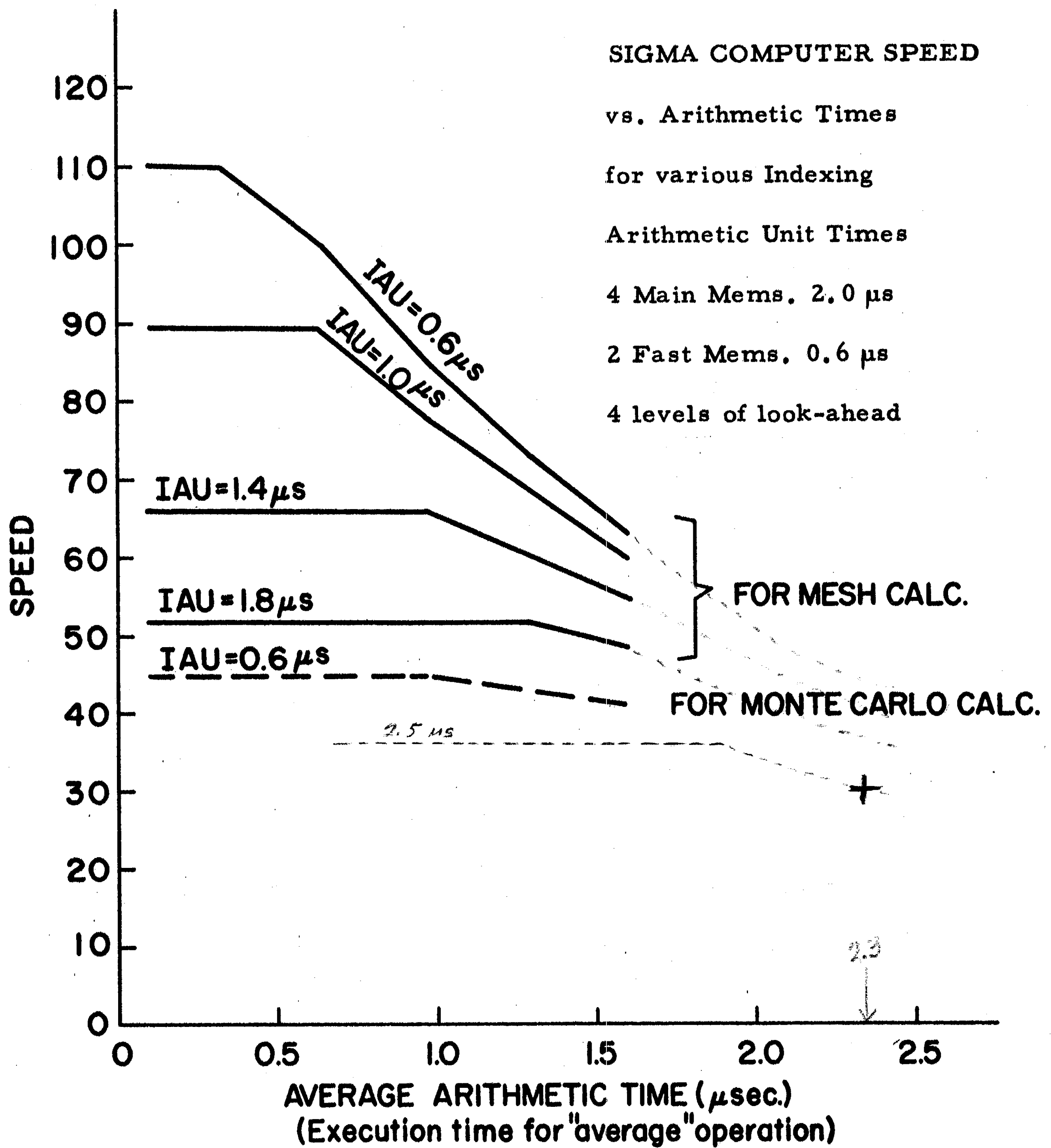


Figure 16




SIGMA COMPUTER SPEED
vs. Arithmetic Times
for various Indexing
Arithmetic Unit Times
4 Main Mems. 2.0 μs
2 Fast Mems. 0.6 μs
4 levels of look-ahead
IAU=0.6μs
IAU=1.0μs
IAU=1.4μs
IAU=1.8μs
IAU=0.6μs
FOR MESH CALC.
FOR MONTE CARLO CALC.
SPEED
120
110
100
90
80
70
60
50
40
30
20
10
0
0
0.5
1.0
1.5
2.0
2.5
AVERAGE ARITHMETIC TIME (μsec.)
(Execution time for "average" operation)

The Monte Carlo problem is much less sensitive to arithmetic speed than is the Mesh problem. Their roles are reversed for the indexing arithmetic speed since the indexing arithmetic unit controls the rate of instruction preparation and the Monte Carlo problem is instruction-access limited.

6. Speed vs Instruction Memory Speed and Instruction Buffering

Figure 18 shows the effect on overall performance of the instruction memory cycle time. The most striking result shown is the reduction in speed of the Mesh Problem with the removal of the indexing arithmetic unit instruction buffer.

Not only as the speed of the problem cut almost by a factor of two, but it clearly assumes the behavior of an instruction-access-limited problem instead of a compute-limited problem. This instruction buffer (called Y_2 in STRETCH) really serves as a 2 level Virtual Memory for the indexing arithmetic unit and gives many of the same advantages to instruction preparation which the regular Virtual Memory does to data preparation.

For more detail concerning instruction memory speed see the section on the Half microsecond memory below.

7. Arithmetic Unit Efficiency

One fallacy which is frequently quoted is that the goal of improved computer organization is to increase the arithmetic unit efficiency. Actually there are two reasons why this is not the goal in itself. The first is that arithmetic efficiency depends strongly on the mixture of arithmetic and logic in a given problem so that a general purpose computer cannot hope to give equally high percentage utility to all.

The second reason is apparent in Figure 19 which shows that the best way to increase the arithmetic unit efficiency is to slow down the arithmetic unit!

The real goal of improved organization is maximum overall computer performance for minimum cost. One will tend to increase the arithmetic unit speed as long as its percent efficiency is reasonable for a variety of problems. One will stop this process when the overall performance gain no longer matches the increase in hardware and complexity. Thus the arithmetic unit efficiency is a by-product of this design process not the prime variable.

Figure 18

SIGMA COMPUTER SPEED

vs. Instruction Memory Time

4 Main Mems. 2.0 μs

2 Fast Mems. - (varied)

4 Levels of look-ahead

0.6 IAU time

0.64 AU time

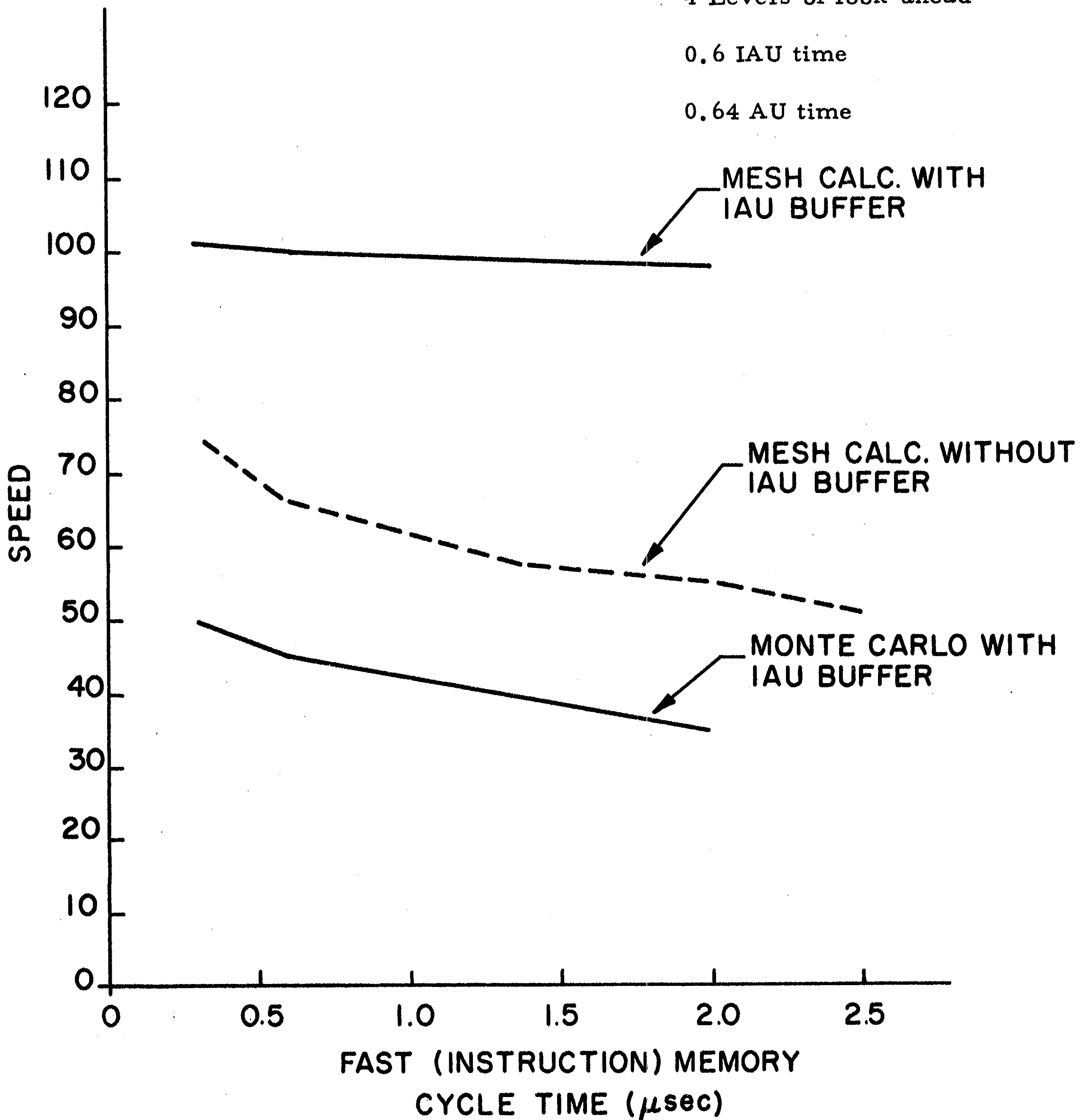


Figure 19

SIGMA ARITHMETIC UNIT EFFICIENCY

vs. Ave. Arithmetic Time

for various cases

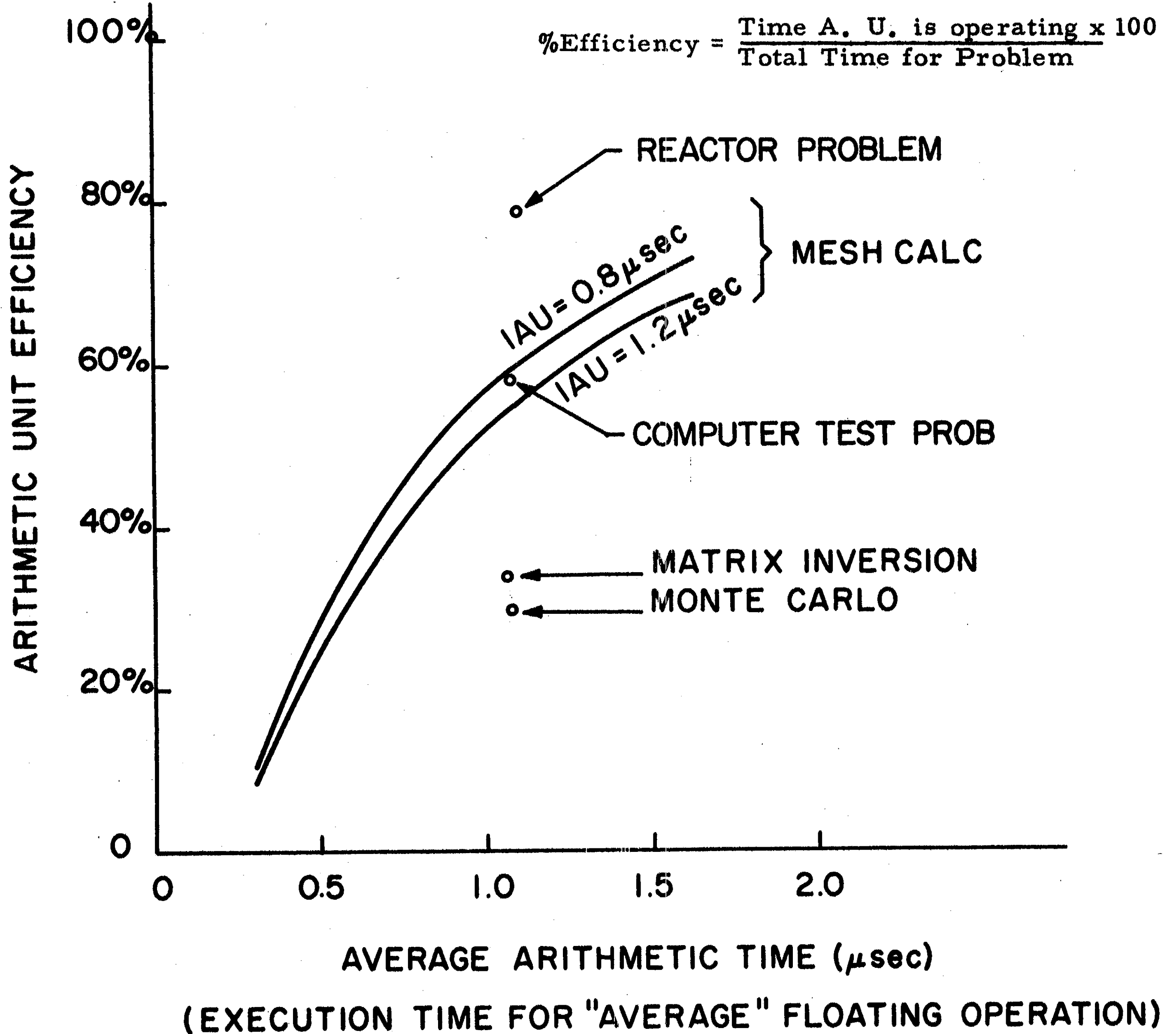
1 Index Mem. 0.8 μs

4 Main Mems 2.0 μs

2 Fast Mems 0.6 μs

4 levels of look-ahead

$$\%\text{Efficiency} = \frac{\text{Time A. U. is operating}}{\text{Total Time for Problem}} \times 100$$



8. Speed vs Concurrent Input-Output Activity

Because of the relative time scales of I/O activity and the CPU processing speeds the Simulator cannot take in account the availability or non-availability of data from I/O on the program being run. However, we can observe the effect on the computation of the I/O devices operating at different rates simultaneously with computing.

