ILLIAC IV

CHARACTERISTICS
AND
PROGRAMMING
MANUAL

66000C
IL4-PM1

ILLIAC IV

SYSTEMS CHARACTERISTICS AND PROGRAMMING MANUAL

Burroughs Corporation
DEFENSE, SPACE AND SPECIAL SYSTEMS GROUP

66000D
IL4-PM1
1 May 1972

ILLIAC IV

SYSTEMS CHARACTERISTICS AND PROGRAMMING MANUAL

This document replaces in entirity all previous issues of IL4-PM1 information.

Burroughs Corporation
Defense, Space and Special Systems Group

PROJECT SUPPORTED BY ADVANCED RESEARCH PROJECTS AGENCY
UNDER CONTRACT NO. AF 30(602)4144 AS ADMINISTERED
BY UNIVERSITY OF ILLINOIS, URBANA, ILLINOIS.

CONTENTS

CONTENTS (CONT'D)

Page

CONTENTS (CONT'D)

ILLUSTRATIONS

TABLES

INTRODUCTION

The ILLIAC IV is a large digital computing system that employs an advanced concept of parallel design to achieve a major increase in processing capacity. The nucleus of the system is the ILLIAC IV array, a matrix of 256 identical processing units and four identical control units, configured into four identical quadrants, each having 64 processing units under the direction of a control unit. These array elements perform the computational tasks for the system.

Although the user of this manual will be concerned mainly with operations internal to the array elements, he should have a working knowledge of the ILLIAC IV system as a whole. To provide this coverage, Section I is devoted in its entirety to a general discussion of the system organization and of the major units within this organization. Emphasis has been given especially to interactions between the major subsystems – ILLIAC IV array and I/O subsystem – and the disk file and B6700 control computer. Section II presents detailed information pertaining to programming characteristics. This section includes bit allocation in the internal word formats, the implementation of the configuration control logic, and the basic registers of the CU. It also includes descriptions of the Interrupt Control (including Interrupt Handling) and Input-Output Control. Section III provides a discourse and flow chart for each of the machine instructions executed by ADVAST of the Control Unit. Section IV similarly treats FINST/PE instructions executed by the array processing elements. A detailed description of the functional characteristics and instructions executed by the Test and Maintenance Unit is presented in Section V. Section VI is devoted to a discussion of instruction timing. Section VII presents a discussion on peripheral timing. A glossary of terms which includes abbreviations frequently used throughout this manual, and a quick reference table for ADVAST, array processing element, and TMU instructions are presented in the Appendices. These quick reference tables have also been included on the inside tabs of sections III, IV, and V respectively.

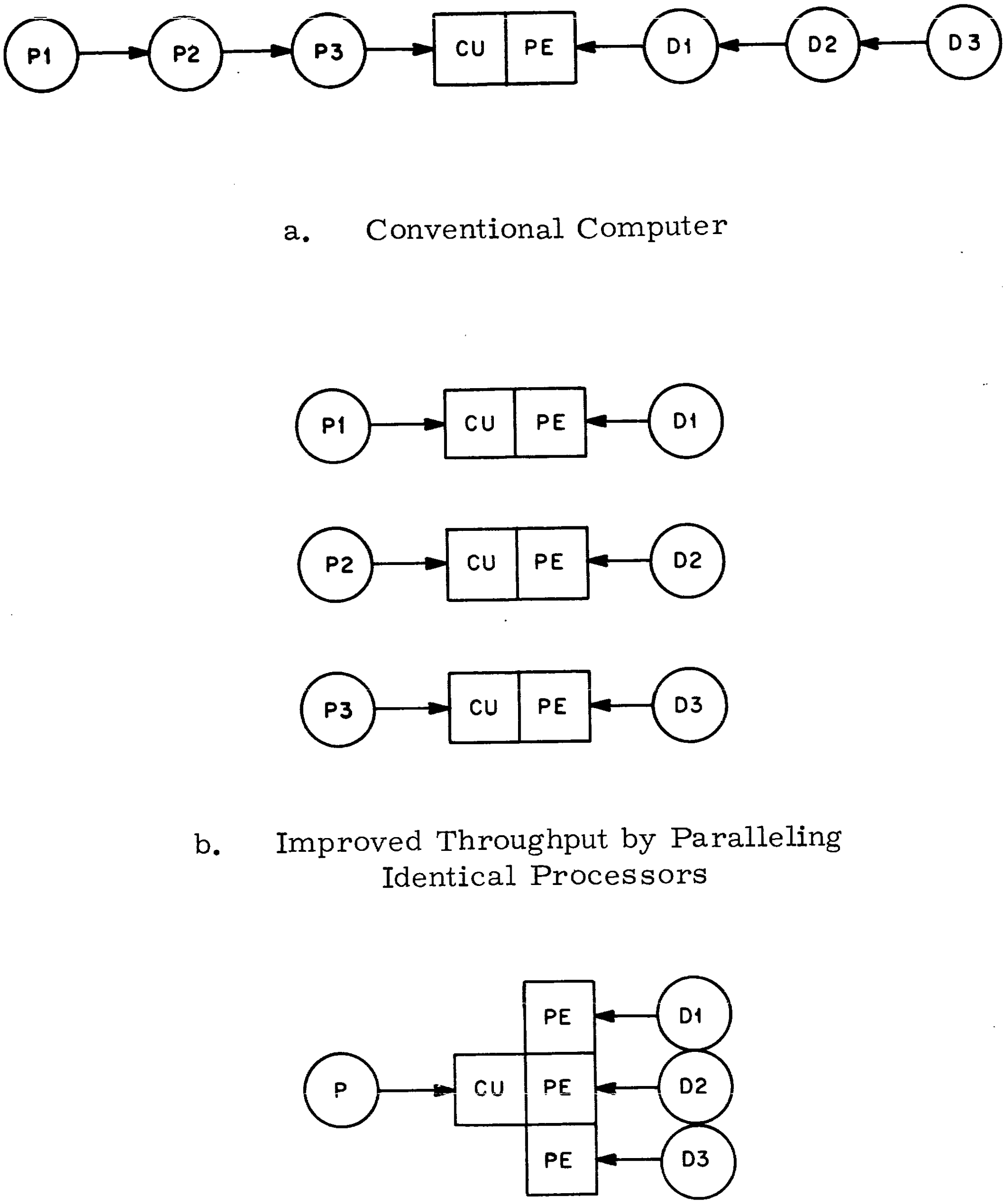
CONTENTS

SECTION I
SYSTEM CHARACTERISTICS

SYSTEM

ILLIAC IV is a milestone in computer development in that it provides a level of parallel processing many times that of conventional designs. To achieve this, a new and fundamentally different approach is used. For important classes of problems many repetitive loops of the same instruction string are executed with different and independent data blocks for each loop. Parallelism may be applied here by using N computers, each executing the identical program concurrently on separate data blocks. This improves execution time by a factor of N for that program. Similarly, since each computer is executing the identical program, much of the control logic of the computers could be made common. This is the fundamental proposition of the ILLIAC IV computer.

Figure 1-1 shows a three-step evolution from conventional design to the ILLIAC IV. The top schematic (Figure 1-1a) shows three identical program loops (P1, P2, P3) operating on three different data blocks (D1, D2, D3) in series. The block element shown is a computer, without input-output or memory, that is functionally separated into a control part (CU) and an execution part (PE). Figure 1-1b shows a simple application of parallelism that produces a threefold increase in throughput. The final schematic, Figure 1-1c, shows the ILLIAC IV approach with its simplifications and economies over the above method.



a. Conventional Computer

b. Improved Throughput by Paralleling Identical Processors

c. ILLIAC IV Approach, Using Common Control Logic and Parallel Identical Processors

Figure 1-2. Development of Parallelism Toward Improving Program Throughput

The ILLIAC IV has a distributed memory system which allows each execution element uninhibited access to an assigned data block within its own memory. If a conventional, centralized memory were used, much time would be wasted in routing data to and from such a memory.

SYSTEM ELEMENTS

The five major elements of the ILLIAC IV are the Control Unit (CU), the Processing Unit (PU), the Input-Output (I/O) Subsystem, the Disk File, and the B6500 Control Computer. Each PU is a combination of a Processing Element (PE), Memory Logic Unit (MLU), and Processing Element Memory (PEM). A CU directly governs 64 PUs configured in an array as illustrated in Figure 1-2. In the ILLIAC IV system there are four such identical arrays called quadrants, making a total of four CUs and 256 PUs. Quadrants may function separately or in combination with one another.

Each PU is labeled with a unique three-digit octal number, the first octal position identifying the quadrant number and the second two positions the PU number within a quadrant. The four "nearest neighbor" connections within the array are defined in terms of direct, parallel, word transfer paths between one PU and others having assigned labels differing by plus and minus eight, and plus and minus one, from the value of the connected PU's label (Figure 1-3). Thus for example, PU 033 can transfer directly to PUs 023, 032, 034, and 043. This connectivity is maintained for both separate and joined quadrants, and enables a variety of physical images to be modeled – for instance, weather maps – by means of a combination of these transfer paths. All CUs have full-word data interconnections for programs that operate in a multiquadrant mode.

The Burroughs parallel disk file is the principal secondary storage element, the main storage being furnished by the PEMs. The disk file provides a storage capacity of 79×10^6 bits per Storage Unit and has a transfer rate of 502×10^6 bits per second. Data is routed in and out of the disk files through

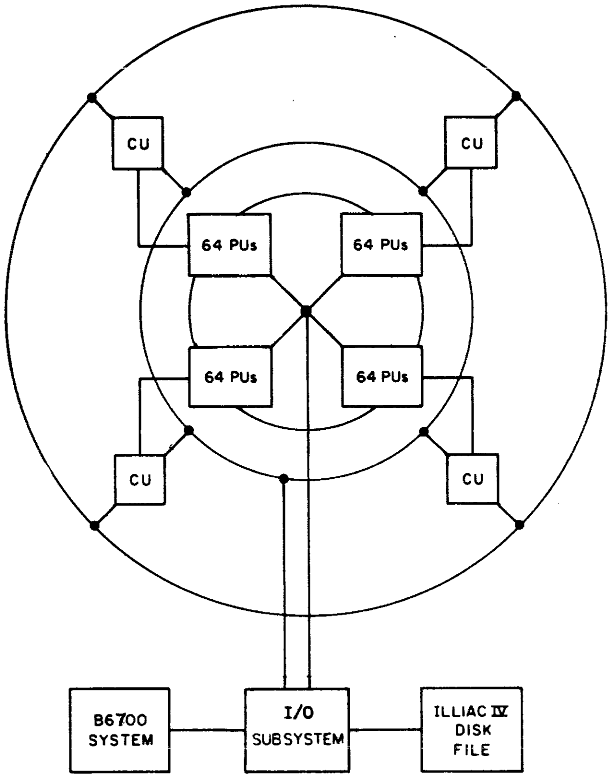


Figure 1-2. ILLIAC IV System Configuration

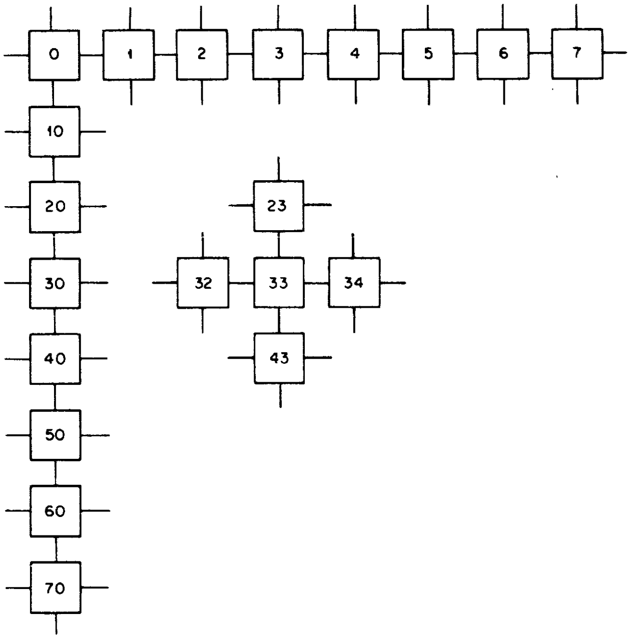


Figure 1-3. Array Connectivity

the Input-Output (I/O) Subsystem which includes the Descriptor Controller (DC), the I/O Switch (IOS), the Buffer I/O Memory (BIOM), and the Disk File Controller (DFC).

The final unit of ILLIAC IV is a B6700 control computer system which serves as the principal managing element for ILLIAC IV. Resident in this system are executive control, facility allocation, peripheral-equipment control, I/O processing and initialization, fault recovery, and program assembly. The linkage between the B6700 system and the Control Units is via the I/O subsystem, as shown in Figure 1-2. It is this link that the B6700 uses to initialize the state word in each CU, that is, setting the initial value of the program counter, the control state, and array configuration. The array configuration identifies the quadrants that are working jointly on the same program and the quadrants, if any, that are operating independently. The B6700 also generates controls to initiate the transfer of program and operands from disk to array memory before allowing the CUs to proceed with program execution.

To effect a data transfer, the B6700 supplies the DC with a start address and the number of words to be transferred. The DC then sends an intermediate memory address to the CU and the disk file to initiate the transfer. Data transfers are made directly between disk and the PEMs. Once the required number of instructions and operands have been transferred from disk, the CU will begin program execution by sending an instruction fetch to the PEMs. Operation then proceeds in the conventional manner, under control of the program stored in the PEMs. Instructions as well as operands may be transferred across quadrant boundaries, so they need be stored only once, regardless of the configuration.

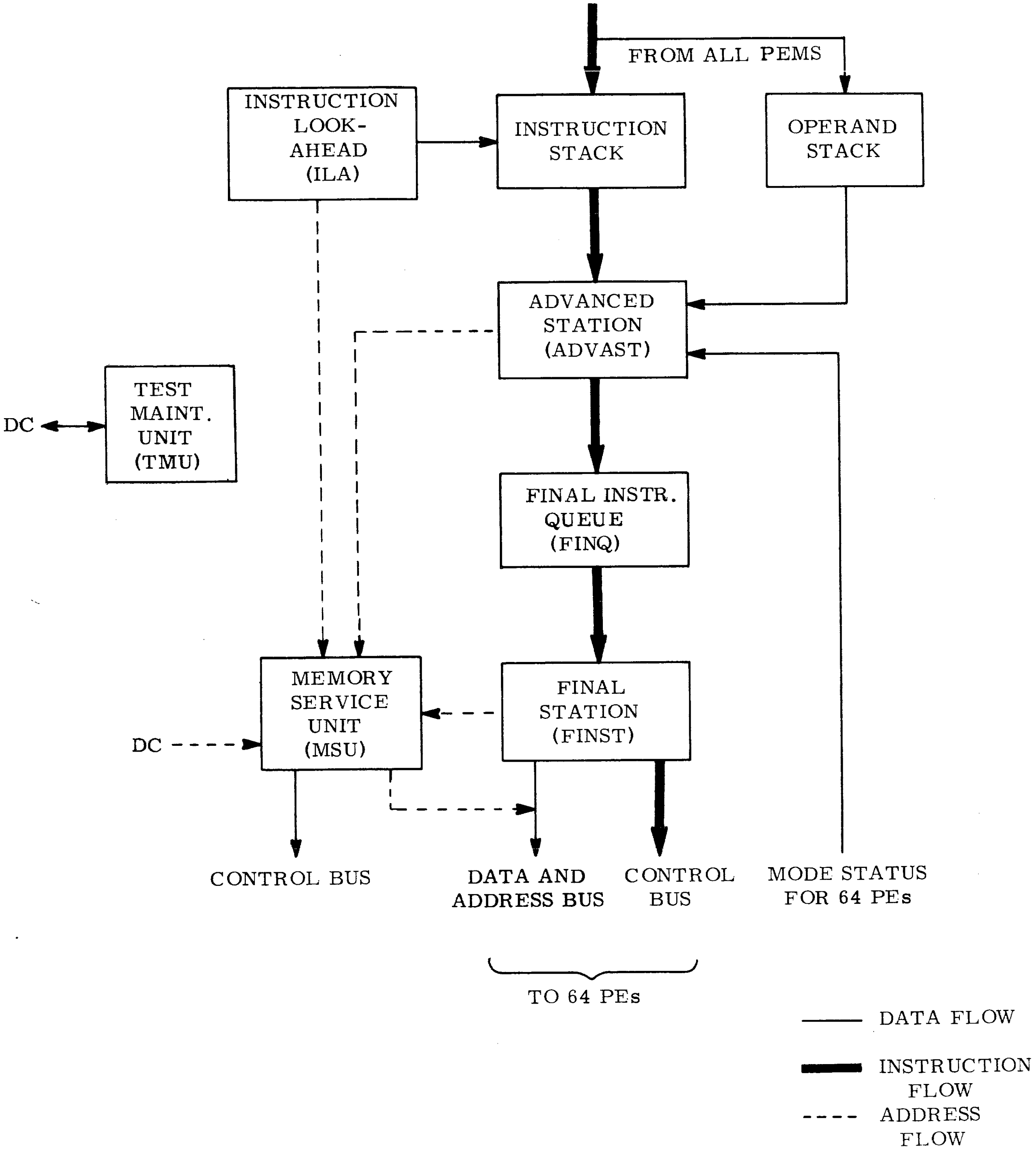


Figure 1-4. Control Unit

CONTROL UNIT FUNCTIONS

The Control Unit is that portion of the computer system which performs the initial processing of instructions up to and including the generation of instruction microsequences for a step-by-step control of the Processing Element. Figure 1-4 is a general block diagram of this single cabinet unit. Contained within the CU are five operating sections which perform specialized processing tasks on a semi-independent basis: Instruction Look-Ahead (ILA), Advanced Station (ADVAST), Final Station (FINST), Memory Service Unit (MSU), and Test Maintenance Unit (TMU).

The Instruction Look-Ahead (ILA) fetches instruction words in 8-word blocks from array memory, placing them in a 64-word content-addressable memory which serves as an instruction word stack. Individual instruction blocks are assigned locations by an associative memory which holds all but the four low-order bits of the instruction addresses. To access a word in the stack, the instruction counter in ILA sends the instruction address to the associative memory to locate the proper 8-word group in the instruction stack, and then the four low-order bits will select the appropriate instruction. Program loops of up to 128 instructions can be contained within the instruction stack.

From the instruction stack, instructions are fed in turn to the Advanced Station (ADVAST), which is the principal housekeeper of the system. Such functions as address arithmetic, loop control, mode control, interrupt processing, and configuration control are performed here. The hardware complement of ADVAST consists of a 64-word operand stack, four full-word accumulators, and a combinatorial logic unit, the latter used to perform functions such as adds, compares, shifts, bit testing, etc. ADVAST permits all those functions generally associated with program control to be performed concurrently with, but in advance of and separate from, the main processing activity.

The primary function of the Final Station (FINST) is to act as an intermediary between the main control, in ADVAST, and the 64 array elements, called Processing Elements (PEs). The repertoire for ILLIAC IV has two general categories of instructions: those executed at ADVAST and those executed in the array elements but controlled by FINST. Since all instructions are first at ADVAST, those instructions intended for execution at FINST are transferred to FINST through the Final Queue (FINQ). This latter element is composed of eight instruction storage positions, which perform a time-smoothing function between ADVAST and FINST. FINST decodes each instruction into control microsequences, which are then broadcast to 64 array elements over a common control bus. FINST also broadcasts full-word operands, shift counts, test values, and other common array parameters on a data bus. In actual operation, the timing sequences of FINST and the 64 array elements are lock-stepped, except for the fixed transmission delay of the intervening control bus.

The Memory Service Unit (MSU) resolves the conflicts of the three users of array memory: I/O, FINST, and ILA. It also transmits the appropriate address to memory and exercises control over the memory cycle. As a hardware expedient, the addresses are transmitted over the same common data bus mentioned above.

The Test Maintenance Unit (TMU) provides the control channel to the Control Unit from the B6700 and the manual maintenance panel.

The array element, or PE, is the execution portion of the configuration depicted in Figure 1-1c. This unit is devoid of all independent control with the exception of mode and some data-dependent conditions. Mode control permits a PE to accept or ignore a broadcast control sequence from the CU, dependent on its current status. The PE is essentially a four-register arithmetic unit, as shown in Figure 1-5, capable of executing a full repertoire of instructions having 64-bit, 32-bit, or 8-bit operands. Further, operations involving 64-bit and 32-bit words can be done in either fixed-point or floating-point representation.

An arithmetic unit in the PE combines a carry-save adder tree and parallel adder with carry look-ahead logic to give a floating-point multiply time on the order of 450 nanoseconds and a floating-point add time of 250 nanoseconds. Both times include post-normalization. Other logic elements include a barrel switch for rapid data-shifting, a leading-one's detector, and a logic unit for Boolean operations. Instruction operands may originate in any of the PE registers, the common data bus, the nearest orthogonal neighboring PEs, or the array memory.

The array memory consists of 64 independent memory modules, called the Processing Element Memory (PEM). Each PEM is collocated with and assigned to a specific PE, providing storage for 2048 words of 64 bits each. The memory is designed for a 250-nanosecond read or write cycle. Memory addresses are supplied to the PEM from the memory address register located in the PE. An address adder and an index register within the PE permit memory indexing and addressing independent of FINST control. Such independence provides important flexibility for addressing data stored in a variety of ordered forms.

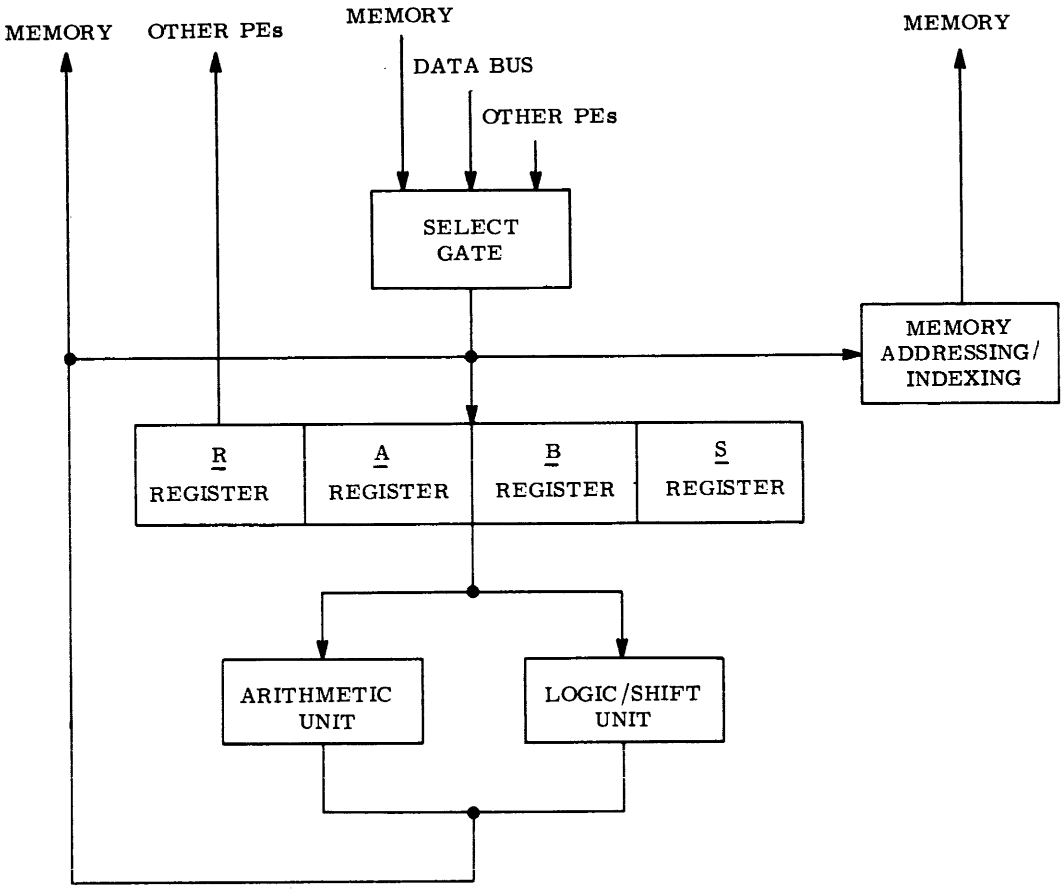


Figure 1-5. Processing Element

CONTROL UNITS

Each Control Unit (CU) directly controls a subarray of 64 Processing Units (PU) in a quadrant. Four identical quadrants, or a total of four CUs and 256 PUs, comprise the ILLIAC IV system. Associated with each subarray of 64 PUs are certain common registers and logical elements which can be manipulated by instructions; decoding of instructions for the Processing Elements (PE) of the PUs is also common. Both the decoding functions and the common registers and logic are contained within the CU. In the performance of its primary task the CU manipulates two types of instructions in the instruction stream: those which it decodes for specifying commands to the PEs – called FINST/PE instructions – and those which control the internal operation of the CU—called ADVAST instructions. A block diagram showing the principal logic elements of the CU is given in Figure 1-6. A block diagram of the CU showing the general relationships of its main functional areas appeared previously as Figure 1-4.

CUs may function individually to control single quadrants (64PUs) or in arrays of two (128 PUs), three (192 PUs), or four (256 PUs) quadrants. When operating in parallel, the CUs specified by the array size control register (MC0) will receive identical program instructions during fetches from array memory. The program instructions will be fetched from that portion of array memory controlled by the CUs specified by the instruction fetch register (MC1). Program instructions which are independent of array size, such as COR, CADD, ADD, CHSA, etc., will be executed by all of the CUs specified by MC0. However, program instructions which are array size dependent, such as LOAD, TCW, RTL, RTG, etc., will be executed by the array whose CUs are specified by the instruction execution register (MC2). Thus, the individual CUs will execute identical instructions in parallel, but in combination will appear indistinguishable from a single control unit having 64, 128, 192, or 256 processing units assigned to it. Note that three-quadrant operation is restricted by the MC1 and MC2 control bits, which may not specify three quadrants.

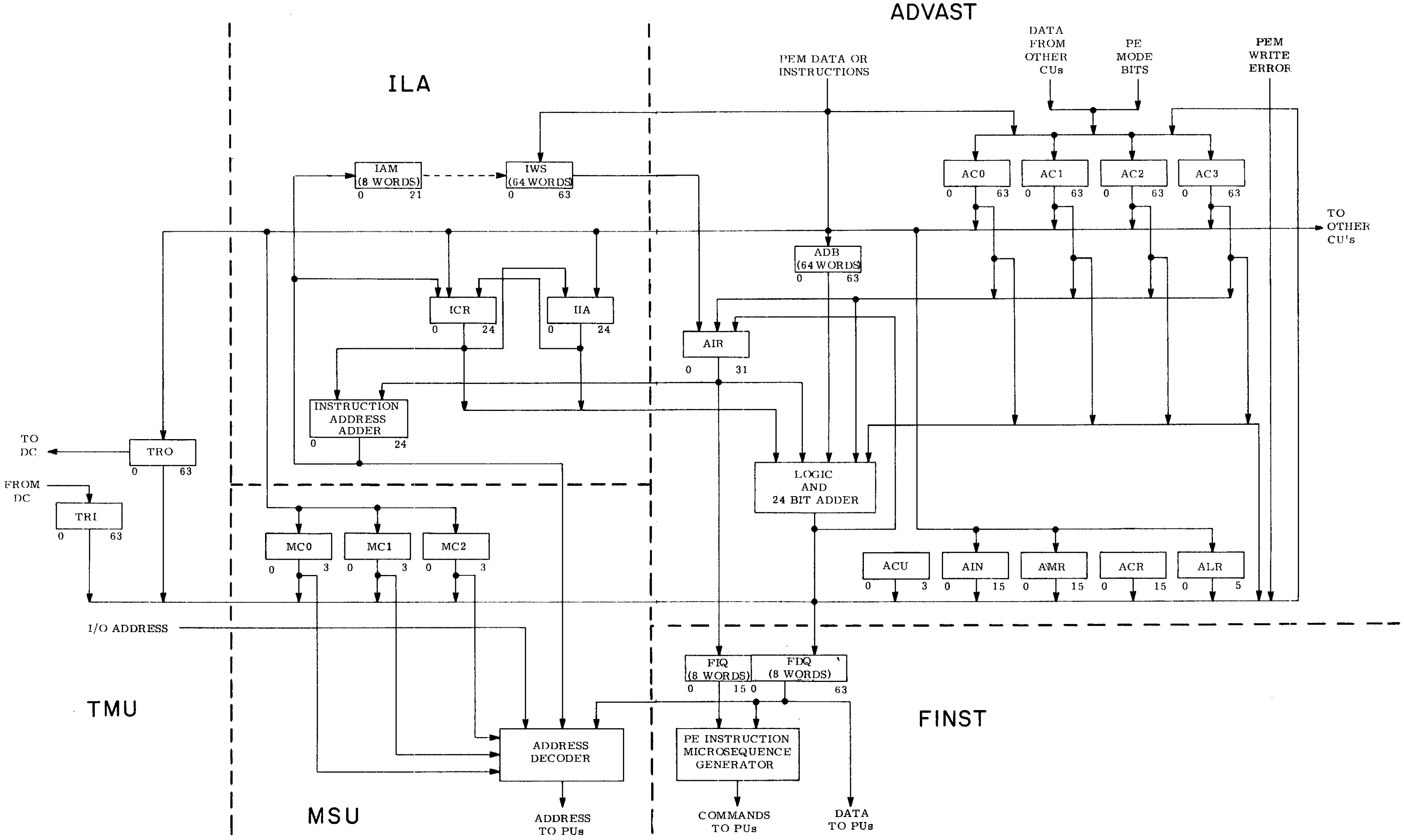


Figure 1-6. Control Unit Block Diagram

A CU shares the same physical array memory (PEMs) with the PEs in its quadrant. The Memory Service Unit (MSU) of the CU determines which PEMs shall be addressed for various memory operations according to the type of memory request received. For example, FINST/PE instructions which involve memory access require that all 64 PEMs of the quadrant must be addressed, whereas the ADVAST instruction BIN requires that only eight specific PEMs of the quadrant be addressed (see Section III, ILLIAC IV Addressing). The various users of array memory include in their memory request to the MSU the following number of bits to identify which PEMs are to be addressed:

FINST/PE	-	none
I/O	-	2
ILA	-	3
ADVAST (via FINST)	-	6

Program steps are fetched in blocks from memory, and executed one at a time. Although there is rather extensive machinery in the CU to reduce the actual number of memory fetches from one fetch per program step, as in conventional machines, to 0.0025 or 0.015 fetch per instruction, this machinery requires no attention on the part of the user programmer.

The registers in the CU which can be manipulated by the program are as follows:

AC0, AC1, AC2, AC3 – A set of 4 registers, 64-bits each, general purpose accumulators (ACARs)

ACR – ADVAST control register, which contains CU status information (read only)

ACU – Own quadrant number register (read only)

ADB – A set of 64 registers of 64 bits each, used as a scratch-pad memory

AIN - ADVAST interrupt register

AMR - ADVAST interrupt mask register

ALR - A register which holds the address of pending memory fetches

MC0, MC1, MC2 - Array configuration control registers

IIA - ILA interrupt storage for ICR

ICR - ILA instruction counter

TRI - TMU input register (read only)

TRO - TMU output register.

CONTROL UNIT STRUCTURE

As noted previously, the order code has two general types of instructions, those used primarily to control the internal operations of the CU, called ADVAST instructions, and those that are used primarily to control PU operation, called FINST/PE instructions. Since there is almost no interaction between the two instruction types, they can be viewed as two interlaced but distinct instruction streams. The hardware of the CU takes advantage of this partial independence to execute the two streams independently but concurrently with one another. The CU has five main functional areas, as follows:

Instruction Look Ahead (ILA). The instructions are fetched, in 8-word blocks of contiguous code, to a section of the CU called the instruction look-ahead (ILA). An associative memory (IAM) detects which blocks of instruction are currently in ILA storage. The instruction counter (ICR) is also contained in ILA.

Advanced Station (ADVAST). Each instruction is passed in sequence to the instruction register (AIR) of ADVAST. In ADVAST, each instruction is first examined to determine whether it is an ADVAST instruction (to be executed exclusively in ADVAST) or a FINST/PE instruction (to be executed by the PEs). FINST/PE instructions will be indexed, if required, within ADVAST and then placed in the final queue (FINQ) for execution by the PEs. Some instructions (e.g., BIN, LOAD) may require additional processing in ADVAST after they have been executed by the PEs.

Final Station (FINST). Instructions from ADVAST enter a section of the CU called the final station (FINST), the outputs of which manipulate the Processing Units. Instructions enter FINST through a final queue (FINQ) so that the instruction execution time at FINST is decoupled from that at ADVAST. Some instructions (e.g., LOAD) are executed partially at ADVAST and partially at FINST because of the need for PU operations to complete instruction execution. In general, the programmer need not be aware of overlap operation between the two sections, if it occurs.

Memory Service Unit (MSU). The memory service unit (MSU) receives requests for access to memory from three sources: from FINST, from ILA, and from the Descriptor Controller (DC) of the I/O subsystem. The MSU resolves conflicts among the three sources as well as conflicts concerned with other FINST uses of the common paths from CU to memory. ADVAST memory requests are serviced through FINST.

Test Maintenance Unit (TMU). The Test Maintenance Unit (TMU) of the CU contains registers TRI and TRO (which are addressable by instructions in ADVAST) and provides paths to the maintenance panel, the display, and the B6700 (via the DC). The display will, on external command, indicate the state of certain CU registers. A portion of TMU serves as a "test instruction" register for diagnostics, testing, and initialization.

TIMING CONSIDERATIONS

Potential program difficulties are introduced by the asynchrony between ADVAST and FINST since ADVAST may be executing instructions which occur later in the instruction stream than those which are in FINQ awaiting execution. The hardware automatically detects this potential problem and introduces the necessary synchronism to prevent any difficulty. The only exceptions are bits ACR(09) and ACR(13). A change in either of these bits is effective immediately, even on any previous instructions which may remain unexecuted in FINST. If this presents a problem, an instruction "FINQ" may precede the CACRB which changes ACR(09) or ACR(13). On the other hand, changes in ACR(10), word size, are synchronized; only those FINST/PE instructions which follow a CACRB(10) instruction will be executed with the new word size. Other cases of asynchronism are also found. For instance, the effects of some interrupts, such as arithmetic fault interrupts, are somewhat delayed in reaching the interrupt register, AIN; they may halt the program several instructions after the one which cause the interrupt. Also, STORE instruction, whose address is in the program area, will not change the execution of the program if the instruction in ILA were fetched before the STORE was effective in memory.

SEQUENCE OF OPERATIONS

The operation of the ILLIAC IV system is somewhat complex due to the close coupling of intraquadrant operations and the largely decoupled operation of interquadrant functions. Superimposed on this structure are communications with the B6700 and the DC, which can be considered as being asynchronous with the ILLIAC IV system itself. The program flow described here traces the actions of the various system components during the execution of a program.

System Start-Up

Upon receipt of a job request, the B6700 transfers the program and data base to the ILLIAC IV disk system. The quadrants of the system which will be used by the program are than selected, and a command issued to the TMU section of the selected CU(s) causing operation to halt and initialization for the new program to proceed. Then, by issuing commands to the DC, the B6700 initiates the loading of the disk-held program and data to the appropriate array memory locations. Following the loading operation, the B6500 sends commands to the TMU(s) which will start program execution after setting the instruction counter (ICR) in the ILA section to the address of the first program instruction.

Fetching the Program

During initialization, the instruction look-ahead unit (ILA) is set to indicate that there are no instructions in its instruction word storage (IWS). Immediately upon start-up, the ILA will recognize this condition and request a block of instructions – via the MSU – from the PE memories that contain the instruction addressed by the ICR.

The IWS acts as an instruction queue for ADVAST. It holds up to 128 instructions which are fetched in blocks of eight words, two instructions per word (16 instructions per block). Eight of these blocks are stored in IWS.

The conditions for initiating the fetch of a new block of instructions are, either ICR has changed or the instruction currently being executed is one of the last eight instructions in the block and the next block of instructions to be executed is not found in IWS. A ring-of-eight counter is used to implement a first-in-first-out discipline on the eight blocks of instructions in IWS. Thus, the instruction block which will be overlayed by the newly fetched instruction block is the oldest block in terms of the time at which the blocks were fetched from array memory. If, however, the block presently being executed is the oldest block (an exceptional case), the ring-of-eight counter is incremented a second time such that the next oldest instruction block will be overlayed.

ADVAST Processing

As noted previously the primary function of ADVAST is to handle the housekeeping tasks for the quadrant. From a programming point of view, FINST and the PEs perform the "inner-loops" of a program while ADVAST handles most of the "outer-loop" and control functions. Included in its tasks are the processing of exception conditions, decision-making for interquadrant transfers, and the handling of interrupts.

When ILA holds the instruction addressed by the ICR, the instruction is sent to the ADVAST instruction register (AIR) which determines whether it is a FINST/PE instruction or one that ADVAST will process. FINST/PE instructions are passed on to the final queue (FINQ) to await execution by FINST and the PEs, whereas ADVAST instructions remain in the AIR while they are being executed.

The ADVAST registers ACAR are primarily index/limit/increment registers that are used to supply addresses for PE instructions, but can also be used as accumulators for performing logical functions such as decision making and data formatting. The ADVAST data buffer (ADB) is used in conjunction with the ACARs in data formatting and information broadcasting to the PEs. The other registers controlled by ADVAST are manipulated to effect program sequencing and control.