Using the STRETCH control word philosophy it is possible to have a number of input-output units operating at the same time the Central Processing Unit is running. The Basic Exchange can reach a peak rate of 1 word every 10 microseconds. The high speed disk normally operates at 1 word every 4 microseconds. Since the mechanical devices take priority over the CPU in addressing memory, the computation slows down because of memory-busy conflicts.

Figure 20 shows an example of how internal computing speed is slowed as the I/O word rates are varied continuously. At the theoretical "choke off" the I/O devices take all the memory cycles available and stop the calculation. Notice that this condition can never arise for any I/O rates presently attainable.

9. Speed vs Number of Memory Units with and without High Speed Disk Running

Because there are fewer memory cycles available when there are fewer memory units, the High Speed disk unit will cause a larger percentage slow-down for a smaller STRETCH system. Figures 21 and 22 show this effect for two typical problems--one which is normally arithmetic limited and one which is instruction-fetch limited. The former is less sensitive to such interference mainly because the Virtual Memory has more of an averaging effect on its data memory references.

The following table shows the approximate quantitative reduction in internal computing speed caused by the disk running at the same time, using the speed without I/O as 100% for each configuration.

Number of Memories	For Monte Carlo Problem	For Reactor Problem
6	- 5%	- 2%
4	-15%	- 4%
2	-24%	-22%
1	-55%	-55%

Figure 20

SIGMA INTERNAL COMPUTING SPEED
Percentage Reduction in Speed caused by Input-Output devices referencing memory at different rates while the calculation is proceeding.

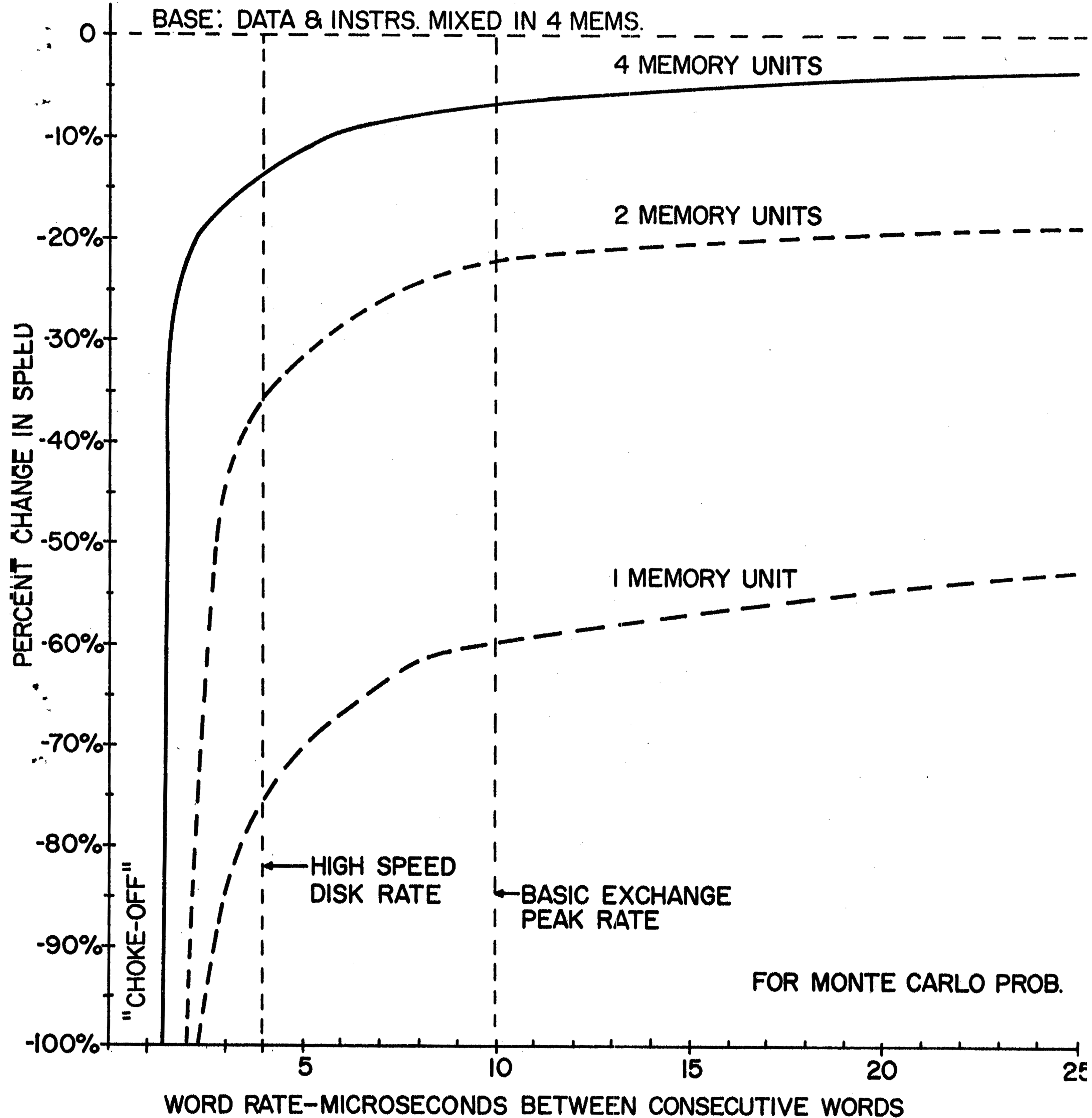
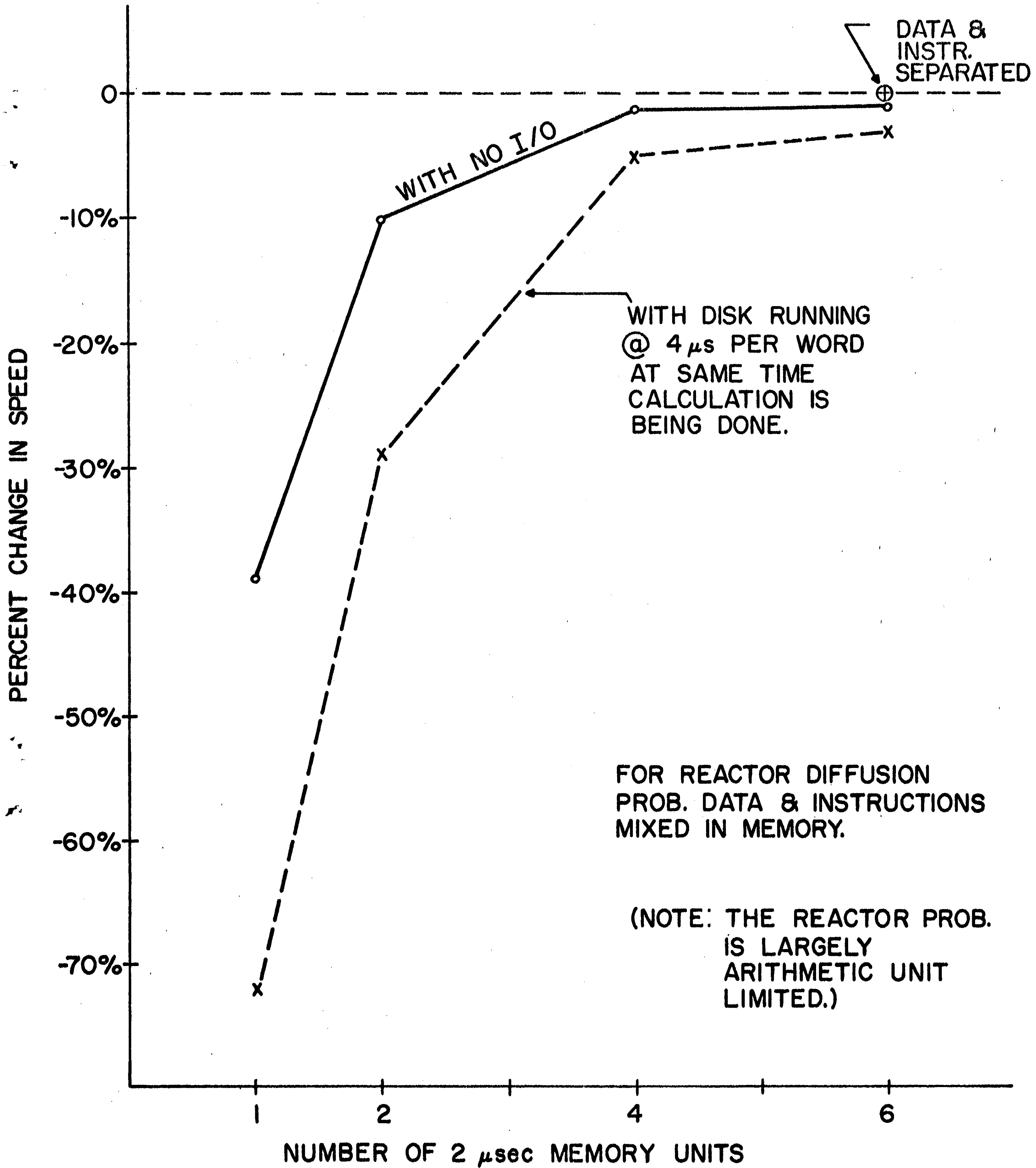
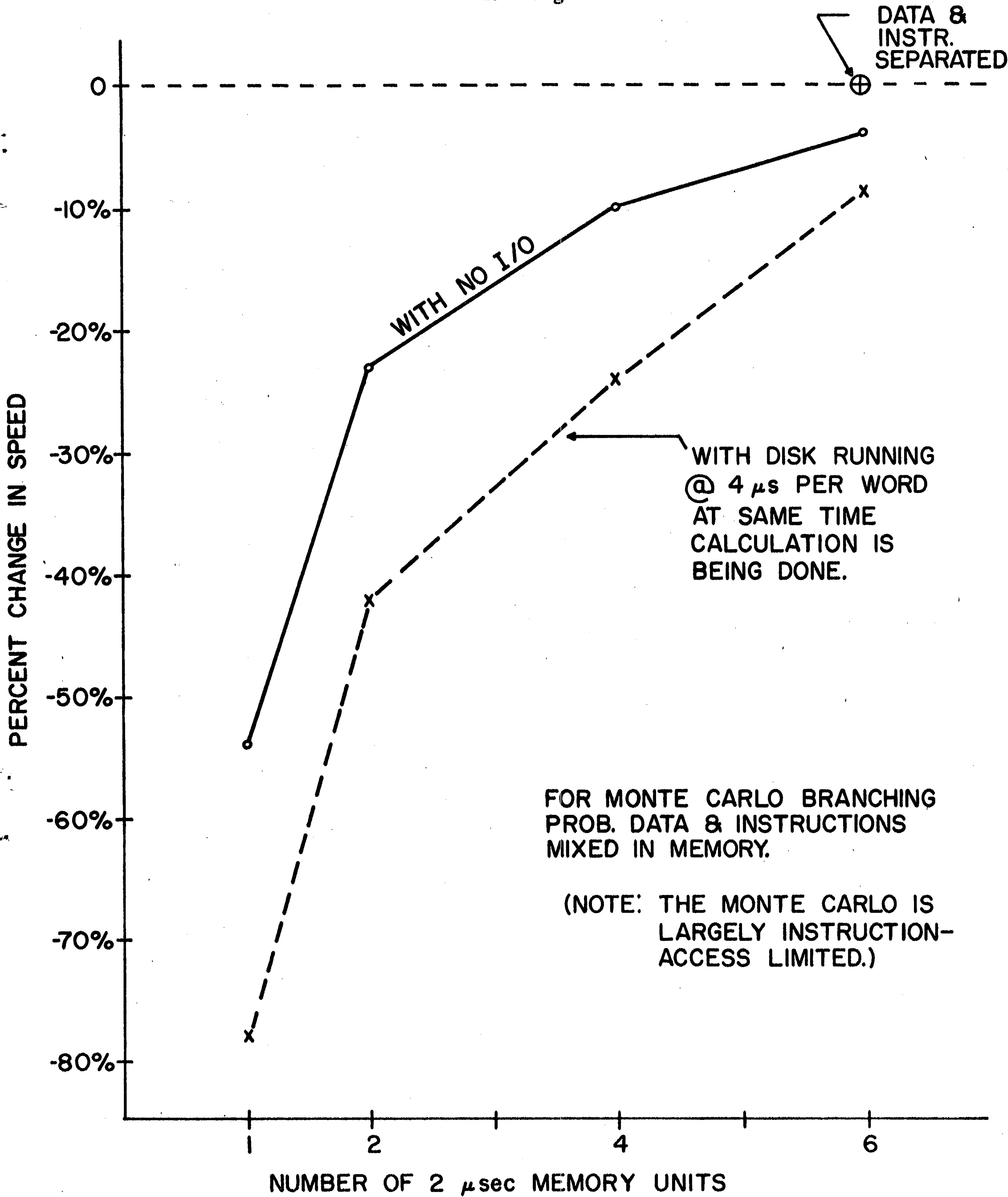


Figure 21

SIGMA Internal Computing Speed vs. Number of Memory Units with and without Disk running



SIGMA Internal Computing Speed vs. Number of Memory Units with and without Disk Running



The user of a small STRETCH system is thus penalized three times compared to a large system user: (1) The top speed of his system is reduced by the loss of memory overlap, (2) He has a larger I/O penalty when it is run concurrently with the computation, and (3) the smaller amount of data which he can hold in the memory at one time increases the amount of I/O activity he needs to do the job.

Looking at it more positively, the user who in the past purchased larger memories for his 704 obtained only the benefits of the third effect, while a STRETCH user also gets a "bonus" of the first two as he expands his system.

D. The Effect of the Half-Microsecond Instruction Memory on STRETCH Performance

1. Introduction:

During July and August of 1958 a series of Timing Simulator runs were made to evaluate the status of the SIGMA and HARVEST computers. One parameter studied was the speed of the instruction memory. Several runs were also made in which there was no separate instruction memory but instructions and data were stored in the same boxes.

The results of these runs proved to be quite important in evaluating the importance of the half-microsecond memory to the STRETCH program. The following analysis is taken from Project 7000 File Memo which was published at that time.

2. Advantages and Disadvantages of the Half-Microsecond Memory:

A. The primary advantage of the half-microsecond memory is, of course, its speed. This speed is beneficial in the following two cases:

(1) Programs can be instruction access limited either because they consist of a series of short operations, or because they contain many branch orders. If the half-microsecond memory is used for instructions it will help reduce the limitation by furnishing the instructions at a faster rate.

(2) In programs which are data-access limited, putting the data in the faster memory will cut down the time required for fetching the data. VFL operations with short fields are in this category. These are of particular importance for HARVEST applications.

In both of the above cases it is the speed of the memory compared to the arithmetic speed which is the important ratio--the faster the arithmetic speed the faster the memory required to service it properly.

B. The main disadvantage of the half-microsecond memory is its size. Each memory box contains only one-sixteenth as many words as a comparable two microsecond memory box. This decreased size certainly must result in reduced performance because more time will be spent reallocating programs. Unfortunately this reduction cannot be evaluated quantitatively by simulation since it depends on the nature of the future problems, and on the nature of future methods of scheduling machine use.

There is another advantage in larger memories which is even harder to evaluate and that is the removal of programming restrictions which exist when programs must be cut to fit a small memory.

3. Simulator Input Data:

The test problems were run with most of the recent design changes simulated, including the 0.8 microsecond I-Box repetition rate and the 0.2 microsecond bus slots. The arithmetic speeds used for the runs were:

	STANDARD	SIGMA	HARVEST
Load, Store	0.2 us	0.4 us	0.4 us
Floating Add	0.6	1.0	1.0
Floating Multiply	1.2	2.5	7.5
Floating Divide	1.8	7.0	7.5
6-6-3-1 average	0.64	1.43	2.40

The average times listed on the last line are used for convenience of plotting only. These arithmetic speeds are not intended to represent present STRETCH values.

4. Results:

Results of some of the runs are given in Table V. A short summary of the pertinant results are given in Table VI.

Straight averages of the percentage losses do not tell the whole story. There are abrupt changes in behavior for some of the problems from one case to another. Upon examination, the reason in each case was due to the problem becoming instruction-access limited where it had previously been arithmetic limited. Each problem crosses over under different circumstances because of its own particular combination of instructions.

Table VI also lists the programs which seem to be instruction-access limited for each memory and arithmetic speed configuration.

The phenomena which has been observed so many times before, still holds here---the higher the machines overall performance, the more sensitive it becomes to each individual component's performance. Thus, all of the problems are prone to become instruction-access limited at STANDARD speeds, where only the faithful Monte Carlo code is limited at HARVEST speeds.

The magnitude of the losses must be considered as well as the pattern. Clearly the memory interferences caused by not having a separate instruction memory is as large or larger than the speed of the memory. The average percentages are given in table VII.

5. <u>Rough Estimate of the Effect of Having a Larger Instruction Memory on Computer Speed:</u>

As was mentioned in section 1, the favorable speed advantage gained by having a larger instruction memory is hard to assess quantitatively. The following is intended to be a rough order-of-magnitude estimate only.

In a given time T, assumed to be long enough to do several problems, the computer will divide its activities between the time spent on useful calculation and the time spent on swapping codes in and out of instruction memory. We may write

$$T = n\, t_c + n\, R_{tc} = ntc\,(1 + R)$$

where n = the number of useful instructions executed

R = the ratio of the number of words swapped per useful instruction executed. (R should be much less than 1)

t_c = average time per calculation executed.

(For simplicity the time for swapping an instruction is taken the same as t_c.)

The speed of the computer, S, is proportional to n/T, the number of useful operations per unit time. So we may write the ratios of the speeds of two systems as:

$$\frac{S_2}{S_1} = \frac{n_2}{n_1} = \frac{tc1\ (1 + R_1)}{tc2\ (1 + R_2)}$$

The tc_1/tc_2 factor is the regular speed-up caused by the faster memory. The term involving the R's is the new factor resulting from the effect of swapping codes. As a guess, we can take R as being inversely proportional to the memory size, so that

$$R_1 = R_2 \frac{N_2}{N_1}$$

also since the R's are both much less than 1, we may write

$$\frac{S_2}{S_1} = \frac{tc_1}{tc_2} \times (1 + R_2 \frac{N_2}{N_1} - R_2)$$

In the present case, consider a 10% computer speed differential on tc's between the half and two-microsecond memories, which differ in size by a 1 to 16 ratio. We can ask what value of R_2 will be necessary to make the half microsecond memory result in an increase in speed over the two microsecond memory. The answer is approximately:

$$R_2 \lesssim 0.1$$

That is, each instruction in the half microsecond memory must be used at least 10 times in an average program before it is replaced in order that the half microsecond memory show a net increase in speed over the larger, slower 2 us memory.

Very roughly speaking, each instruction must be used at least once for each percent loss in speed under the configurations tested here to break even. It seems likely that this condition will be easily satisfied in practice, so that the faster memory will indeed result in a faster computer even though part of its advantage is lost.

The other factor mentioned which favors larger memories is the effect of being able to write less complicated codes when they need not be cut to size. One can express this factor as a $(1 + f)$ term times the speed of the computer to gives its effective speed. This speed gain is because the machine has to do a fraction f fewer instructions to accomplish the same job with a larger memory as it would take with the smaller. Since this fraction is so strongly a function of the problem involved, one can only guess what it will be as an average for all SIGMA problems. It should be in the 0 to 10% range, however.

6. Conclusions:

(a) Whether a problem is instruction-access limited or not is the main property which determines its behavior under changes in instruction memory.

(b) The property of being instruction-access limited depends considerably on the individual sequence of instructions in a problem itself, and on the relative speeds of the arithmetic unit and the instruction memory.

(c) The higher the performance of the computer, the more sensitive is its speed to changes in instruction memory configuration. At the SIGMA speeds, replacing the two 0.6 us memory boxes by two 2.0 us memories results in an average of 2.5% loss in performance in the cases tested.

(d) At SIGMA speeds, intermixing data and instructions causes an average loss of 3.9% in performance over having a separate 2.0 us instruction memory. This is because conflicts between data and instructions delay instruction accesses. Note that this is larger than the effect of memory speed itself.

(e) The speed gains from having a faster memory are reduced somewhat by the fact that it is smaller and more time must be spent swapping codes. This seems to be a small effect timewise, however.

The effective performance increase possible because bigger programs may be put into the larger memory at once is hard to assess. It is probably also in the 1 to 10% area.

TABLE V

Computer Performance as functions of Memory Configurations and Arithmetic Unit Speeds.

SPEED OF TEST PROBLEMS (times 704 Speeds)

Configuration	Mesh		Monte Carlo		Reactor		Computer Test		Simultaneous Equations	
Instruction Memories	Speed	%	Speed	%	Speed	%	Speed	%	Speed	%
STANDARD										
1. 2 0.6 us Mems.	85.9	0	45.5	0	122.4	0	89.5	0	48.7	0
2. 2 2.0 us Mems.	82.9	-3.5	41.2	-9.5	121.3	-0.9	87.6	-2.1	47.5	-2.5
3. No. Instr. Mem.	74.7	-13.1	36.3	-20.2	97.5	-20.3	73.2	-18.3	45.0	-7.4
SIGMA										
1. 2 0.6 us Mems.	59.2	0	41.9	0	75.9	0	58.1	0	45.1	0
2. 2 2.0 us Mems.	59.0	-0.4	38.2	-8.8	75.7	-0.3	57.9	-0.4	43.8	-2.8
3. No. Instr. Mem.	55.2	-6.8	35.4	-15.4	75.6	-0.4	55.0	-5.3	43.2	-4.2
HARVEST										
1. 2 0.6 us Mems.	39.3	0	38.8	0	38.0	0	40.8	0	31.7	0
2. 2 2.2 us Mems.	39.2	-0.2	35.6	-8.2	37.9	-0.1	40.7	-0.3	31.6	-0.4
3. No. Instr. Mem.	37.5	-4.6	33.8	-12.9	37.9	-0.2	40.7	-0.3	31.3	-1.2

Each has 4 0.6 microsecond total cycle instruction memory
2.0 microsecond total cycle data memories

TABLE VI

Summary of Results: Average Computer speed changes caused by Instruction memory speeds and Arithmetic Speeds, straight averages for all five test problems.

STANDARD AU Speeds	Average Percent Decrease	Problems*which are Instr. - access limited
1. 2 1/2 us Mems.	0	(2)
2. 2 2 us Mems	-3.7%	(2) (4) (5)
3. No. Instr. Mem.	-15.9%	(1) (2) (3) (4) (5)
SIGMA AU Speeds		
1. 2 1/2 us mems	0	(2)
2. 2 2 us Mems	-2.5%	(2) (5)
3. No. Instr. Mem.	-6.4%	(1) (2) (4) (5)
HARVEST AU Speeds		
1. 2 1/2 us Mems.	0	(2)
2. 2 2 us Mems.	-1.8%	(2)
3. No. Instr. Mem.	-3.8%	(1) (2)

*The Problem numbers are those given in Section VB.

TABLE VII

Average Percentage Losses for all problems.

	Arithmetic Speeds		
	STANDARD	SIGMA	HARVEST
Ave. loss caused by replacing 0.6 us Instr. Memory by 2.0 us Memory.	-3.7%	-2.5%	-1.8%
Average additional loss caused by having no separate Instr. Memory.	-12.2%	-3.9%	-2.0%
Maximum loss caused by replacing 0.6 us Instr. Memory by 2.0 us Memory	-9.5%	-8.8%	-8.2%
Max. additional loss caused by having no separate Instr. Memory.	-19.5%	-6.6%	-4.7%

E. A Study of Branching on Arithmetic Results in STRETCH

1. Introduction:

The asynchronous organization of STRETCH allows many of the components of the Computer System to be operating at the same time on different jobs and thus by overlapping greatly increases the overall efficiency of the system.

Unfortunately this organization also has its drawbacks. In particular, one of the curses of the non-sequential preparation and execution of instructions is that if there is a Branch in the problem code it spoils the smooth flow of instructions to the Indexing Arithmetic Unit. Any branch in a program will cause some delay, but the ones which hurt the most are the branches on arithmetic results which cannot be detected by the Indexing Arithmetic Unit in advance.

2. Ways in Which Arithmetic Result Branches can be Handled:

There are two fundamental ways in which branches on Arithmetic Unit results can be handled by the computer:

(1) The computer can stop the flow of instructions until the Arithmetic Unit has completed the preceeding operation so that the result is known, then fetch the next correct instruction. This places a delay on every AU result Branch whether taken or not.

(2) The computer can "guess" which way the branch is going to go before it is taken and proceed with fetching and preparing the instructions along one path with the understanding that if the guess was wrong, these instructions must be discarded and the correct path taken instead.

Under the second alternative there are four possible ways in which the guessing can be made. The branches in question are indicator branches on the Arithmetic Unit result indicators. These operations have a modifier which allows the branch to be taken either if the specified indicator is on or off. Since one can guess that the indicator is on or off for each, the four combinations are:

Case	Name	Operation		Guess	Assumed Result of Operation
I	NN-FF	Ind Branch	on	Ind on	branch
			off	off	branch
II	NF-FN	Ind Branch	on	Ind off	no branch
			off	on	no branch
III	NN-FN	Ind Branch	on	Ind on	branch
			off	on	no branch
IV	NF-FF	Ind Branch	on	Ind off	no branch
			off	off	branch

3. Simulation Results:

To study the effects of wrong-way branches on the SIGMA Timing Simulator, the Monte Carlo Branching Code was chosen as the guinea pig. The code was rewritten so that every arithmetic result branch was a wrong guess and again so that every one was guessed correctly. (Note that neither of these extremes is actually possible in a program with branches unless they are essentially unconditional.)

Several runs were made varying the instruction memory speed and the AU and IAU times. The regular (NF-FN) case had two wrong branches out of thirteen encountered in one loop of the program which consists of fifty-nine operations executed per loop.

By examining the timing charts drawn by the Simulator for many of the individual branches, the average time delays listed in Table VIII were derived.

Table VIII: Average Time Delay per Individual Branch

assumed	Guessed	For 0.6 us Instr. Mem.	For 2.0 us Instr. Mem.
no branch	right	0 us	0 us
no branch	wrong	2.5 us	3.2 us
branch	right	1.5 us	3.2 us
branch	wrong	3.7 us	4.8 us

For "Standard" Times (AU = 0.64 us, IAU = 0.6 us)

If one takes the actual times to complete the problem in each case and divides the total delay by the number of wrong-way branches, one obtains the times listed in Table IX. The approximate delay due to the memory interferences, etc., caused by starting the processing of the wrong instructions, can be estimated by comparing the times in Table VIII with those in Table IX. These interference times are listed in Table IX.