Final Station Processing

FINST accepts instructions from ADVAST and places them in the final queue (FINQ), which is composed of an instruction queue (FIQ) and a data queue (FDQ). FDQ holds the address values or data associated with the instruction in FIQ. The eight locations in FINQ are serviced on a first-in, first-out basis. It is FINQ that permits the concurrent operation of ADVAST and FINST.

Instructions are taken from FINQ in largely undecoded form, for execution in the PE. FINST decodes these instructions into sets of microsequence commands for the array of 64 PUs. In some cases synchronism with other quadrants in an array is required and is also accomplished in this process. The generated microsequences contain the individual enable signals that control the information flow – both in direction (register to register) and in time – within the PUs. The generated microsequences are then broadcast to all of the PUs selected to accomplish the execution of the instruction.

Communication and Input-Output

Following the completion of processing on a block of data, additional data and/or program, or the output of the processed data, may be required for subsequent operations. Since the system has no input-output commands of its own, the CU can place a request code in its TMU output register (TRO) to interrupt the B6700 for servicing. This may be accomplished in either of two ways.

1. TRO loaded interrupt - TCI 04: occurs when a word is loaded into the TRO. The word is generated and loaded into the TRO programmatically.

2. CU halted interrupt - TCI 05: occurs when the CU has processed a HALT instruction. The HALT instruction causes the CU to complete current operations and then wait for further instructions.

In either case, the B6700 reads the interrupt, via the DC, and interprets its meaning. Numerous methods are available to the B6700 control program to assume control of the array (see Section V, B6700-TMU Communications).

Other CU Functions

Other CU functions are largely ADVAST controlled. Synchronism requirements are delineated in the individual instruction descriptions and are accomplished at either ADVAST or FINST, depending on the instruction set. The Configuration Control description in Section II details the grouping of quadrants into arrays and the synchronism that this implies. The interrupt system is described under Operational Control in Section II, which explains in more detail the uses and effects of the associated registers. The content of the control registers is also described so that the features for programming utility and service routines are available to the systems programmer.

PROCESSING UNITS

The Processing Unit (PU) functions as a general purpose computer under the direction of an ILLIAC IV Control Unit (CU). All of the 256 Processing Units in the ILLIAC IV system are electrically, mechanically, and functionally identical, each PU consisting of a Processing Element (PE), a Memory Logic Unit (MLU), and a Processing Element Memory (PEM). Data inputs to and outputs from the PE and PEM are shown in Figure 1-7.

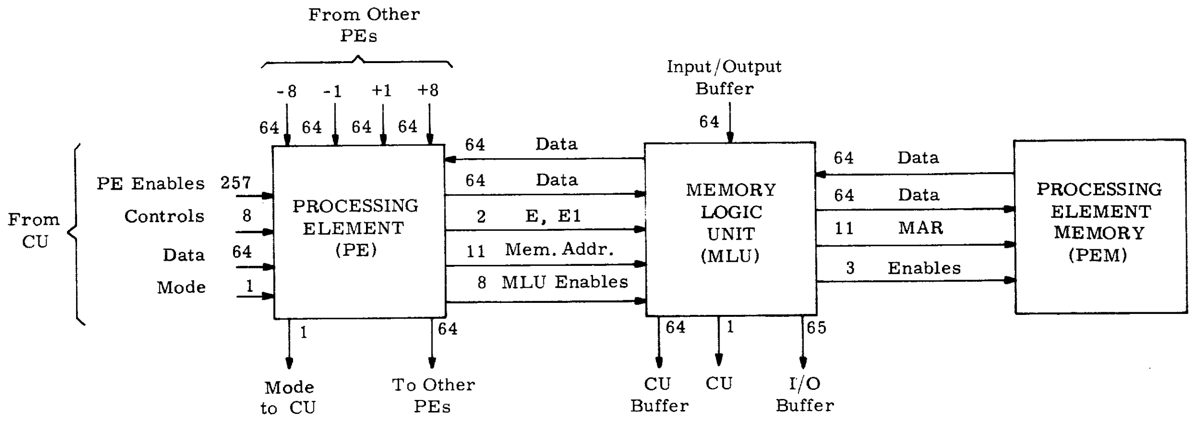


Figure 1-7. Processing Unit Data Inputs and Outputs

For control, the PE and PEM receive enable signals from the CU for the sequential enabling of data paths and logic during instruction execution and for controlling the reading and writing in the PEM. In addition, the CU monitors the control status of the PE by using one input and one output of the PE mode logic. Similarly, it monitors the memory protect error status of the PEM by using one input and one output of the MLU.

PROCESSING ELEMENT (PE)

The portion of the PE in which data manipulation is carried out is shown in Figure 1-8. The principal registers are the five 64-bit data registers, called the A, B, C, R, and S registers, the 16-bit indexing register, called the X register, and the 16-bit memory address register. For speed in addition, multiplication, and shifting, the logic gating is structured for register-length parallel operation. Although devoid of many of the controls usually associated with the conventional processing unit, the PE, under main control of the FINST portion of the CU, can execute a full complement of instructions involving arithmetic and data manipulations. Various instructions allow the use of 64-bit or 32-bit word sizes, in fixed-point or floating-point representation, or combinations of 8-bit bytes using unsigned notation. All operations are fully synchronized in the PE using a clock supplied to it from the CU. A receiver-retiming register accomplishes this function, synchronizing the controls with this clock before they are buffered for distribution within the PE. Although most of the controls originate externally to the PE, some data-dependent controls, such as used in normalization and signed-arithmetic operations, are generated within the PE.

Registers and Logic

Data Registers

The five 64-bit data registers are A, B, C, R, and S. The A register functions as an accumulator, holding one of the operands in arithmetic operations and receiving the output of the adder at the conclusion of the operation. The B register holds the second operand in arithmetic

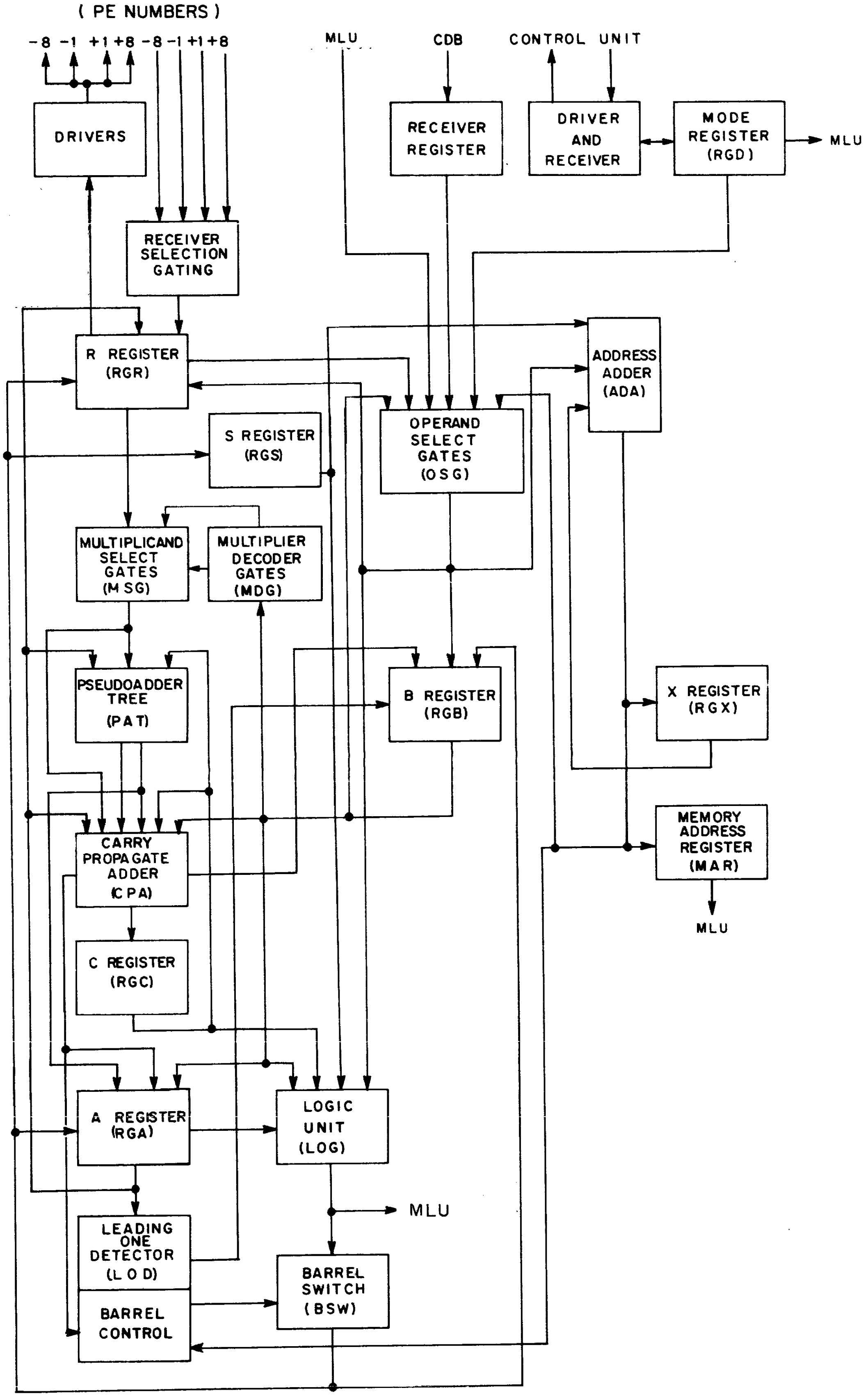


Figure 1-8. Processing Element Block Diagram

operations (with the exception of multiply) and communicates most directly with external data via the operand select gates. The C register is used in certain instructions to save carries from the adder. The R register is the routing register, used principally for communications with other PEs, and at times for temporary storage of operands. The S register is used for programmatic storage of an operand within the PE.

Mode Register

This register contains eight bits which control some of the operations in the PE and store the PE faults and test results. Two of these bits, E and E1, called enable bits, are used to protect the A register, the S register, and the memory information register (MIR) by controlling the gating of clocks to the outer (bits 0-7, 40-63), and inner (bits 8-39) half-words. The E bit alone also protects the 16-bit X register, which is the PE index register. In 32-bit mode, E and E1 are independent; however, in 64-bit mode, E should equal E1. The two F bits (F and F1) are used to store faults (underflow, overflow, etc.). The other four bits (G, H, I, and J) are used primarily for temporary storage of test results and can be manipulated in conjunction with the E's and F's. By instruction, any one mode bit may be sent from the CU or any one mode bit may be sent to the CU.

Shifting

A 64-bit, right shifting, end-around shift network, called the barrel switch (BSW), is used in the PE. With the logic unit to select the input and with full distribution of the output, the BSW allows generalized, one-clock-period shifting of registers in the PE. BSW control is extensive to allow 64-bit or 32-bit words to be shifted left, right, end-off or end-around. Inputs to the BSW control include shift amounts calculated by the address adder (ADA), fixed amounts required in certain instructions, and variable amounts derived from operands to be normalized or aligned. The normalization amount is generated in a fast, parallel logic network, called the leading one detector (LOD). From the output of the A register, the LOD finds the position of the most significant nonzero bit in the 48-bit or 24-bit mantissa and generates both the shift controls for the BSW and a binary number to be used for exponent correction.

Adding and Multiplying

The requirements for the utmost speed in the addition and multiplication instructions demand a fast parallel adder. The one chosen can function as either a carry propagating adder using three levels of look-ahead, four bits in the first group, four groups in the second section, and four sections in the final level (achieving a 64-bit sum in a single clock period), or as a carry save adder. To distinguish this adder from the other adders, it is called the carry propagating adder (CPA) in spite of its dual purpose.

Eight-bit byte gating allows the interruption of carry propagation for 8-bit mode, and a carry register allows the saving of carries for use in the multiplication sequence.

For speed in multiplication, the eight least significant bits of the multiplier are decoded for each iteration and the proper multiples of the multiplicand are generated by the multiplicand select gates (MSG) which are added in a quadruple layer of parallel carry save adders (CSA) with the CPA acting as the fourth CSA. This logic accomplishes a single multiplication iteration, but without full carry propagation, in one clock time.

Addressing

The 16-bit address adder (ADA) has inputs selected from among the X register, the S register, and the operand select gates (OSG). Sums may be sent to the X register, to the 16-bit memory address register (MAR), and to the BSW controls. The sum output is also sent to the OSG, but is used only for transfers from the X register. With these data paths, all shift counts and memory addresses are indexable by either the X or S register, and the X register may be modified with the ADA.

Figure 1-8 shows that the portion of the PE used for memory addressing is largely separated from the remainder of the PE. The sending of a memory address over the common data bus through the OSG and the ADA (where it may be indexed) into the MAR may be overlapped with any instruction not using this part of the PE. This feature is valuable in decreasing PE or CU idle time caused by waiting for information from memory, which takes approximately seven clock times to complete one memory cycle.

Instructions

The instruction set of the PE is that of a large scale, general purpose digital computer. Floating point arithmetic in both 64-bit and 32-bit words is provided with options for rounding and normalization. Full word operations, 8-bit byte operations, operations ignoring exponents, operations using exponents only, and operations ignoring the signs are provided in the arithmetic group. A full set of tests is generated by making all registers addressable and providing all possible comparisons. Test results are set into a mode bit which may then be used to programmatically direct the flow of the instructions. Swaps of parts of 32-bit words, bit manipulation, shifts and logical operations complete the instruction set.

Control

The PE is driven by a control unit to execute the instruction string contained in the CU. The PE does not receive the raw instructions but rather the fully decoded controls for the enabling of data paths and internal control of the PE as in a microprogrammed computer. While many of these external control inputs are used directly, some must be modified according to the data in the PE. Extensively used modifiers include the mode bits E and E1, the signs of the A and B registers, and the output of the LOD.

There are a few internal control signals of the PE which are generated in conjunction with data dependent operations such as multiplier decoding and mantissa normalization. These will be formed in the PE and are timed to coincide with external controls.

PROCESSING ELEMENT MEMORY (PEM)

The PEM provides a high speed random access storage function for the ILLIAC IV Processing Unit (PU), of which it is a subunit. The other subunits of the PU are the Processing Element (PE) and the Memory Logic Unit (MLU). The PEM provides storage for 2048 words, each word being 64 bits in length. The memory operates with a read cycle time of 250 nanoseconds (maximum), a write cycle time of 250 nanoseconds (maximum), and a data access time of 188 nanoseconds (maximum). The PEM interfaces with, and is directly controlled by, the MLU.

The first 128 words of the PEM can be write-protected by setting the control bit ACR 13. If a write is attempted in any of the word locations 0 through 127 when ACR 13 is set, the memory write cycle will not occur.

MEMORY LOGIC UNIT (MLU)

The MLU controls and effects the transfer of data between the PEM, the Control Unit Buffer (CUB), the PE, and I/O Subsystem. The MLU also enables non-memory data transfers between the CUB and the PE. In addition to the control and timing circuitry for PEM operations, the MLU contains a memory information register (MIR) used for the temporary storage of data to be written into or read from the PEM.

I/O SYSTEM

The three elements which perform the I/O function are:

1. A Burroughs B6700 data processing system which, together with its peripherals, performs all the functions of the control computer;

2. A Model II disk file system providing approximately one billion bits of storage;

3. An ILLIAC IV I/O subsystem which interfaces between the above elements and the ILLIAC IV array subsystem.

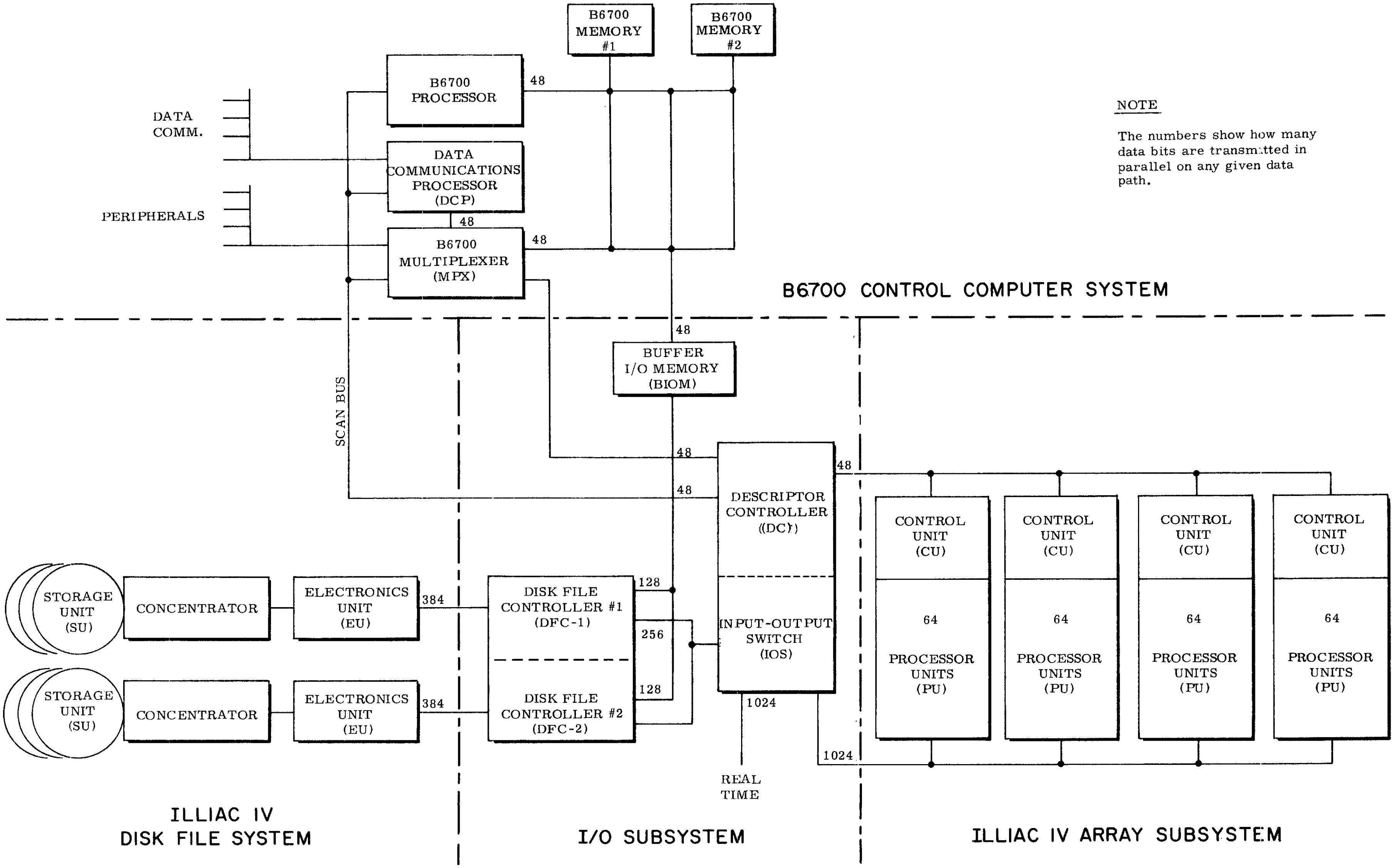
The relationship of these elements to one another and to the array is illustrated in Figure 1-9 and described in the following paragraphs.

B6700 I/O CONTROL COMPUTER

The primary functions of the I/O control computer are to execute the supervisory program for the ILLIAC IV complex and prepare programs for ILLIAC IV. The supervisory program controls the operation of ILLIAC IV; schedules jobs for the array; maintains the Model II disks; transmits control words (descriptors) to the I/O Descriptor Controller, which directs the I/O transactions in and out of the array; responds to interrupt conditions from the array or elsewhere; and communicates with the operator.

The initial B6700 data processing system* necessary to run the supervisory program and prepare user programs consists of: one processor, 32 K words of memory, an I/O multiplexer with one peripheral control cabinet, and suitable peripherals including a disk file with 10^7 bytes of storage. Associated with the multiplexer are controller units which interface with the various peripherals. These are Burroughs units for the standard peripherals: magnetic tape, disk file, line printer, card reader, card punch, and console printer/keyboard. The B6700 can be expanded from this initial complement

* Refer to B6700/B7700 Characteristics Manual.



NOTE

The numbers show how many data bits are transmitted in parallel on any given data path.

Figure 1-9. ILLIAC IV Interface Diagram

of equipment to include an additional processor and multiplexer as well as additional memory (up to 512 K words). On-line communication may be added by including a Datacom processor, multiline controls, and line adapters.

The interface between the I/O subsystem and the I/O control computer is designed to take advantage of the existing properties of the B6700, while keeping simple the interface to the ILLIAC IV array. Control words are received over the scan bus interface provided from the B6700 processor, and results are described in words transmitted back over this same interface.

Two data paths exist between the B6700 subsystem and the I/O subsystem, one being the Buffer I/O Memory (BIOM), and the other being directly into the Descriptor Controller (DC). The BIOM functions as a module of B6700 memory, as seen from the B6700 side, handling data transfers from the B6700 into the ILLIAC IV I/O subsystem. On the ILLIAC IV side, the BIOM can transfer either into the disk file, or directly into array memory. The data path to the DC uses the 48-bit word interface of the multiplexer, being a connection into the multiplexer's memory bus whenever the multiplexer is not using it. All interfaces between B6700 and I/O subsystem use bidirectional cables; 20 lines for address, 48 bits for data, 3 bits for tag bits accompanying the data, 8 bits of control, and 1 bit of parity.

ILLIAC IV DISK FILE SYSTEM

The ILLIAC IV disk file system (not to be confused with disk file which is part of the B6700 control computer equipment) will initially consist of two Model II disk files with thirteen storage units. Each Model II disk file includes an electronics unit, a concentrator, and Burroughs Model II mechanisms, with sufficient electronic circuitry for reading or writing simultaneously on 128 tracks of one disk. Each disk has a capacity of 79,257,600 bits and a maximum of sixteen such disks may be connected to an electronics unit and its associated concentrator. The electronics unit houses certain common electronics, registers for providing conversion of information from disk-serial to control-unit-parallel form, control logic, power, motor control, and the air pressure system. Read amplifiers are housed in the concentrator. Approximate transfer rate to and from the Disk File Controller is 502×10^6 bits per second and the average access time is 19.6 milliseconds. The interface between each electronics unit and its controller in the DFC is 384 bidirectional data lines and 25 control-address lines. The track layout consists of 256 active information tracks per disk face, arranged in one zone.

ILLIAC IV I/O SUBSYSTEM

The I/O subsystem is shown in Figure 1-9 as consisting of the I/O Descriptor Controller (DC), I/O Switch (IOS), Buffer I/O Memory (BIOM), and Disk File Controller (DFC). The functions performed by these elements are briefly described below.

The DFC consists of two controllers which execute descriptors held in DC for transfers between disk and array, disk and BIOM, BIOM and array, and real-time link and array. All transfers involving the array are via the IOS.

As previously noted, the BIOM acts as a memory module for the B6700 system. Within the I/O subsystem, the BIOM has a 128-bit bidirectional interface with each of the two DFC units. All transfers through this interface are under the control of DFC descriptors.

The IOS unit buffers and distributes data between the DFC and the ILLIAC IV array. The DC is also located in the IOS cabinet. The IOS has a 256-bit bidirectional interface with each of the two DFC units and initially a 1024-bit bidirectional interface with the ILLIAC IV array. The IOS design provides for possible future expansion of the real-time link with the array to 4096 bits.

The DC receives pairs of control words, called "scan descriptor, area descriptor", over the scan bus interface. The DC fetches I/O descriptors over the multiplexer word interface in response to the control words; sometimes an entire sequence of I/O descriptors will be initiated by one pair of control words. The DC sends result descriptors back over the scan bus upon the completion of I/O transactions. Certain I/O descriptors cause the DC to send words of data, fetched over the multiplexer word interface, to the CU, where they are treated as instructions by the TMU. There is a 48-bit bidirectional interface between DC and TMU for these transfers.

CONTENTS

SECTION II
PROGRAMMING CHARACTERISTICS

WORD FORMATS

There are two general categories of instructions in ILLIAC IV, called ADVAST instructions and FINST/PE instructions. The main distinction between them is that ADVAST instructions are used primarily for quadrant-related control functions (that is, manipulating logical control elements common to all 64 PEs in a quadrant), whereas FINST/PE instructions are associated more with the control of individual PEs within a quadrant. In this respect, ADVAST instructions are primarily CU related instructions, in that they are executed in the advanced station (ADVAST) section of the CU, while FINST/PE instructions are primarily PE related instructions, although principal control resides in the FINST section of the CU. Both types of instructions employ a 32-bit word length. Data words, on the other hand, are 64 bits in length. Operations involving data words may employ 32-bit or 64-bit word sizes, in either floating-point or fixed-point representation, or may use combinations of 8-bit bytes in unsigned notation. These are more fully explained in this section.

INSTRUCTION WORD FORMATS

All instruction words are 32 bits in length, although some may not use all fields available. Most instruction words must contain an operation field, that is, nine bits which specify the particular operation code, and a parity bit (exceptions are the ADVAST instructions ALIT, SLIT, and JUMP). Most instruction

ADVAST INSTRUCTIONS

NO. OF BITS IN GROUP:	5	3	8	2	1	1	4	8
	FIELD A OP CODE	INDEX INFORMATION	SKIP FIELD	FIRST OPERAND	GLOBAL/ LOCAL	PARITY	FIELD B OP CODE	SECOND OPERAND
BIT NO.:	0 — 4	5 — 7	8 — 15	16 — 17	18	19	20 — 23	24 — 31

FINST/PE INSTRUCTIONS

NO. OF BITS IN GROUP.	5	3	4	1	3	16
	FIELD A OP CODE	INDEX INFORMATION	FIELD B OP CODE	PARITY	ADDRESS USE	ADDRESS
BIT NO:	0 — 4	5 — 7	8 — 11	12	13 — 15	16 — 31

Figure 2-1. Instruction Word Formats

words also contain a 3-bit index field, the first bit of which indicates if an accumulator register is to be used for address modification, and the next two bits identify the specific accumulator to be used. Figure 2-1 shows the general formats for the two types of instruction words. Field usage is explained in detail on page 3-1 for ADVAST instructions and on page 4-1 for FINST/PE instructions.

DATA WORD FORMATS

Various options in the FINST/PE instruction repertoire permit the use of six basic bit configurations for data words, as shown in Figure 2-2. For the most part, bit usage is obvious in referring to the formats. For floating-point quantities, however, the bit assignments for the various fields are relative to the equation:

$$A = (-1)^S M \cdot 2^{(E - D)}$$

where A represents the floating-point quantity being expressed, S is the content of the sign field, M is the content of the mantissa field stated as a binary fraction, and E is the content of the exponent field (D = 16384 or 64, in decimal, respectively for 64-bit and 32-bit word sizes).

64-Bit, Floating Point

Bit No.:	0	1 — 15	16 — 63
	SIGN OF MANTISSA	EXPONENT	MANTISSA

32-Bit, Floating Point

Bit No.:	0	1 — 7	8	9 — 15	16 — 39	40 — 63
	SIGN OF OUTER MANTISSA	OUTER EXPONENT	SIGN OF INNER MANTISSA	INNER EXPONENT	INNER MANTISSA	OUTER MANTISSA
Working Bits:	0_{outer}	1_{outer} — 7	0_{inner}	1_{inner} — 7	8_{inner} — 31	8_{outer} — 31

64-Bit, Fixed Point (no sign) or Logical

Bit No.:	0 — 63
	OPERAND (NO SIGN BIT)

48-Bit, Fixed Point (no exponent)

Bit No.:	0	1 — 15	16 — 63
	SIGN OF OPERAND	▨	OPERAND

24-Bit, Fixed Point (no exponents)

Bit No.:	0	1 — 7	8	9 — 15	16 — 39	40 — 63
	SIGN OF OUTER OPERAND	▨	SIGN OF INNER OPERAND	▨	INNER OPERAND	OUTER OPERAND
Working Bits:	0_{outer}		0_{inner}		8_{inner} — 31	8_{outer} — 31

8-Bit, Fixed Point (no signs)

Bit No.:	0 — 7	8 — 15	16 — 23	24 — 31	32 — 39	40 — 47	48 — 55	56 — 63
	BYTE #1	BYTE #2	BYTE #3	BYTE #4	BYTE #5	BYTE #6	BYTE #7	BYTE #8

Figure 2-2. FINST/PE Data Word Formats

The ADVAST instruction repertoire permits the use of three different bit configurations for data words, two of which are shown in Figure 2-3. These data words are used in conjunction with the ADVAST instructions INCRXC, TXE, TXG, and TXL (Figure 2-3a) and DUPI and DUPO (Figure 2-3b). The third format is that of a 64-bit logical operand, as shown in Figure 2-2 (64-Bit, Fixed Point or Logical).

Bit No.:	0	1	2——15	16——39	40——63
	HALF-WORD IND.	SIGN OF INCREMENT	MAG. OF INCREMENT	LIMIT	CURRENT INDEX VALUE

(a) For Instructions INCRXC, TXE, TXG, and TXL

Bit No.:	0——7	8——39	40——63
	OUTER HALF WORD	INNER HALF WORD	OUTER HALF WORD

(b) For Instructions DUPI and DUPO

Figure 2-3. ADVAST Data Word Formats

NOTATION CONVENTIONS

All words, registers, and fields in ILLIAC IV are numbered starting from zero and ascending in a left to right direction or from most significant to least significant bit position. The length of a data field is specified using the following notation:

(REGISTER NAME) (High order bit position): (Length of field in bits)

For example, bit positions 40 through 63 of an accumulator register (ACAR) are referenced as ACAR 40:24.

It is sometimes necessary to replace the position or length value by an expression which is also described using the same notation. These expressions are contained in parentheses which have the meaning "the value represented by". An example of this notation is the identification of a bit position in an ACAR register that is defined in the low order bits of the ADVAST instruction register (AIR). This would be written as ACAR (AIR 26:6):1.

Exponents in ILLIAC are represented by an "excess code" notation (also called the "offset code"), rather than by sign and magnitude. This means that the zero value (offset base) of the exponent is represented by a "one" in the most significant bit followed by all "zeros". Positive exponents are formed by adding the exponent value to the offset base value; negative exponents are formed by subtracting the exponent absolute value from the offset base value.

Examples of excess code notation are given below for the 32-bit and 64-bit modes. For convenience, octal numbers are used.

Exponent Value	32-Bit Mode (Exponent has 7 bits)	64-Bit Mode (Exponent has 15 bits)
0	100	40000
+1	101	40001
-1	077	37777

Note that the peculiarity of having two representations for zero in sign-magnitude notation, namely +0 and -0, does not apply to excess code notation.

CONFIGURATION CONTROL LOGIC

The purpose of the configuration control logic is to specify the array configuration (one, two, three, or four CUs) during the execution of program instructions. The array configuration is specified by the following:

1. The configuration control registers MC0, MC1, and MC2;

2. The local/global bit (bit 18) of the ADVAST instruction to be executed, when either the instruction is executed entirely in ADVAST or passed to FINQ for subsequent execution by FINST and the MSU;
3. The FINST/PE instructions RTL and RTG;
4. For certain ADVAST instructions (CCB, COPY, ORAC, etc.), bits ADR 0:2;
5. For ADVAST memory instructions (LOAD, STORE, etc.) bit(s) 56 and/or 57 of the specified accumulator which are passed to FINQ for subsequent execution by FINST and the MSU.

All of the functional units of the CU except the TMU (ILA, FINST, ADVAST, and MSU), make use of the configuration control logic for synchronization. In addition, it is used by FINST to direct routing commands, by the MSU for address interpretation and data or instruction steering, and finally, by ADVAST for receiving data for checking results of test and skip instructions.

A "one" in bit 18 indicates a "local" action. In this case, the CU that is executing the instruction performs the operation independently; no synchronization with other CUs occurs, and there is no exchange of information or control signals. A "zero" in bit 18 denotes a "global" process. Here the CUs in the array execute the instruction in combination with one another; synchronization of the CUs is required, along with an exchange of information.

Each of the three configuration control registers (MC0, MC1, and MC2) is four bits long. The registers are readable – they can be stored in an accumulator register or in main memory – and they are writeable – they can be loaded from an accumulator or from main memory. Each register is considered as one local memory location. The three registers are also accessible by the TMU instructions SOC and SOD for either display or transmission to the DC. They may also be loaded by the TMU set and transmit instructions SAT, SLT, and SRT. The use of these registers is described in the following paragraphs.

MC0, Array Size Control Register. Each bit in MC0 corresponds to one specific, permanently-numbered quadrant. A set bit represents a particular quadrant assigned to work on the current program. Every quadrant represented in MC0 receives instructions during the instruction fetch process of ILA, receives operands from main memory during the operand fetch process of ADVAST, participates in all synchronizing at the various stations of the CU, and receives the T/F indicator during Test-Skip instructions and the CTSB instruction.

Example 1:

	Bit Position			
Register	0	1	2	3
MC0	0	1	1	1

In Example 1 there are three quadrants in the array: CU1, CU2, and CU3. CU0 is not part of the system defined by MC0.

MC1, Instruction Fetch Register. MC1 provides information concerning the location of the program in main memory. The contents of MC1 are relative to MC0, that is, the position of a set bit in MC1 refers to ONLY the bits SET in MC0, starting with the first set bit in MC0.

Example 2:

	Bit Position			
Register	0	1	2	3
MC0	1	1	0	1
MC1	0	1	1	0

In Example 2 there are three quadrants in operation: CU0, CU1, and CU3. MC1 has the second and third bits set to show that the program is stored in the second and third CUs in the array defined by MC0. Therefore of the three bits set in MC0, the second and third define the CUs that contain the program, so that in this example CU1 and CU3 are the program CUs.

MC2, Instruction Execution Register. MC2 furnishes information needed for concerted performance of information exchanges of the CUs. The contents of MC2 are relative to MC0, just as the contents of MC1 are relative to MC0.

Example 3:

	Bit Position			
Register	0	1	2	3
MC0	1	0	1	1
MC2	1	0	1	0

In example 3 there are three quadrants in operation: CU0, CU2, and CU3. This configuration indicates that CU0, CU2, and CU3 should participate in all sync operations at the various stations of the CU and that CU0 and CU3 are to exchange information (such as data for inter-CU instructions TCW, TCCW, etc.). CU0 and CU3 will also attempt to fetch operands from their memory, however, the one determined by bit 57 of the accumulator will fetch it and send it to CU0, CU2, and CU3.

The error settings of the MC register are:

Any "all zero" setting of any register;
Any "three ones" setting of MC1 or MC2;
Any setting of MC1 or MC2 in which the number of "ones" exceeds the number of "ones" in MC0.

The various valid settings are:

MC0 holds four "ones": MC1 and MC2 can have settings consisting of four, two, or one "one(s)" resulting in eleven valid settings in either register.

MC0 holds three "ones": The first three bits of MC1 and MC2 can have settings of two or one "one(s)" resulting in six valid settings for either register.

MC0 holds two "ones": The first two bits of MC1 and MC2 can have settings of two or one "one(s)", resulting in three valid settings for either register.

MC0 holds one "one": The first bit of MC1 and MC2 must have a setting of one "one" resulting in one valid setting for either register.

Certain actions occur when the registers are changed:

Changing MC0 or MC1 causes the "IWS present" bits to be reset. This results in fetching new blocks for IWS using the present contents of the instruction counter (ICR) when the current block is exhausted.

Changing MC0 or MC2 causes ADVAST to stop executing instructions and empty FINQ. The new value to be loaded into MC0 or MC2 does not replace the old value until FINST is idle. At that time the transfer to MC0 or MC2 takes place.

The ADVAST instruction repertoire can be divided into four sets, considering the configuration control logic:

1. The following instruction set is executed independently by each CU: ALIT, CACRB, CADD, CAND, CEXOR, CLC, COMPC, COR, CROTL, CROTR, CSHL, CSHR, CSUB, DUPI, DUPO, EXCHL, EXEC, FINQ, HALT, INCRXC, INR, JUMP, LDC, LDL, LIT, SETC, SKIP, SLIT, STL, and any other instruction in which bit 18 (Local/Global) is "one".

2. This set causes the array specified by MC0 to be synchronized at the beginning of the instruction and causes the CUs specified by MC2 to execute the instruction: LEADO, LEADZ, ORAC, TCCW, TCW, the TEST-SKIP instructions, and WAIT. (FINST executes RT similarly.) Note that in the case of the TEST-SKIP instructions, the CUs of the array specified by MC0 will test the result and skip.

3. This set requires examination of instruction word bits 24 and 25 to select the CU to execute the instruction: CCB, COPY, CRB, CSB, CTSBF, and CTSBT. Note that COPY is a special instruction in this group; an ACAR of the CU which is selected by examination of bits 24 and 25 is copied by the other CUs in the array. (In single-quadrant array, this instruction is a no-op.) For the remainder of the instructions in this category, bits 24 and 25 are used as follows: The configuration control logic establishes how many and which CUs are in the array. When only one CU is in the array, bits 24 and 25 are irrelevant. When two CUs comprise the array, only bit 25 is pertinent; if it is "zero", the lower-numbered CU executes the instruction. In a four-CU array, bits 24 and 25 specify the CU to execute the instruction.

4. This set includes the six ADVAST main memory operations: LOAD, LOADX, BIN, BINX, STORE, and STOREX. The main memory address is contained in the specified ACAR. To determine which CU performs the fetch, ACAR bits 56 and 57 are examined identically to AIR bits 24 and 25 above.

The LOAD, LOADX, BIN, BINX instructions are processed at both the ADVAST and FINST stations. Following ADVAST processing, the local memory location(s) referenced by these instructions are considered "empty" until processed and filled by the FINST operation. Should an ADVAST instruction reference one of these "empty" locations, the ADVAST unit will stall until the referenced location has been filled. Although this protection feature insures against timing dependent programming errors, it may slow down system throughput. Where feasible, the programmer should access data as far as possible in advance of its ADVAST use in order to minimize this delay.