Table IX. Average Time Delay in Total Problem per Wrong-way Branch

	0.6us Instr. memory.	0.2 Instr memory:
For "Standard Times" (AU=0.64 us, IAU=0.6 us)	2.9 us	3.5 us
For "Recommended Times"(AU-1.09 us, IAU-0.9us	3.6 us	4.3 us
Extra Delay due to memory Interferences	0.5 us	1.0 us

Presumably if one holds up on every branch (Case O) the time loss will be about that of assuming no Branch and guessing wrong. (line 2 in Table VIII). If one guesses according to one of the four other cases, the time loss will depend on (1) the percentage of branches which are Br-ons, (2) the percentage of Br-ons which are actually taken, and (3) the percentage of Br-offs which are actually taken.

The calculation will be delayed by each branch taken even when they are guessed correctly, however, since we are interested in examining the additional time lost due to guessing wrong or holding up, the delays due to correct branching should be removed. The following times in Table X may be used to compute actual combinations of branches.

Table X: Average Time Delay per Branch

Computer Guessed	Should Have Guessed	0.6 us Instr. Memory	2.0 us Instr. Memory
Hold up	no branch	2.2 us	1.6 us
Hold up	branch	2.5 us	3.2 us
no branch	no branch	0 us	0 us
no branch	branch	3.0 us	4.2 us
Branch	branch	0 us	0 us
Branch	no branch	2.7 us	2.6 us

The temptation in evaluating the individual cases is to assume 50% for all the combinations and essentially average the time losses. Actually, by examining a few problems superficially, we have found that considerably fewer than half the arithmetic result branches encountered in a code are actually taken. About 20% seem to be more typical. This seems to be due to the tendency of coders to think of the branches as being exceptional cases. They normally write the main flow of the code continuously and the exceptions elsewhere.

There seems to be a tendency to link indicators turning on with exception cases. In time this would result in fewer Br-ons being taken and more Br-offs being taken. These generalizations are admittedly uncertain mainly because very few relevant statistics are available.

There is also a "feedback" in such statistics because the way in which the machine guesses the branches will influence future programmers to write their codes to take advantage of the speed gain, so that the statistics of the future will be biased in favor of the system chosen now!

Table XI compares the five cases for several assumed values of percentages. The last two lines are my guesses as to the averages to be expected.

Table XI: Average Time Delays per Branch for the Different Cases

% Br-ons	% Br-ons taken	% Br-offs taken	Case 0 Hold-up	Case I NN-FF	Case II NF-FN	Case III NN-FN	Case IV NF-FF
for 0.6 us. Instruction Memory							
50%	50%	50%	2.35 us	1.30 us	1.45 us	1.38 us	1.38 us
50%	20%	20%	2.26	2.14	0.54	1.33	1.42 us
80%	20%	80%	2.30	1.69	0.89	1.19	0.47
for 2.0 us Instruction Memory							
50%	50%	50%	2.40 us	1.00 us	1.80 us	1.40 us	1.40 us
50%	20%	20%	1.92	1.96	0.36	1.16	1.64
80%	20%	80%	2.11	1.45	0.84	2.22	0.10

4. Conclusions:

(1) The performance variation in a problem with a lot of arithmetic data branching can vary by approximately ≠ 15% depending on the way in which the branches are handled.

(2) Holding-up on every branch seems to be less desirable than any of the guessing procedures.

(3) It is very unlikely that one ever get fewer than 15% or more than 85% wrong-way branches regardless of his procedure.

(4) It seems possible to get a fairly low loss by picking Case IV, provided the percentages on the last line of Table IX really are correct. However, if the percentages should be different, Case IV is much more sensitive to them than Case II.

(5) To be really effective Case IV needs the existance of the indicators ≥ 0, ≤ 0 to make the distinction between off and on precise. At present one must code "Br-on ≤ 0", as "Br-off > 0," so that the equating of "on" to "exceptional case" is spoiled somewhat.

(6) The highest performance would be obtained if each branch had an extra "guess bit" which would permit the programmer to specify which way he estimates each branch will most likely go. This seems to be impossible in the present format schemes. It also would place a considerable extra burden on the programmer for the gains promised.

5. Recommendations Finally Presented as a Result of the Simulator Runs

Case II (NF-FN) should be adopted as the guessing scheme. This means that for any branch for which the IAU cannot compute the correct outcome, it should guess that the branch is not taken and proceed with the processing of the next instruction.

Case II was chosen over case IV because:

1) Its time loss is low (at least second best)

2) It does not require special controls for deciding whether to assume a branch is taken or not

3) It does not require that new indicators be defined.

4) It should not confuse the programmer with complicated rules of coding the way Case IV might.

VI. APPENDIX: Details of Timing Simulation Program SIM-2

The following pages give detailed symbol definitions and flow diagrams for the SIM-2 code. The diagrams accurately represented the code at the time they were drawn. There have been some additions to the program since then, particularly in the I/O simulation section, but they do not change the main logic of the flow.

The simplified flow diagram, Figure 11, shows the major sections of the program. The following pages elaborate upon this figure. The logic of the Virtual Memory operation is described in Section III. The logical diagrams given there have direct counterparts in the flow diagrams which follow.

STRETCH Timing Simulator Program SIM-2
List of Quantities Used in Flow Diagram

Table 1

Quantities Concerning Instructions fed through Simulator

	Instr. Input	Look-ahead Ready Reg.	Look-ahead levels	Instr. Fetch	Arith. Unit	Indexing Arithmetic Unit
Instr. Number	II1	LR3	LAU8	IMRS IRUM	IP1 NRAU1	(II 1)
Op. Code	II2	LR1	LAU2	ICAN	NRAU2	(IST = "STATE")
Instr. Location	II3		LAU1	ICR1 ICR2		
#I Index Addr.	II4					IRO
#II Index Addr.	II5					IRO
Data Address	II6	LR2	LAU3	IBUG2		IRO
Special Desig.	II7					
Return Tag	II8					
Sp. Sp. Desig.	II9					
Compare Bit			LAU4			
Forward Addr.			LAU5			
O.K. Bit			LAU6			IRO1 (Fetch) IRI (Return)
Forward Bit			LAU7			
Mem. Bring Bit			LAU9			
Unit Clock					JAUT	INS

	Main or Instr. Mem.	Index Core Mem.	Central Control Decode	Bus to Mem.	Bus From Mem.	Exchanges
Instr. Number	IMM2	IXM4	IBD1(or 6)	JF1	NF1	(90,91,92,93)
Return Addr.	IMM1	IXM2	IBD (or 5)	JF	NF	32
Bring Bit	IMM3	IXM6	IBD2(or 7)	JF2		1 or 0
Read out Clock	IMM4	IXM8				
End Sig. Clock	IMM5					
Mem. Cy. Clock	IMM6	IXM10	IBD4(or 9)	JF4	NF2	CLC
Mem. Res. Bit	IMM7					
Mem. Box No.			IBD3(or 8)	JF3		IOM
				JF-fast memory JM-main memory	NF-Fast memory NM-main memory	

List of Symbols

TABLE II

Control and Tally Quantities

1. Look-Ahead Symbols:

NCTRA	Instruction Fetch Counter
NCTRB	Data Fetch Counter
NCTRC	Data Store Counter
NSTOB	Store Bit (an unexecuted Store)
NLH	Number of Look-Ahead levels
NBFR	Modular value of NCTRC

2. Conflict Counters, and Tallys:

CTT	Total Time Tally
CAU	Arithmetic Unit Tally
CIAU	Indexing Arithmetic Unit Tally
CADLA	Average depth of Look-Ahead Tally
CLAF	Look-Ahead Full Tally
CWI	Arithmetic Unit Waiting on Instruction Tally
CWM	Arithmetic Unit Waiting on Data Tally
CDLA	Look-Ahead level use Tally
CIF	In-Bus from Fast Memory Tally
CIM	In-Bus from Main Memory Tally
COF	Out Bus (Read) to Memory Tally
COM	Out Bus (Write) to Memory Tally
CIST	Index State Tally
CIMM	Memory use Tallys
CMMC	Main Memory Conflict Tally
CFMC	Fast Memory Conflict Tally
CXMC	Index Memory Conflict Tally

3. Miscellaneous Symbols:

MARK	Time Counter for Listing
BIB	Break-in Bit on Wrong-Way Branches
SKIP	Signal to "Run-Dry" at End
TALLY	Count of Number of Executed Ops.
A, B, AD, THINK	Temporary Locations
IDR, IDW, LDA	Pseudo-op Controls, etc.
LASCB	Look-Ahead, Self Compare Bit
PBIT, CBIT, PHDB TBIT, WBIT	Controls for Printing
IB	Block for Input from Control Cards
IP, RP, SIP, SSP	Various Printing Blocks

List of Symbols

Table III

Input Constants Appearing on Summary Listing

Symbol on Listing	Name in Code	Description		
LA	NLH	No. levels of look-ahead		
FM	NUFM	No. of fast memory boxes		
MM	NUMM	No. of main memory boxes		
IR	INX-4	Index reset		IAU State 4
IS	INX-3	Index store-1		IAU State 3
IA	INX-2	Index add-1		IAU State 2
ID	INX-1	Index Decode		IAU State 1
	INX	(not used)		
MB	NMBT	Main (or write) bus time		
FB	NFBT	Fast (or read) bus time		
FD	IDMT	Fast Memory bus decode time (CCU)		
HM	IDMT+1	Hamming check time		
HE	IDMT+2	High Speed Exchange word rate		
X1	IDMT+3	Index Memory read-out time		
X2	IDMT+4	Index Memory cycle time		
MD	IDMT+5	Main Memory bus decode time (CCU)		
F1	MFT 1	Fast Memory read-out time		
F2	MFT 2	Fast Memory end signal time		
F3	MFT 3	Fast Memory Total cycle time		
M1	MMT 1	Main Memory read-out time		
M2	MMT 2	Main Memory end signal time		
M3	MMT 3	Main Memory Total cycle time		
15	JT-15	Op. Code	15	Square Root
14	JT-14		14	Divide
13	JT-13		13	
12	JT-12		12	Cumulative Multiply
11	JT-11		11	Multiply
10	JT-10		10	
9	JT-9		9	Add
8	JT-8		8	
7	JT-7		7	Load
6	JT-6		6	
5	JT-5		5	
4	JT-4		4	Immediate Ops.
3	JT-3		3	Immediate Ops.
2	JT-2		2	Immediate Ops.
1	JT-1		1	Immediate Ops.
LE	JT	Low Speed Exchange word rate		

List of Symbols

Table IV

Output Results on Summary Listing

Symbol on Listing	Name in Code	Description
XMC	(CXMC)	Index Memory Conflicts (in % of total time)
TT	CTT	Total Time of problem (XXX. X microseconds)
AU	(CAU)	Arithmetic Unit busy (in % of TT
IAU		Indexing Arithmetic Unit busy
ADLA		Average depth of look-ahead
LAF		Look-ahead full
WI		Arithmetic Unit waiting on instructions
WM		Arithmetic Unit waiting on data
DLA-		% Time Look-ahead has depth specified
IBF		In bus from fast Memory busy
IBM		In bus from Main Memory busy
RB		Read bus to Memory busy
WB		Write bus to Memory busy
IS-	IST	Time spent in Indexing State specified
M-		Time Memory Box specified is busy (M12 to M5 are Main Memories, M4 to M1 are Instr. Mems.
MMC		Main Memory conflicts
FMC		Fast Memory conflicts
WBC		Write bus conflicts
RBC		Read bus conflicts

Op codes:
(1) 1 thru 4 Immediate (1 = wrong-way branch)
(2) 5 thru 34 bring type (See Table III)
(3) 35 indexing type
(4) 36 to 97 store type
(5) Instruction No. 98 Stop in AU and Tr to Summary

Return Addresses:
(1) 20 = IAU data
(2) 21 = Instruction fetch
(3) 1, 2, 3, . . . 8 = Look-ahead levels
(4) 32 = Exchange

SIM-2 Flow Diagram

(1) Initialize
(2) Arithmetic Unit
(3) Load Look-Ahead
(35) Summary Print-out

A0
START

A0 initialize
Rewind Tape #2
clear Data Mem.
set 9 → Mark
4 → ICAN
START → ICR1
1 → IST

A1
TIME CYCLE

A1 Test for halt
$IP_1 - 98$ =0 → SUMMARY
≠0

Test Instr No
A2 NRAU1 =0 → A4 NCTRA-NCTRC (Test if instr. is available) ≠0 → A15 JAUT (Test A.U. clock) =0 → A12
NCTRA-NCTRC =0 → A13
JAUT ≠0 → A13
NRAU1 ≠0

A12
NCTRC Mod(NLH) → X1
LAU8, 1 → NRAU1
LAU2, 1 → NRAU2
Transfer Instr. from Look-Ahead to A.U.
→ A13

Decode Op. Code
A7 NRAU2 − 35 − → BRING TYPE
+ or 0 → STORE or INDEX TYPE

Test Store Bit
A11 NSTOB ≠0 → A13
=0

A12 START AU
NCTRC Mod(NLH) → NBFR
1 → NSTOB
1 → JAUT
NRAU1 → IP_1
0 → NRAU1
NCTRC+1 → NCTRC
→ A13

Test for wrong-way Transfer
A10 NRAU2 − 1 ≠0 → AB10
=0
NSTOB ≠0 → A13
=0
NCTRC Mod(NLH) → X1
LAU3, 1 → ICR1
ICR1+1 → ICR2
1 → BIB
→ AC10
To wipe out Look-Ahead + start over

AB10
NCTRC Mod(NLH) → X1
Test OK bit
LAU6, 1 =0 → A13
≠0

A14 Start A.U.
NRAU2 → X1
JT, 1 → JAUT
NRAU1 → IP_1
0 → NRAU1
NCTRC+1 → NCTRC
→ A13

A13 Load Look-Ahead

Test Ready Register
A13 LR1 =0 → A17
≠0
A18 NCTRA − NCTRC − NLH =0 → A17
≠0 → NCTRA Mod(NLH) → X1

A19 NSTOB ≠0 → X1 − NBFR =0 → A17
NSTOB =0 → A23
X1 − NBFR ≠0 → A23

Put in Look-Ahead
A23
LR1 → LAU2, 1
LR2 → LAU3, 1
LR3 → LAU8, 1
0 → LAU4, 1
→ AB23

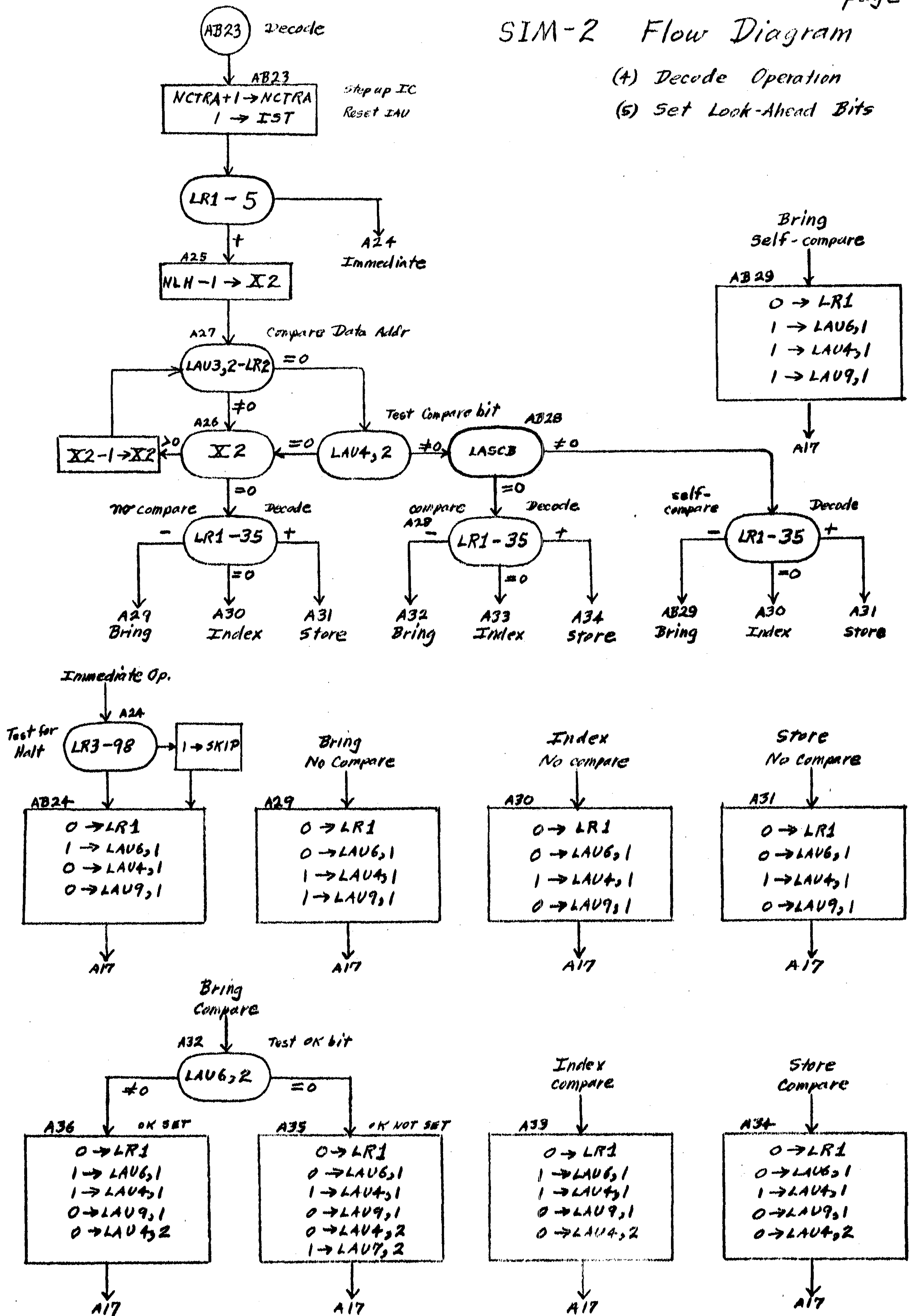
SUMM Summary Program.

SUMM
Prepare and Print Summary Table

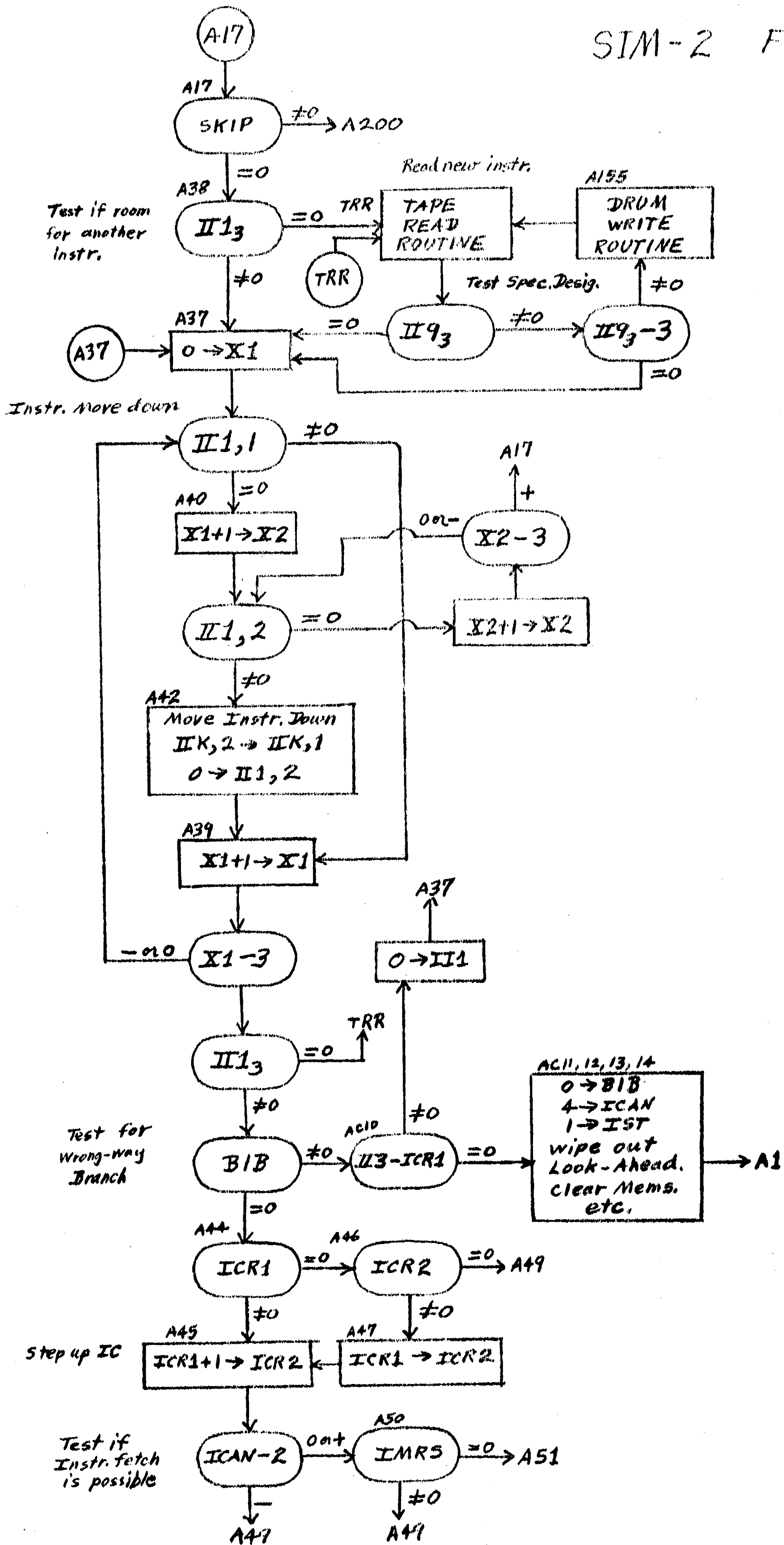
RP1
Read New Control Cards

LAST? yes → Stop End of Prob. Series
no → A0 Start Next Problem

HGK
Mar 11, '58


page 2
SIM-2 Flow Diagram
(4) Decode Operation
(5) Set Look-Ahead Bits
AB23 Decode
AB23
NCTRA+1→NCTRA
1 → IST
Step up IC
Reset IAU
LR1 − 5
A24 Immediate
A25
NLH −1 → X2
A27 Compare Data Addr
LAU3,2-LR2
=0
≠0
A26
X2
X2-1→X2
>0
=0
LAU4,2
Test Compare bit
≠0
AB28
LASCB
≠0
=0
no compare
Decode
LR1 −35
=0
A29 Bring
A30 Index
A31 Store
compare
A28
LR1-35
A32 Bring
A33 Index
A34 Store
self-compare
Decode
LR1-35
AB29 Bring
A30 Index
A31 Store
Bring Self-compare
AB29
0 → LR1
1 → LAU6,1
1 → LAU4,1
1 → LAU9,1
A17
Immediate Op.
A24
Test for Halt
LR3−98
1→SKIP
AB24
0 →LR1
1 → LAU6,1
0 → LAU4,1
0 →LAU9,1
A17
Bring No Compare
A29
0 →LR1
0 →LAU6,1
1 →LAU4,1
1→LAU9,1
A17
Index No compare
A30
0→ LR1
0 →LAU6,1
1 →LAU4,1
0 →LAU9,1
A17
Store No Compare
A31
0 →LR1
0 →LAU6,1
1→LAU4,1
0 →LAU9,1
A17
Bring Compare
A32
Test OK bit
LAU6,2
≠0
=0
A36
OK SET
0 →LR1
1→ LAU6,1
1→LAU4,1
0→LAU9,1
0 →LAU4,2
A17
A35
OK NOT SET
0→LR1
0 →LAU6,1
1 →LAU4,1
0 →LAU9,1
0 →LAU4,2
1→LAU7,2
A17
Index compare
A33
0→ LR1
1 →LAU6,1
1 →LAU4,1
0 →LAU9,1
0 →LAU4,2
A17
Store Compare
A34
0 →LR1
0→LAU6,1
1→LAU4,1
0→LAU9,1
0→LAU4,2
A17
NGK
Mar 11,'58
Rev

SIM-2 Flow Diagram

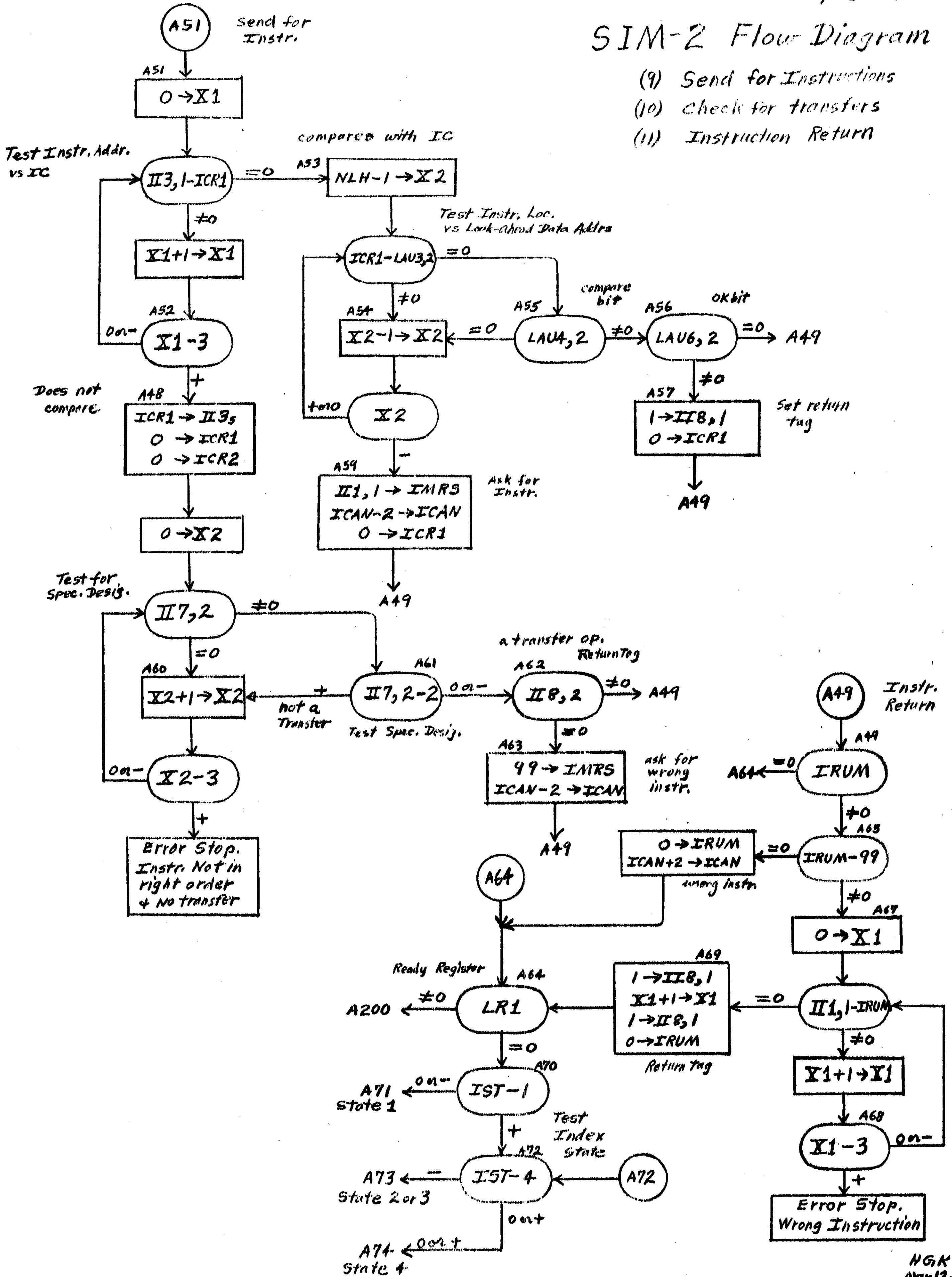


(6) Read in Instrs. from Tape.