FORKING AND JOINING

Occasionally it becomes necessary, when running in multiquadrant array, for the quadrant to separate (fork) and then rejoin. This capability allows for the running of local subroutines.

Forking is accomplished by resetting the configuration control registers and then branching to the area in main memory where the local subroutine resides. It must be understood that the procedures leading up to forking and the actual forking instructions are executed in multiquadrant configuration. Since changing MC0 and MC1 causes the "IWS present" bits to be reset, care must be taken that at least the resetting of MC0 and MC1, and the JUMP instruction are in the same 8-word block in IWS.

In order to fork, each CU must know what its quadrant number is so that it may properly include itself in the new configuration control register settings. This can be determined by reading the ACU, a local register which contains "own CU number", via the LDL instruction. This register, which is readable only, has four bits in the same bit arrangement as MC0. The bit which corresponds to this CU number is hard-wired ON, all others being hard-wired OFF. Prior to forking, it is recommended that the configuration control registers be stored to facilitate joining.

At the completion of the local subroutine, joining can be accomplished as follows: the desired array configuration must be determined, this condition must be set into a WAIT instruction (together with bit 27), and then the "request join" option of the WAIT instruction (bit 27 ON; bits 28:4 set to the desired configuration) must be executed (this can be facilitated by the use of ACAR indexing). If all quadrants in the desired array have executed this instruction, then all the quadrants in the new array will be in sync and their MC0 will reflect the desired configuration setting. ACR bit 5 can be tested to determine whether this did indeed occur. If it did, then MC1 can be changed to the desired setting and the program may JUMP to the global routine. If sync had not in fact been achieved, the program has the option of either repeating the special WAIT instruction until the quadrants do sync, or it may disregard the other quadrants and continue in single-quadrant array.

BASIC CU REGISTERS

Following is a list of the common registers accessible via the local address contained in ADVAST instructions.

Octal Address	Register Mnemonic	Function
000 - 077	Dnn (ADB)	General registers, local data buffer, broadcast buffer
100 - 103	AC0-3 (ACAR0-3)	Accumulators, Index Registers
104	ICR	Instruction Counter
105	IIA	Interrupted Instruction Address
140	ACR	ADVAST Control Register (see p. 2-20)
142	AIN	Interrupt Register
144	ALR	ADVAST Local Address Register
145	AMR	Mask Register
151-153	MC0-2	Configuration Control Registers
154	ARE	Memory Write Error Indicators
155	TRI	Input Communication Register (from B 6700)
156	TRO	Output Communication Register (to B 6700)
157	ACU	Own Quadrant Number (read only)

OPERATIONAL CONTROL

This section presents the procedure that mechanizes interrupts within the ILLIAC IV system and also describes the registers which control interruption and normal operation of the Control Unit (CU). The basic philosophy behind this organization is emphasized at times to convey to the user the precautions required in the programmatic manipulation of these controls.

In the ILLIAC IV repertoire there are no "privileged" instructions. In this sense, all programs, whether performing interrupt or noninterrupt (normal) processing, have complete access to the system elements. As a result, the user must maintain rigid accountability for proper manipulation of these facilities. Since few provisions for an executive or control program have been incorporated to facilitate the selection and activation of user requests, it is incumbent upon the user to conform to the programming conventions established for the system so that optimum exploitation of the facilities is assured.

INTERRUPT HANDLING

Interrupts in ILLIAC IV are recognized in the ADVAST section of a CU. An interrupt is caused by the occurrence of a masked condition, that is, a bit is set (by program) in the interrupt mask register (AMR), and if subsequently, the masked condition occurs, the interrupt will be recognized. Regardless of the setting of the mask bit (and, therefore, of the recognition of the interrupt) the occurrence of the condition will cause a bit to be set in the interrupt register (AIN), which bit corresponds with the bit in AMR. The purpose of the interrupt feature is to provide automatic recognition of a condition which either may require immediate response, or the occurrence of which is unexpected and asynchronous with regard to the rest of the program (for example, intercommmunication, etc.). When such recognition is not required, the interrupt feature may be bypassed for a particular condition by reloading the AMR with the appropriate bit reset. If the interrupt feature is bypassed, the information content of the AIN may be sampled at the convenience of the program being executed, as a part of its normal execution cycle.

Interrupt Recognition

Interrupt recognition occurs on an array basis; that is, if one CU in an array (as determined by MC0) is interrupted, then all CUs in the array will be interrupted. However, those CUs which actually caused the interrupt may be determined by ascertaining which CUs have a masked AIN bit (by reading each AIN and comparing it against the respective AMR).

It should be noted that there is no automatic change to the configuration control registers (MC0, MC1, MC2) when an interrupt occurs. Therefore if an interrupt occurs while operating in multiquadrant configuration at least the initial processing must be a multiquadrant routine, using the same array size as just prior to the interrupt.

Types of Interrupts

There are two types of interrupts: recoverable and nonrecoverable. Type 1 (recoverable) interrupts will occur only when all quadrants in the array are synchronized. That is, the condition will be recognized immediately, but the CUs will not be interrupted until all the CUs are synchronized for the execution of an ADVAST global instruction, or at the end of every instruction when there is only one quadrant in the array.

Type 2 (nonrecoverable) interrupts will occur at the next clock pulse after the condition is recognized. The CUs will be initialized by signaling the TMU to execute a SIV instruction (with bits 39-44 of TCR set), which will cause ADVAST, FINST, and ILA registers and control latches to be reset or set to the idle state. The MSU will be similarly initialized at the completion of I/O cycles. Interrupt processing would then proceed as for Type 1 interrupts.

Recoverability and nonrecoverability relate to the ability of the machine to return, following an interrupt, to a known state relative to the point of interruption without special provisions in the interrupted program.

Table 2-1 indicates interrupt type for all interrupts.

Interrupt Status Storage

In order to resume the execution of a program after interrupt processing, it is required that the status of the CU be stored before entry into the interrupt state and some known state restored upon exit. This storage function is accomplished in register storage within the CU and in the PE memory array.

Instruction Counter (ICR) and Interrupt Address Register (IIA)

The instruction counter (ICR) is a 25-bit register which is used in both normal and interrupt programs to control the sequence in which instructions are executed. Another register, the interrupt address register (IIA), which is of the same length as the ICR, is loaded from the ICR when an interrupt occurs. The IIA reloads the ICR when the "interrupt return" instruction (INR) is given and is read and set as a local register.

Control Information Storage

No additional storage is required to store the logical condition of the quadrant when an interrupt occurs. However, if interrupt processing alters the setting of the interrupt controls (specifically, in the interrupt, mask, and control registers), the programmatic storage and restoration of their settings must be accomplished within the interrupt routine. Note that all control registers which are not part of the local memory cannot be stored in the accumulators or array memory but must be stored outside the array.

Data Storage

The initiation of an interrupt causes the contents of ACAR0 to be stored in memory location eight or nine, depending on the setting of the "alternate interrupt base in use" bit in the ADVAST control register (ACR). ACAR0 is then loaded with an index value containing an increment field of +1, a limit field of 32, and an index field of 16. Words 16 to 31 can be used for storing up to 16 words of the ADVAST data buffer (ADB) should this space be required for interrupt processing. ACAR0 provides the address for the store operations. Words 9 (or 10) through 15 are for the storage of other ADVAST registers (for example, other ACARs and the AMR) that may be altered during interrupt processing.

Upon the execution of the INR instruction, ACAR0 will be reloaded with the contents of memory location eight or nine, depending on the setting of the "alternate interrupt base in use" bit in the control register (ACR).

Interrupt Routine Instruction Storage

The instruction counter (ICR) will be set to "zero" (or "one", depending upon the setting of the "alternate interrupt base in use" bit in ACR) to cause the first block of interrupt processing instructions to be fetched from array memory words 0 through 7. As the local memory information is stored in words 8 through 31, a branch is required within this area of the block. (The alternate interrupt base was provided to allow the programmer to bypass a suspected memory failure in PEM 0.)

Entering the Interrupt Mode

Because of the interrupt mechanism described above, there can be only one level of recoverable interrupt (multiple interrupts would destroy interrupt return information). However, the simultaneous occurrence of interrupt conditions is considered a single level of interrupt. To protect the return information until the execution of an interrupt routine, use of the AMR as the interrupt enabling device is suspended and a "hardware" mask is employed instead. The hardware mask prohibits the recognition of any interrupt conditions except those indicating hardware malfunctions, and remains in use until ACR02 is reset via INR or CACRB-2. Also, while the hardware mask is in use, "storage protect" is disabled, regardless of the setting of ACR13, until the hardware mask is replaced by AMR (via INR or CACRB-2).

During interrupt processing (that is, when ACR bit 1 is "one"), a masked interrupt condition will cause the CU to stop and interrupt the B 6700 system by setting TCI bit 5 (CU halted). When a "branch trace" interrupt occurs, the contents of the present ICR are loaded into TRO 40:24, ICR (24:1) goes into TRO (0:1), the other TRO bits are set to zero, and TCI bit 7 is set (indicating the right half of TRO is loaded).

INTERRUPT, MASK, AND CONTROL REGISTER FUNCTIONS

This section identifies the information content of the interrupt (AIN), interrupt mask (AMR), and control (ACR) registers. The length of these registers is 16 bits.

The functions of the bits in the AIN and AMR registers are listed in Table 2-1, and for the ACR register in Table 2-2. The setting of any of these bits indicates to the program that the condition, as stated, is true. In the case of the AIN and AMR, the bit positions in each register relate to the same function and thus have the same number. For ACR, a correspondence in bit positions 11 through 15 exists with the AIN/AMR. In general, bits that might be used or interrogated together are grouped together.

The first four AIN/AMR positions are grouped together because they represent what are most probably hardware malfunctions. The "hardware mask" (used immediately after an interrupt occurs until the mask register is loaded) is implemented to permit a second interrupt (i. e., a stop) condition to become true, and contains bits 0-3 of AMR.

Table 2-1. Functions of Bits in Interrupt (AIN) and Mask (AMR) Registers

AIN/AMR Bit No.	Interrupt/Mask Name	Type*	Condition(s) for Setting or Functions Masked
0	Spare	2	Available for indicating power failure to operator or to the B6700.
1	Parity error in instruction	2	Sum of bits loaded from IWS to AIR modulo 2 is zero, except for instructions SLIT, ALIT, JUMP, or instruction loaded in AIR due to EXEC.
2	Undefined instruction	2	Op code fields of AIR indicate instruction not one of ILLIAC IV instruction set (see page 2-21, Illegal CU Addresses).
3	CU stalled	2	CU has waited 15 milliseconds for another instruction, or HALT was executed, or breakpoint was reached, or second interrupt has halted all operations.
4	Improper setting of MC0, MC1 or MC2	2	Any configuration register with all zeros; MC1 or MC2 contains three ones; or, bit position is set that is greater than the number of ones in MC0.
5	Improper local address	2	Nonexistent or inaccessible ADVAST local address requested as effective local address (after indexing, if specified). Not set for BIN or BINX instructions. (See page 2-21, Illegal CU Addresses).
6	ADB wrap-around	2	Effective ADB address is greater than octal 77 in BIN or BINX instruction.
7	Execute loop	2	AIR contents replaced by ACAR value which has identical op code and ACAR address.
8	Skip distance equals minus one	2	Skip field of instruction for modification of ICR has value of minus one (endless loop on the same instruction).
9	User program request	1	INR instruction executed and ACR bit 1 (processing interrupt) is reset (zero).
10	Spare	2	
11	PE overflow	2	F mode bit in any of the 64 PEs is set; or, ACR bit 10 (32-bit mode) is set and F1 mode bit in any of the 64 PEs is set (see page 4-15, F Bits).
12	Spare	1	Available for indicating that block of real-time information has been stored in predesignated area of ILLIAC IV storage.
13	Attempted write to protected storage	2	ACR bit 13 (storage protect enable) is set and the hardware mask is not in effect, and an attempt was made to write into PE memory at effective address less than octal 1000. (Note that write operation is inhibited.)
14	Branch trace	1	ACR bit 14 (branch trace enable) is set and ICR has been altered by EXCHL, STL, LOAD(X), SKIP, or JUMP instruction.
15	TRI loaded	1	Set by controls in Test Maintenance Unit when a set-transmit to TRI is executed.

* Type 1 interrupt - recoverable,
Type 2 interrupt - nonrecoverable.

Table 2-2. Functions of Bits in CU Control Register (ACR)

Bit No.	Can be Set/Reset via CACRB	ACR Function	Description
0	Yes/Yes	Test result	ADVAST comparison indicator - When set, previous test was true, and when reset, previous test was false.
1	No / No	Processing interrupt	Processing interrupt indicator - When set, the CU is in the interrupt processing mode. Set whenever corresponding bits of the AIN and AMR are set. Reset during the execution of INR.
2	No/Yes	Hardware mask in use	Hardware mask in use indicator - When set, the CU does not enter the interrupt processing mode for any interrupt except bits 0-3. Set when CU is initialized, and concurrent with the setting of ACR1. Reset automatically by the INR instruction.
3	No / No	ALR busy	LOAD or BIN pending indicator - When set, indicates that a "read data from array memory" is in progress at one of the CU stations or in the FINST queue. The indicator is set at the beginning of the operation and reset at the end of the operation.
4	Yes/Yes	Alternate interrupt base in use	Base address indicator - Used to determine the starting address for interrupt programs (ICR = 00... (ACR4)0) and the location of array memory in which to store accumulator zero (00...100 (ACR4)).
5	Yes/Yes	Quadrants awaiting synch indicator	This bit, when used in conjunction with the "join" option of the WAIT instruction, will indicate whether all quadrants, which were specified in the ADR portion of the WAIT instruction, are ready to synchronize. Care must be taken to ensure that this bit is reset before entering the "join" routine.
6	No / No	FINST idle	FINST complete indicator - Reflects the status of the FINST idle level during the previous clock period.
7	No / No	BIN/LOAD in progress	BIN/LOAD indicator - When set, indicates that the last memory operation processed at ADVAST was a BIN; when reset, the memory operation was a LOAD.
8	Yes/Yes	Non-overlap mode	When set, the ADVAST, FINST, and PUs must complete the current operation, all CUs in the array must be synchronized, and enough time left for any interrupt to reach AIN before the next instruction is executed.
9	Yes/Yes	Exponent underflow inhibit	When set, FINST inhibits transfer of exponent underflow condition(s) to F or F1 bits in all PEs.
10	Yes/Yes	32-bit mode	32/64 bits mode indicator - A copy of this indicator is transmitted to FINST with every PE instruction. When set, the PE will receive the proper enables to operate in 32-bit mode; when reset, the PE will receive the proper enables to operate in 64-bit mode.
11	Yes/Yes	Spare	
12	Yes/Yes	Spare	
13	Yes/Yes	Storage protect enable	Write in protected area indicator - When set and the hardware mask is not in effect, the protected area of memory is protected against writes. If such a write is attempted . while bit 13 is on, by any memory user other than the I/O, then error latches in memory are set which cause the setting of AIN13. Error latches are reset whenever ACR13 is reset or whenever INR resets ACR1.
14	Yes/Yes	Branch trace enable	Branch trace enable indicator - When set in the non-interrupt mode and a change of ICR is attempted, then AIN 14 is set and the old contents of ICR are stored in TRO 40:24.
15	Yes/Yes	TRO loaded	TRO loaded indicator - Set by LOAD(X) to TRO and reset when the data has been read by the CDC. It is not set by other actions which may load TRO such as STL, EXCHL, and branch trace interrupt.

Interrupt (AIN) and Mask (AMR) Registers

The function of the AIN register is to preserve an indication to the program that a condition has occurred, except as specifically indicated otherwise in Table 2-1. The function of the AMR register is to permit or inhibit recognition of an interrupt condition by use of the interrupt mechanism. The reading of the AIN (by a STORE (X) or LDL instruction) causes it to be cleared.

ADVAST Control Register (ACR)

The function of the ACR register is to provide indications to the program of the state of an ILLIAC IV quadrant which are required for the programming of executive or control programs. Unless specifically excluded (as noted in Table 2-2), all ACR bits can be set or reset by the CACRB instruction. The bits at all times indicate the actual state of the system.

ILLEGAL INSTRUCTION/ADDRESS HANDLING

This section identifies the procedures for handling the various types of illegal instructions/addresses.

Illegal CU Instructions

All undefined instructions are illegal. The following cases are included as undefined instructions:

(a) All the blanks of Table 3-1;

(b) Most of the blanks of Table 4-1, the exceptions being octal codes 2600, 2601, 2602, 2603, 2011, 2012, 2610, 2611 (which are used for communication from ADVAST to FINST), and 3416, 3417, 3616 and 3617 (which are meaningless variants of the FINST/PE instructions AD and SB);

(c) Any FINST/PE instruction whose ADR field is 110 or 100 (register code), and whose address field specifies an invalid combination of source and destination registers (see FINST/PE transmit instruction LD).

Illegal CU Addresses

Three categories of CU operations are of interest: ADVAST arithmetic instructions, all other ADVAST instructions, and TMU instructions. Each of these categories makes unique use of CU registers which are identified or addressed by an 8-bit address. These registers/addresses (spare included) are divided into four groups with legal or illegal status according to Table 2-3.

In Table 2-3, the following ADVAST instructions are classed as "ADVAST Arithmetic":

CADD
CAND
CEXOR
COR
CSUB
LESS--
GRTR--
TXE--
TXG--
TXL--

and the following ADVAST instructions are classed as "All Other ADVAST":

BIN
BINX
DUPI
DUPO
EXCHL
LDL
LOAD
LOADX
STL
STORE
STOREX

ADVAST Valid Registers

ADVAST instructions that may address registers and yield results are defined in Table 2-4.

Interrupt Selection

Interrupts resulting from illegal instructions/addresses are always returned for processing to the originating source. That is, if the illegal instruction

Table 2-3. Legal/Illegal CU Addresses

Group	Octal Address	Advast Arithmetic	All Other Advast	TMU
1	000-137	Legal*	Legal	Legal
2	140-177	Illegal	Legal	Legal
3	200-277	Illegal	Illegal	Legal
4	300-377	Illegal	Illegal	Illegal

*ICR and IIA valid for CADD and CSUB only.

Table 2-4. Valid Registers for ADVAST Instructions

	REGISTERS												
ADVAST INSTRUCTIONS	AC0-3	ADB	AMR	AIN	ALR	ARE	ACR	ACU	ICR	IIA	MC0-2	TRI	TRO
BIN		Yes											
BINX		Yes											
CADD	Yes	Yes							Yes	Yes			
CAND	Yes	Yes											
CEXOR	Yes	Yes											
COR	Yes	Yes											
CSUB	Yes	Yes							Yes	Yes			
DUPI		Yes											
DUPO		Yes											
EXCHL	Yes	Yes	Yes	Yes	Yes	UN		UN	Yes	Yes	Yes	UN	Yes
LDL	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
LOAD	Yes	Yes	Yes	Yes	Yes	UN		UN	Yes	Yes	Yes	UN	Yes
LOADX	Yes	Yes	Yes	Yes	Yes	UN		UN	Yes	Yes	Yes	UN	Yes
STL	Yes	Yes	Yes	Yes	Yes	UN		UN	Yes	Yes	Yes	UN	Yes
STORE	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
STOREX	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
LESS--	Yes	Yes											
GRTR--	Yes	Yes											
EQLX--	Yes	Yes											
TXE--	Yes	Yes											
TXG--	Yes	Yes											
TXL--	Yes	Yes											

UN = Undefined. The instruction can address these registers but the results are undefined.

Yes = The instruction can address these registers and the results are defined.

Blank = The instruction cannot address these registers.

was received from ILA via IWS, an AIN bit is set which allows ADVAST to process the interrupt. However, if the instruction was received from the DC-TMU (that is, a TMU or EFA ADVAST instruction, or EFF FINST instruction) a TCI bit is set which allows the B6700 control computer to process the interrupt.

INPUT-OUTPUT CONTROL

This section describes the input-output control functions performed by the various elements of the I/O subsystem. These elements are the Descriptor Controller (DC) and Disk File Control (DFC), the Buffer I/O Memory (BIOM), and the Input-Output Switch (IOS). Since the main input-output control functions involve the use of descriptors for defining operations to be performed, emphasis is placed on use of descriptors - scan, I/O subsystem, and result - for controlling I/O operations within the ILLIAC IV system. This discussion is included with the description of the DC.

DESCRIPTOR CONTROLLER (DC)

The main control functions of the I/O subystem reside in the Descriptor Controller (DC). The DC (Figure 2-4) is comprised of the following control logic areas:

Scan and Result Descriptor: Handles the scan bus from the processor and assembles a result descriptor when requested by the processor.

Word Bus Control: Buffers I/O descriptors and data between the B6700 memory and other areas DC. It also accepts B6700 memory addresses from the scan bus, CU descriptor control, and list control areas of the DC.

CU Descriptor Control: Stores and executes all I/O descriptors for the CUs. It controls the transmission of data between the B6700 and CUs, sends instructions to the CUs, and handles interrupts from the CUs.

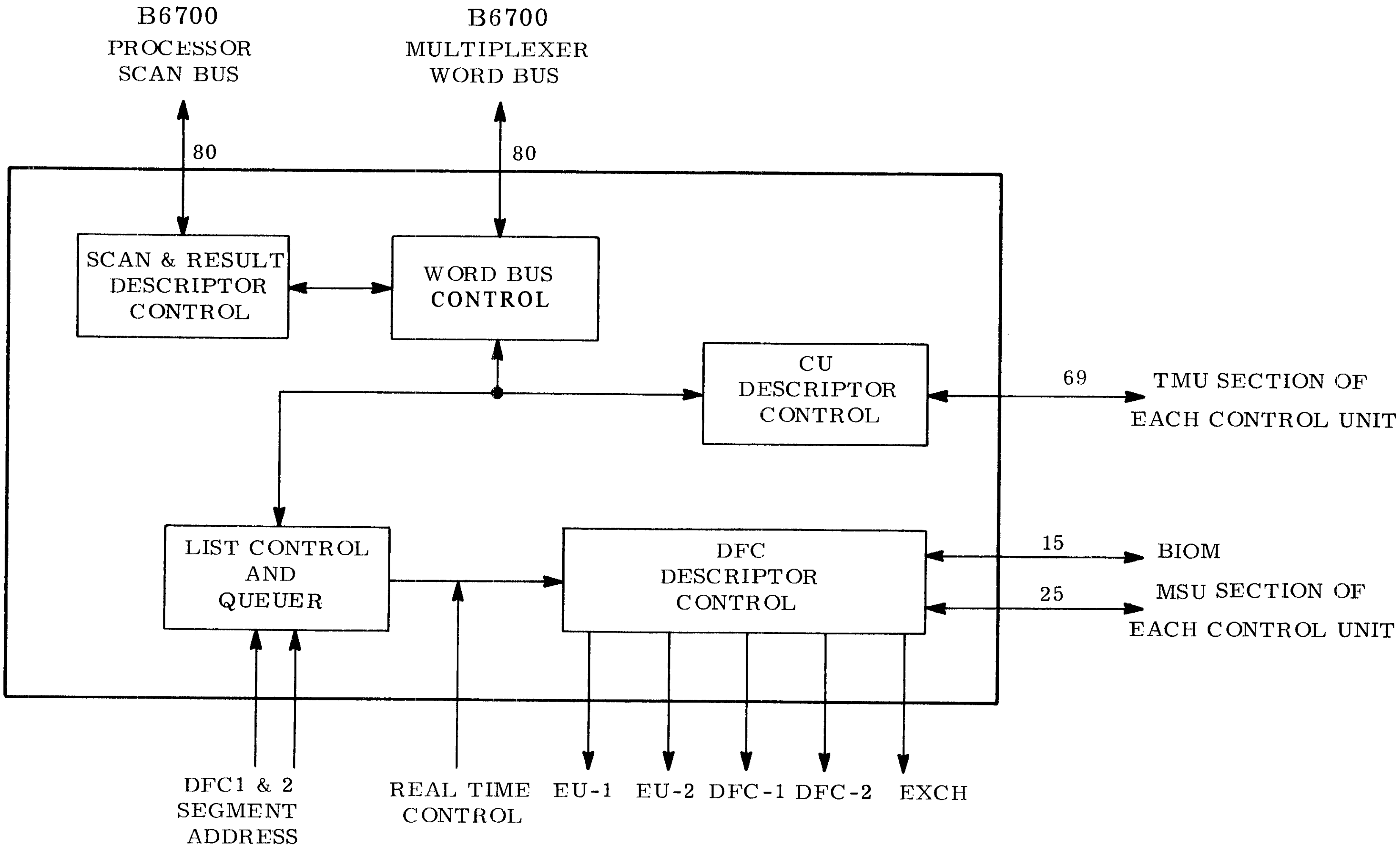


Figure 2-4. Descriptor Controller

Disk File Control 1 and 2 Descriptor Control: This section combines the control functions of the two identical DFC controls, the BIOM address register, and the disk exchange select registers into one logical grouping. Real time external controls enter here into the designated disk descriptor control. It receives information from the queuer, and controls the address for the array memory or BIOM for transfer to the proper memory location when needed. Control lines are provided to effect data flow to the DFCs and to send the disk address to the selected EU.

Queuer: This is a functionally separate logic area of the DC and represents that portion of the hardware which allows the sequential execution of disk transactions to minimize latency time. The queuer contains storage area for 24 disk transactions.

List Control: The function of the list control is to keep the queuer full and to allow the processor to complete the execution of a group of I/O descriptors without being interrupted for a descriptor fetch operation. The descriptors are fetched from B6700 memory as a group as directed by the List I.D. in the first word of the respective I/O descriptor. The list is a sequential group of I/O descriptors in the B6700 memory with List I.D.'s not equal to zero. A list is started by either a List Head I/O descriptor or by any other I/O descriptor with a List I.D. not equal to zero. The List I.D. field is two bits and permits three different lists to be active at one time. A maximum of 15 descriptors is permitted in each list. A second list (or third) can be started by an I/O descriptor within the first list having a different List I. D. A list is ended when a List Tail I/O descriptor is executed having a corresponding List I. D. value. A result descriptor is sent to the processor when the last I/O descriptor of a particular list is executed. If an error is detected in a list descriptor being fetched from the B 6700 memory, under control of the list controls, the remainder of the descriptors in the list are not fetched. The active lists are tailed. A result descriptor is returned for the descriptor in which the error was detected (see Queuer Result Descriptors for the type of relevant error detection).

Scan Bus

The scan bus of the DC is used for initiating operations between the B6700 processing system and other elements of ILLIAC IV, or between various elements of ILLIAC IV, under the control of the B6700. At present it is used only for operations that involve the B6700-DC interface but it has the capability for future expansion to accommodate additional devices if required. This is evident in the bit utilization of the scan descriptors as discussed below, wherein certain bits of these control words are defined only for operations with the DC.

The scan bus consists of a standard memory interface comprising 80 signal lines. Eight of these are used as control lines, 20 are address lines, and 52 are data lines. The address lines are used only for fetching scan descriptors from B6700 stack top in the processor and the data lines are used only for sending/receiving descriptors to/from the B6700 processor. Descriptors involving exchanges with B6700 memory utilize the word bus.

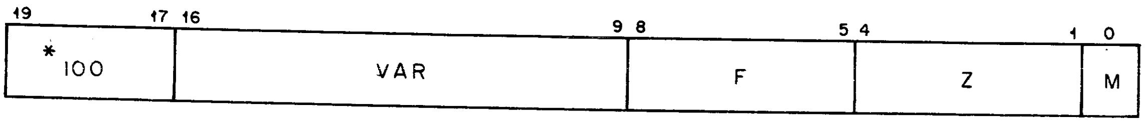
Scan Descriptor Usage

The B6700 supervisory program uses scan descriptors to initiate operations in the DC. As presently defined, the five types of operations that may be performed using these descriptors are:

Initiate I/O (scan out);
Interrogate peripheral status (scan in);
Read result descriptor (scan in);
Set exchange (scan out);
Clear queuer (scan out).

Each of these operations is initiated by the execution of an associated scan command in the B6700 processor. Upon execution of the scan command, the top word in the B6700 processor stack (scan descriptor) is placed on the address lines of the scan bus to identify to the DC the function to be performed. The second word of the stack (area descriptor) provides the sink or source for additional data required for scan in/scan out. The format for a scan descriptor is as follows:

Scan Descriptor Format



*Bit 19 = 1 for DC transfers.

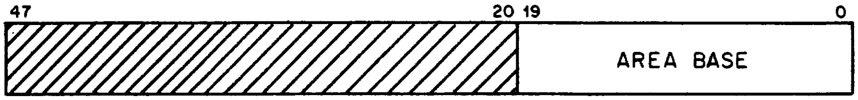
Note: For all descriptors, the left-most bit in a field is the most significant bit and the right-most bit is the least significant bit.

Field	Function
M	Designates unit(s) addressed by scan descriptor. M = 0 indicates all units on scan bus are addressed; M = 1 indicates only those units identified by the Z field are addressed.
Z	Z = 00 identifies DC as unit addressed by scan descriptor.
F	Function code. Identifies scan command to DC.
VAR	Used for variants of the "interrogate" descriptor and "set exchange" descriptor.

The function field (F) identifies the operation to be performed by the DC. The execution of these commands is described below:

Initiate I/O (F = 0000): Initiate the specified operation on the unit designated. The DC uses the area base address specified by the area descriptor to fetch an I/O descriptor from B6700 memory. The command is not accepted by the DC if the DC is already performing an "initiate I/O" operation. In this event, the processor will detect an invalid address interrupt. The format for the area descriptor is as follows:

Area Descriptor Format



Interrogate Peripheral Status (F = 0001): This scan-in descriptor will cause the DC to interrogate the status of the List Controls, the Queuer, the Disk and Disk Controls, and the Control Units, as determined by the variant bit set into the VAR field. The bit assignments are as follows:

Variant Bits								
16	15	14	13	12	11	10	9	Description
0	0	0	0	0	0	0	1	Interrogate status of List
0	0	0	0	0	0	1	0	Interrogate status of Queuer
0	0	0	0	0	1	0	0	Interrogate status of Disk
0	0	0	0	1	0	0	0	Interrogate status of CU

The result descriptor to be returned following the execution of this scan-in descriptor will have one of the following formats, as appropriate:

List Status Variant

Bits	Description
0, 1	1 1 = Software attention with or without hardware exception
5-2	Number of descriptors in the Queuer from the three lists
6	1 = List head FF for List 1 Set 0 = List head FF for List 1 Reset
7	1 = List tail FF for List 1 Set 0 = List tail FF for List 1 Reset
11-8	Number of descriptors in Queuer from List 1
12	1 = List head FF for List 2 Set 0 = List head FF for List 2 Reset
13	1 = List tail FF for List 2 Set 0 = List tail FF for List 2 Reset
17-14	Number of descriptors in Queuer from List 2
18	1 = List head FF for List 3 Set 0 = List head FF for List 3 Reset
19	1 = List tail FF for List 3 Set 0 = List tail FF for List 3 Reset
23-20	Number of descriptors in Queuer from List 3
47-28	Contents of the List Address Register

Queuer Status Variant

Bits	Description
0, 1	1 1 = Software attention with or without hardware exception
2, 3	1 1 = Queuer full 0 1 = Queuer not full, not empty 0 0 = Queuer empty
4	1 = Queuer location 00 contains a descriptor 0 = Queuer location 00 does not contain a descriptor
27-5	Same as bit 4 for Queuer locations 01 through 23 respectively

Disk Status Variant

Bits	Description
0, 1	1 1 = Software attention with or without hardware exception
2	1 = DFC-1 busy 0 = DFC-1 not busy
18-3	0 = Storage units 00 through 15, respectively, associated with "DFC-1 ready" 1 = Storage units 00 through 15, respectively, associated with "DFC-1 not ready"
22-19	Specifies the electronic unit controlled by DFC-1
23	1 = DFC-2 busy 0 = DFC-2 not busy
39-24	0 = Storage units 00 through 15, respectively, associated with "DFC-2 ready" 1 = Storage units 00 through 15, respectively, associated with "DFC-2 not ready"
43-40	Specifies the electronic unit controlled by DFC-2

CU Status Variant

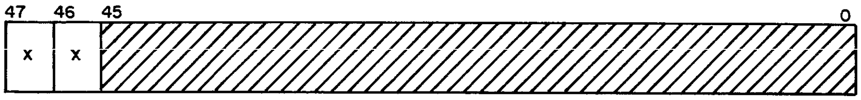
Bits	Description
0, 1	1 1 = Software attention with or without hardware exception
8	1 = CU #0 not ready 0 = CU #0 ready
10	1 = CU #1 not ready 0 = CU #1 ready
12	1 = CU #2 not ready 0 = CU #2 ready
14	1 = CU #3 not ready 0 = CU #3 ready

Read Result Descriptor (F = 0010): When access has been granted to the DC, a result descriptor will be placed on the scan bus. (See Result Descriptor discussion, page 2-36, for word format and bit content of the responding descriptor.)

Set Exchange (F = 0011): The DC will respond to this command by setting the disk exchange configuration registers to the value on the scan bus. One of two area descriptor formats applies, as determined by the setting of variant bit 9. (The queuer in DC must be empty of I/O descriptors before this command can be accepted.)

If variant bit 9 = 1, then the following area descriptor applies to the real-time device:

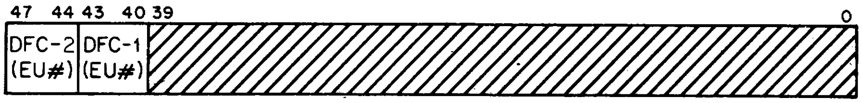
Area Descriptor Format



Bits	Description
46	1 = DFC-1 assigned to real-time device
47	1 = DFC-2 assigned to real-time device

If variant bit 9 = 0, then the following applies:

Area Descriptor Format



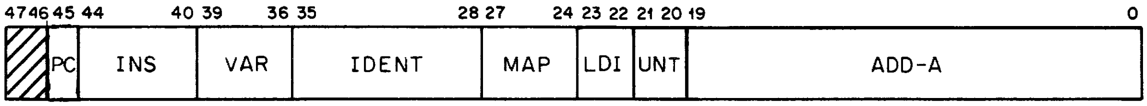
Bits	Description
40-43	Switch designated EU to DFC-1 control
44-47	Switch designated EU to DFC-2 control

Clear Queuer (F = 0100): This instruction clears the 24 occupied and 24 priority FF's. All list counters are cleared, as is the list address register.

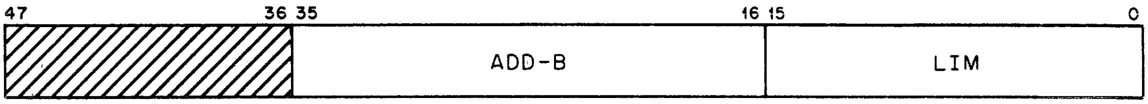
I/O Descriptor Usage

The I/O descriptors are comprised of one or two 48-bit words, according to the function to be performed. Before the I/O descriptor can be fetched from B6700 memory, a scan descriptor must be executed by the B6700 specifying an "I/O initiate" command. Execution of the scan descriptor results in an area descriptor being sent from the B6700 to the DC via the B6700 scan bus. The area descriptor gives the B6700 memory address where the I/O descriptor is stored. Upon receipt of this information, the DC uses the area descriptor to fetch the I/O descriptor from B6700 memory via the B6700 multiplexer and the word bus. The operation specified by the I/O descriptor is then performed as required. The I/O descriptor word formats are as follows:

First I/O Descriptor Word



Second I/O Descriptor Word



Field	Function
PC	Parity Control. When zero, the I/O functions normally (with odd parity). When one, the parity generator of the DFC is disabled. This bit is used for diagnostic purposes only.
INS	Instruction field. Specifies instruction to be performed, as listed in Table 2-5.
VAR	Variant field.
IDENT	Identifier field. Identifies I/O descriptor and the B 6700 control program where it originated. Program uses field to compare against respective result descriptor, which must bear same program identification.
MAP	Control field. Contains routing information for controlling distribution of data to and from array quadrants, for both CUs and PEMs.
LDI	Descriptor identifier (List I. D.). LDI = 00 indicates result descriptor required upon execution of I/O descriptor. LDI = 01, 10, or 11 indicates result descriptor to be delayed until last I/O descriptor bearing List I. D. 01, 10, or 11, respectively, is executed. Note that if an error is detected when any descriptor is being stored in the Queuer or when any descriptor is being executed, a result descriptor is returned for that descriptor.
UNT	Designates which of the two DFCs is to be used in the operation.
ADD-A	Address field. Used to specify one of the start addresses for data transfer operations between storage devices.
ADD-B	Address field. Used to specify one of the start addresses for data transfer operations between storage devices.
LIM	Specifies limit for data transfer operations; a count of data transfers (1024 bits each) for array memory, or word count for B6700 memory or BIOM.