(7) Move Instrs. down in Memory.

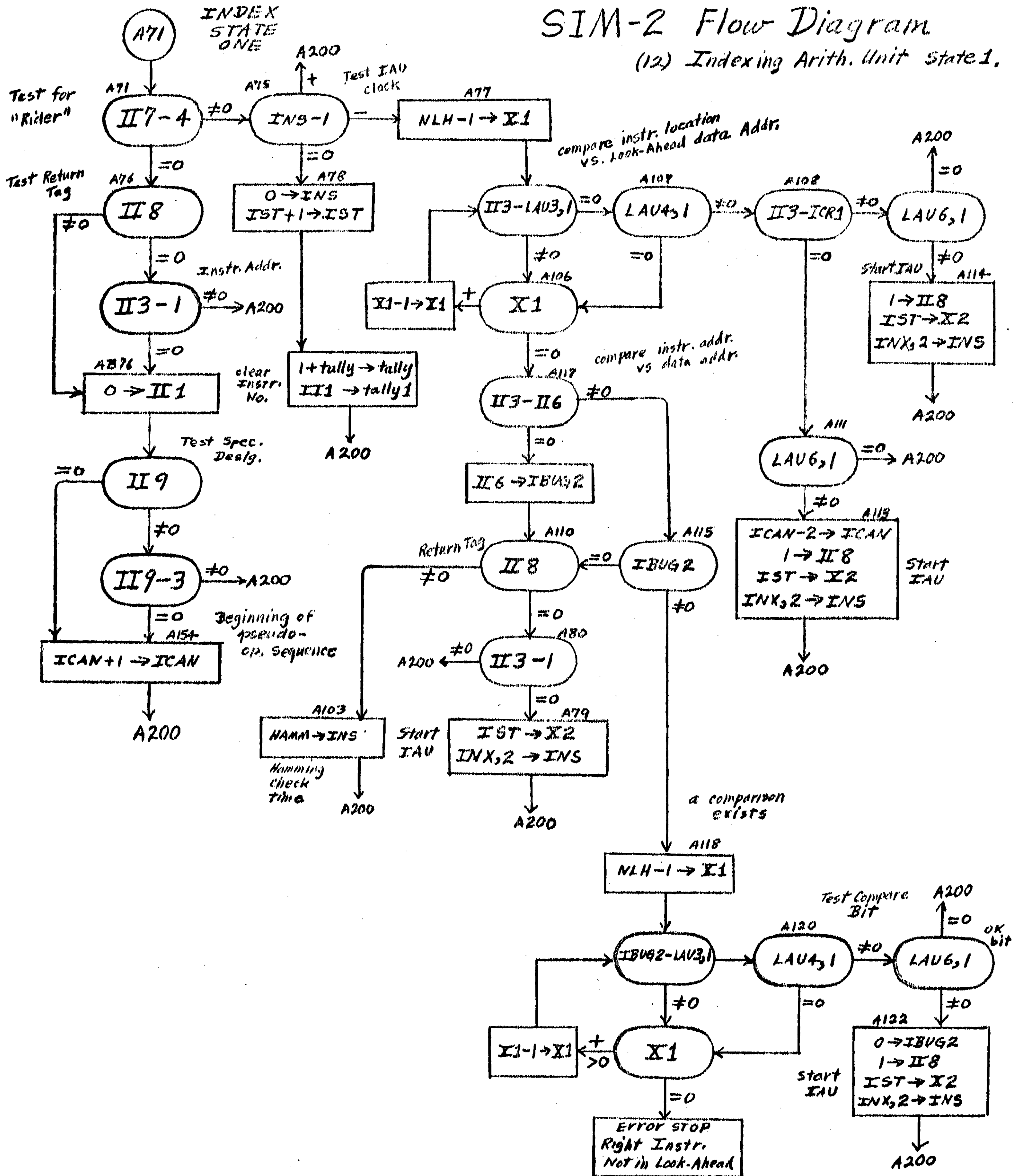
(8) Wipe out + start over on wrong-way Br

HGK
Mar 12, '58


page 4
SIM-2 Flow Diagram
(9) Send for Instructions
(10) check for transfers
(11) Instruction Return
A51
Send for Instr.
0 → X1
Test Instr. Addr. vs IC
II3,1-ICR1
compare with IC
A53
NLH-1 → X2
X1+1 → X1
Test Instr. Loc. vs Look-ahead Data Addrs
ICR1-LAU3,2
compare bit
OK bit
A52
X1-3
A54
X2-1 → X2
A55
LAU4,2
A56
LAU6,2
A49
Does not compare
A48
ICR1 → II3,5
0 → ICR1
0 → ICR2
X2
A57
1 → II8,1
0 → ICR1
Set return tag
A59
II1,1 → IMRS
ICAN-2 → ICAN
0 → ICR1
Ask for Instr.
0 → X2
Test for Spec. Desig.
II7,2
a transfer op.
Return Tag
A60
X2+1 → X2
not a Transfer
A61
II7,2-2
Test Spec. Desig.
A62
II8,2
Instr. Return
A49
IRUM
A64
A63
99 → IMRS
ICAN-2 → ICAN
ask for wrong instr.
X2-3
A65
0 → IRUM
ICAN+2 → ICAN
wrong instr.
IRUM-99
Error Stop. Instr. Not in right order + No transfer
A67
0 → X1
Ready Register
A64
LR1
A200
A69
1 → II8,1
X1+1 → X1
1 → II8,1
0 → IRUM
Return Tag
II1,1-IRUM
A70
IST-1
A71
State 1
Test Index State
A72
IST-4
A73
State 2 or 3
A74
State 4
A68
X1-3
Error Stop. Wrong Instruction
HGK
Mar 12, '58

SIM-2 Flow Diagram

(12) Indexing Arith. Unit State 1.



HGK
Mar 13, '58

SIM-2 Flow Diagram

(13) Index State 2 or 3
(14) Index State 4

A73 INDEX STATE TWO or THREE

Test IAU clock
A73 INS-1: + → A200; =0 → A82; − → A81
A82: 0 → INS; IST+1 → IST → A72
A81 IRO: =0 → A83; ≠0 → A84
A84 IRI: =0 → A200; ≠0 → A85
A85: 0 → IRI; 0 → IRO; 0 → IRO1; IST → X2; INX,2 → INS (START IAU) → A200
A83: 10(IST)+20 → AD
C(AD): =0 → A90; ≠0 → A86
A90: IST+1 → IST → A72
A86: NLH-1 → X1
C(AD)−LAU3: =0 → A88; ≠0 → A87
A88 LAU4,1 (indexed?): ≠0 → LAU6,1; =0 → A87
LAU6,1 (ok bit): =0 → A200; ≠0 → A92
A92: IST → X2; INX,2 → INS (Start IAU) → A200
A87 X1: + → X1-1 → X1 → C(AD)−LAU3; =0 → C(AD) → IRO → A200

A74 INDEX STATE FOUR

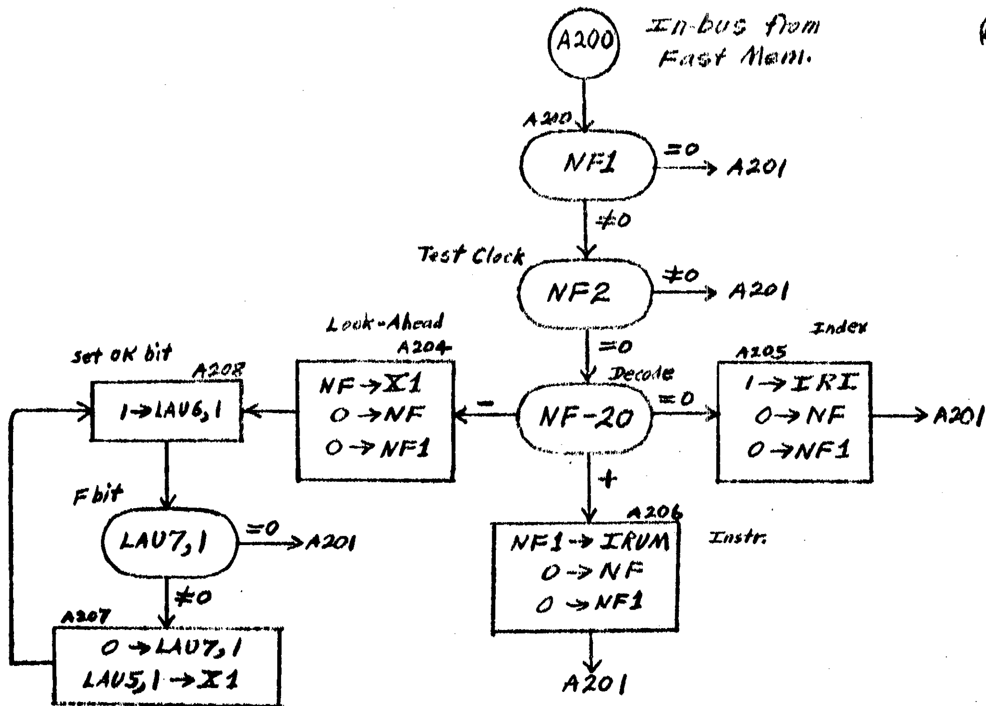
Test IAU clock
A74 INS-1: + → A200; − → A93; =0 → A94
A93: IST → X2; INX,2 → INS (Start IAU) → A200
A94 II7 (Test for spec. Desig.): =0 → A95; ≠0 → A96
A96 II7-2: + ("a rider") → A98; − → A95; =0 (a transfer) → A97
A98: 0 → II1; 1 → IST; ICAN+1 → ICAN → A200
A97: 1 → X1
II3,1-1: ≠0 → A100; =0 → A99
A100: II3,1 → ICR1 → A95
A99: X1+1 → X1
X1-3: 0 or − → II3,1-1; + → Error Stop No "1" Address
A95: II2 → LR1; II6 → LR2; II1 → LR3; 0 → II1
II9 (Test for pseudo-op.): =0 → A101; ≠0 → II9-2
II9-2 (Test end of pseudo-ops.): ≠0 → A102; =0 → A101
A101: ICAN+1 → ICAN; 1 → IST → A200
A102: DRUM READ ROUTINE; 1 → IST → A200

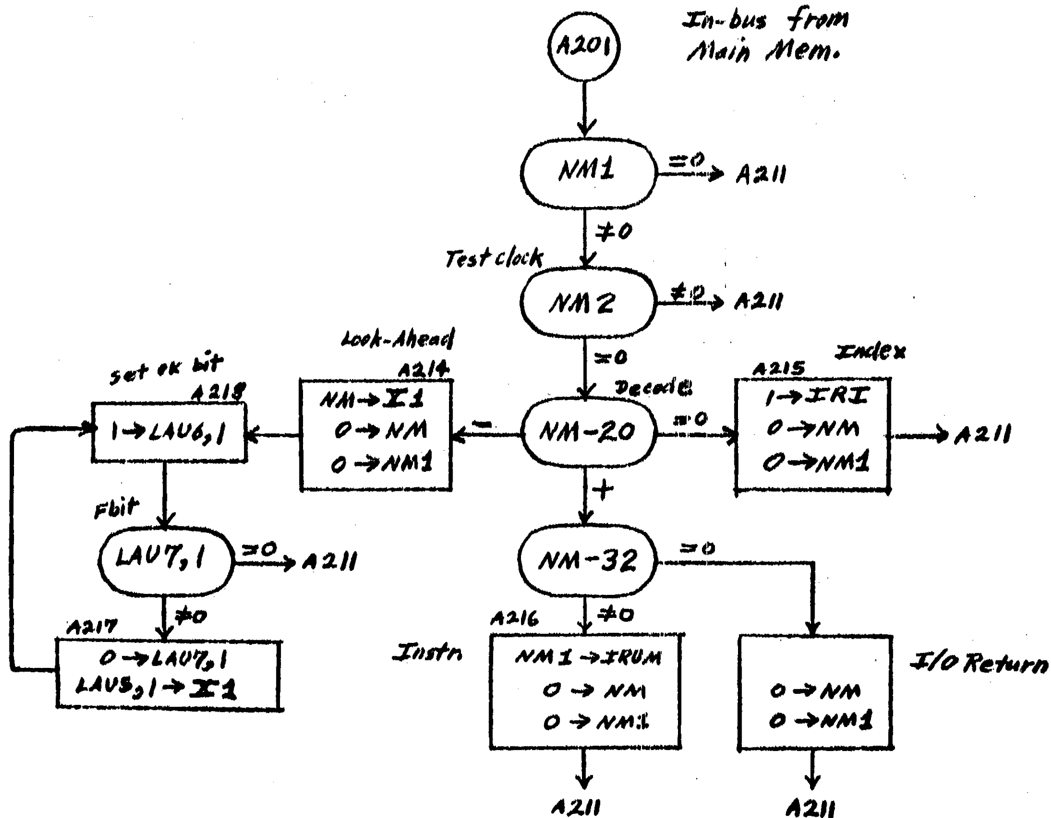
HGK
Mar 13,'58

SIM-2 Flow Diagram

(15) In-bus Fast Mem.

(16) In-bus Main Mem.

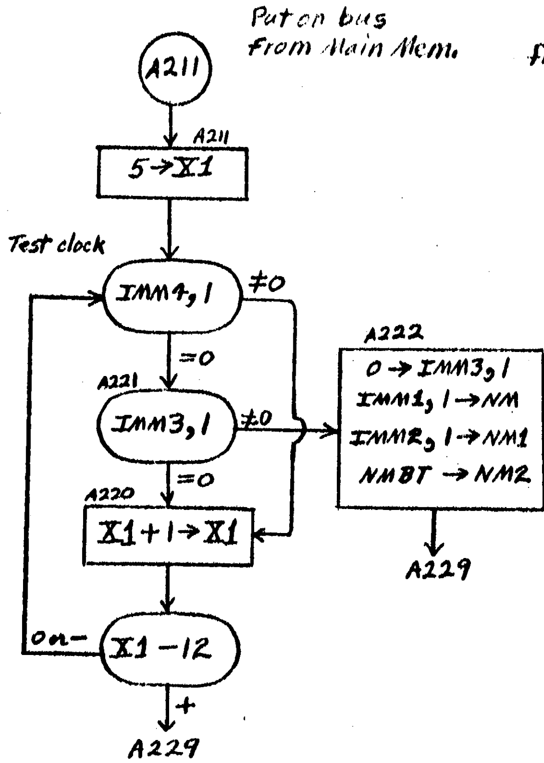


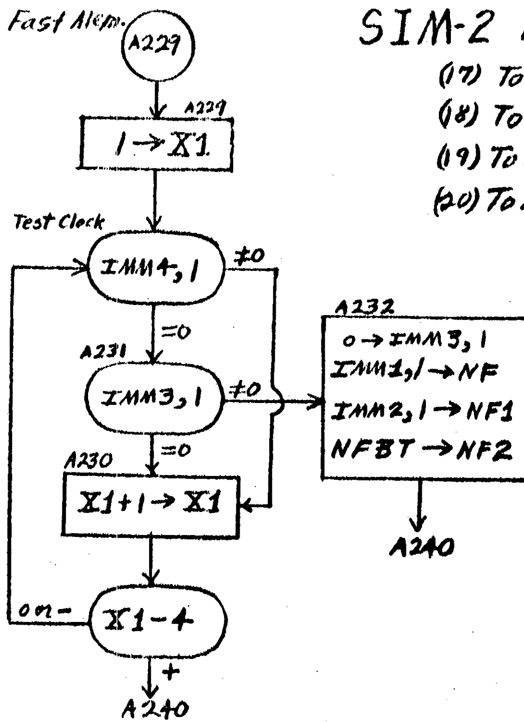


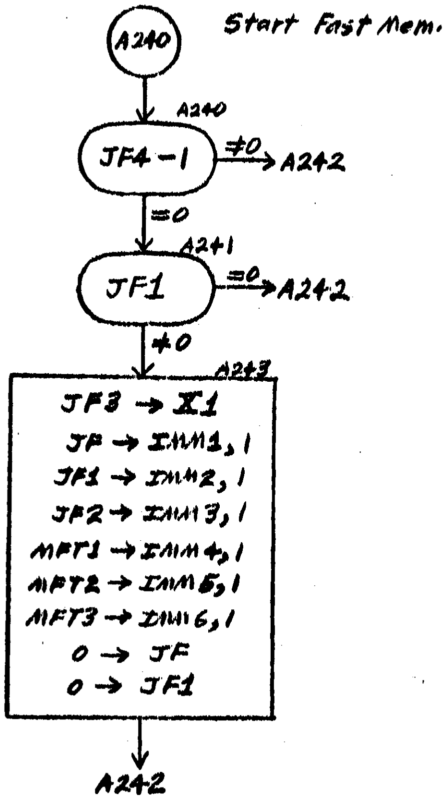
HGK
Mar 13, '58

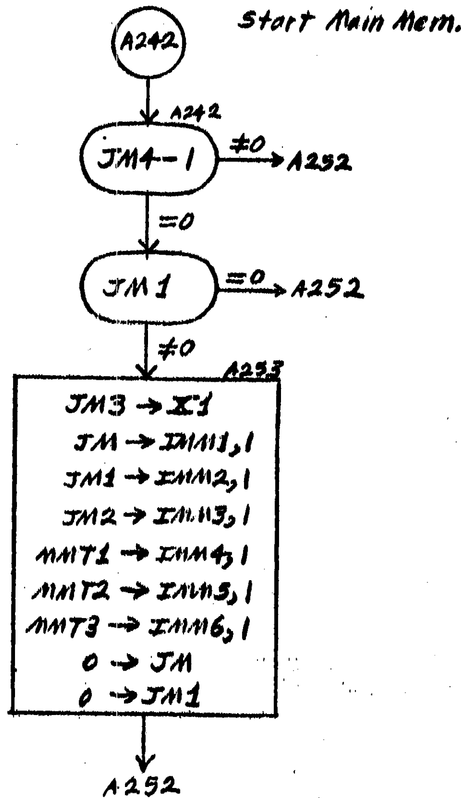
SIM-2 Flow Diagram

(17) To bus from Main Mem.
(18) To bus from Fast Mem.
(19) To Fast Mem. from bus
(20) To Main Mem. from bus





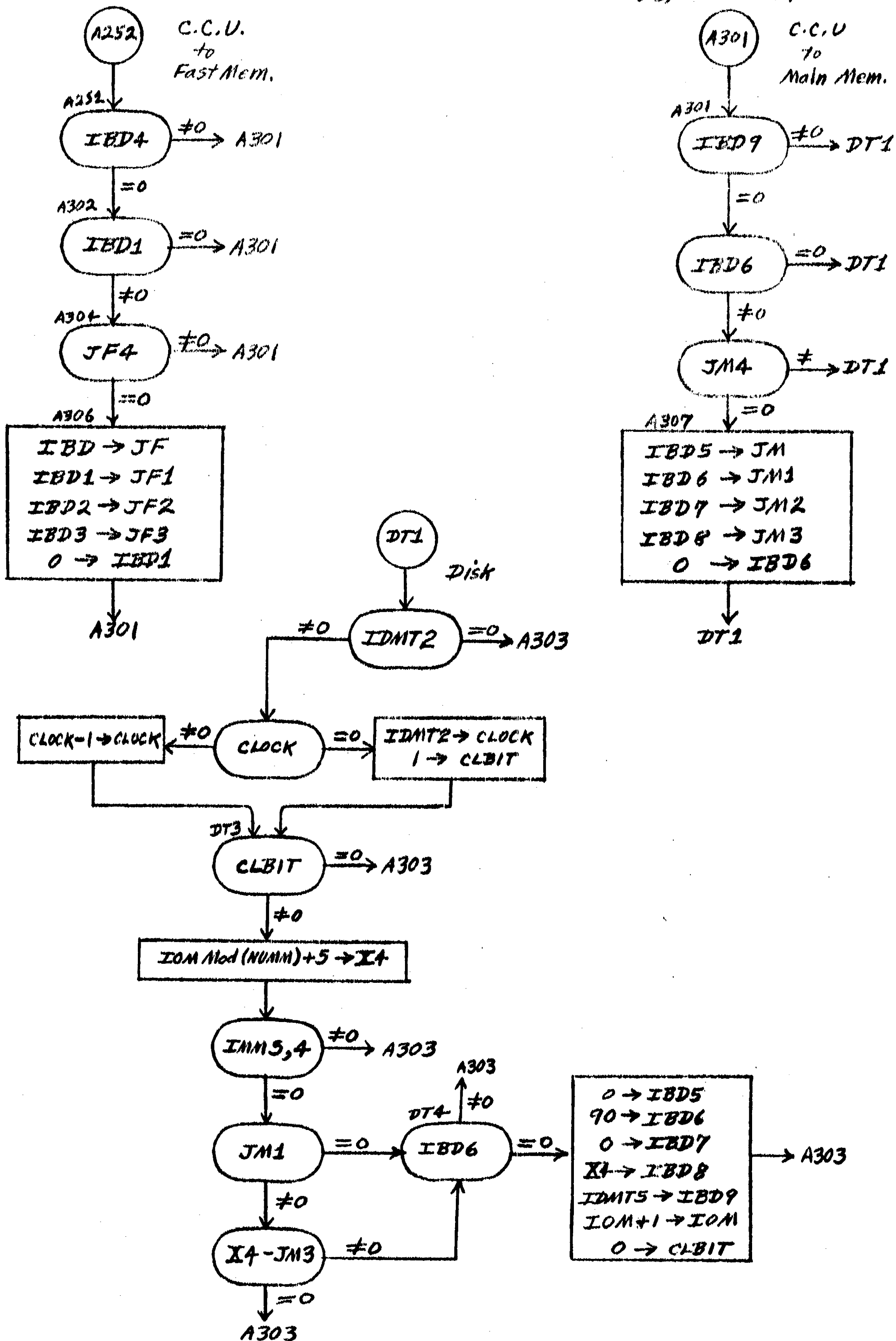




HGK
Mar 13,'58

SIM-2 Flow Diagram

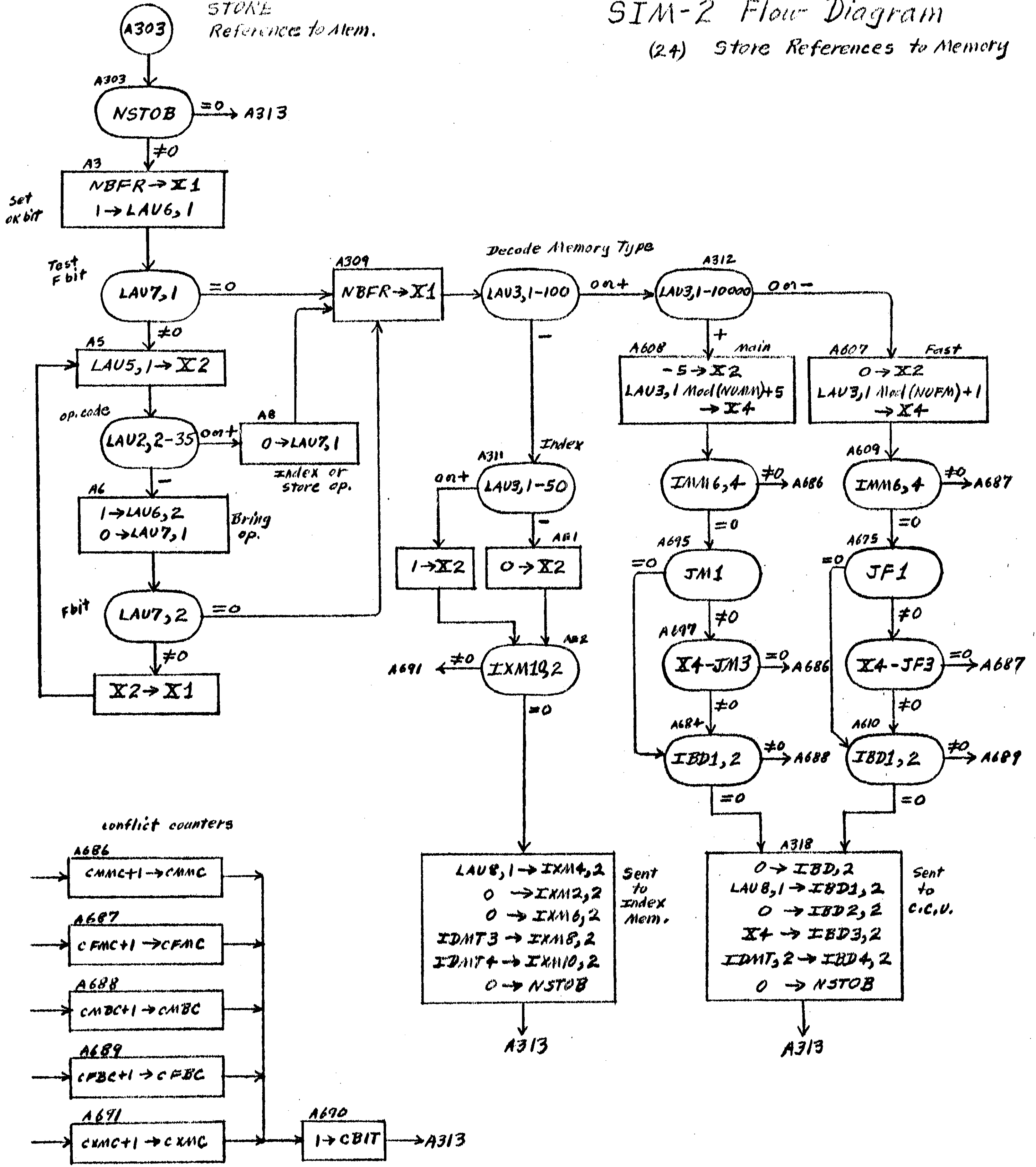
(21) C.C.U. to Fast Mem. Bus
(22) C.C.U. to Main Mem. Bus
(23) Disk Input Section