The specific I/O descriptor fields for the various operations are listed in Table 2-5. Each entry indicates the operation to be performed, its corresponding INS field contents, and the other fields of the descriptor that are used. A detailed description of the various operations follows:

Table 2-5. Operations Specified by Instruction Field of I/O Descriptor

Operation	First Word						Second Word	
	INS	VAR	MAP	LDI	UNT	ADD-A	ADD-B	LIM
Write CU	01	Y	Y				B6700	Y
Read CU	02		Y				B6700	Y
Scan CU	03		Y				B6700	Y
Stop CU	04		Y					
List Head	14			Y		List(B6700)		
List Tail	15			Y		List(B6700)		
Transfers:								
Disk-Array	16	Y	Y	Y	Y	Disk	Array	Y
Array-Disk	17	Y	Y	Y	Y	Disk	Array	Y
Disk-BIOM	18	Y	Y	Y	Y	Disk	BIOM	Y
BIOM-Disk	19	Y	Y	Y	Y	Disk	BIOM	Y
BIOM-Array	20	Y	Y	Y	Y	BIOM	Array	Y
Array-BIOM	21	Y	Y	Y	Y	BIOM	Array	Y
Array-Real Time Device	24	Y	Y		Y		Array	Y
Real Time Device-Array	25	Y	Y		Y		Array	Y

Y = Field is used.

Write CU: Transfer "n" words sequentially, singularly or jointly, to the TMU input register (TRI) of the designated CUs from B6700 memory starting at address ADD-B. The number of words, n, is specified by the word count in LIM. The transfer is jointly if bit 1 of VAR is set; otherwise it proceeds singular in a sequential manner. The CUs are designated by MAP, which has four bits, one for each CU. If all four bits of MAP are set, all four CUs are designated as destinations.

Read CU: Transfer "n" words (48 bits) from the TMU output register(s)(TRO) of the designated CU(s) and store sequentially in B6700 memory starting at the address specified in "ADD-B" (CMAR). If more than one CU is specified in the map field, the transfers will be from the specified CUs in a sequential manner (bit 1 of "VAR" (bit 36) is ignored). If bit 2 of "VAR" (bit 37) is a zero, the number of words specified will be read one at a time, in sequence, from each CU specified in the map field, and stored in sequential locations in the B6700 memory. If bit 2 of "VAR" (bit 37) is a one, the number of words specified will be read two at a time, in sequence, from each CU specified in the map field.

Scan CU: Used for read out of addressed CU registers during diagnostic operation. First, a word is "written" (as in the "write CU" operation above) to send an instruction word to the CU (usually an "SOC" or "SOD" to indicate to the TMU what CU register should be placed in the TMU's output register (TRO)). Next, a word is "read" (as in the "read CU" operation above) from one half of the TRO into the same B6700 memory location from the word just written. Then, if bit 2 of "VAR" (bit 37) equals a one, a second word will be read from the same CU (the second half of TRO) into the next sequential B6700 memory location. If bit 2 of "VAR" (bit 37) equals zero, a second word is not read from the CU. The B6700 memory address is then incremented and the above procedure repeated until the word count in the "LIM" field is reached. The CU's are designated by the map field. If more than one CU is specified in the map field, the transfers will be to/from the specified CUs in a sequential manner (bit 1 of "VAR" (bit 36) is ignored).

Stop CU: Stop instruction issued to CU(s) designated in MAP. This is a one-word descriptor.

List Head: This is a one-word descriptor which identifies the descriptor list in LDI and specifies the B6700 memory location (ADD-A) which contains the first I/O descriptor in the list. The I/O descriptors comprising the list must be in sequential order in B6700 memory. The first I/O descriptor of a list may act as the head, in which case its location is the list address.

List Tail: This is a one-word descriptor which denotes the end of the descriptor list identified in LDI. After locating the list tail, the DC will write a "lock" of all zeros in the associated memory location. If the program is to extend the list, this must be done before the lock is effected, otherwise it must create a new list

Disk-to-Array Transfer: Transfer "n" words (1024 bits) from the designated electronics unit (EU) to the designated section of the array. UNT identifies the DFC to be used in the transfer and ADD-A specifies the start for the disk segment. The destination section of the array is identified by MAP and bits 3 and 4 of VAR as described below; the array memory start address is given by ADD-B. The number of 1024-bit words transferred is defined by the word count in LIM. Each bit in the MAP corresponds to a particular PE quadrant (PEQ).

If one PEQ is desired, its respective MAP bit should be set. If two PEQs are desired, their respective MAP bits should be set. The specification of three PEQs is not allowed. If all four quadrants are desired, all four MAP bits should be set. Data is always transferred sequentially to the designated sections of the array. Bit 3 of VAR is set if only one-fourth of each mapped PEQ is desired; bits 3 and 4 are set if only one-half of each PEQ designated is desired - and must start at the first or third quarter. The least significant bit of VAR is set to indicate priority, overriding all other I/O descriptors in the queue for disk access. ADD-A provides the disk address, as follows: bits 13 through 16 are the storage unit (SU) designation; bits 11 and 12 are the track number, and bits 0 through 10 are the segment address. ADD-B is the array memory address of a block of 16 PEM words.

Array-to-Disk Transfer: Transfer "n" words (1024 bits) from the designated section of the array starting at array memory address ADD-B to the DFC unit designated by UNT starting at the disk segment address in ADD-A. Field usage is as indicated above for disk-to-array transfers.

Disk-to-BIOM Transfer: Transfer "n" words (256 bits) from disk starting at segment address in ADD-A, via the DFC designated by UNT, to BIOM starting at the address in ADD-B. If it is a priority transfer, bit 1 of VAR is set. The MAP field must be all zeros. The format of ADD-A is as described for disk-to-array transfers. LIM specifies a 256-bit word count.

BIOM-to-Disk Transfer: If VAR bits 2, 3, 4 equal "0, 0, 0", transfer "n" words (256 bits) from BIOM starting at address in ADD-B, via the DFC designated by UNT, to disk starting at segment address in ADD-A. Other conditions are as above for disk-to-BIOM transfers. If VAR bit 2 equals "1", 16 words of 256 bits are transferred, then 48 words of 256 bits are skipped, alternately until the word count "n" is reached, otherwise as above. If VAR bits 2, 3 equal "0, 1", 32 words of 256 bits are transferred, then 32 words of 256 bits are skipped, alternately until the word count "n" is reached. For either of the last two variants, the starting address should be zero, modulo 16 (or 32 for the second variant), if the first group of words transferred is to be the same size as all the others.

BIOM-to-Array Transfer: Transfer "n" words (1024 bits) from BIOM starting at address ADD-A, via the DFC specified by UNT, to the designated section of the array starting at array memory address ADD-B. Other conditions are as indicated for disk-to-array transfers.

Array-to-BIOM Transfer: Transfer "n" words (1024 bits) from the designated section of the array starting at array memory address ADD-B, via the DFC designated by UNT, to BIOM starting at address ADD-A. Other conditions are as indicated for disk-to-array transfers.

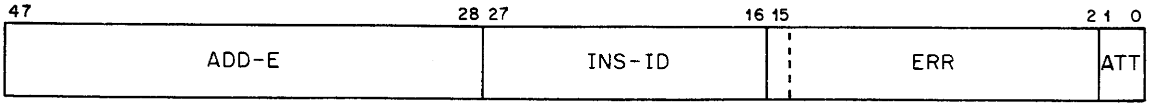
Array-to-Real Time Device: Transfer "n" words (1024 bits) from the designated section of the array starting at array memory address ADD-B, under control of the DFC designated by UNT via IOS, to the real-time device. Other conditions are as indicated for disk-to-array transfers.

Real Time Device-to-Array Transfer: Transfer "n" words (1024 bits) from the real-time device under control of the DFC designated by the UNT via IOS, to the designated section of the array starting at array memory address ADD-B. Other conditions are as indicated for disk-to-array transfers.

Result Descriptor Usage

Result descriptors are comprised of one 48-bit word. Access for the execution of this descriptor is granted upon execution of a "read result descriptor" scan command by the B 6700. The request for the scan command is initiated by the DC which sends an interrupt (to the B6700) which was caused by an "I/O complete", an error, or an interrupt condition. The format for a result descriptor is as follows:

Result Descriptor Format



Field	Function
ADD-E	Address field. Indicates last address used by array, B 6700, or BIOM.
INS-ID	Identity field. Bit 27 identifies type of descriptor. Bits 23-26 contain the four least significant bits of the instruction, which combined with the restrictions on the use of the instructions, unambiguously identifies the instruction being executed. Bits 16-22 provide an identification number for software use. Bits 13 and 14 serve as list ID in some types of descriptor. Bit 7 identifies the type of descriptor.
ERR	Error field. Denotes type of error or interrupt. As noted in the above comment, bits 13 and 14 carry an ID function in some descriptor types. Bit 7 is used for "CU3 not ready" in the DFDC result descriptors.
ATT	This field denotes the type of attention required by the program for the result descriptor.

Specific bit usage is described in the following subparagraphs for the various result descriptors.

CU Attention Result Descriptor: This descriptor is formed whenever a CU which is not being addressed by an active CU instruction, generates an interrupt (see note). It is read in the same manner as a result descriptor. Bit usage is described below:

> Note: In the event that a CU interrupt is generated by a CU which is being addressed by an active instruction, the operation in progress is terminated and a CU Result Descriptor (described next) is returned instead of the CU Attention Result Descriptor.

Bits	Description
0, 1	1 1 = Software attention with or without hardware exception
7	Zero
9	CU #0 requires attention
11	CU #1 requires attention
13	CU #2 requires attention
15	CU #3 requires attention
27-23	Always bit pattern 00111 (octal 07) in INS field to indicate unique code for this descriptor.

CU Result Descriptor: As noted above, this descriptor is generated upon the occurrence of an interrupt in a CU being addressed by an active CU instruction. It is also generated by certain error conditions. Bit usage is as follows:

Bits	Description
0, 1	1 1 = Software attention with or without hardware exception
4	Descriptor error. Indicates that an incorrect instruction code was received by the CU Descriptor Control (CUDC), or that a MAP field is equal to 0
5	Time-out error. Indicates that during operation with CU, a transfer is not completed within 10 msecs.
6	Word bus error. Indicates that CUDC was thwarted in an attempt to use the word bus because of error. The word bus controls will also return a word bus result descriptor in response to the error.

Bits	Description
7	Zero
8	CU #0 not ready
9	CU #0 error
10	CU #1 not ready
11	CU #1 error
12	CU #2 not ready
13	CU #2 error
14	CU #3 not ready
15	CU #3 error
22-16	Identifier bits (ID). All but the LSB of the IDENT field of the original CU I/O descriptor
27-23	Unique code of 01, 02, 03, or 04 (octal)
47-28	Address field (ADD-E)

Word Bus Result Descriptor: The word bus control is used by various control areas to initiate data transfers involving the word bus, as follows: by the scan bus controls to fetch I/O descriptors; by the list controls to fetch I/O descriptors; by the CU descriptor controls (CUDC) to execute read/write operations between the array (CUs) and the B 6700. The word bus result descriptor is generated by the word bus control. Bit usage is as follows:

Bits	Description
0,1	1 1 = Software attention with or without hardware exception
4	Descriptor error. This bit is set under any of the following conditions: a. I/O descriptor fetched has an invalid instruction code; b. List head descriptor specifies a list already in progress;

4 (Cont'd)

c. List tail descriptor specifies a list that does not have its corresponding list head flip-flop set;

d. List tail descriptor specifies a list that already has its list tail flip-flop set;

e. List head or list tail descriptor does not specify a list;

f. Linked-list tail descriptor does not specify a list or the list specified does not have its corresponding list head flip-flop set;

g. CUDC already has an active CU descriptor when new CU descriptor is received for processing by CUDC

5 Read address parity error. Indicates that the B6700 memory detected a parity error for the read address information transmitted

6 Read data parity error. Indicates that the data read from B6700 memory was received with a parity error

or

6,5 Write address/data parity error. Set to 1,1 to indicate that the B6700 memory detected a parity error for the address or data transmitted to it for a write operation.

7 Zero

8 No access to memory. Indicates that the word bus requested memory access and the memory ready signal was not returned for a period of 8 clocks.

22-16 Identifier bits. All but the LSB of the IDENT field of the original I/O descriptor.

27-23 All zeros in INS field; unique code for this descriptor.

47-28 Address field (ADD-E)

Queuer Result Descriptor: This is an I/O result descriptor that is generated by the Queuer controls of the DC. Bit usage is as follows:

Bits	Description
0, 1	1 1 = Software attention with or without hardware exception
2	Indicates a Queuer busy condition, as follows: a. BIOM-Array operation specified but BIOM-Array descriptor already stored; b. Priority for DFCn specified but DFCn already has priority descriptor stored; c. Queuer is full (non-list descriptors)
3	Unit not ready
4	Descriptor error. Indicates the existence of one of the following error conditions: a. Illegal unit specified in descriptor; b. Descriptor specifies UNT field equal to 0 when INS field specifies a disk operation; c. Invalid INS code specified; d. List specified but list tail is already set for that list
5	LSB of DFC specified
6	MSB of DFC specified
7	One
8	Queuer full
12-9	Identification of storage unit specified
14, 13	List ID
15	Priority bit
22-16	Identifier field. All but the LSB of the original field.
26-23	Instruction field, last four bits of "transfer" type instruction
27	Zero

Bits	Description
47-28	Address field; array or BIOM memory address if list ID is zero. If list ID is not zero, contents of list address register.

DFDC Result Descriptor: This is an I/O result descriptor that is generated by the DFC controls of the DC. If bits 2-14 are all zero, successful completion of a non-list descriptor is indicated. Bit usage is as follows:

Bits	Description
0, 1	1 1 = Software attention with or without hardware exception
2	CU #0 not ready
3	CU #2 not ready
4	List complete. When bit 4 is set, the remainder of the descriptor pertains to the last descriptor executed in a given list.
5	Invalid MAP/VAR specified
6	Disk read parity error
7	CU #3 not ready
8	Disk missed access
9	EU manual write lockout
10	CU #1 not ready
11	SU not ready
12	Disk time-out
14, 13	List ID
15	Track counter incrementing indicator
22-16	Identifier field. All but the LSB of the original field.
26-23	Instruction field; last four bits of "transfer" type instruction
27	One
47-28	Last memory address used by array or BIOM.

DISK FILE CONTROL (DFC)

The I/O Disk File Control (DFC) contains two identical controls (Figure 2-5) which communicate with the two Model II Electronic Units (EU), the I/O Switch (IOS), the Buffer I/O Memory (BIOM), and the Descriptor Controller (DC). The DFC controls the flow of data to and from an EU under the direction of an I/O descriptor executed by the DC. Data flow between the B6700 system and the ILLIAC disk is via the BIOM, through the designated DFC and EU, to a Storage Unit (SU). Data flow between array memory and the ILLIAC disk is via the IOS, from its associated IOR, through the designated DFC, through an EU, to an SU. The designate lines (SU, track, and read/write) go directly from the DC to the EU.

The smallest addressable area of the disk is a segment which contains 16,384 bits of data and 128 bits of parity. The total segment size of 16,512 bits is contained in 43 disk words of 384 bits each. The data size of 16,384 bits is equal to 256 ILLIAC words of 64 bits each. There are 1200 segments per revolution

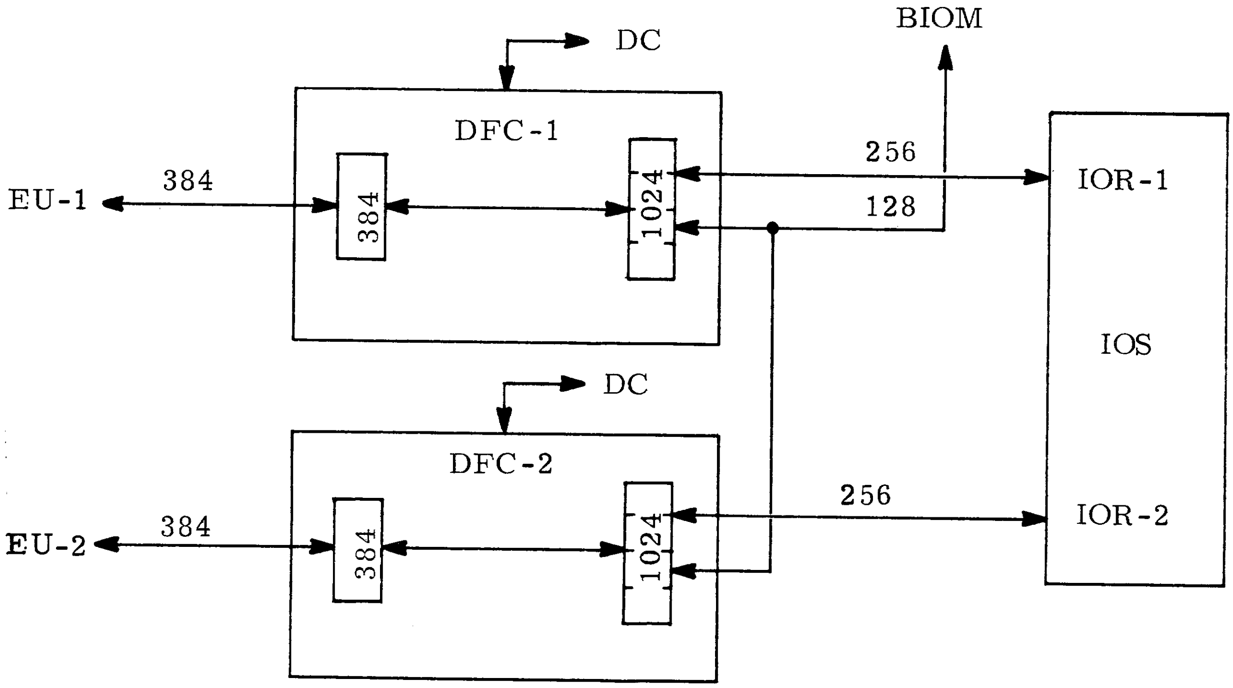


Figure 2-5. Disk File Control

and 4800 segments per disk (SU). Data transfer may start at any designated segment but cannot cross SU boundaries.

A DFC has a 384-bit word register to interface with the EU, and a 1024-bit register to interface with IOR in the IOS or BIOM. These registers plus the 1024-bit IOR are required to match the combined data rates of the two disks (2 × 384 bits every 680 nsec) to the worst-case delay for array memory cycles (1024 bits every 900 nsec). The 384-bit word register is also used for receiving 16 sets of addresses during the address mode. The address mode occurs whenever the DFC is not busy executing an I/O descriptor for read or write. Each SU provides a set of 11 address bits for every segment. These bits are continually compared with the queuer I/O descriptors in the DC. When a match is found, that I/O descriptor is the next executed by the DFC.

BUFFER I/O MEMORY

The BIOM is treated at the B6700 system as though it were a B6700 memory module having four ports, one each for two B6700 processors and for the two B6700 multiplexers, as shown in Figure 2-6. The advantage of considering the BIOM a B6700 memory module is that the control program can transfer data between B6700 memory (disk or tape) and the BIOM via the multiplexer without requiring the B6700 processor to cycle its main memory. The B6700 side of the BIOM is independent of the ILLIAC IV I/O subsystem, with the control program handling memory protection and resolving conflicts in memory access. The I/O descriptor for any multiplexer operation must be in BIOM at the head of its assigned area.

The BIOM contains four PEM modules, each providing storage for 2048 words, 64 bits in length. It has two functional interfaces, one with the ILLIAC IV I/O subsystem , the other with the B6700 system. These are shown in the simplified block diagram of the BIOM, Figure 2-6.

The interface with the ILLIAC IV I/O subsystem is used for transferring data to or from the ILLIAC IV disk system or the ILLIAC IV array. All data transfers on this interface are for a 256-bit word per memory cycle. The four PEMs are used in parallel to store 2048 words of 256 bits each; that is, each PEM stores 64 bits of the data word. Addressing of the BIOM is controlled by the designated DFC descriptor control in the DC. Eleven address bits specify one of 2048 locations in a PEM such that the same address refers to the same relative location in each of the four PEMs.

The interface with the B6700 system is used for transferring data to or from the B6700. On this interface, data may be transferred as either a 32-bit or a 48-bit word. The BIOM module is designated by address bits A19 through A15 inclusive equal to "1" which assigns the BIOM as the two top memory modules on the B6700 system. Address bit A14, the least significant module address bit, is the mode bit; A14 = 0 for 32-bit mode transfers and A14 = 1 for 48-bit mode transfers. Address bits A13 through A0 are the BIOM internal addresses.

In the 32-bit mode, two 32-bit words are stored for each 64-bit PEM word. Address bits A2 and A1 in combination designate one of the four PEMs. Address bit A0 designates the half of the PEM word to be used. The formats are illustrated in Figure 2-7.

In the 48-bit mode, sixteen 48-bit words are stored for every three 256-bit words (four PEMs), as shown in Figure 2-8. The four least significant address bits, A3 through A0, designate and control the 16 different positions within three 256-bit words (four PEMs). The octal addresses for a 16-word subset are shown in Figure 2-8. Address bits A13 through A4 define 1024 subsets, of which 682 are usable. In both Figures 2-7 and 2-8 the least significant end of the word is on the right side of the figure; bit 0 for a B6700 and bit 63 for a PE word.

The memory address as received from either the processor or the multiplexer is interpreted in the BIOM logic so that it may be used directly in addressing a PEM. The BIOM-Internal addresses of the three words (of four PEMs) in a subset N are (see Figure 2-9):

$$3N + W0; \quad 3N + W1; \quad 3N + W2$$

where

N = bits A13 through A4

W0, W1, and W2 are derived from A3 through A0.

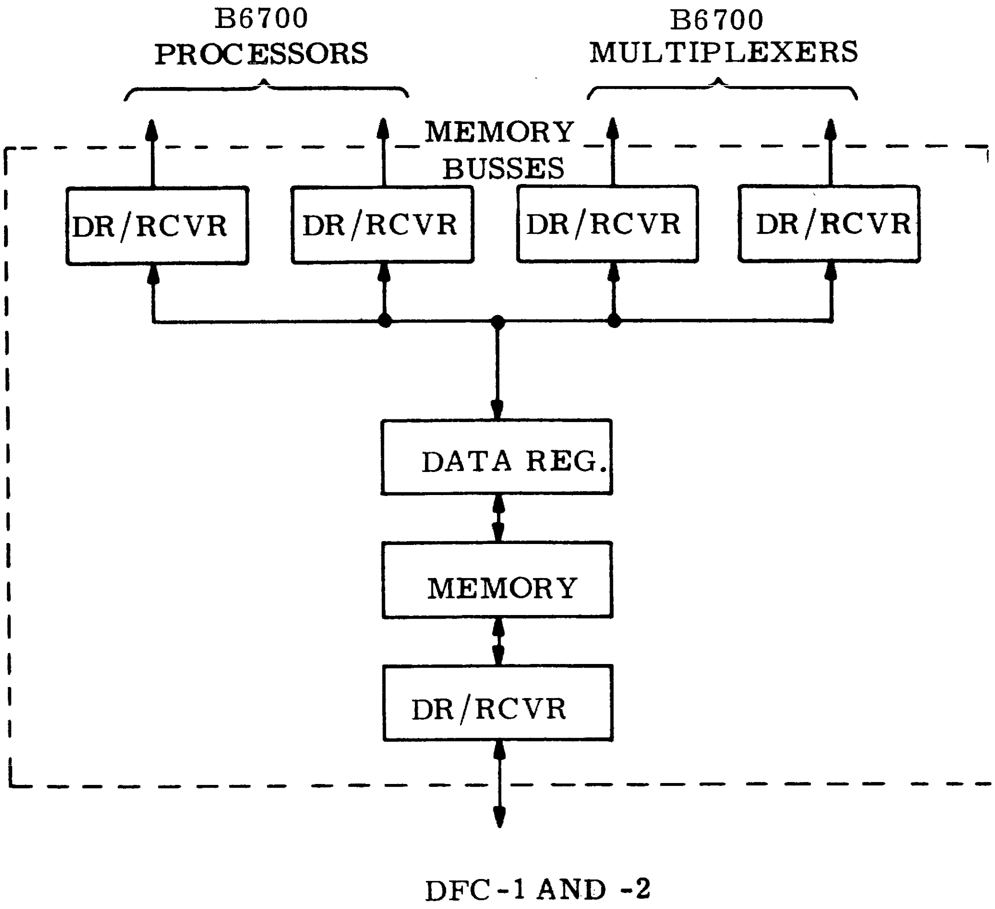


Figure 2-6. Buffer I/O Memory

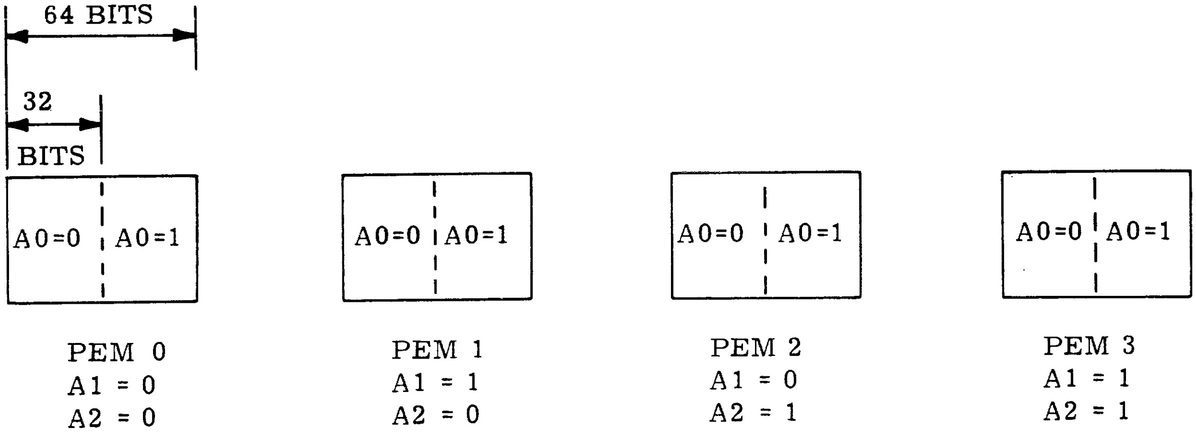


Figure 2-7. PEM 32-Bit Mode Format

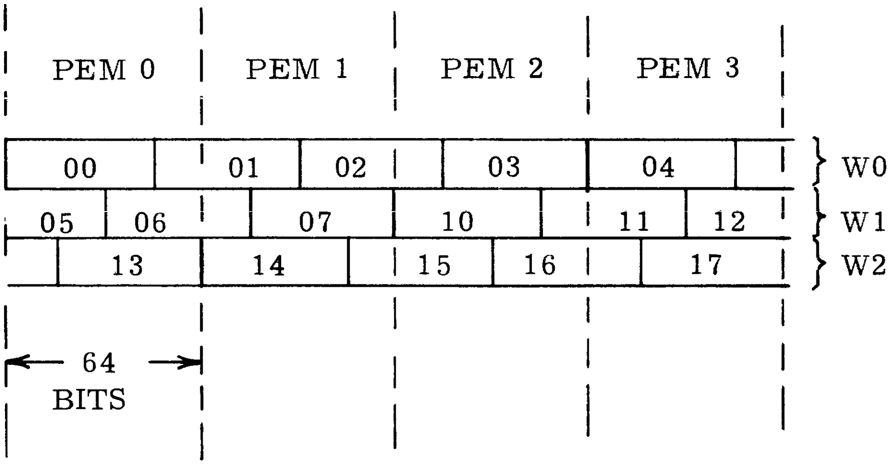


Figure 2-8. PEM 48-Bit Mode Format

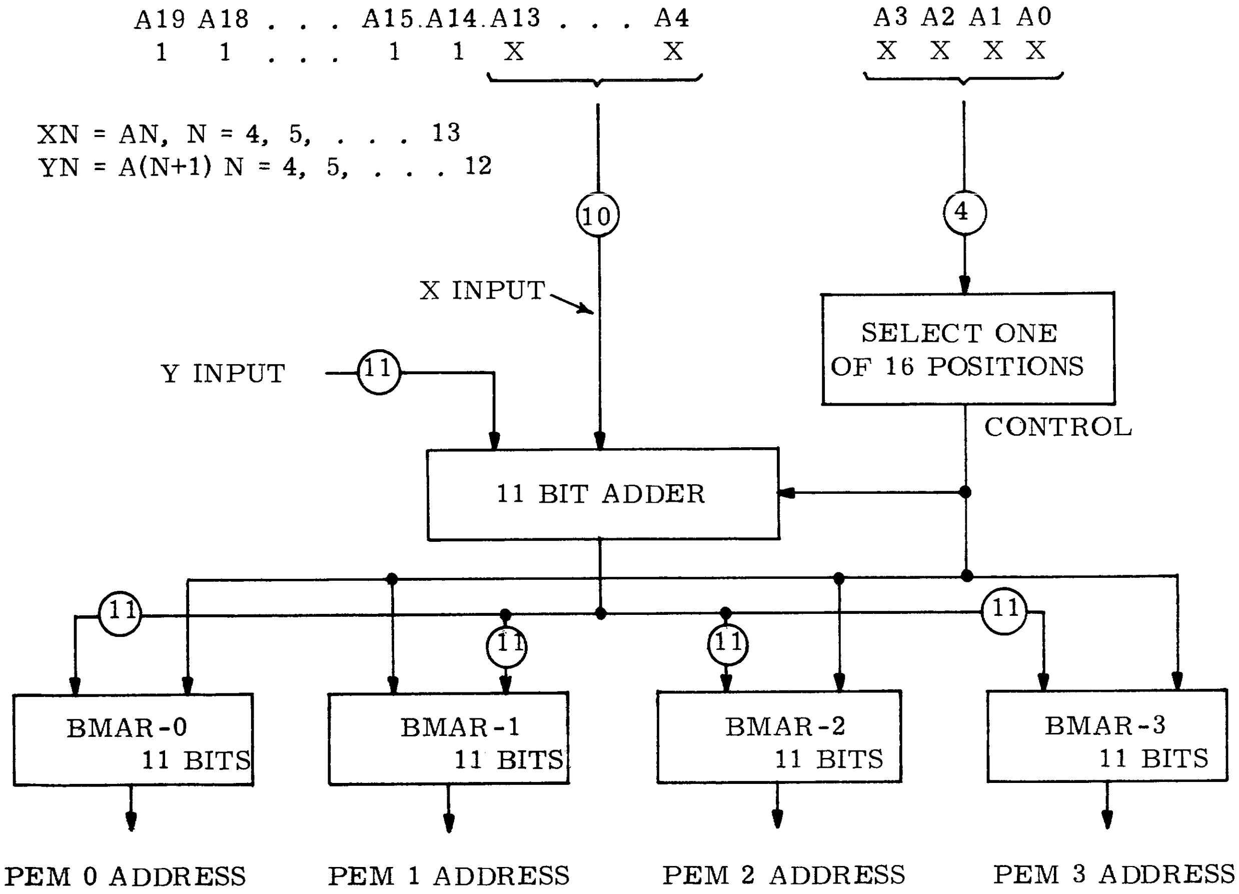


Figure 2-9. 48-Bit Mode PEM Address Modification

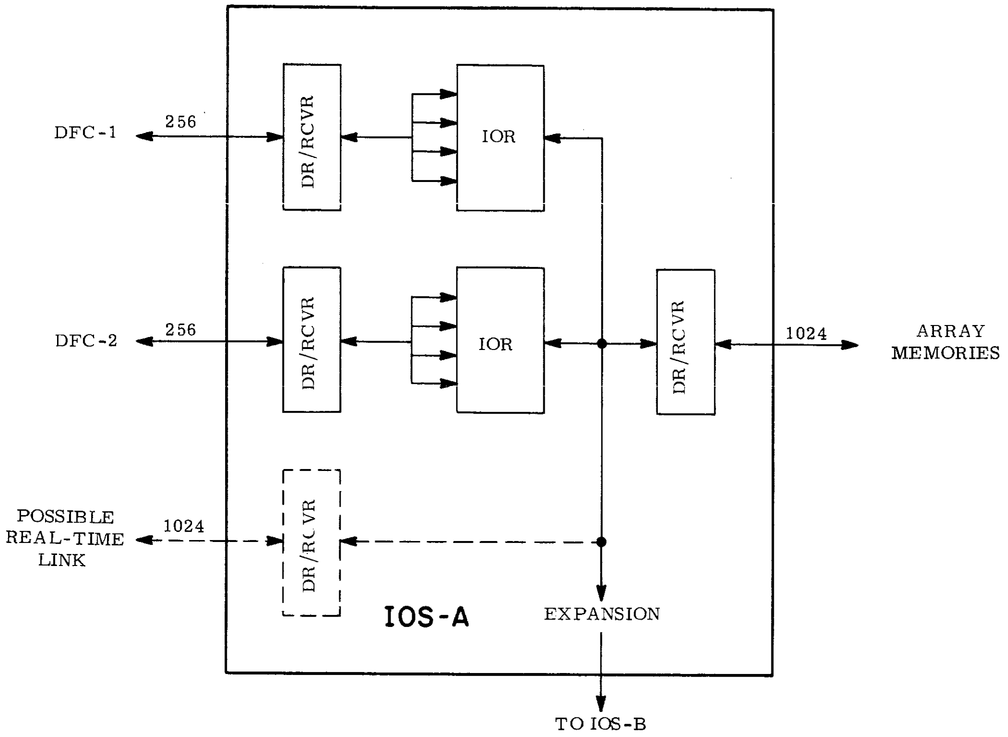


Figure 2-10. I/O Switch Configuration for 1024-Bit Transfer Capability

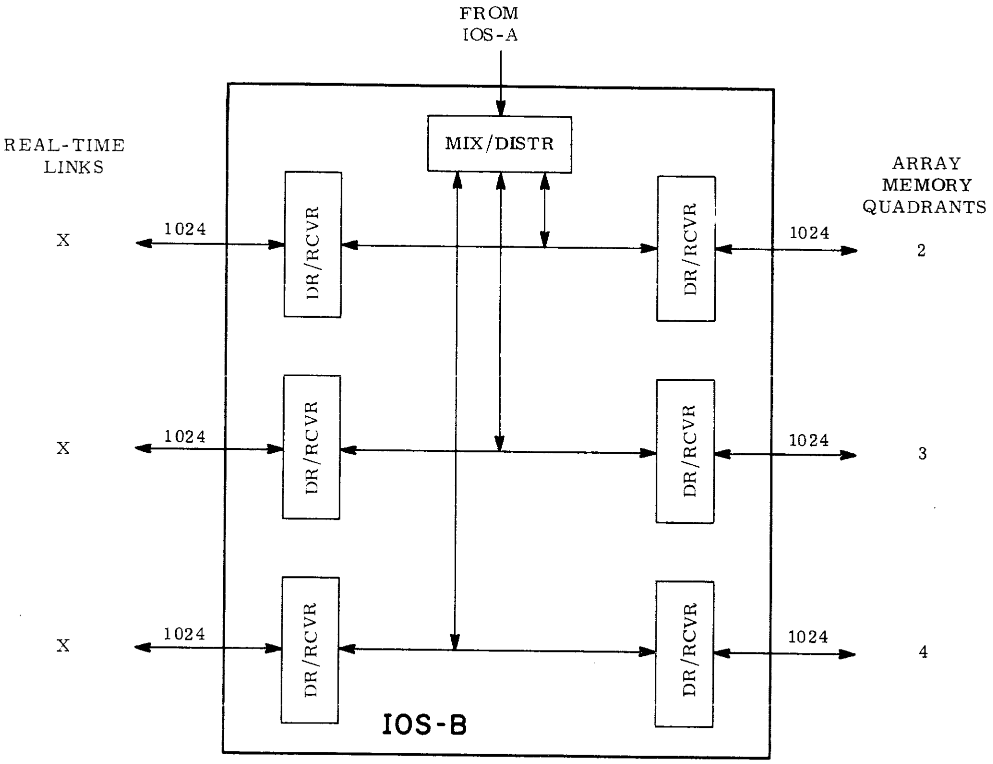


Figure 2-11. Possible Expansion Elements to Basic IOS Configuration for 4096-Bit Transfer Capability

Addresses from the four ports and DC are mixed and priority resolved for allowing access to BIOM. The order of priority, from highest to lowest is: DC, Port 1, Port 2, Port 3, Port 4. Note that if a BIOM-to-disk operation is in process, no other unit (the four ports) is allowed access until the operation is complete. The path between BIOM and disk is critical in timing to allow the BIOM to keep pace with the disk.

I/O SWITCH

The I/O switch (Figure 2-10) isa unit which is used for data buffering and distribution, and which provides expansion capability for the real-time link. For buffering purposes a 1024-bit I/O register (IOR) is provided for each disk file control. This buffering allows the transfer of 1024 bits to array memory every microsecond, alternating between the DFCs. The transfer channel between each DFC and the IOR is a 256-line bidirectional cable. The transfer channel between either IOR and the array memory is a 1024-line bidirectional cable. When the fourth or last group of 256 bits is being transferred between an IOR and its DFC, a PEM memory cycle is requested by the descriptor control of DC to effect the transfer. (The I/O registers are not used with the real-time link.)

A second function of the IOS is to distribute data between the three I/O ports and the array memory. The three ports are the two DFCs and the real-time link. The distribution method depends on whether the IOS is a 1024-bit unit in the IOS-A configuration (Figure 2-10) or an expanded version (IOS-A, IOS-B) capable of handling 4096 bits (Figure 2-11). For a 1024-bit IOS, the two IORs are switched alternately to the single 1024-line cable to array memory. The 1024 bits are sufficient to accommodate 16 PEMs of 64 bits each per IOS transfer. Thus, each line need be routed to only 16 different PEMs for the entire array of 256 PEMs. The real-time link requires one of the disk file controllers for control purposes although it has its own 1024-bit input to the IOS. For a 4096-bit IOS, the IORs are distributed to one of four 1024-bit cables to the array memory so that each 1024-bit cable group connects to a separate quadrant of 64 PEMs. In this configuration, the real-time link has an input/output of 4096 lines and uses all four cable groups to the array.