HGK
Mar 13, 58

SIM-2 Flow Diagram

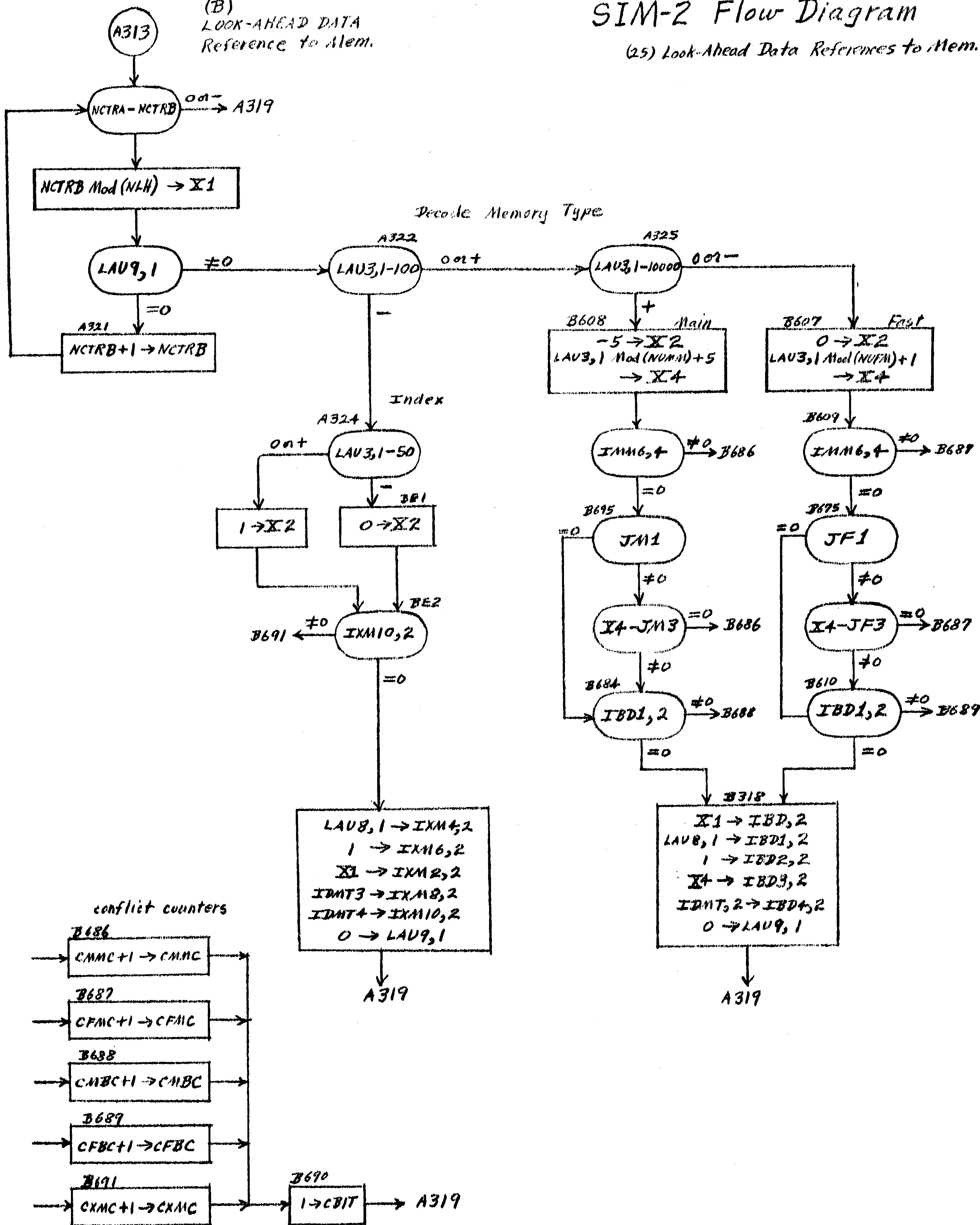
(24) Store References to Memory



HGK
Mar 14, '58

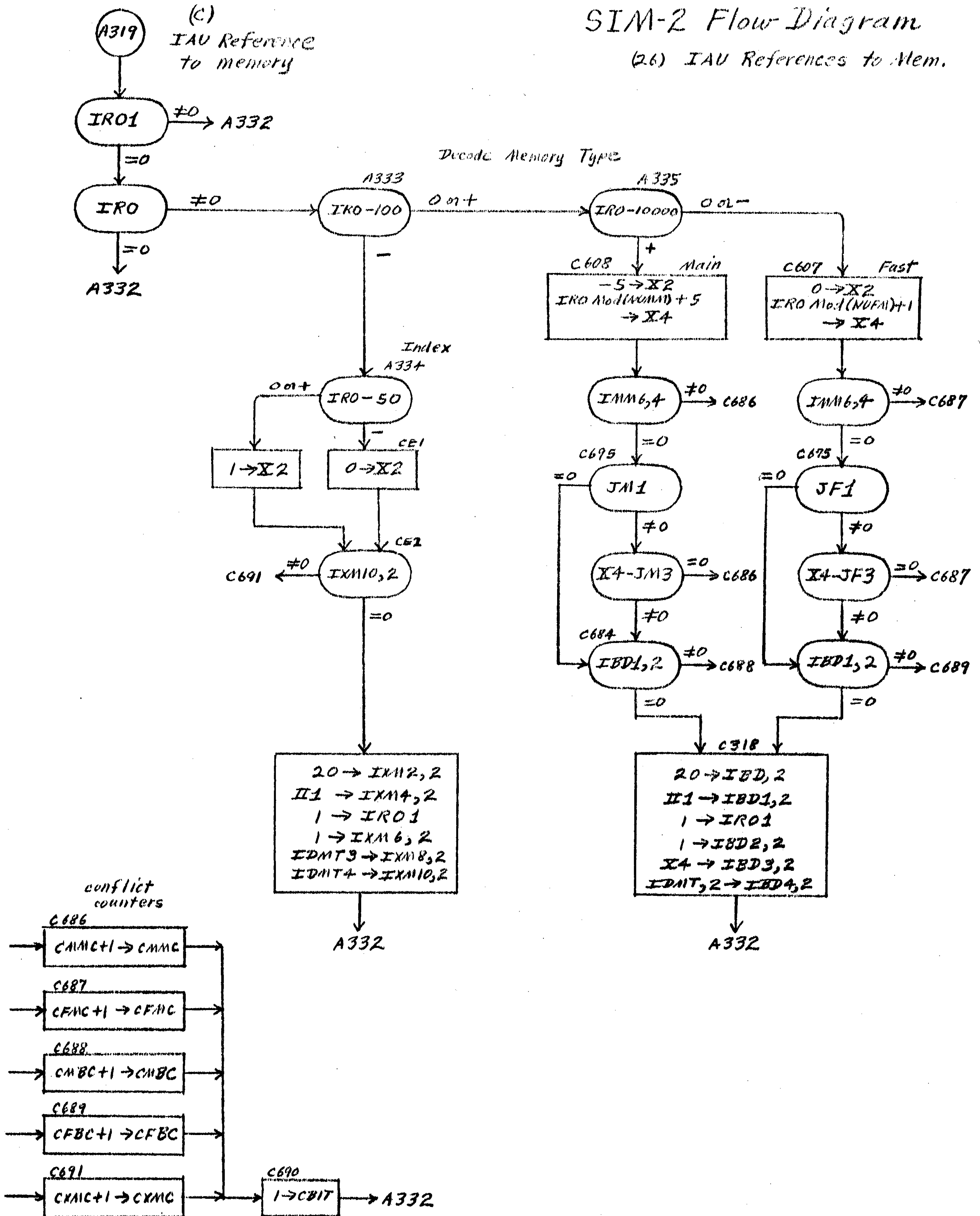
SIM-2 Flow Diagram

(25) Look-Ahead Data References to Mem.



SIM-2 Flow Diagram

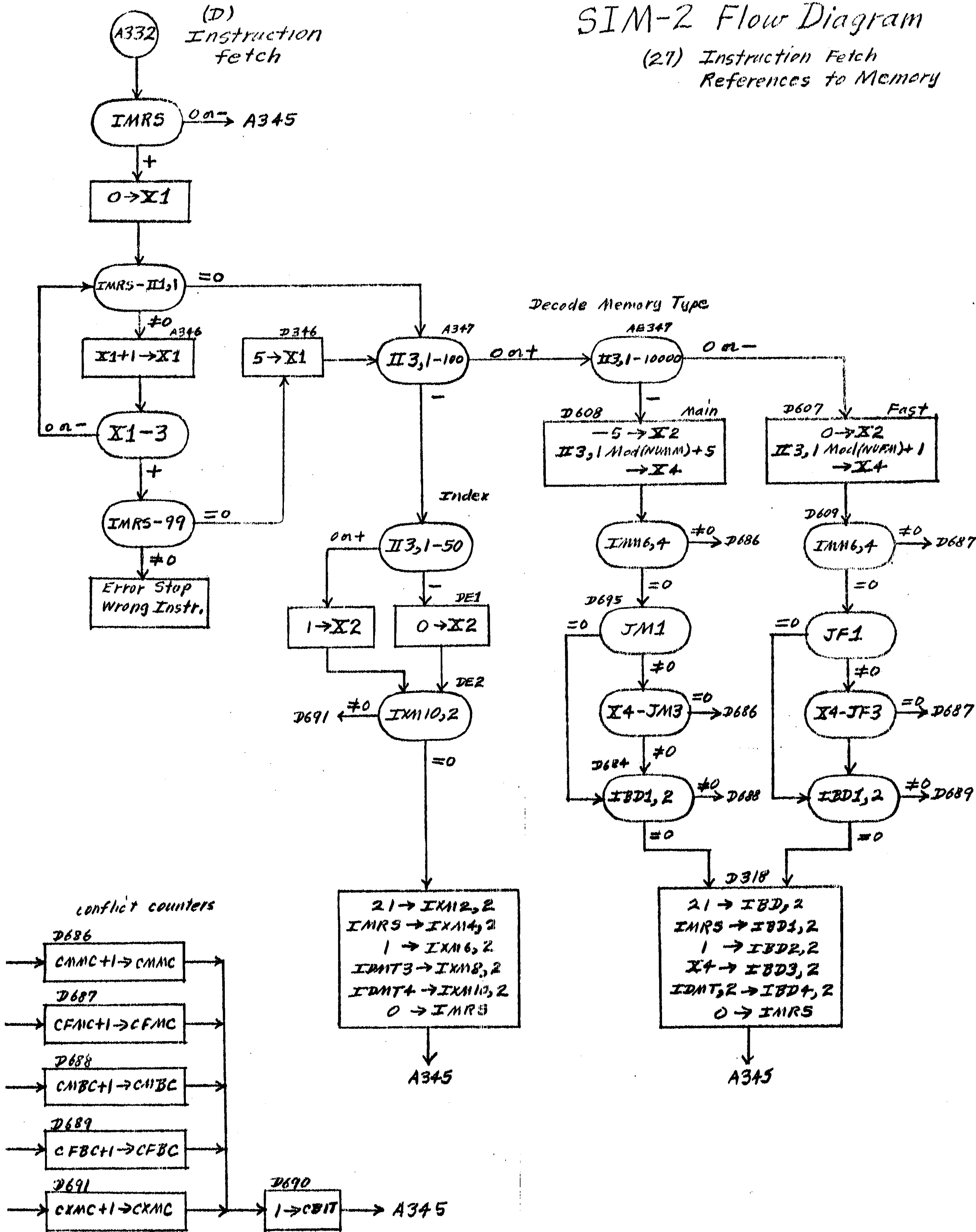
(26) IAU References to Mem.



HGK
Mar 14,'58

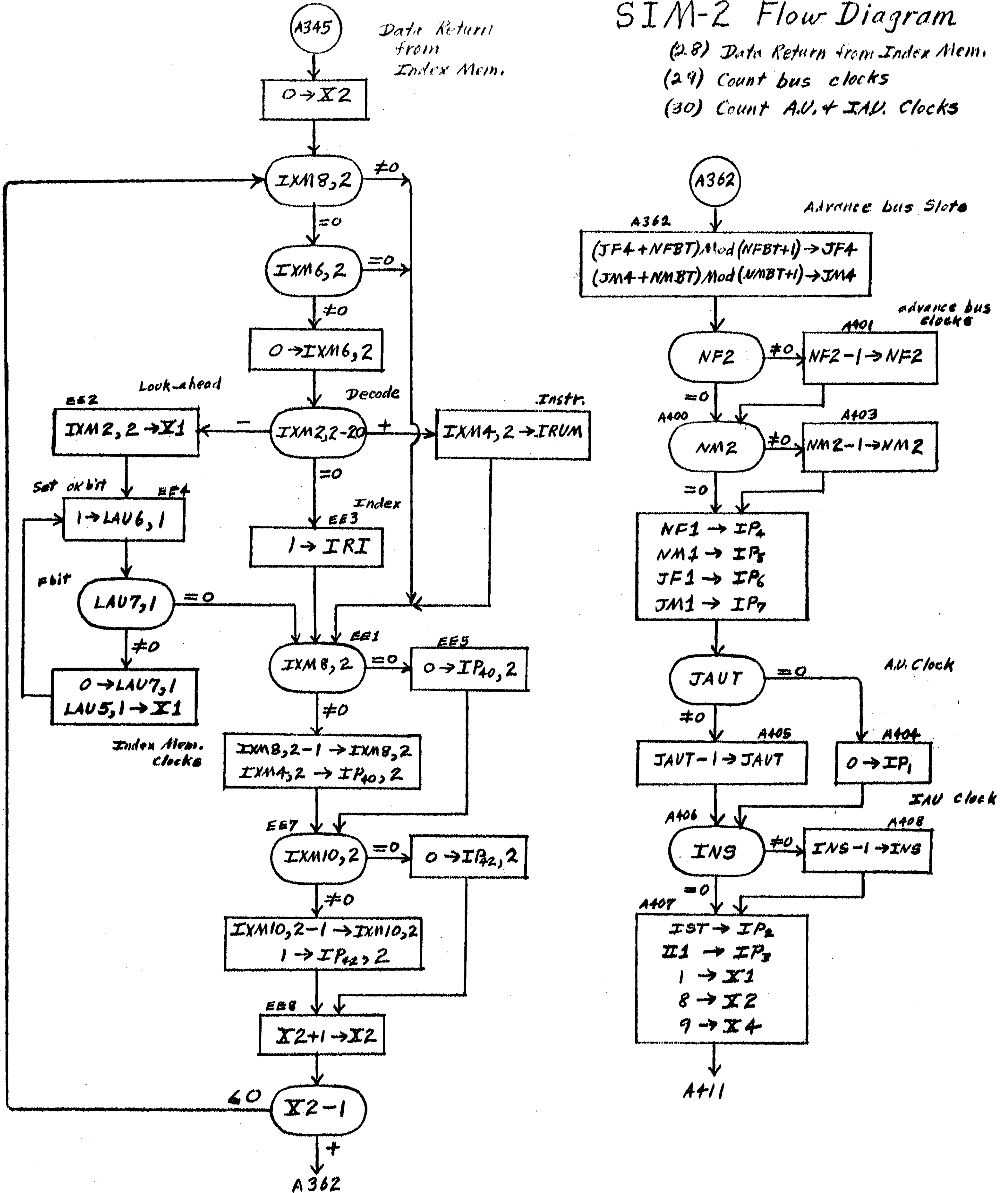
SIM-2 Flow Diagram

(27) Instruction Fetch References to Memory



SIM-2 Flow Diagram

(28) Data Return from Index Mem.
(29) Count bus clocks
(30) Count A.U. + IAU. Clocks



HGK
Mar 14,'58

SIM-2 Flow Diagram

(31) Count Main + Fast Mem. Clocks
(32) Count C.C.U. Clocks
(33) Prepare Data for Print
(34) Print Data Lines