The initial I/O switch will be in the IOS-A configuration, that is, with a bandwidth of 1024 lines to the array memory. This is sufficient for the two disk files and most real-time links. However, the IOS is organized so that it can be expanded to include the IOS-B configuration to provide an additional 3072 data lines for the real-time link and array memory. This, of course, would require that the cabling between the IOS and the quadrants be slightly modified so that the first cable of 1024 lines goes only to quadrant 1; the other three sets of cables would go to quadrants 2, 3, and 4.

CONTENTS

(See Index on Reverse Side)

ADVAST INSTRUCTION INDEX

Mnemonic Code	Octal Code	Ref. Page	Mnemonic Code	Octal Code	Ref. Page	Mnemonic Code	Octal Code	Ref. Page
ALIT	16XX	3-12	INCRXC	0002	3-41	TXEF	1413	3-70
BIN	0610	3-13	INR	0007	3-42	TXEFA	1412	3-70
BINX	0611	3-13	JUMP	17XX	3-43	TXEFAM	1216	3-71
CACRB	0001	3-15	LDC	0011	3-44	TXEFM	1217	3-71
CADD	0402	3-17	LDL	0405	3-45	TXET	1411	3-70
CAND	0410	3-18	LEADO	0201	3-46	TXETA	1410	3-70
CCB	1101	3-19	LEADZ	0200	3-46	TXETAM	1214	3-71
CEXOR	0407	3-20	LESSF	1507	3-66	TXETM	1215	3-71
CLC	0005	3-21	LESSFA	1506	3-66	TXGF	1403	3-72
COMPC	0006	3-22	LESST	1505	3-66	TXGFA	1402	3-72
COPY	0204	3-23	LESSTA	1504	3-66	TXGFAM	1302	3-73
COR	0411	3-24	LIT	0003	3-48	TXGFM	1303	3-73
CRB	0207	3-25	LOAD	0600	3-49	TXGT	1401	3-72
CROTL	0015	3-26	LOADX	0601	3-49	TXGTA	1400	3-72
CROTR	0017	3-27	ONESF	1007	3-67	TXGTAM	1300	3-73
CSB	0013	3-28	ONESFA	1006	3-67	TXGTM	1301	3-73
CSHL	0014	3-29	ONEST	1005	3-67	TXLF	1407	3-74
CSHR	0016	3-30	ONESTA	1004	3-67	TXLFA	1406	3-74
CSUB	0403	3-31	ONEXF	1017	3-68	TXLFAM	1306	3-75
CTSBF	1102	3-32	ONEXFA	1016	3-68	TXLFM	1307	3-75
CTSBT	1100	3-32	ONEXT	1015	3-68	TXLT	1405	3-74
DUPI	0401	3-34	ONEXTA	1014	3-68	TXLTA	1404	3-74
DUPO	0400	3-35	ORAC	0205	3-52	TXLTAM	1304	3-75
EQLXF	1417	3-64	SETC	0012	3-53	TXLTM	1305	3-75
EQLXFA	1416	3-64	SKIP	1103	3-54	WAIT	0206	3-78
EQLXT	1415	3-64	SKIPF	1107	3-69	ZERF	1003	3-76
EQLXTA	1414	3-64	SKIPFA	1106	3-69	ZERFA	1002	3-76
EXCHL	0406	3-36	SKIPT	1105	3-69	ZERT	1001	3-76
EXEC	0004	3-38	SKIPTA	1104	3-69	ZERTA	1000	3-76
FINQ	0010	3-39	SLIT	16XX	3-55	ZERXF	1013	3-77
GRTRF	1503	3-65	STL	0404	3-56	ZERXFA	1012	3-77
GRTRFA	1502	3-65	STORE	0602	3-58	ZERXT	1011	3-77
GRTRT	1501	3-65	STOREX	0603	3-58	ZERXTA	1010	3-77
GRTRTA	1500	3-65	TCCW	0203	3-60			
HALT	0000	3-40	TCW	0202	3-61			

TMU INSTRUCTION INDEX

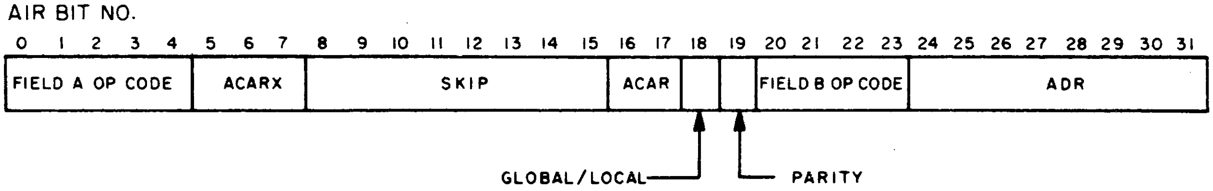
Mnemonic Code	Octal Code	Ref. Page	Mnemonic Code	Octal Code	Ref. Page	Mnemonic Code	Octal Code	Ref. Page
EFA	160	5-16	SAT	047	5-25	SR	005	5-24
EFF	164	5-18	SIS	120	5-26	SRT	045	5-25
LICR	041	5-20	SIV	100	5-27	TIC	121	5-33
LISR	040	5-21	SL	006	5-24	TOC	002	5-34
RPT	001	5-22	SLT	046	5-25	WIS	044	5-35
RUN	020	5-23	SOC	011	5-30			
SA	007	5-24	SOD	010	5-32			

SECTION III

ADVAST INSTRUCTIONS

INSTRUCTION FORMAT AND FIELD USAGE

The format for ADVAST instruction words is given below, followed by an explanation of field usage. Note that all bit positions are stated relative to their location in the ADVAST instruction register (AIR).



Field — Description

FIELD A OP CODE — AIR BITS 0:5. Bit 0 is "zero" for ADVAST instructions. Refer to Table 3-1 for the ADVAST Op Codes.

ACARX — AIR BITS 5:3. When bit 5 is "one", the contents of the ACAR specified by bits 6 and 7 are used to index the quantity found in the ADR field. When bit 5 is "zero", the ADR field is used without indexing, and the values in bits 6:2 are irrelevant (except in the SLIT/ALIT instruction, where 6:2 specify the recipient ACAR).

SKIP — AIR BITS 8:8. This field is used in the test and skip instructions to show sign and magnitude of the skip distance, if a skip is to be executed. Bit 8 is the sign ("one" means subtract; "zero" means add), while bits 9:7 specify the magnitude.

Table 3-1. ADVAST Instruction Op Codes

FIELD B (AIR BITS 20:4)

FIELD A (AIR BITS 0:5)

	00	01	02	03	04	05	06	07	10	11	12	13	14	15	16	17
00	HALT	CACRB	INCRXC	LIT	EXEC	CLC	COMPC	INR	FINQ	LDC	SETC	CSB	CSHL	CROTL	CSHR	CROTR
01																
02	LEADZ	LEADO	TCW	TCCW	COPY	ORAC	WAIT	CRB								
03																
04	DUPO	DUPI	CADD	CSUB	STL	LDL	EXCHL	CEXOR	CAND	COR						
05																
06	LOAD	LOADX	STORE	STOREX					BIN	BINX						
07																
10	ZER TA	ZER T	ZER FA	ZER F	ONES TA	ONES T	ONES FA	ONES F	ZERX TA	ZERX T	ZERX FA	ZERX F	ONEX TA	ONEX T	ONEX FA	ONEX F
11	CTSBT	CCB	CTSBF	SKIP	SKIPTA	SKIPT	SKIPFA	SKIPF								
12													TXE TAM	TXE TM	TXE FAM	TXE FM
13	TXG TAM	TXG TM	TXG FAM	TXG FM	TXL TAM	TXL TM	TXL FAM	TXL FM								
14	TXG TA	TXG T	TXG FA	TXG F	TXL TA	TXL T	TXL FA	TXL F	TXE TA	TXE T	TXE FA	TXE F	EQLX TA	EQLX T	EQLX FA	EQLX F
15	GRTR TA	GRTR T	GRTR FA	GRTR F	LESS TA	LESS T	LESS FA	LESS F								
16	SLIT/ ALIT															
17	JUMP															

Field — Description

ACAR — AIR BITS 16:2. Each instruction describes the particular usage of the ACAR specified in this field. Usually, the designated ACAR is the source of the first operand and/or the destination of the result.

GLOBAL/LOCAL — AIR BIT 18:1. A "zero" indicates "global"; "one" indicates "local." Global means that the execution of the instruction is dependent upon the array configuration control logic; in a multiquadrant array, it is assumed that all CUs are executing the same program. Local means that this CU executes the instruction independently, without synchronizing or interchanging data with other CUs.

PARITY — AIR BIT 19:1. This is an odd parity bit. ALIT, SLIT, JUMP, and instructions executed by means of the EXEC instruction do not utilize the parity bit.

FIELD B OP CODE — AIR BITS 20:4. Refer to Table 3-1 for the ADVAST Op Codes.

ADR — AIR BITS 24:8. Each instruction describes the particular usage of this field. Generally, it is indexable (see ACARX), and specifies the local memory address to be used as the source of the second operand, or the source or destination of a data transfer. It indicates the shift amount in the shift instructions. For some instructions (e.g., CCB, CRB, CSB, etc.) bits 24:2 designate the number of the CU to perform the operation (as interpreted by the array size and configuration control logic shown in the CU Determination Chart, Table 3-2). For the ALIT, SLIT, and JUMP instructions, AIR bits 8:24 comprise the ADR field.

For instructions of the type noted in the ADR field description, Table 3-2 may be used to determine which CU will be used in instruction execution. The array defined by MC0 and MC2 should be the same in all CUs executing an instruction of this type. There is no provision in the hardware to enforce

Table 3-2. CU Determination Chart

				AIR 18:1				
				= 1 (LOCAL)	= 0 (GLOBAL)			
CU Number*				All Values of AIR 24:2	AIR 24:2 (Relative CU Number)			
0	1	2	3		00	01	10	11
1	0	0	0	ALL CUs DEFINED BY MC0	0	0	0	0
0	1	0	0		1	1	1	1
0	0	1	0		2	2	2	2
0	0	0	1		3	3	3	3
1	1	0	0		0	1	0	1
1	0	1	0		0	2	0	2
1	0	0	1		0	3	0	3
0	1	1	0		1	2	1	2
0	1	0	1		1	3	1	3
0	0	1	1		2	3	2	3
1	1	1	1		0	1	2	3

*Number of CU(s) executing instruction as determined by MC0, MC2, and the configuration control logic.

this conformity, and it is conceivable that there may be a reason for having them not the same in the various CUs. Therefore, it is necessary to examine the settings of these registers for each CU to determine if any and which CUs will perform the function.

ILLIAC IV ADDRESSING

Addressing is primarily a function of the Memory Service Unit (MSU). The three users of MSU are I/O and the Final Station (FINST) and Instruction Look-Ahead (ILA) portions of the CU. ADVAST requests its memory cycles via FINST.

The I/O has two requests, IOA and IOB, the difference between them being only in priority. FINST generates its request for three distinct purposes:

1. FINST Own Request – PE read/write request.
2. FINST Request A – ADVAST LOAD(X), BIN(X), or STORE(X).
3. FINST Request B – Transfer from PE register to an ACAR in ADVAST.

The priorities for these requests in order of highest to lowest is the following:

1. IOA
2. FINST Request
3. ILA
4. IOB

The addresses sent to the PEs may be indexed by either the X or S registers in the PE before loading into the Memory Address Register (MAR). The register selection is determined by FINST.

The addresses (instructions and data) in a four-quadrant array (as defined by MC1 for instruction addresses and MC2 for data addresses) are interpreted as follows:

IIA|ICR	0 — 4	5 — 15	16 — 17	18 — 20	21 — 23	24
	For future expansion	PEM Subaddress (Bits 5-15)	Selects the CU No.	Selects the PEM within the PUC	Selects the PUC	Half-word designator in ICR
ACAR	40 — 55		56 — 57	58 — 60	61 — 63	0

EXAMPLE

For the addresses specified (bits 5-23) the ILLIAC components will be as follows:

Address (Bits 5-23)	Selected Components			
	PUC	PEM	CU	Subaddress
0000000	0	0	0	0000
0000110	0	1	1	0000
0000777	7	7	3	0001
1777777	7	7	3	3777

In a two-quadrant array as defined by MC1 for instruction addresses and MC2 for data addresses, the address is interpreted as follows:

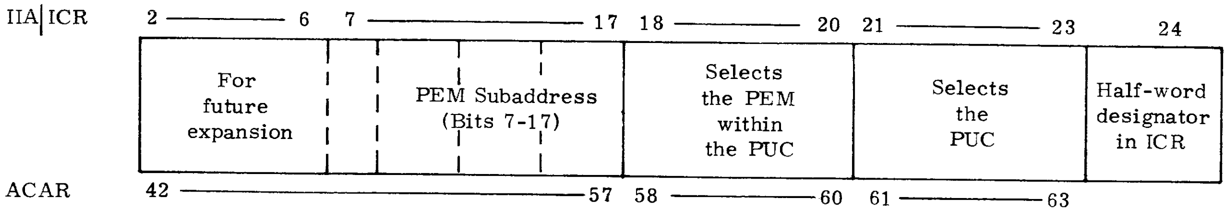
IIA|ICR	1 —— 5	6 —— 16	17	18 —— 20	21 —— 23	24
	For future expansion	PEM Subaddress (Bits 6-16)	Selects higher or lower CU No.	Selects the PEM within the PUC	Selects the PUC	Half-word designator in ICR
ACAR	41 —— 56		57	58 —— 60	61 —— 63	

EXAMPLE

For the addresses specified (bits 6-23) the ILLIAC components will be as follows:

Address (Bits 6-23)	Selected Components			
	PUC	PEM	CU	Subaddress
000000	0	0	L	0000
000001	1	0	L	0000
000010	0	1	L	0000
000077	7	7	L	0000
000177	7	7	H	0000
000277	7	7	L	0001
000377	7	7	H	0001
777777	7	7	H	3777

In a one-quadrant array as defined by MC1 and MC2, the addresses are interpreted as follows:


IIA|ICR
2
6
7
17
18
20
21
23
24
For future expansion
PEM Subaddress (Bits 7-17)
Selects the PEM within the PUC
Selects the PUC
Half-word designator in ICR
ACAR
42
57
58
60
61
63

EXAMPLE

For the addresses specified (bits 7-23) the ILLIAC components will be as follows:

Address (Bits 7-23)	Selected Components		
	PUC	PEM	Subaddress
000000	0	0	0000
000001	1	0	0000
000010	0	1	0000
000077	7	7	0000
000100	0	0	0001
377777	7	7	3777

Various memories will be selected in a quadrant by the memory select lines from MSU. The I/O request will select eight PUCs and two PEMs within each PUC (that is, IIA|ICR bits 20 - 23 are ignored). The BIN(X) and ILA requests will select eight PUCs and one PEM within each PUC (that is, IIA|ICR bits 21 - 23 are ignored). LOAD(X) and STORE(X) will select one PUC and one PEM within the PUC. PE requests will select eight PUCs and eight PEMs within each PUC (that is, IIA|ICR bits 18-23 are ignored).

The half-word designation is the least significant address bit for the 32-bit instructions. It is ONE to designate the right or less significant half-word, and ZERO for the left or more significant.

ADVAST INSTRUCTION REPERTOIRE

Following is a list of the instructions that comprise the ADVAST instruction repertoire. They are arranged in alphabetical order according to mnemonic or functional group, and in the same order of appearance as the instruction descriptions which comprise the remainder of this subsection. Timing for the instructions is given in Section VI.

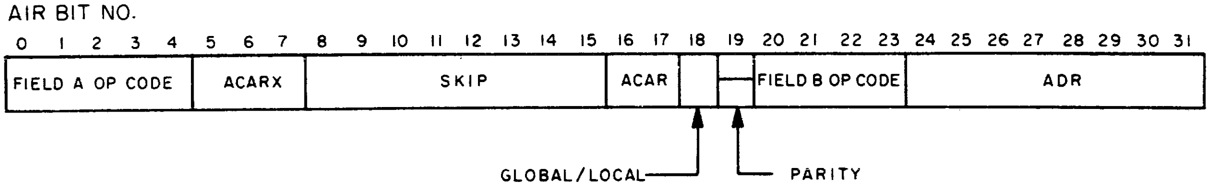
	Octal Op Code		
Mnemonic Code	Field A 0:5	Field B 20:4	Operation
ALIT	16	XX	Add literal to address field of ACAR
BIN	06	10	Block fetch from PE memory to ADB
BINX	06	11	Block fetch (RGX-indexed) from PE memory to ADB
CACRB	00	01	Set/Reset nth bit in ADVAST control register
CADD	04	02	Add local memory to ACAR
CAND	04	10	Logical AND of local memory and ACAR
CCB	11	01	Complement nth bit of ACAR
CEXOR	04	07	Logical exclusive-OR of local memory and ACAR
CLC	00	05	Clear ACAR
COMPC	00	06	Complement ACAR
COPY	02	04	Copy ACAR
COR	04	11	Logical OR of local memory and ACAR
CRB	02	07	Reset nth bit in ACAR
CROTL	00	15	Rotate ACAR left (end around)
CROTR	00	17	Rotate ACAR right (end around)
CSB	00	13	Set nth bit in ACAR
CSHL	00	14	Shift ACAR left (end off)
CSHR	00	16	Shift ACAR right (end off)
CSUB	04	03	Subtract local memory from ACAR
CTSBF	11	02	Skip if nth bit in ACAR is not "one"
CTSBT	11	00	Skip if nth bit in ACAR is "one"
DUPI	04	01	Duplicate inner-half of ADB memory word
DUPO	04	00	Duplicate outer-half of ADB memory word
EXCHL	04	06	Exchange local operand and ACAR
EXEC	00	04	Execute
FINQ	00	10	Stop ADVAST until FINST is idle
HALT	00	00	CU comes to orderly idle state

	Octal Op Code		
Mnemonic	Field A	Field B	
Code	0:5	20:4	Operation
INCRXC	00	02	Modify index field of ACAR by increment field of same ACAR
INR	00	07	Return to normal processing after interrupt
JUMP	17	XX	Jump to address in ADR field
LDC	00	11	Transfer specified PE register to ACAR
LDL	04	05	Load from local address
LEADO	02	01	Find leading "one" in ACAR
LEADZ	02	00	Find leading "zero" in ACAR
LIT	00	03	Store next 64 bits in ACAR
LOAD	06	00	Word fetch from PE memory to CU local memory
LOADX	06	01	Word fetch (RGX-indexed) from PE memory to CU local memory
ORAC	02	05	Inclusive-OR of operand in ACAR of all CUs executing the instruction
SETC	00	12	Specified mode bit from PEs to ACAR
SKIP	11	03	Skip forward/backward
SLIT	16	XX	Replace address field of ACAR
STL	04	04	Store ACAR in local address
STORE	06	02	Store from local address into specified PE location
STOREX	06	03	Store from local address into specified PE location (RGX-indexed)
TCCW	02	03	Transmit ACAR counterclockwise (to next lower numbered CU)
TCW	02	02	Transmit ACAR clockwise (to next higher numbered (CU)
Test-Skip T/F A			
(True/False All/Any)			
EQLXTA	14	14	Skip if ACAR 40:24 equal operand 40:24
T	14	15	
FA	14	16	
F	14	17	
GRTRTA	15	00	Skip if ACAR 40:24 are greater than operand 40:24
T	15	01	
FA	15	02	
F	15	03	
LESSTA	15	04	Skip if ACAR 40:24 are less than operand 40:24
T	15	05	
FA	15	06	
F	15	07	
ONESTA	10	04	Skip if ACAR 0:64 are all "ones"
T	10	05	
FA	10	06	
F	10	07	

	Octal Op Code		
Mnemonic	Field A	Field B	
Code	0:5	20:4	Operation
ONEXTA	10	14	Skip if ACAR 40:24 are all "ones"
T	10	15	
FA	10	16	
F	10	17	
SKIPTA	11	04	Skip dependent upon CU true/false flip-flop
T	11	05	
FA	11	06	
F	11	07	
TXETA	14	10	Skip if ACAR 40:24 equal bits 16:24 in local memory
T	14	11	
FA	14	12	
F	14	13	
TXETAM	12	14	Skip if ACAR 40:24 equal bits 16:24 (also, 40:24 are modified by 1:15) of same ACAR
TM	12	15	
FAM	12	16	
FM	12	17	
TXGTA	14	00	Skip if ACAR 40:24 are greater than bits 16:24 in local memory
T	14	01	
FA	14	02	
F	14	03	
TXGTAM	13	00	Skip if ACAR 40:24 are greater than bits 16:24 (also, 40:24 are modified by 1:15) of same ACAR
TM	13	01	
FAM	13	02	
FM	13	03	
TXLTA	14	04	Skip if ACAR 40:24 are less than bits 16:24 in the local memory
T	14	05	
FA	14	06	
F	14	07	
TXLTAM	13	04	Skip if ACAR 40:24 are less than bits 16:24 (also, 40:24 are modified by 1:15) of same ACAR
TM	13	05	
FAM	13	06	
FM	13	07	
ZERTA	10	00	Skip if ACAR 0:64 are all "zeros"
T	10	01	
FA	10	02	
F	10	03	
ZERXTA	10	10	Skip if ACAR 40:24 are all "zeros"
T	10	11	
FA	10	12	
F	10	13	
WAIT	02	06	Synchronize all CUs in array or join all CUs specified by ADR 4:4

ADVAST INSTRUCTION DESCRIPTIONS

The remainder of this section consists of descriptions of the various ADVAST instructions. These are arranged alphabetically according to instruction mnemonic, in the same order as presented in the instruction repertoire previously listed. Each description includes the mnemonic code, the operation performed, the AIR contents (the ADVAST instruction register) for the specific instruction, and a brief functional description and flow chart of major operations performed during instruction execution. The word format for ADVAST instructions is as follows:



The general format used in the instruction descriptions is as shown below. Shaded fields are used to indicate irrelevant fields for specific instructions.

XX	ACARX	SKIP	ACAR	G/L	—	XX	ADR

For all instructions that load the ICR, the stepping of the ICR upon completion of the instruction is inhibited. Certain instructions (e.g., INCRXC, TXE, TXG, and TXL) treat an ACAR as an index register, utilizing the ACR bits as follows:

Half-Word Ind.	Sign of Increment	Magnitude of Increment	Limit	Current Index Value
0:1	1:1	2:14	16:24	40:24

Two abbreviations used in the flow charts are "ILA NI" for "fetch the next instruction in sequence using the ILA" and "JUMP", for "fetch the instruction corresponding to the new contents of the instruction counter".

MNEMONIC CODE: ALIT

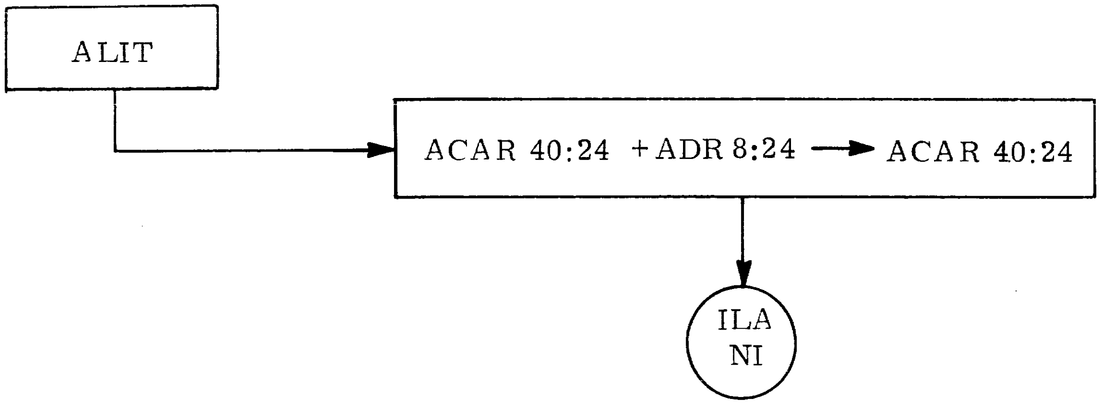
OPERATION: Add Literal to Address Field of ACAR

AIR:

16	1	XX	ADR
0 4	5	6 7	8 31

DESCRIPTION: This instruction causes the address field of the specified ACAR (bits 40:24) to be replaced by the sum of the address field of the instruction (ADR 8:24) and the address field of the specified ACAR. Bit 5 of the instruction must be "one". Bits 6 and 7 (see XX in format) designate the ACAR which is used to index the ADR field (bits 8:24) of the AIR. The results of the indexing operation are returned to the specified ACAR (bits 40:24). ACAR bits 0:40 are not changed; any overflow is disregarded.

FLOW CHART:



MNEMONIC CODE: BIN(X)

OPERATION: Block Fetch (RGX Indexed) from PE Memory to ADB

AIR:

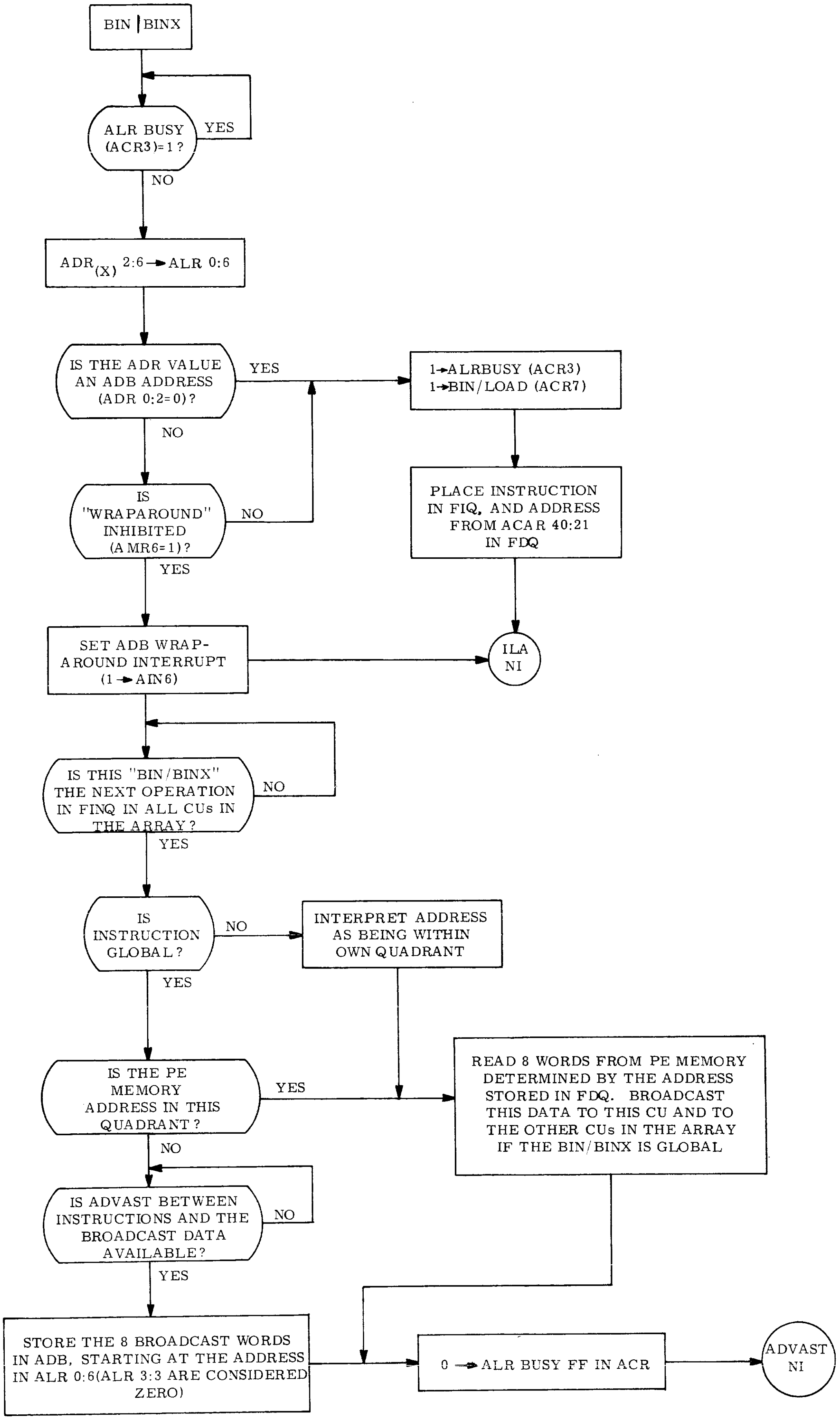
BIN	06	ACARX	▨	ACAR	G/L	—	10	ADR
	0 4	5 7		16 17	18	19	20 23	24 31
BINX	06	ACARX	▨	ACAR	G/L	—	11	ADR
	0 4	5 7		16 17	18	19	20 23	24 31

DESCRIPTION: This instruction causes a block of eight words to be read from PE memory and stored in ADB. The PE memory address is taken from the specified ACAR and, if BINX, is modified by RGX in the selected PUs. The resultant PE memory address is treated as though the three least significant bits were "zero" and then incremented until eight words are transferred. The ADB address is taken from the ADR field of the instruction and is indexable. The resultant ADB address is limited to the ADB address area and is treated as though the three least significant bits were "zero" and then incremented until eight words are transferred.

BIN/BINX causes the ADVAST station to stall if ACR3 is set, that is, if a previously requested BIN/BINX or LOAD/LOADX instruction using ADB has not been completed. The ALR register is used to hold the unfilled local address until the operation is executed at FINST. Note that any instructions that reference unfilled location(s) as specified in ALR will cause ADVAST to stall until the location(s) are filled. ACR7 specifies whether one or eight locations are locked out.

When this instruction is ready for execution at FINST and MSU, all CUs in the array are synchronized. The address in PE memory is interpreted according to ACAR 56:2 and the setting of MC0 and MC2 to determine which CU accesses the data. If the instruction is global then only one CU in the array will fetch the eight words from the PE. The eight words will then be broadcast to other CUs in the array. If the instruction is local then only this CU in the array will receive the eight words.

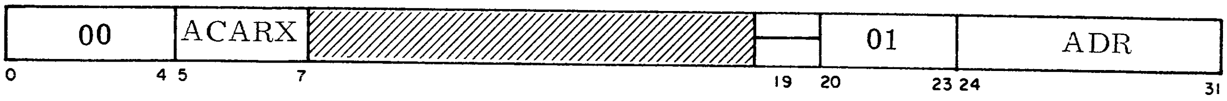
FLOW CHART: See next page.


BIN | BINX
ALR BUSY (ACR3)=1 ?
YES
NO
ADR$_{(X)}$ 2:6 → ALR 0:6
IS THE ADR VALUE AN ADB ADDRESS (ADR 0:2=0)?
YES
NO
1→ALRBUSY (ACR3)
1→BIN/LOAD (ACR7)
IS "WRAPAROUND" INHIBITED (AMR6=1)?
NO
YES
PLACE INSTRUCTION IN FIQ, AND ADDRESS FROM ACAR 40:21 IN FDQ
SET ADB WRAP-AROUND INTERRUPT (1 → AIN6)
ILA NI
IS THIS "BIN/BINX" THE NEXT OPERATION IN FINQ IN ALL CUs IN THE ARRAY?
NO
YES
IS INSTRUCTION GLOBAL?
NO
YES
INTERPRET ADDRESS AS BEING WITHIN OWN QUADRANT
IS THE PE MEMORY ADDRESS IN THIS QUADRANT?
YES
NO
READ 8 WORDS FROM PE MEMORY DETERMINED BY THE ADDRESS STORED IN FDQ. BROADCAST THIS DATA TO THIS CU AND TO THE OTHER CUs IN THE ARRAY IF THE BIN/BINX IS GLOBAL
IS ADVAST BETWEEN INSTRUCTIONS AND THE BROADCAST DATA AVAILABLE?
NO
YES
STORE THE 8 BROADCAST WORDS IN ADB, STARTING AT THE ADDRESS IN ALR 0:6(ALR 3:3 ARE CONSIDERED ZERO)
0 → ALR BUSY FF IN ACR
ADVAST NI

MNEMONIC CODE: CACRB

OPERATION: Set/Reset nth Bit in ADVAST Control Register

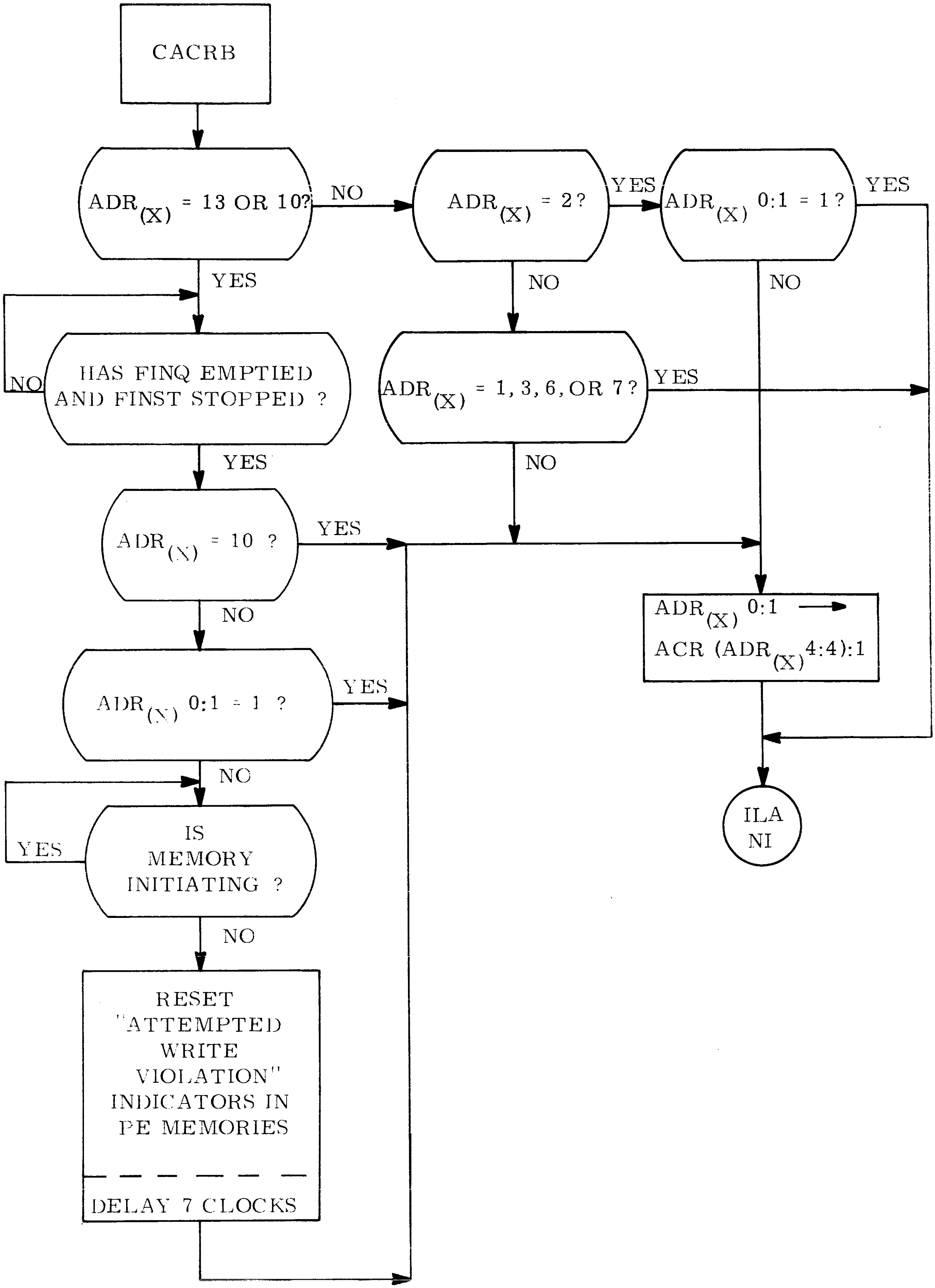
AIR:



DESCRIPTION: This instruction changes a bit in the ADVAST control register. The bit number is specified in ADR 4:4 and is indexable. The most significant bit of the local address field (ADR 0:1) will contain a "one" if the bit is to be set and a "zero" if it is to be reset. If ADR 4:4 equals 1, 3, 6, or 7, then the ADVAST control register will not be changed. If ADR 4:4 equals 2, then the ADVAST control bit will not be set.

Three bits, ACR9, ACR10, and ACR13, control operations in the PEs but are set and reset by CACRB which is an ADVAST instruction. A new value of the bit should be effective only on the instructions which follow the CACRB instruction. Since the PEs may still be executing, from FINQ, instructions which preceded the CACRB, there is a potential problem in synchronization. Hardware interlocks automatically resolve this potential problem in the case of bits 10 and 13. A change in ACR9, however, will apply as soon as the change is effective, and will apply even to those instructions still remaining unexecuted in FINQ. In case of doubt, a CACRB9 can be preceded by a FINQ instruction; CACRB10 or CACRB13 need not.

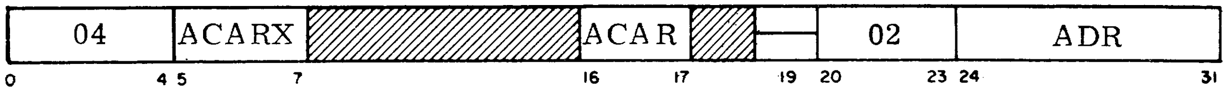
FLOW CHART: See next page.


CACRB
ADR(X) = 13 OR 10?
NO
YES
ADR(X) = 2?
YES
NO
ADR(X) 0:1 = 1?
YES
NO
HAS FINQ EMPTIED AND FINST STOPPED ?
NO
YES
ADR(X) = 1, 3, 6, OR 7?
YES
NO
ADR(X) = 10 ?
YES
NO
ADR(X) 0:1 = 1 ?
YES
NO
IS MEMORY INITIATING ?
YES
NO
RESET "ATTEMPTED WRITE VIOLATION" INDICATORS IN PE MEMORIES
DELAY 7 CLOCKS
ADR(X) 0:1 →
ACR (ADR(X) 4:4):1
ILA
NI

MNEMONIC CODE: CADD

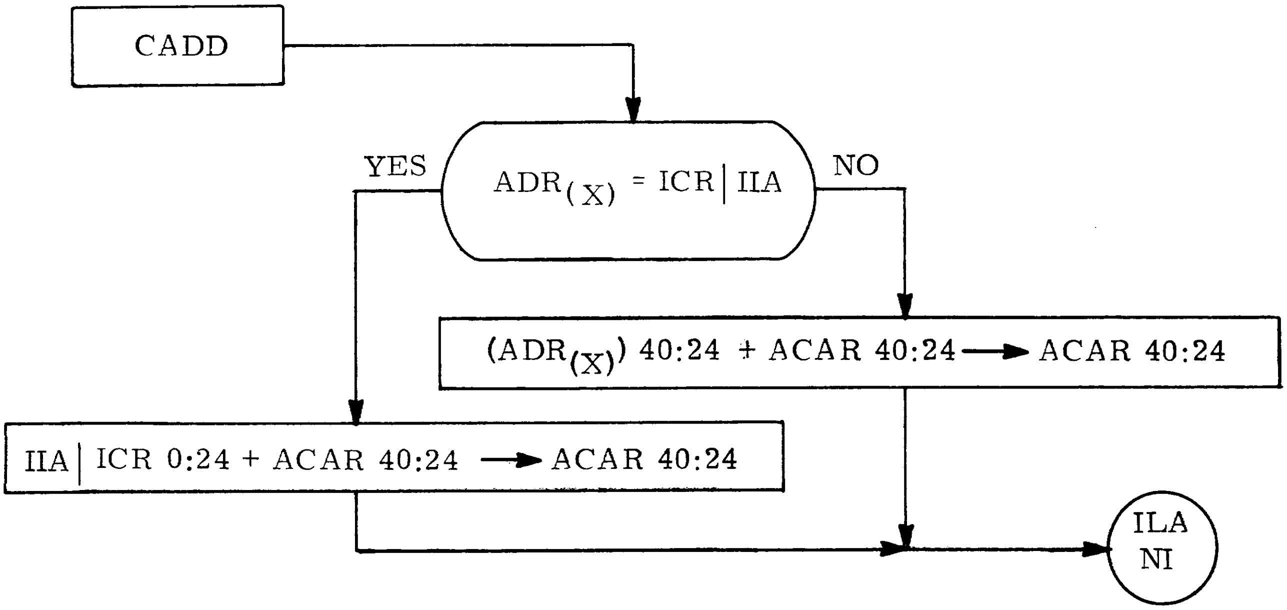
OPERATION: Add Local Memory to ACAR

AIR:



DESCRIPTION: This instruction adds the operand in the local address to the contents of the specified ACAR. The local address field is indexable. Operation is limited to the least significant 24 bits of the ACAR and the local operand, except that the least significant bit of ICR and IIA are not used. Overflow is disrеgarded. The result is stored in the least significant 24 bits of the ACAR. The most significant portion of the ACAR is not changed. The address is restricted to ADB, the ACARs, the ICR, or the IIA.

FLOW CHART:


CADD
YES
ADR(X) = ICR | IIA
NO
(ADR(X)) 40:24 + ACAR 40:24 ⟶ ACAR 40:24
IIA | ICR 0:24 + ACAR 40:24 ⟶ ACAR 40:24
ILA
NI

MNEMONIC CODE: CAND

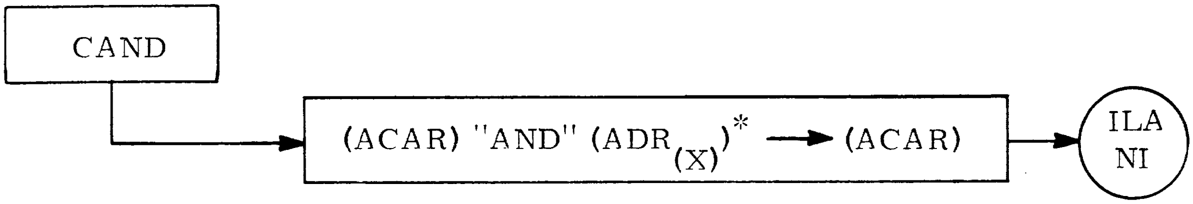
OPERATION: Logical AND of Local Memory and ACAR

AIR:

04	ACARX	▨	ACAR	▨	—	10	ADR
0 4	5 7		16 17		19	20 23	24 31

DESCRIPTION: This instruction performs the logical AND between the specified ACAR and the operand in the local address. The local address is indexable and limited to the ADB and the four ACARs. The result is stored in the specified ACAR.

FLOW CHART:



*Local address limited to ADB and the four ACARs.

MNEMONIC CODE: CCB

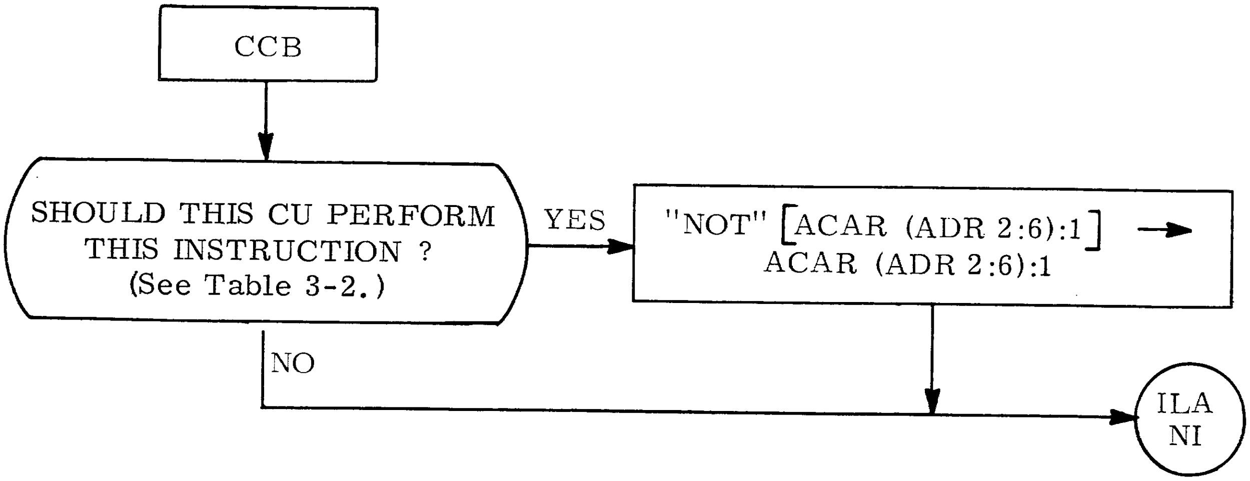
OPERATION: Complement nth Bit of ACAR

AIR:

11	ACARX	▨	ACAR	G/L	—	01	ADR
0 4	5 7		16 17	18	19	20 23	24 31

DESCRIPTION: The local address field of this instruction specifies a CU to complement a bit in the specified ACAR. The CU number is specified in ADR 0:2 and is relative to MC2 and MC0. The bit number to be complemented is specified in ADR 2:6. The local address field of the instruction is indexable. This is a NO-OP for the CUs not selected to perform the operation.

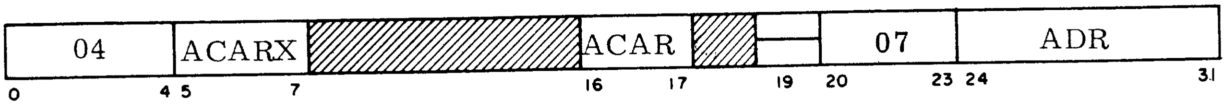
FLOW CHART:


CCB
SHOULD THIS CU PERFORM THIS INSTRUCTION ? (See Table 3-2.)
YES
"NOT" [ACAR (ADR 2:6):1] → ACAR (ADR 2:6):1
NO
ILA NI

MNEMONIC CODE: CEXOR

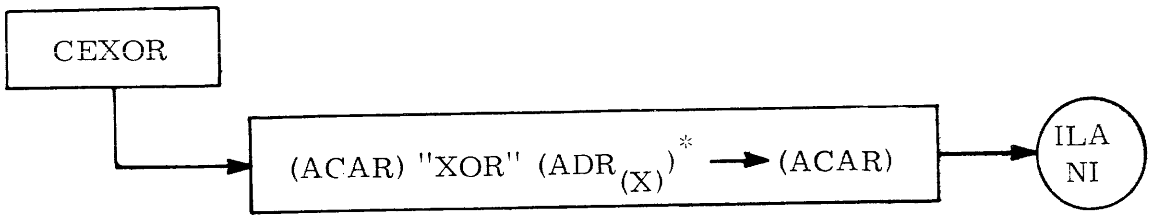
OPERATION: Logical EXCLUSIVE-OR of Local Memory and ACAR

AIR:



DESCRIPTION: This instruction performs the logical EXCLUSIVE-OR between the specified ACAR and the operand in the local address. The local address is indexable and limited to the ADB and the four ACARs. The result is stored in the specified ACAR.

FLOW CHART:

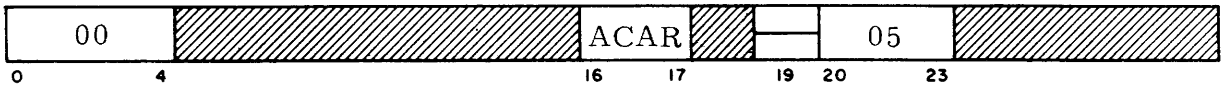


*Local address limited to ADB and the four ACARs.

MNEMONIC CODE: CLC

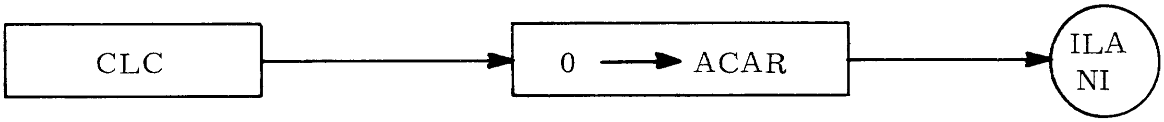
OPERATION: Clear ACAR

AIR:



DESCRIPTION: This instruction causes the CU to reset the specified ACAR to all zeros.

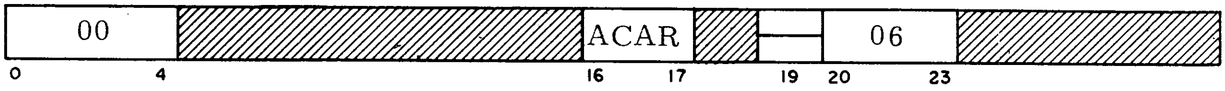
FLOW CHART:



MNEMONIC CODE: COMPC

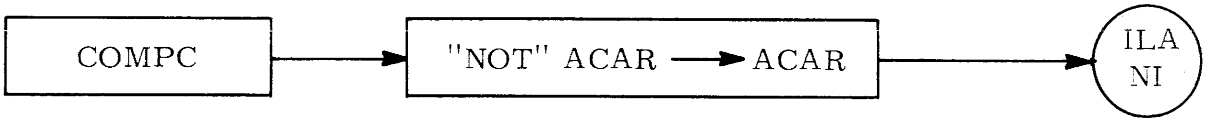
OPERATION: Complement ACAR

AIR:



DESCRIPTION: This instruction causes each bit of the specified ACAR to be inverted.

FLOW CHART:



MNEMONIC CODE: COPY

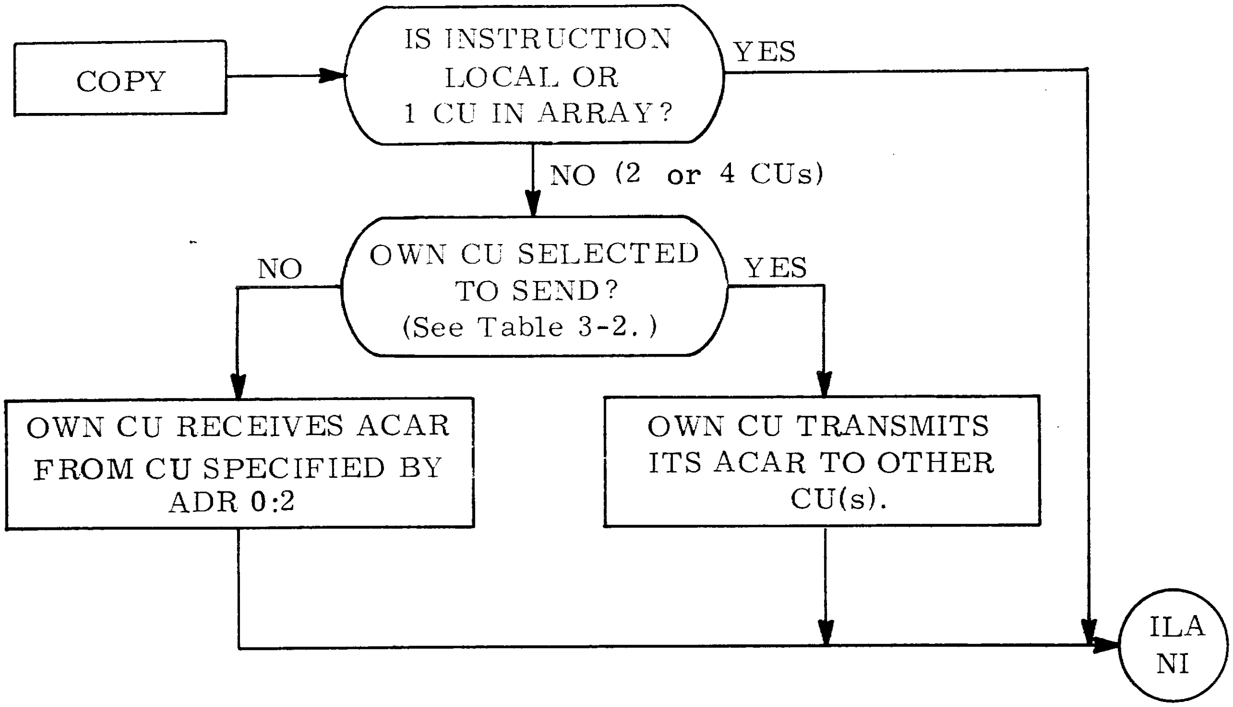
OPERATION: Copy ACAR

AIR:

02	ACARX	▨	ACAR	G/L	—	04	ADR
0 4	5 7		16 17	18	19	20 23	24 31

DESCRIPTION: This instruction causes all CUs in the array to be synchronized at the beginning of the instruction. The local address field of the instruction selects a CU whose specified ACAR is to be copied into the same ACAR of the non-selected CUs. The selected CU sends its ACAR to the other CUs each of which stores it in its ACAR. The local address field is indexable. The CU number is specified in ADR 0:2, and is relative to MC2 and MC0. This instruction is a NO-OP in single quadrant array.

FLOW CHART:


COPY
IS INSTRUCTION LOCAL OR 1 CU IN ARRAY?
YES
NO (2 or 4 CUs)
OWN CU SELECTED TO SEND? (See Table 3-2.)
NO
YES
OWN CU RECEIVES ACAR FROM CU SPECIFIED BY ADR 0:2
OWN CU TRANSMITS ITS ACAR TO OTHER CU(s).
ILA NI

MNEMONIC CODE: COR

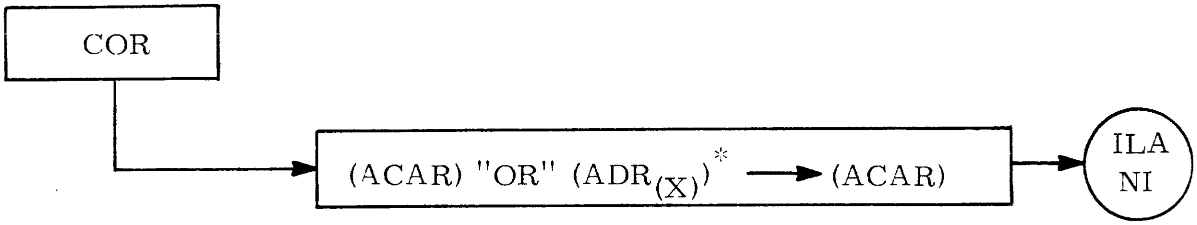
OPERATION: Logical OR of Local Memory and ACAR

AIR:

04	ACARX	▨	ACAR	▨	—	11	ADR
0 4	5 7		16 17		19	20 23	24 31

DESCRIPTION: This instruction performs the logical OR between the specified ACAR and the operand in the local address. The local address is indexable and limited to the ADB and the four ACARs. The result is stored in the specified ACAR.

FLOW CHART:

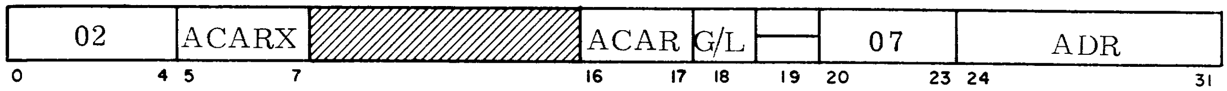


*Local address limited to ADB and the four ACARs.

MNEMONIC CODE: CRB

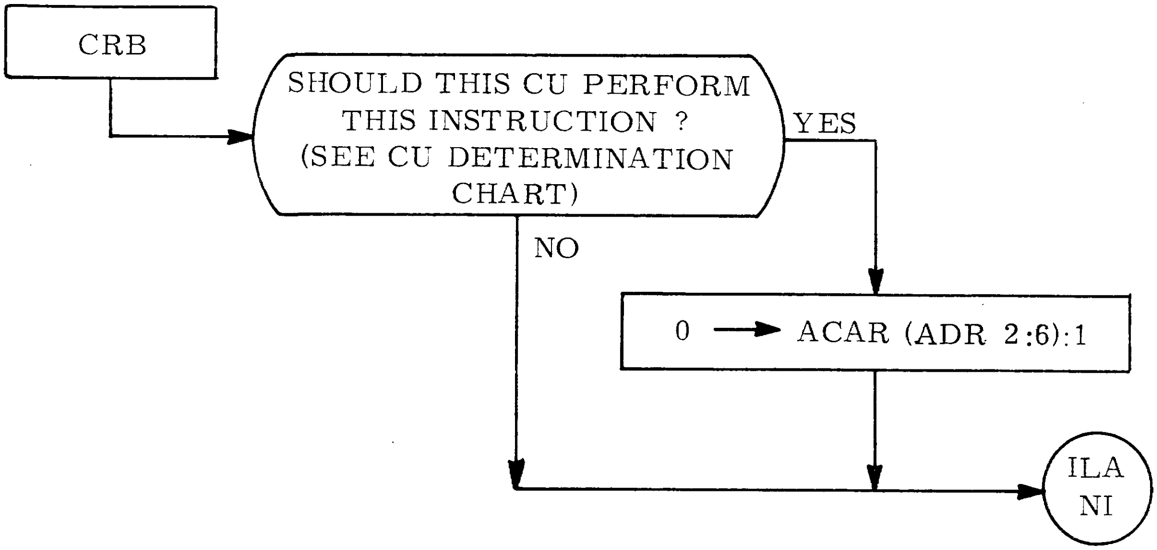
OPERATION: Reset nth Bit in ACAR

AIR:



DESCRIPTION: The local address field of this instruction specifies a CU to reset a bit in the specified ACAR. The CU number is specified in ADR 0:2, as interpreted by the array size and configuration control logic. The bit number to be reset is specified in ADR 2:6. The local address field of the instruction is indexable. This is a NO-OP for the CUs not selected to perform the operation.

FLOW CHART:


CRB
SHOULD THIS CU PERFORM THIS INSTRUCTION ? (SEE CU DETERMINATION CHART)
YES
NO
0 → ACAR (ADR 2:6):1
ILA NI

MNEMONIC CODE: CROTL

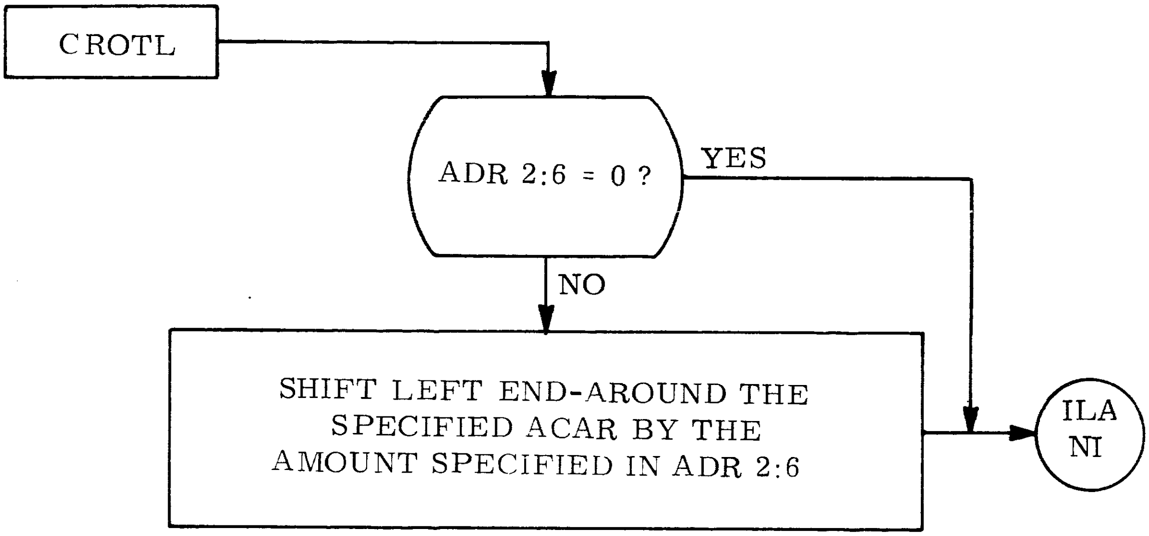
OPERATION: Rotate ACAR Left (End Around)

AIR:

00	ACARX	▨	ACAR	▨	—	15	ADR
0 4	5 7		16	17	19	20 23	24 31

DESCRIPTION: This instruction shifts the specified ACAR to the left end-around, by an amount specified in ADR 2:6. The ADR field is indexable.

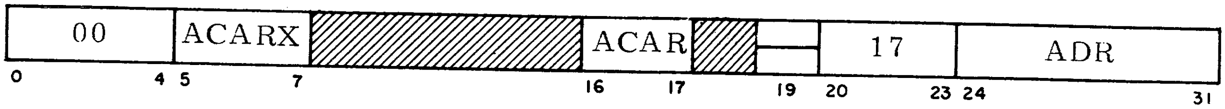
FLOW CHART:


CROTL
ADR 2:6 = 0 ?
YES
NO
SHIFT LEFT END-AROUND THE SPECIFIED ACAR BY THE AMOUNT SPECIFIED IN ADR 2:6
ILA NI

MNEMONIC CODE: CROTR

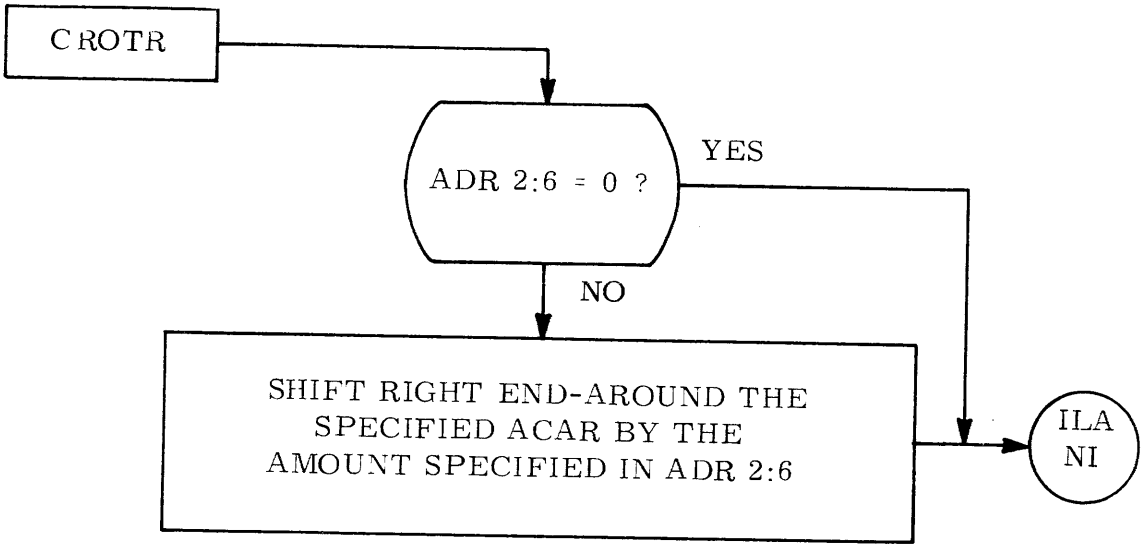
OPERATION: Rotate ACAR Right (End Around)

AIR:



DESCRIPTION: This instruction shifts the specified ACAR to the right end-around by an amount specified in ADR 2:6. The ADR field is indexable.

FLOW CHART:



CROTR
ADR 2:6 = 0 ?
YES
NO
SHIFT RIGHT END-AROUND THE SPECIFIED ACAR BY THE AMOUNT SPECIFIED IN ADR 2:6
ILA NI

MNEMONIC CODE: CSB

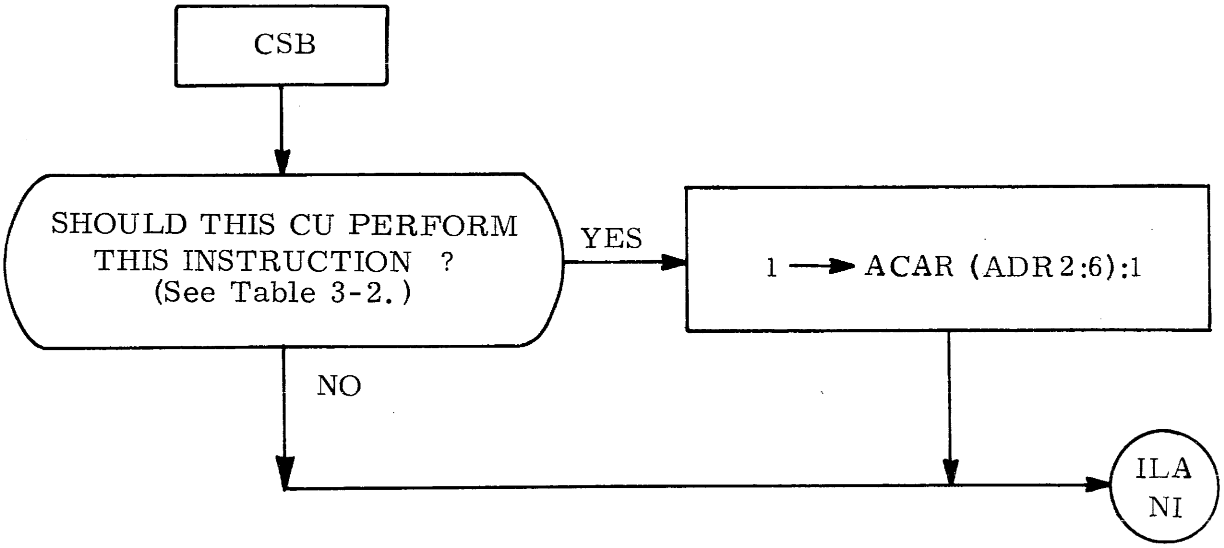
OPERATION: Set nth Bit in ACAR

AIR:

00	ACARX	▨	ACAR	G/L	—	13	ADR
0 4	5 7		16 17	18	19	20 23	24 31

DESCRIPTION: The local address field of this instruction specifies a CU to set a bit in the specified ACAR. The CU number is specified in ADR 0:2, as interpreted by the array size and configuration control logic. The bit number to be set is specified in ADR 2:6. The local address field of the instruction is indexable. This is a NO-OP for the CUs not selected to perform the operation.

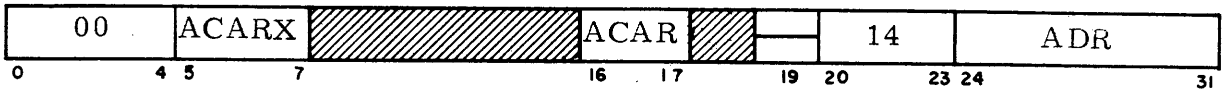
FLOW CHART:



MNEMONIC CODE: CSHL

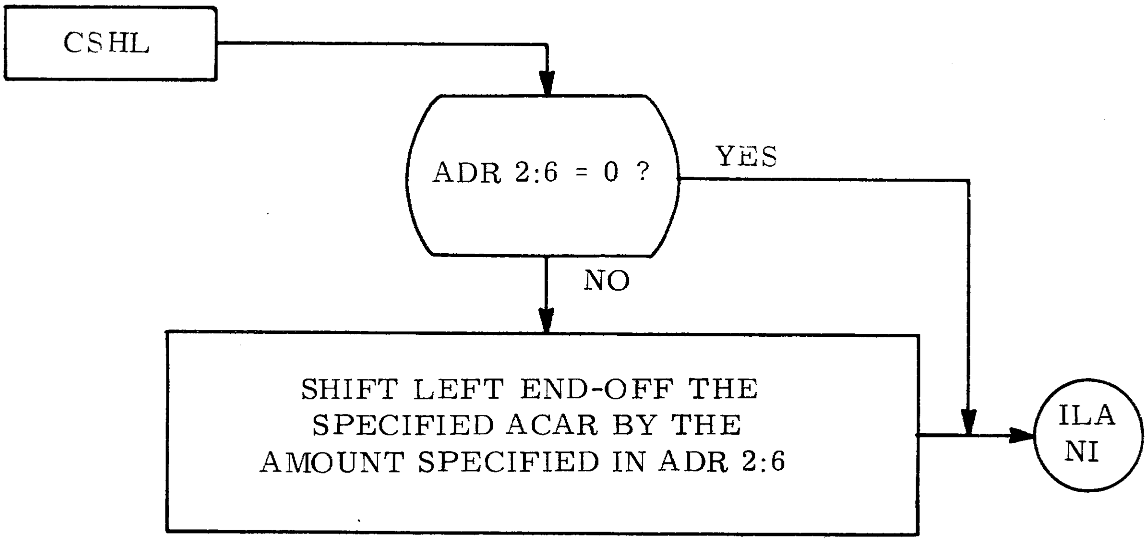
OPERATION: Shift ACAR Left (End Off)

AIR:



DESCRIPTION: This instruction shifts the specified ACAR to the left end-off, by an amount specified in ADR 2:6. The ADR field is indexable. Zeros replace vacated bit positions at the right end of the ACAR.

FLOW CHART:


CSHL
ADR 2:6 = 0 ?
YES
NO
SHIFT LEFT END-OFF THE SPECIFIED ACAR BY THE AMOUNT SPECIFIED IN ADR 2:6
ILA
NI

MNEMONIC CODE: CSHR

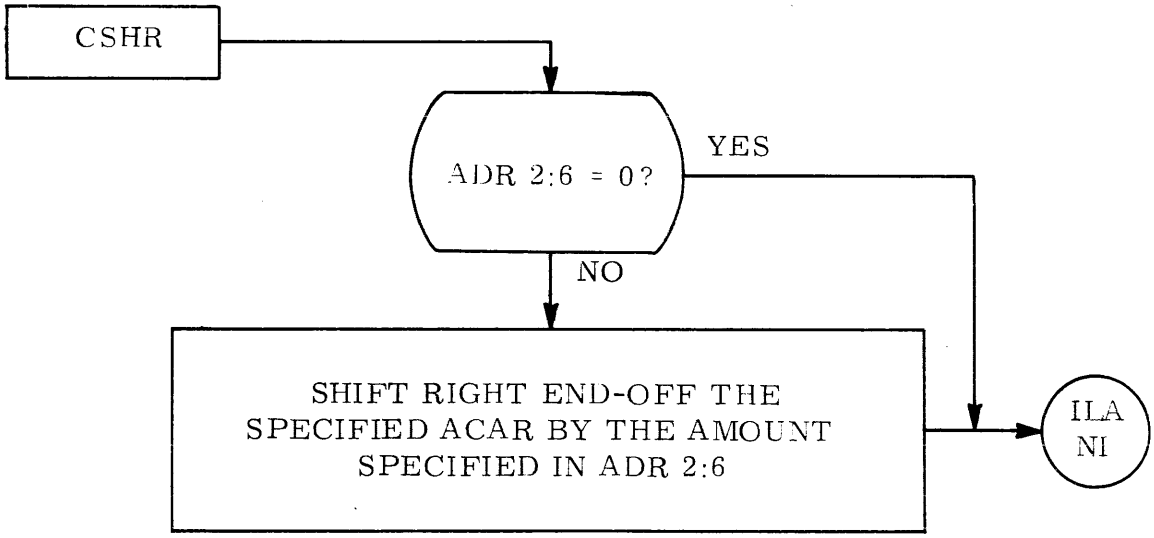
OPERATION: Shift ACAR Right (End Off)

AIR:

00	ACARX	▨	ACAR	▨	—	16	ADR
0 4	5 7		16	17	19	20 23	24 31

DESCRIPTION: This instruction shifts the specified ACAR to the right end-off, by an amount specified in ADR 2:6. The ADR field is indexable. Zeros replace vacated bit positions at the left end of the ACAR.

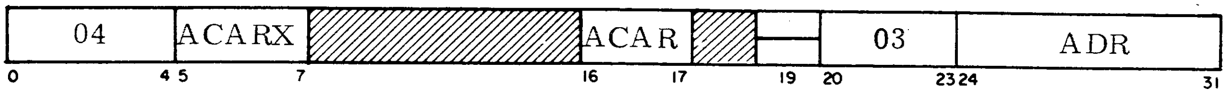
FLOW CHART:


CSHR
ADR 2:6 = 0?
YES
NO
SHIFT RIGHT END-OFF THE SPECIFIED ACAR BY THE AMOUNT SPECIFIED IN ADR 2:6
ILA
NI

MNEMONIC CODE: CSUB

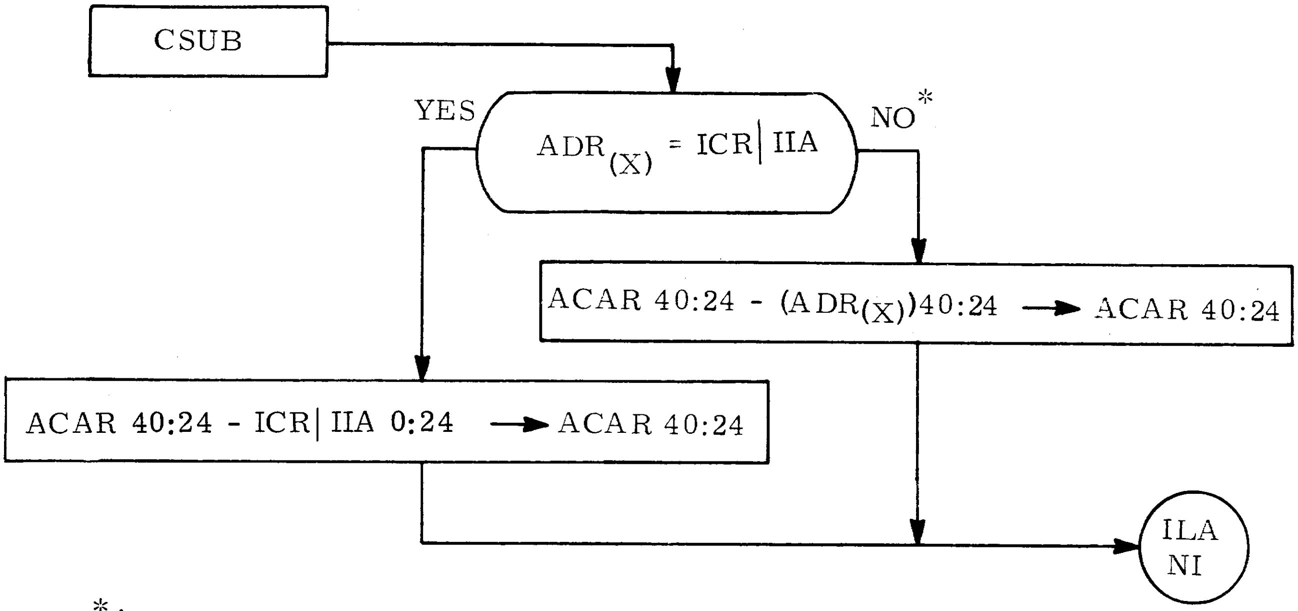
OPERATION: Subtract Local Memory from ACAR

AIR:



DESCRIPTION: This instruction subtracts the operand in the local address from the contents of the specified ACAR. The local address field is indexable; addresses are limited to the ADB, the ICR, the IIA, and the four ACARs. Operation is limited to the least significant 24 bits of the ACAR and the local operand, except that the least significant bit of ICR and IIA are not used. Overflow is disregarded. Underflow (a negative result) is shown in 2's complement form. The result is stored in the least significant 24 bits of the ACAR. ACAR 0:40 is not changed.

FLOW CHART:


CSUB
YES
ADR(X) = ICR| IIA
NO*
ACAR 40:24 - (ADR(X))40:24 → ACAR 40:24
ACAR 40:24 - ICR| IIA 0:24 → ACAR 40:24
ILA
NI

*ADR$_{(X)}$ is limited to ICR, IIA, ADB, and the four ACARs.

MNEMONIC CODE: CTSB (F | T)

OPERATION: Skip if nth Bit in ACAR is (Not One | One)

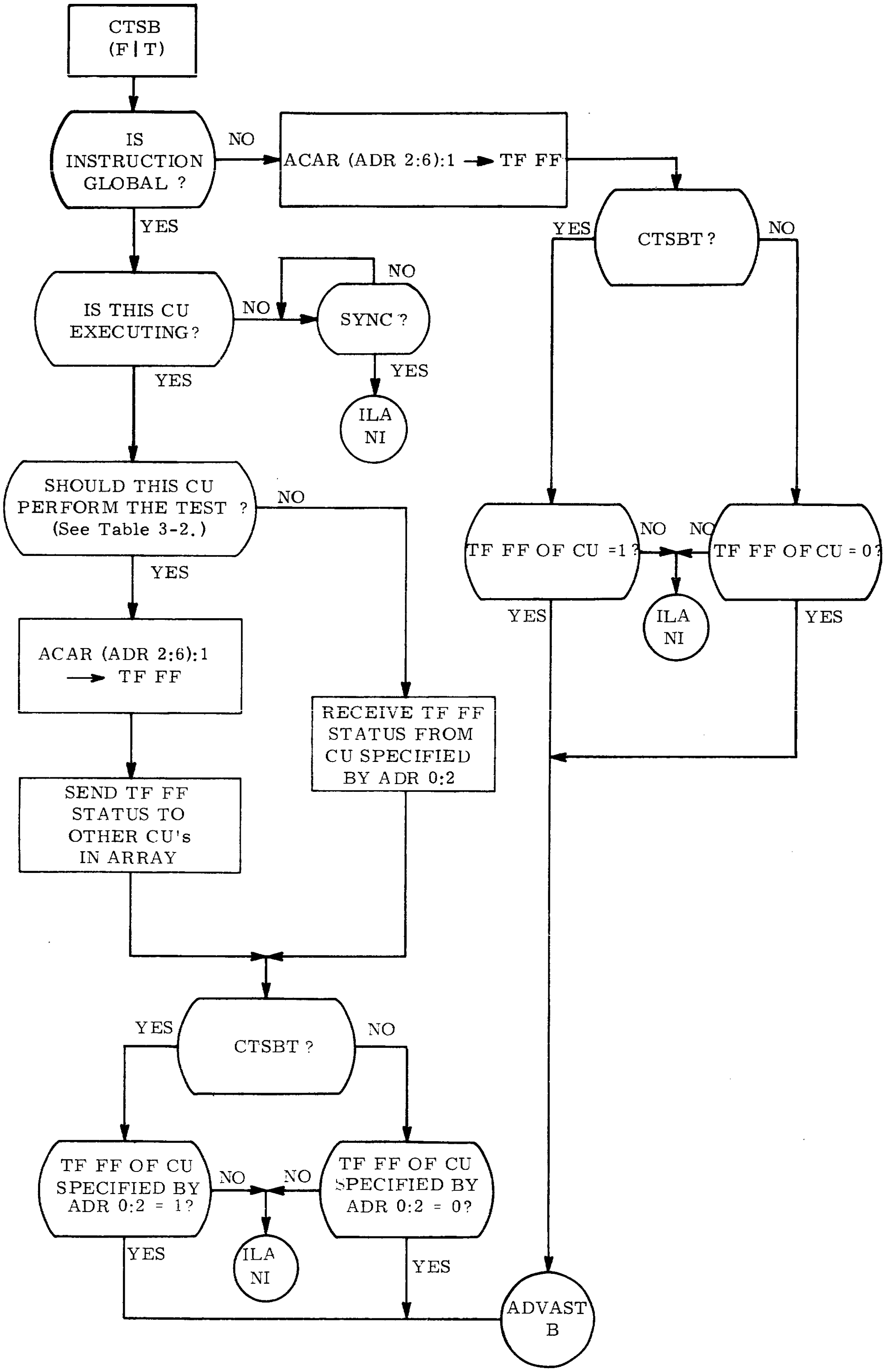
AIR:

CTSBF	11	ACARX	SKIP	ACAR	G/L	—	02	ADR
	0 4	5 7	8 15	16 17	18	19	20 23	24 31
CTSBT	11	ACARX	SKIP	ACAR	G/L	—	00	ADR
	0 4	5 7	8 15	16 17	18	19	20 23	24 31

DESCRIPTION: This instruction causes all CUs specified by MC0 to be synchronized at the beginning of the instruction. ADR 0:2 contains the number of the CU to test the bit of its ACAR. ADR 2:6 designates the number of the bit in the specified ACAR to be tested for logical one. The local address is indexable. The TF flip-flop is set if the bit is true and reset if the bit is false. If the instruction is global, then each CU (relative to MC0 and MC2) executing the test sends its TF flip-flop to the other CUs. The CU will sample the TF flip-flop line indicated by ADR 0:2 of the CUs specified by MC2 relative to MC0 and if the TF FF is as specified in the Op Code then the jump is executed. If the instruction is local then the CU uses its own TF flip-flop for the test.

The jump address is derived by modifying the ICR by the SKIP field of the instruction (after stepping the ICR). SKIP 0:1 is the sign bit, where "1" means subtract and "0" means add. SKIP 1:7 are the magnitude bits of the modifier, where each count corresponds to a 32-bit word.

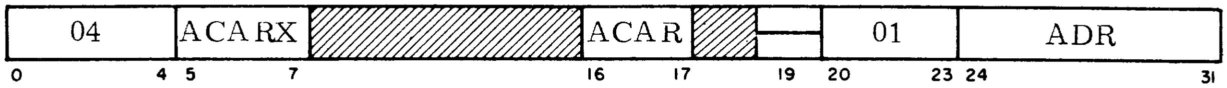
FLOW CHART: See next page.


CTSB (F | T)
IS INSTRUCTION GLOBAL ?
NO
ACAR (ADR 2:6):1 → TF FF
YES
CTSBT ?
YES
NO
IS THIS CU EXECUTING?
NO
SYNC ?
NO
YES
ILA NI
YES
SHOULD THIS CU PERFORM THE TEST ? (See Table 3-2.)
NO
YES
TF FF OF CU =1 ?
NO
NO
TF FF OF CU = 0?
YES
ILA NI
YES
ACAR (ADR 2:6):1 → TF FF
RECEIVE TF FF STATUS FROM CU SPECIFIED BY ADR 0:2
SEND TF FF STATUS TO OTHER CU's IN ARRAY
YES
CTSBT ?
NO
TF FF OF CU SPECIFIED BY ADR 0:2 = 1?
NO
NO
TF FF OF CU SPECIFIED BY ADR 0:2 = 0?
YES
ILA NI
YES
ADVAST B

MNEMONIC CODE: DUPI

OPERATION: Duplicate Inner Half of ADB Memory Word

AIR:



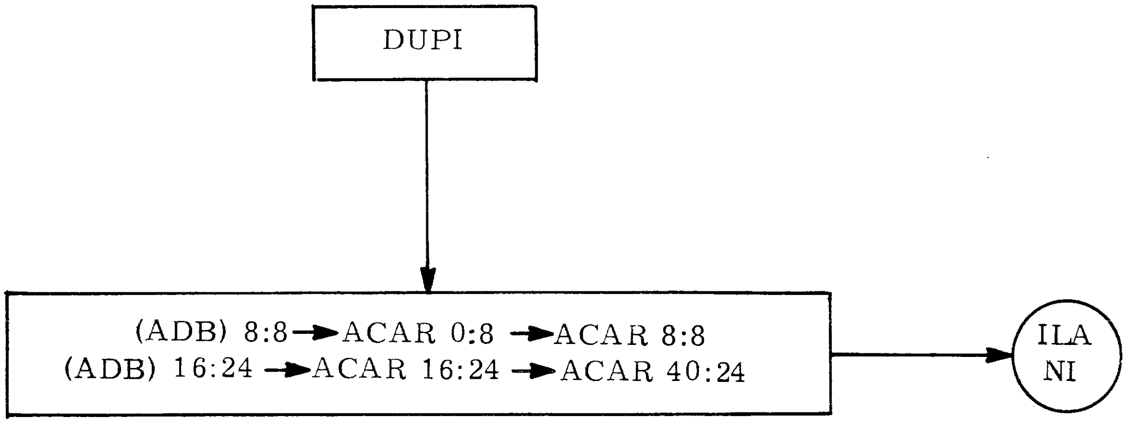
DESCRIPTION: This instruction causes the CU to duplicate the inner half of the word found in ADB memory into both halves of the specified ACAR.

Bit Alignment:

ADB Memory (Inner Word)	ACAR (Duplicate Word)
8-15	0-7
16-39	40-63
8-39	8-39

The local memory address of the instruction is indexable, and is restricted to the addresses of ADB; that is, bits 24 and 25 (in ADR) must contain 00.

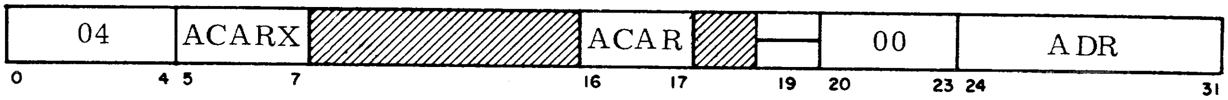
FLOW CHART:


DUPI
(ADB) 8:8→ACAR 0:8 →ACAR 8:8
(ADB) 16:24 →ACAR 16:24 →ACAR 40:24
ILA
NI

MNEMONIC CODE: DUPO

OPERATION: Duplicate Outer Half of ADB Memory Word

AIR:



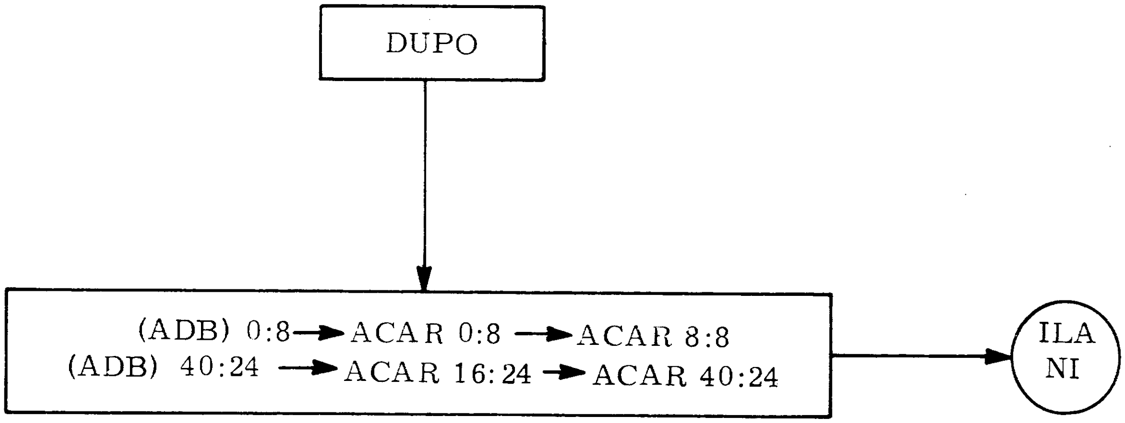
DESCRIPTION: This instruction causes the CU to duplicate the outer half of the word found in ADB memory into both halves of the specified ACAR.

Bit Alignment:

ADB Memory (Outer Word)	ACAR (Duplicate Word)
0-7	8-15
40-63	16-39
0-7	0-7
40-63	40-63

The local memory address of the instruction is indexable, and is restricted to the addresses of ADB; that is, bits 24 and 25 (in ADR) must contain 00.

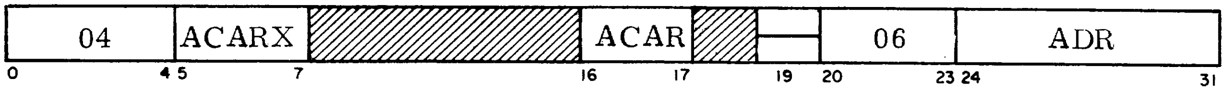
FLOW CHART:


DUPO
(ADB) 0:8 → ACAR 0:8 → ACAR 8:8
(ADB) 40:24 → ACAR 16:24 → ACAR 40:24
ILA
NI

MNEMONIC CODE: EXCHL

OPERATION: Exchange Local Operand and ACAR

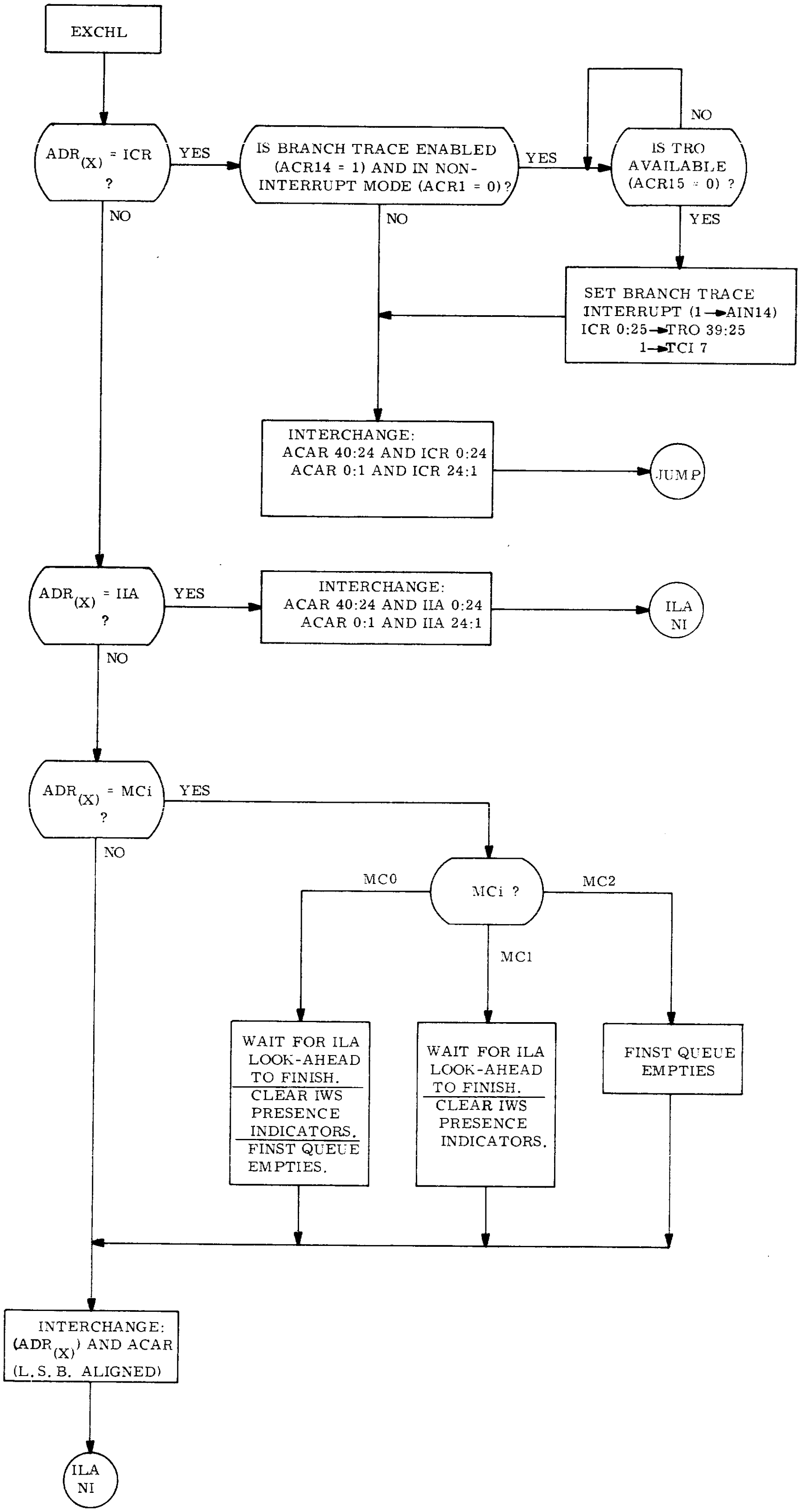
AIR:



DESCRIPTION: This instruction interchanges the contents of the specified ACAR and the operand in the local address. The local address field is indexable and only the following addresses are permitted: ADB, AIN, ALR, AMR, AC 0-3, ICR, MC0-2, IIA, and TRO. Each local address, except the IIA and ICR, has its least significant bit aligned with the least significant bit of the ACAR. The ICR and IIA have their second least significant bit aligned with bit 63 of the ACAR. The most significant bit of the ACAR is interchanged with the least significant bit of the ICR and IIA. When this instruction loads the ICR, the incrementing of the ICR upon completion of the instruction is inhibited and a jump occurs. If this instruction is executed and the ICR is updated and other branch trace conditions are met, then an interrupt will occur and program control will proceed to interrupt processing. Loading MC0 or MC1 causes the IWS presence indicators to be cleared. Resetting of the presence bits does not inhibit execution of the block currently being executed from IWS, but requires that the next block entered must be fetched from memory. Loading MC0 or MC2 causes the FINST queue to empty before the interchange is performed.

All bits not replaced by local memory bits will be reset to zero in the accumulator.

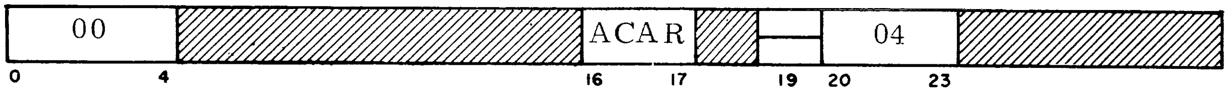
FLOW CHART: See next page.


EXCHL
$\text{ADR}_{(X)}$ = ICR ?
YES
NO
IS BRANCH TRACE ENABLED (ACR14 = 1) AND IN NON-INTERRUPT MODE (ACR1 = 0)?
YES
NO
IS TRO AVAILABLE (ACR15 = 0) ?
NO
YES
SET BRANCH TRACE INTERRUPT (1→AIN14)
ICR 0:25→TRO 39:25
1→TCI 7
INTERCHANGE:
ACAR 40:24 AND ICR 0:24
ACAR 0:1 AND ICR 24:1
JUMP
$\text{ADR}_{(X)}$ = IIA ?
YES
NO
INTERCHANGE:
ACAR 40:24 AND IIA 0:24
ACAR 0:1 AND IIA 24:1
ILA NI
$\text{ADR}_{(X)}$ = MCi ?
YES
NO
MCi ?
MC0
MC1
MC2
WAIT FOR ILA LOOK-AHEAD TO FINISH.
CLEAR IWS PRESENCE INDICATORS.
FINST QUEUE EMPTIES.
WAIT FOR ILA LOOK-AHEAD TO FINISH.
CLEAR IWS PRESENCE INDICATORS.
FINST QUEUE EMPTIES
INTERCHANGE:
$(\text{ADR}_{(X)})$ AND ACAR
(L.S.B. ALIGNED)
ILA NI

MNEMONIC CODE: EXEC

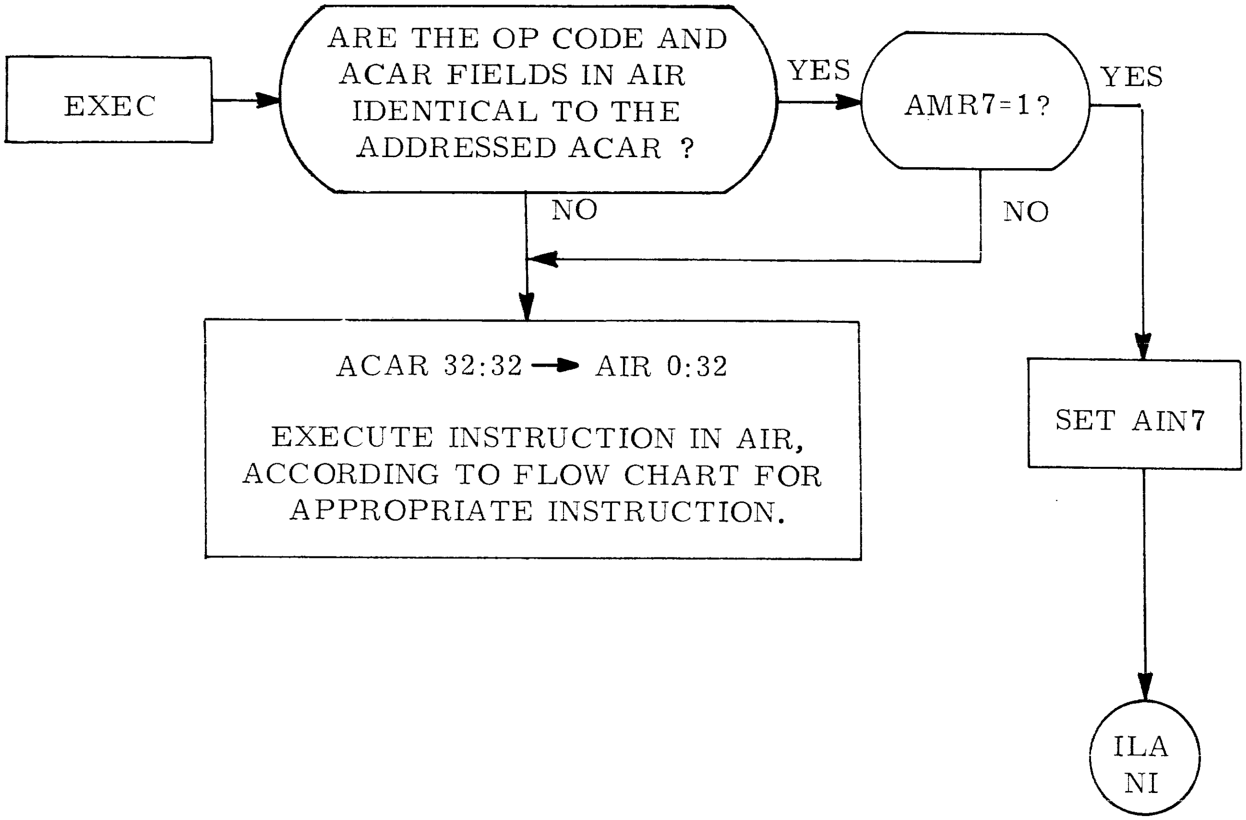
OPERATION: Execute

AIR:



DESCRIPTION: This instruction causes the least significant 32 bits of the specified ACAR to be transferred to the AIR. The transfer from IWS to AIR and incrementing of the ICR are inhibited. Normal operation resumes with the execution of the instruction just loaded into the AIR. No parity check is performed on the instruction accessed from the ACAR.

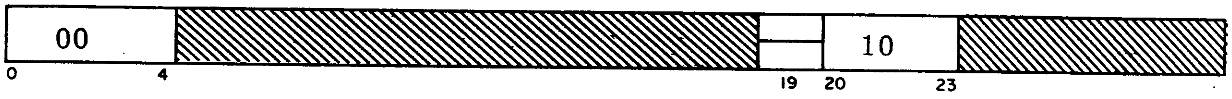
FLOW CHART:


EXEC
ARE THE OP CODE AND ACAR FIELDS IN AIR IDENTICAL TO THE ADDRESSED ACAR ?
YES
AMR7=1?
YES
NO
NO
ACAR 32:32 → AIR 0:32
EXECUTE INSTRUCTION IN AIR, ACCORDING TO FLOW CHART FOR APPROPRIATE INSTRUCTION.
SET AIN7
ILA
NI

MNEMONIC CODE: FINQ

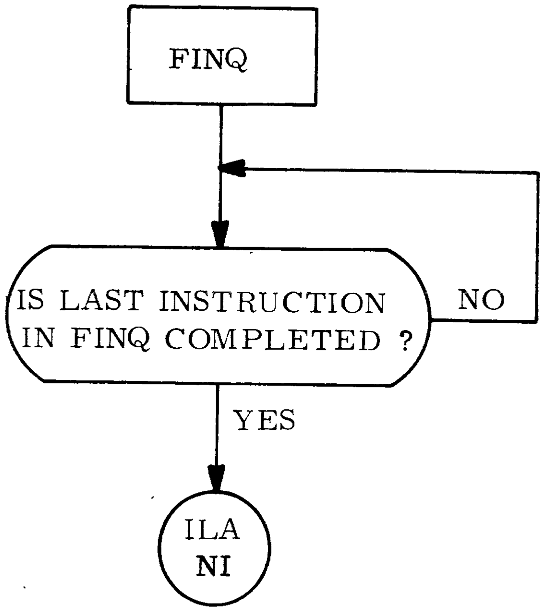
OPERATION: Stop ADVAST Until FINST is Idle

AIR:



DESCRIPTION: This instruction causes ADVAST to stop operating until FINST is idle. ADVAST resumes normal operation at the completion of the last instruction in FINQ. This instruction is a NO-OP when the CU is operating in Single Instruction Mode or in Interrupt Mode.

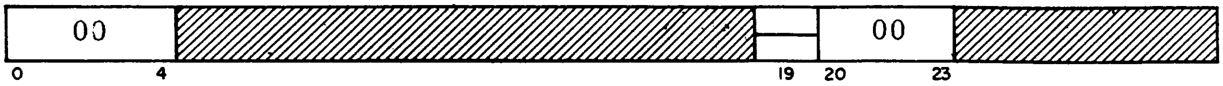
FLOW CHART:



MNEMONIC CODE: HALT

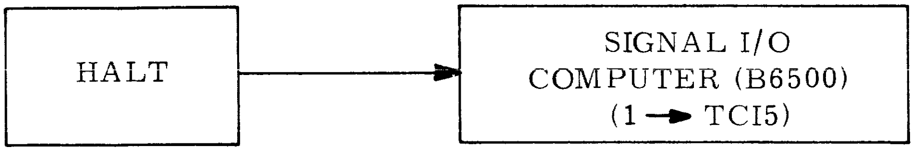
OPERATION: Cu Comes to Orderly Idle State

AIR:



DESCRIPTION: This instruction causes the control unit to stop operating. This is accomplished by causing ADVAST to cease fetching instructions from IWS. In turn, FINST will complete the present queued operations and will stop operating. All pending memory fetches will be completed. However, all communications with the I/O and between the I/O and main memory continue normally.

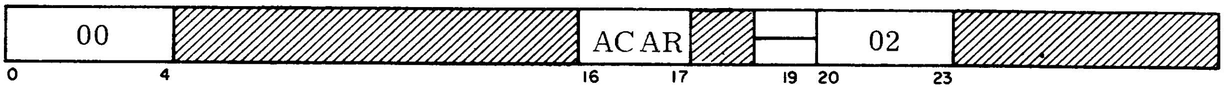
FLOW CHART:



MNEMONIC CODE: INCRXC

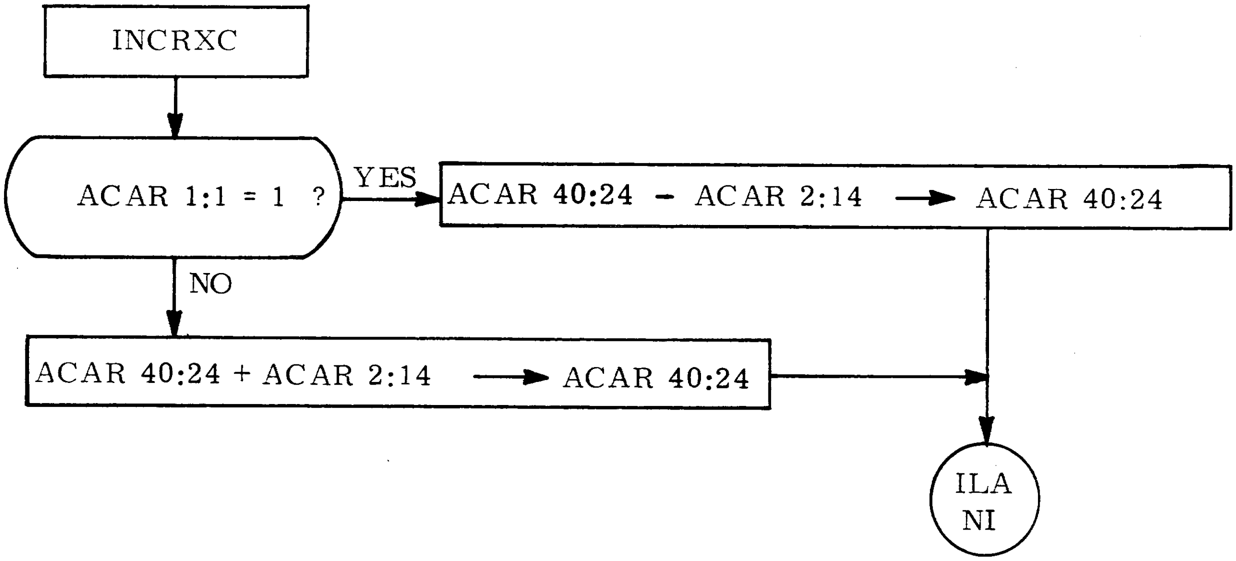
OPERATION: Modify Index Field of ACAR by Increment Field of Same ACAR

AIR:



DESCRIPTION: The increment field (bits 1:15) of the specified ACAR is added to the index field (bits 40:24) of the same ACAR. Bit 15 is justified with bit 63; bit 1 is the sign bit of the increment. The resultant sum is stored back in the index field of the ACAR. The other bits of the ACAR are not disturbed. Any overflow/underflow is disregarded. The most significant ten bits of the increment operand will be treated as logical zeros.

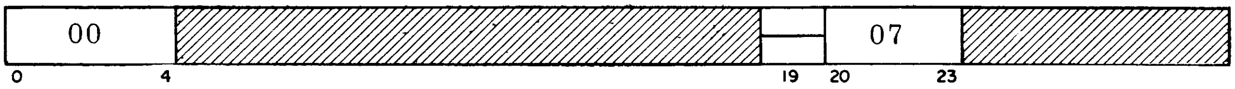
FLOW CHART:


INCRXC
ACAR 1:1 = 1 ?
YES
ACAR 40:24 - ACAR 2:14 ⟶ ACAR 40:24
NO
ACAR 40:24 + ACAR 2:14 ⟶ ACAR 40:24
ILA
NI

MNEMONIC CODE: INR

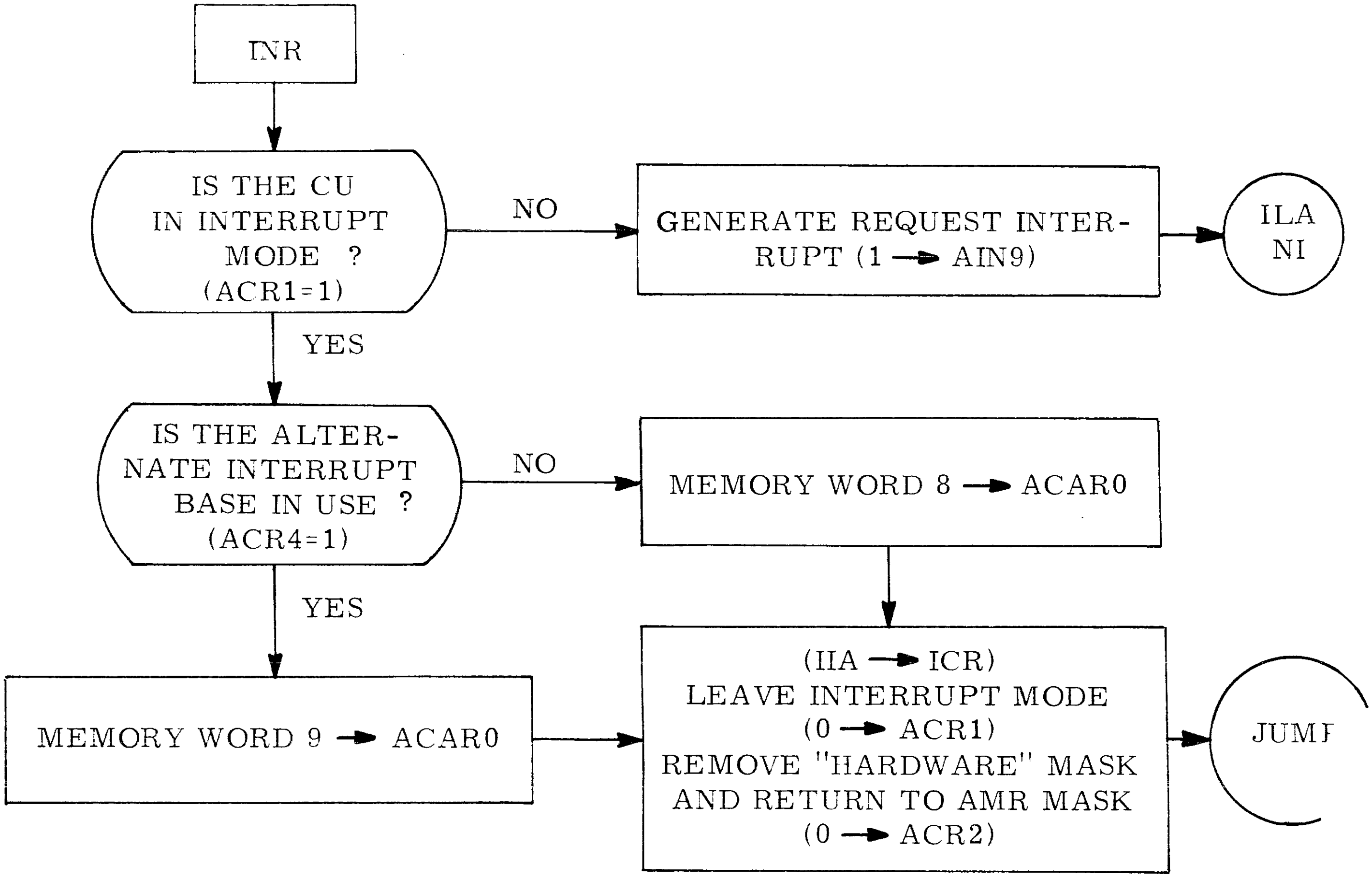
OPERATION: Return to Normal Processing after Interrupt

AIR:



DESCRIPTION: This instruction causes the CU to set pertinent registers and controls to their respective states, as stored in memory relative to the interrupt base address. These states are not necessarily identical to those present prior to the interrupt, since the nature of the interrupt may require the associated interrupt program to modify certain of these data.

FLOW CHART:


INR
IS THE CU IN INTERRUPT MODE ? (ACR1=1)
NO
GENERATE REQUEST INTERRUPT (1 → AIN9)
ILA NI
YES
IS THE ALTERNATE INTERRUPT BASE IN USE ? (ACR4=1)
NO
MEMORY WORD 8 → ACAR0
YES
MEMORY WORD 9 → ACAR0
(IIA → ICR)
LEAVE INTERRUPT MODE
(0 → ACR1)
REMOVE "HARDWARE" MASK AND RETURN TO AMR MASK
(0 → ACR2)
JUMP

MNEMONIC CODE: JUMP

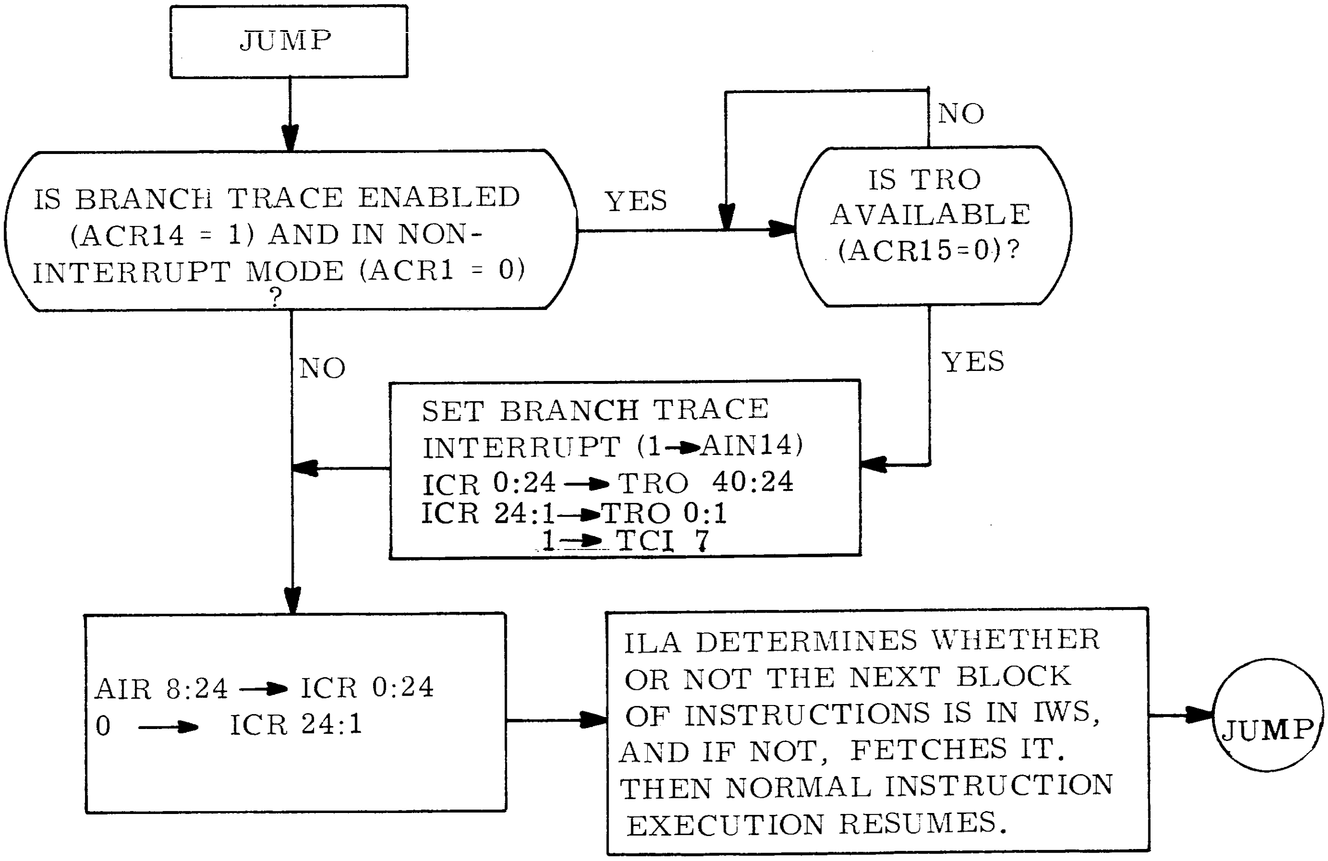
OPERATION: Jump to Address in ADR Field

AIR:

17	ACARX	ADR
0 4	5 7	8 31

DESCRIPTION: This instruction causes the CU to execute a jump to another part of the instruction stream. The last eight bits of the address field of the instruction (24:8) may be modified modulo 256 by ACAR indexing. The result, 24 bits long, is transferred to the most significant bits of the ICR. The least significant bit of the ICR is set to "zero". If this instruction is executed and the ICR is updated and other branch trace conditions are met, then an interrupt will occur and program control will go to the interrupt program.

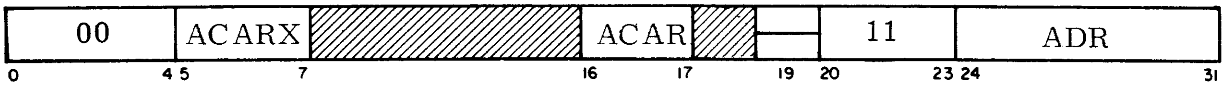
FLOW CHART:


JUMP
IS BRANCH TRACE ENABLED (ACR14 = 1) AND IN NON-INTERRUPT MODE (ACR1 = 0) ?
YES
NO
IS TRO AVAILABLE (ACR15=0)?
NO
YES
SET BRANCH TRACE INTERRUPT (1→AIN14)
ICR 0:24→TRO 40:24
ICR 24:1→TRO 0:1
1→ TCI 7
AIR 8:24 → ICR 0:24
0 → ICR 24:1
ILA DETERMINES WHETHER OR NOT THE NEXT BLOCK OF INSTRUCTIONS IS IN IWS, AND IF NOT, FETCHES IT. THEN NORMAL INSTRUCTION EXECUTION RESUMES.
JUMP

MNEMONIC CODE: LDC

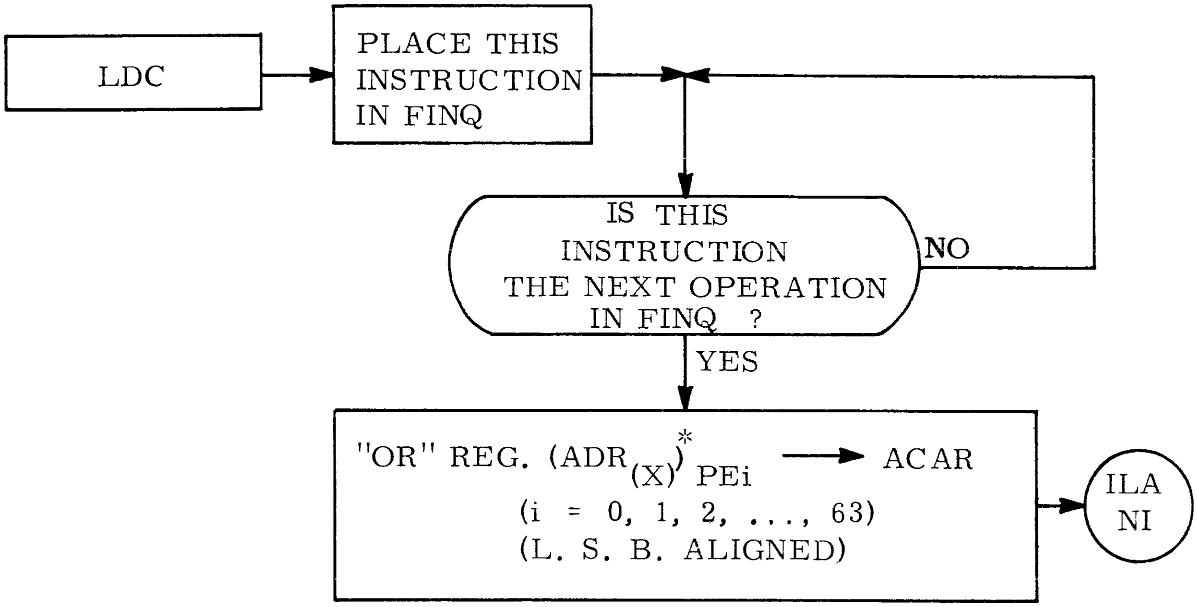
OPERATION: Transfer Specified PE Register to ACAR

AIR:



DESCRIPTION: This instruction causes FINQ to empty before it or another ADVAST instruction is executed. When FINST becomes idle, the corresponding registers addressed in all enabled PEs in the quadrant are ORed together and replace the contents of the specified ACAR. The bit positions in ADR 2:5 correspond to RGA, RGB, RGX, RGS, and RGR respectively. The register code can be modified by ACAR indexing.

FLOW CHART:

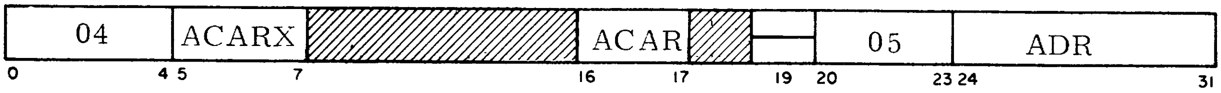

LDC
PLACE THIS INSTRUCTION IN FINQ
IS THIS INSTRUCTION THE NEXT OPERATION IN FINQ ?
NO
YES
"OR" REG. (ADR(X))* PEi → ACAR
(i = 0, 1, 2, . . . , 63)
(L. S. B. ALIGNED)
ILA NI

*Specification of PE register is limited to RGA, RGB, RGX, RGS, or RGR.

MNEMONIC CODE: LDL

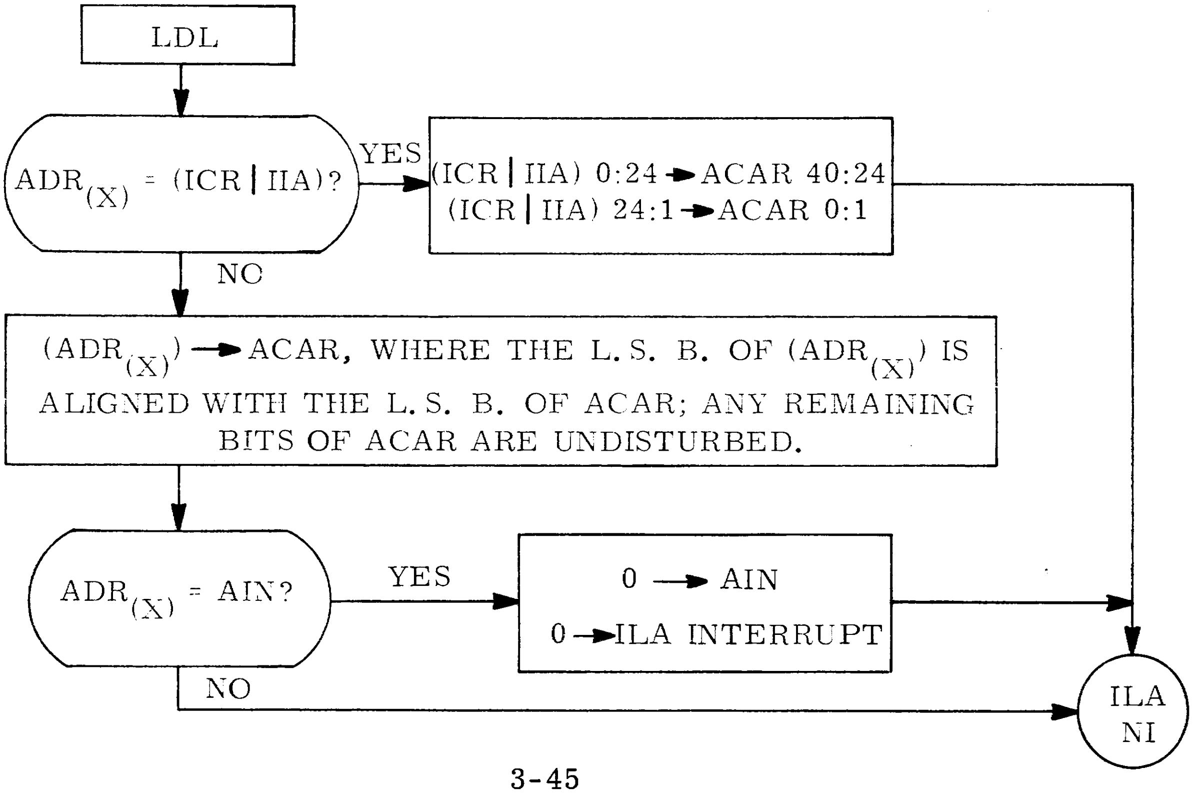
OPERATION: Load from Local Address

AIR:



DESCRIPTION: This instruction transfers the operand in the local address to the specified ACAR. The local address field is indexable and only the following addresses are permitted: ADB 00-77, ACR, AIN, ALR, AMR, AC0-3, ICR, MC0-2, IIA, TRO, TRI, ACU, and PEM (ARE). With the exception of ICR and IIA, all of these registers have their least significant bit aligned with the least significant bit of the ACAR. The ICR and the IIA have their second least significant bit aligned with bit 63 of the ACAR. The least significant bit of the ICR and IIA is transferred to the most significant bit of the ACAR.

FLOW CHART:


LDL
ADR(X) = (ICR | IIA)?
YES
(ICR | IIA) 0:24 → ACAR 40:24
(ICR | IIA) 24:1 → ACAR 0:1
NO
(ADR(X)) → ACAR, WHERE THE L. S. B. OF (ADR(X)) IS ALIGNED WITH THE L. S. B. OF ACAR; ANY REMAINING BITS OF ACAR ARE UNDISTURBED.
ADR(X) = AIN?
YES
0 → AIN
0 → ILA INTERRUPT
NO
ILA
NI

MNEMONIC CODE: LEAD (O|Z)

OPERATION: Find Leading (One|Zero) in ACAR

AIR:

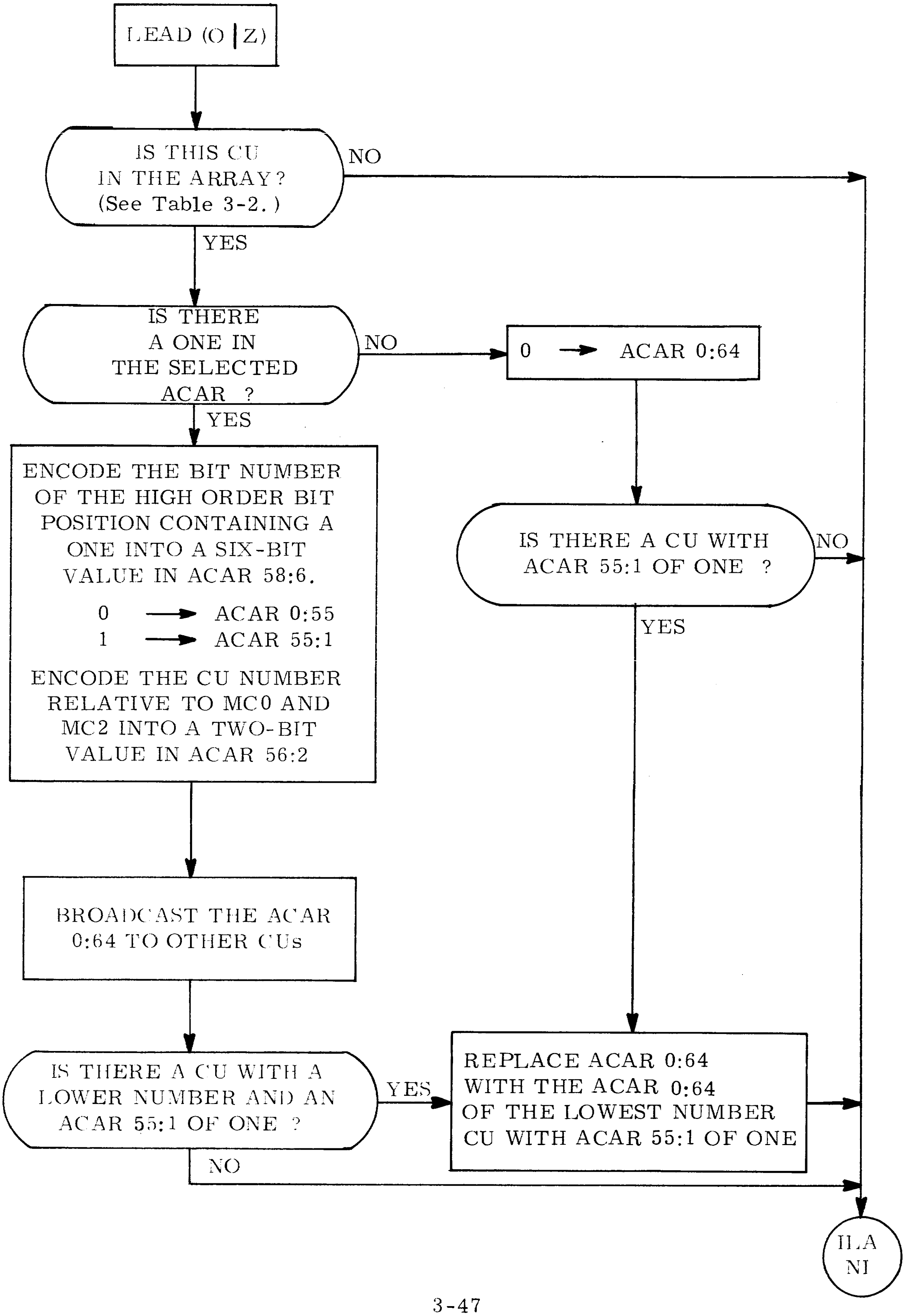
LEADO	02	▨	ACAR	G/L	—	01	▨
	0	4	16	17 18	19 20	23	

LEADZ	02	▨	ACAR	G/L	—	00	▨
	0	4	16	17 18	19 20	23	

DESCRIPTION: This instruction causes all CUs specified by MC0 to be synchronized at the beginning of the instruction. All CUs specified by MC2 relative to MC0 detect the leading "one | zero" in the specified ACAR and notify each other of their findings. All CUs store the information presented by the lowest numbered CU which detects a leading "one | zero". The information is stored in the specified ACAR, after it has been reset. If a "one | zero" is found, the CU number (relative to MC0 and MC2) is in bits 56:2 while the encoded bit position is in bits 58:6. Should no "one | zero" be detected by any CU, all CUs store a "zero" in bit 55. Should a 'one zero" be detected by any CU, all CUs store a "one" in bit 55. (Bit 0 is the leading bit of the ACAR.)

In a single quadrant array, the CU records its own information.

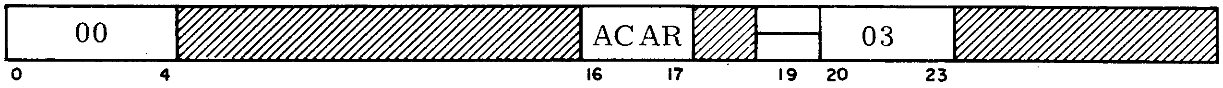
FLOW CHART: See next page.


LEAD (O | Z)
IS THIS CU IN THE ARRAY? (See Table 3-2.)
NO
YES
IS THERE A ONE IN THE SELECTED ACAR ?
NO
0 → ACAR 0:64
YES
ENCODE THE BIT NUMBER OF THE HIGH ORDER BIT POSITION CONTAINING A ONE INTO A SIX-BIT VALUE IN ACAR 58:6.
0 → ACAR 0:55
1 → ACAR 55:1
ENCODE THE CU NUMBER RELATIVE TO MC0 AND MC2 INTO A TWO-BIT VALUE IN ACAR 56:2
IS THERE A CU WITH ACAR 55:1 OF ONE ?
NO
YES
BROADCAST THE ACAR 0:64 TO OTHER CUs
IS THERE A CU WITH A LOWER NUMBER AND AN ACAR 55:1 OF ONE ?
YES
REPLACE ACAR 0:64 WITH THE ACAR 0:64 OF THE LOWEST NUMBER CU WITH ACAR 55:1 OF ONE
NO
II.A NI

MNEMONIC CODE: LIT

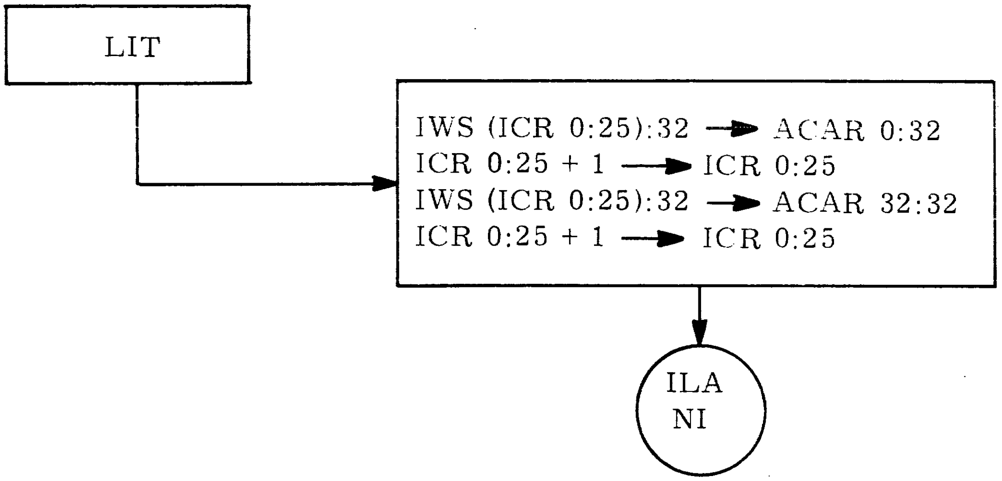
OPERATION: Store Next 64 Bits in ACAR

AIR:



DESCRIPTION: The 64-bit literal value following the LIT instruction is stored in the specified ACAR. The next instruction to be executed is located in the 32 bits following the literal value.

FLOW CHART:



MNEMONIC CODE: LOAD(X)

OPERATION: Word Fetch (RGX Indexed) from PE Memory to CU Local Memory

AIR:

LOAD	06	ACARX	▨	ACAR	G/L	—	00	ADR
	0 4	5 7		16 17	18	19	20 23	24 31

LOADX	06	ACARX	▨	ACAR	G/L	—	01	ADR
	0 4	5 7		16 17	18	19	20 23	24 31

DESCRIPTION: This instruction reads a word from PE memory and stores it in the CU local memory. The PE memory address is indicated in the specified ACAR 40:24. The local memory address is given in the ADR field. It is indexable and only the following local addresses are permitted: ADB 00-77, AIN, ALR, AMR, AC0-3, ICR, MC0-2, IIA, and TRO (see Table 5-1, page 5-6). When the local address is TRO, ACR(15) and TCI(04) are set. The instruction is placed in FINQ to await execution at FINST. The ALR register is used to hold the unfilled local address until the operation is executed at FINST.

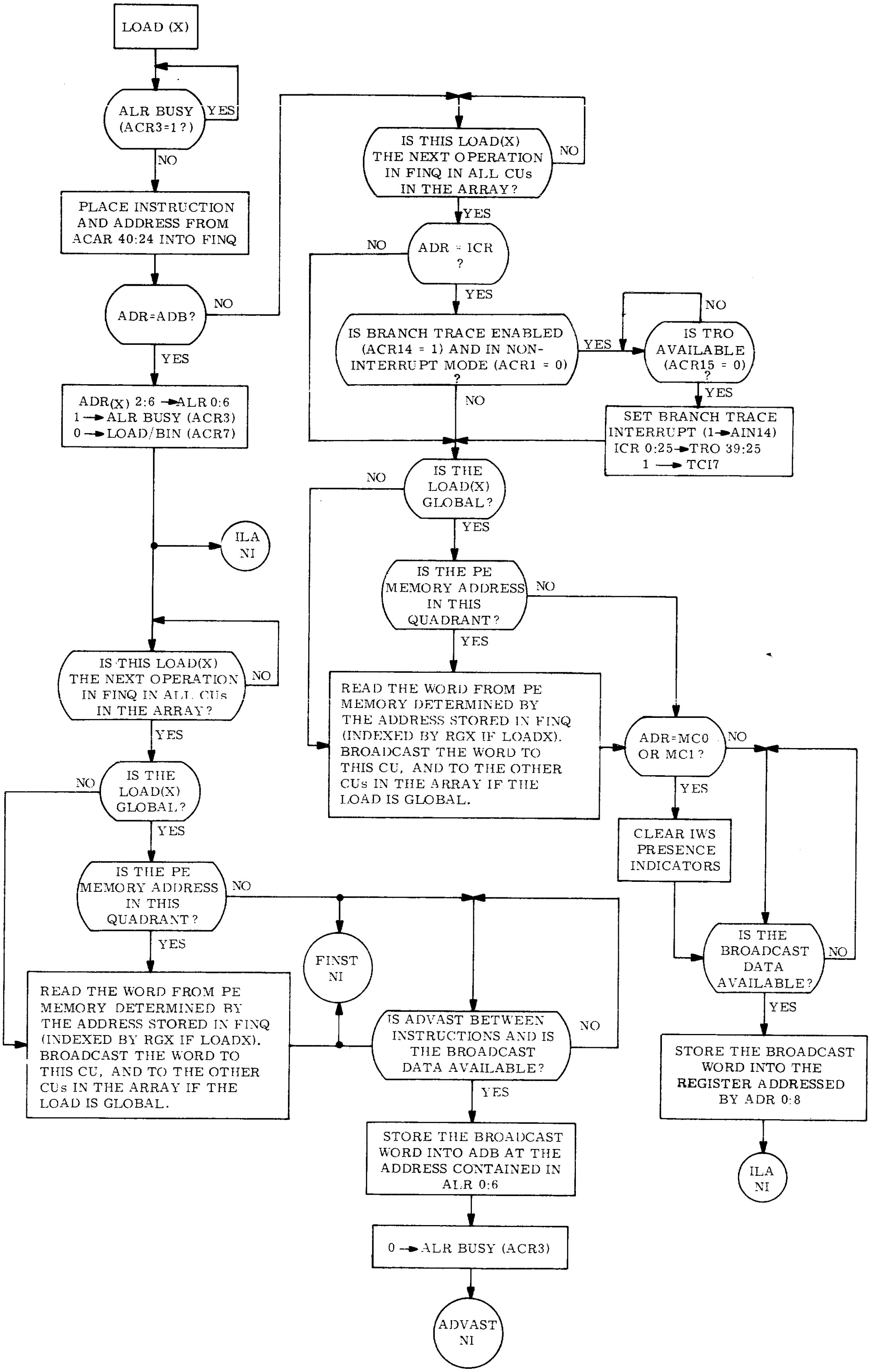
LOAD causes the ADVAST station to stall if a previous BIN/BINX or LOAD/LOADX has not been completed. If ADR references ADB then ADR(X) 2:6 will be stored in ALR 0:6 and a "one" will be set in ACR3 (ALR Busy). The word read from the PE memory will be determined by the address stored in FINQ (indexed by RGX, if LOADX) and will be broadcast to the proper CUs. If the instruction is local then only this CU will store the broadcast data into the ADB as specified by ALR 0:6. If the instruction is global then all CUs specified by MC0 will store the broadcast data into the ADB as specified by ALR 0:6.

If ADR does not reference ADB then the following will occur. The word read from the PE memory will be determined by the address stored in FINQ (indexed by RGX) and will be broadcast to the proper CUs. If the ADR specifies the configuration control registers MC0 or MC1 then the IWS presence indicators will be cleared to indicate there is no valid information in the ILA. If the instruction is local then only this CU will store the broadcast data into the register specified by ADR 0:8. If the instruction is global then all CUs specified by MC0 will store the broadcast data into their registers specified by ADR 0:8.

If this instruction is executed and the ICR is updated and other branch trace conditions are met, then an interrupt will occur and program control will proceed to interrupt processing.

When this instruction is executed and any address other than the ADB is specified, then the FINST queue will be emptied.

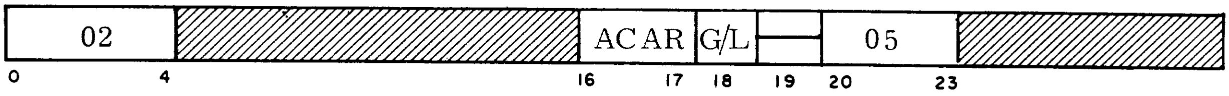
FLOW CHART: See next page.


LOAD (X)
ALR BUSY (ACR3=1 ?)
YES
NO
PLACE INSTRUCTION AND ADDRESS FROM ACAR 40:24 INTO FINQ
ADR=ADB?
NO
YES
ADR(X) 2:6 → ALR 0:6
1 → ALR BUSY (ACR3)
0 → LOAD/BIN (ACR7)
ILA NI
IS THIS LOAD(X) THE NEXT OPERATION IN FINQ IN ALL CUs IN THE ARRAY?
NO
YES
IS THE LOAD(X) GLOBAL?
NO
YES
IS THE PE MEMORY ADDRESS IN THIS QUADRANT?
NO
YES
READ THE WORD FROM PE MEMORY DETERMINED BY THE ADDRESS STORED IN FINQ (INDEXED BY RGX IF LOADX). BROADCAST THE WORD TO THIS CU, AND TO THE OTHER CUs IN THE ARRAY IF THE LOAD IS GLOBAL.
FINST NI
IS ADVAST BETWEEN INSTRUCTIONS AND IS THE BROADCAST DATA AVAILABLE?
NO
YES
STORE THE BROADCAST WORD INTO ADB AT THE ADDRESS CONTAINED IN ALR 0:6
0 → ALR BUSY (ACR3)
ADVAST NI
IS THIS LOAD(X) THE NEXT OPERATION IN FINQ IN ALL CUs IN THE ARRAY?
NO
YES
ADR = ICR ?
NO
YES
IS BRANCH TRACE ENABLED (ACR14 = 1) AND IN NON-INTERRUPT MODE (ACR1 = 0) ?
YES
NO
IS TRO AVAILABLE (ACR15 = 0) ?
NO
YES
SET BRANCH TRACE INTERRUPT (1 → AIN14) ICR 0:25 → TRO 39:25 1 → TCI7
IS THE LOAD(X) GLOBAL?
NO
YES
IS THE PE MEMORY ADDRESS IN THIS QUADRANT?
NO
YES
READ THE WORD FROM PE MEMORY DETERMINED BY THE ADDRESS STORED IN FINQ (INDEXED BY RGX IF LOADX). BROADCAST THE WORD TO THIS CU, AND TO THE OTHER CUs IN THE ARRAY IF THE LOAD IS GLOBAL.
ADR=MC0 OR MC1 ?
NO
YES
CLEAR IWS PRESENCE INDICATORS
IS THE BROADCAST DATA AVAILABLE?
NO
YES
STORE THE BROADCAST WORD INTO THE REGISTER ADDRESSED BY ADR 0:8
ILA NI

MNEMONIC CODE: ORAC

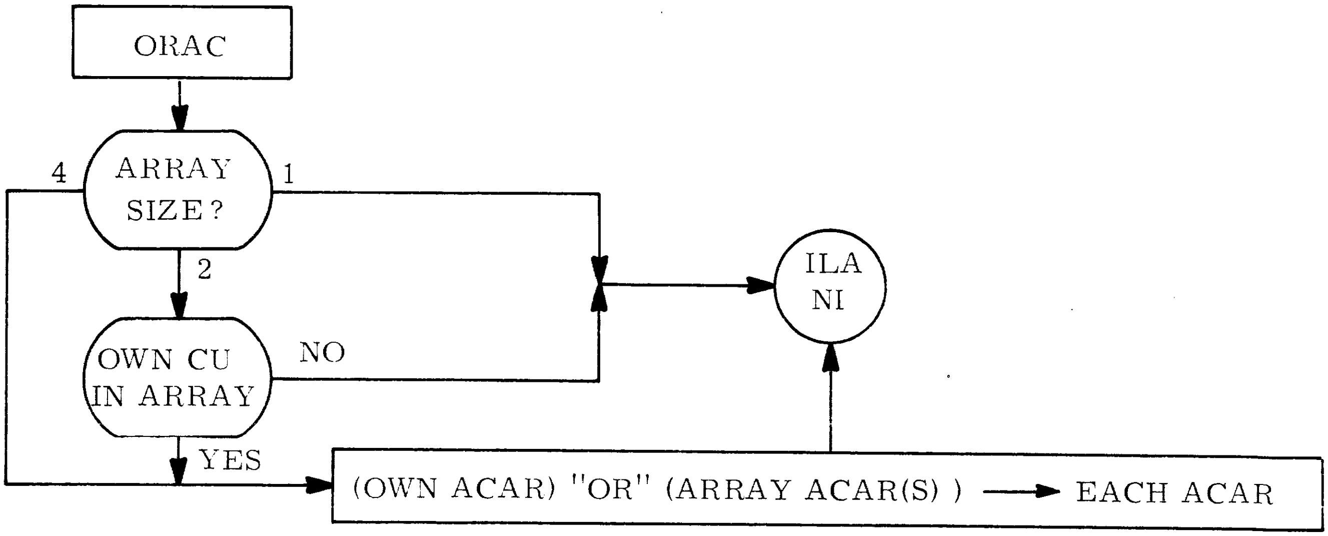
OPERATION: Inclusive OR of Operand in ACAR of All CUs Executing the Instruction

AIR:



DESCRIPTION: This instruction causes all CUs specified by MC0 to be synchronized at the beginning of the instruction. Each CU transmits its specified ACAR to the other CUs in the array. In turn each CU in the array receives the operands from the other CUs, performs an INCLUSIVE-OR of all the operands of the CUs specified by MC2 relative to MC0, and stores the result in its ACAR. This instruction is a NO-OP in a single-quadrant array.

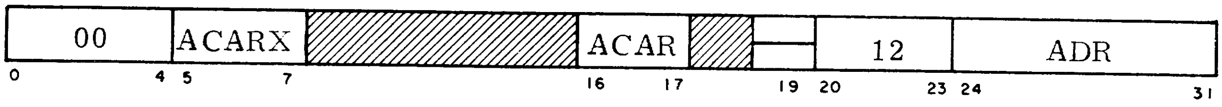
FLOW CHART:



MNEMONIC CODE: SETC

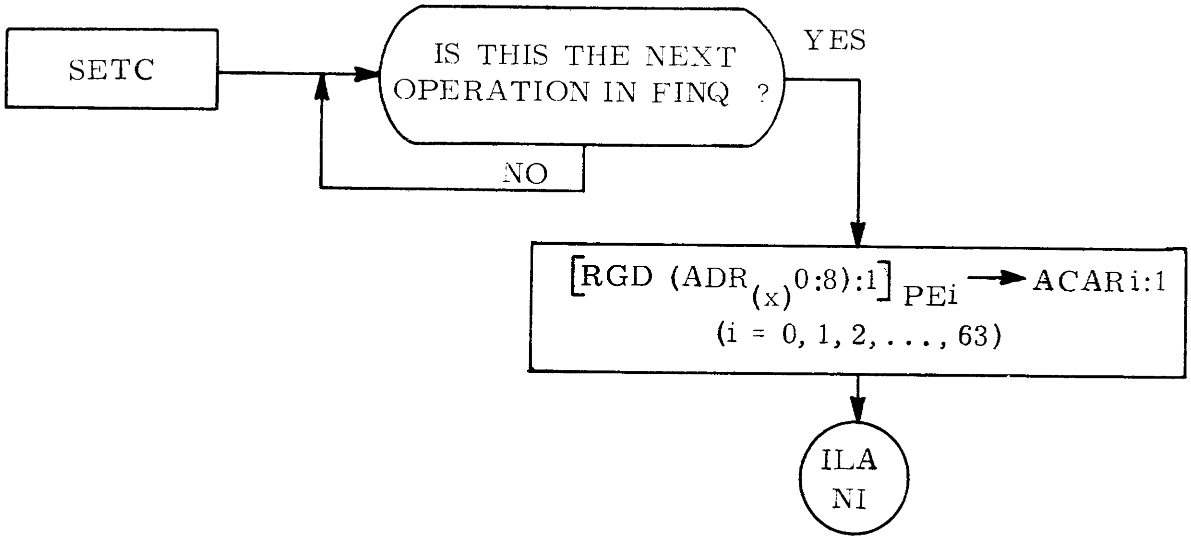
OPERATION: Specified Mode Bit from PEs to ACAR

AIR:



DESCRIPTION: This instruction causes transmission of a particular mode bit from each of the 64 processing elements to the ACAR specified in AIR 16:2. The local address field of the instruction selects the mode bit; this field is indexable. The mode bits correspond to ADR bit positions 0 through 7, for H, G, J, I, E1, E, F1, F respectively. If no ADR bits are set, the result is the logical OR of F and F1. If more than one ADR bit is set, then the results are undefined. SETC causes ADVAST to stop processing instructions from IWS until FINQ is empty and the mode bits are returned by the PE and stored in the ACAR.

FLOW CHART:

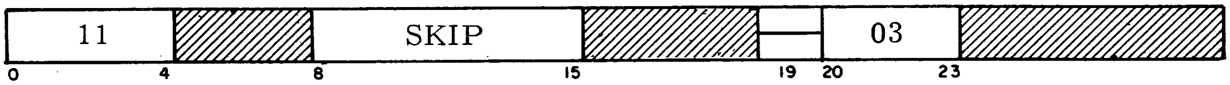


SETC
IS THIS THE NEXT OPERATION IN FINQ ?
YES
NO
[RGD (ADR(x)0:8):1] PEi → ACARi:1
(i = 0, 1, 2, . . . , 63)
ILA
NI

MNEMONIC CODE: SKIP

OPERATION: Skip Forward/Backward

AIR:

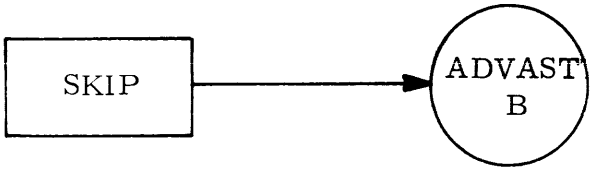


DESCRIPTION: This instruction causes the CU to execute an unconditional skip to another part of the instruction stream. The jump address is derived by modifying the ICR with the SKIP field of the instruction (after stepping the ICR). Bit 0 of the SKIP field is the sign bit ("1" means subtract, "0" means add) and bits 1:7 are the magnitude bits of the modifier. Examples of the skip field values show the following effects:

Skip Value	Effect
-1	Infinite loop
0	No operation
+1	Skip next instruction

If this instruction is executed and the branch trace conditions are met, then an interrupt will occur and program control will proceed to interrupt processing.

FLOW CHART:

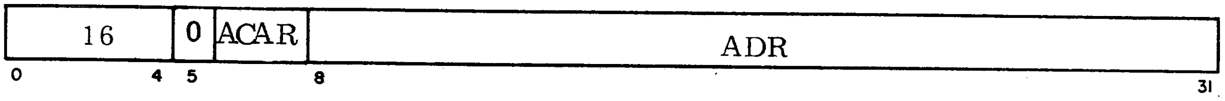


p. 3-63

MNEMONIC CODE: SLIT

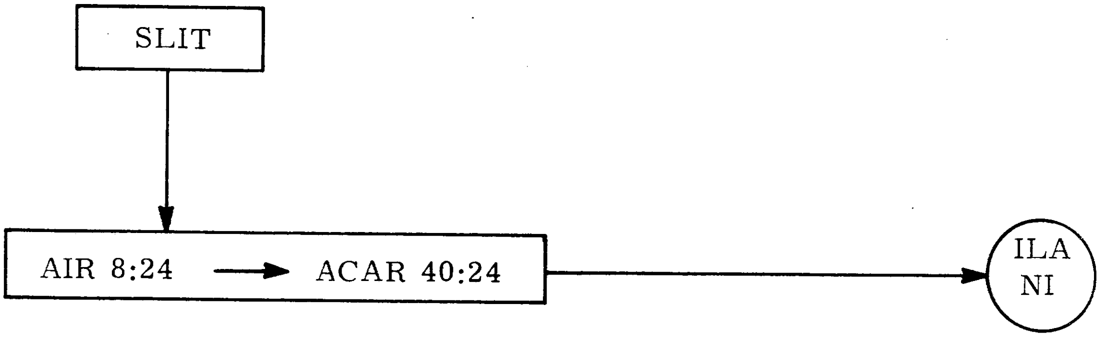
OPERATION: Replace Address Field of ACAR

AIR:



DESCRIPTION: This instruction causes the address field of the specified ACAR (bits 40:24) to be replaced by the address field of the instruction (bits 8:24). Bit 5 of the instruction must be "zero", and bits 6:2 specify the ACAR. ACAR bits 0:40 are not disturbed.

FLOW CHART:



MNEMONIC CODE: STL

OPERATION: Store ACAR in Local Address

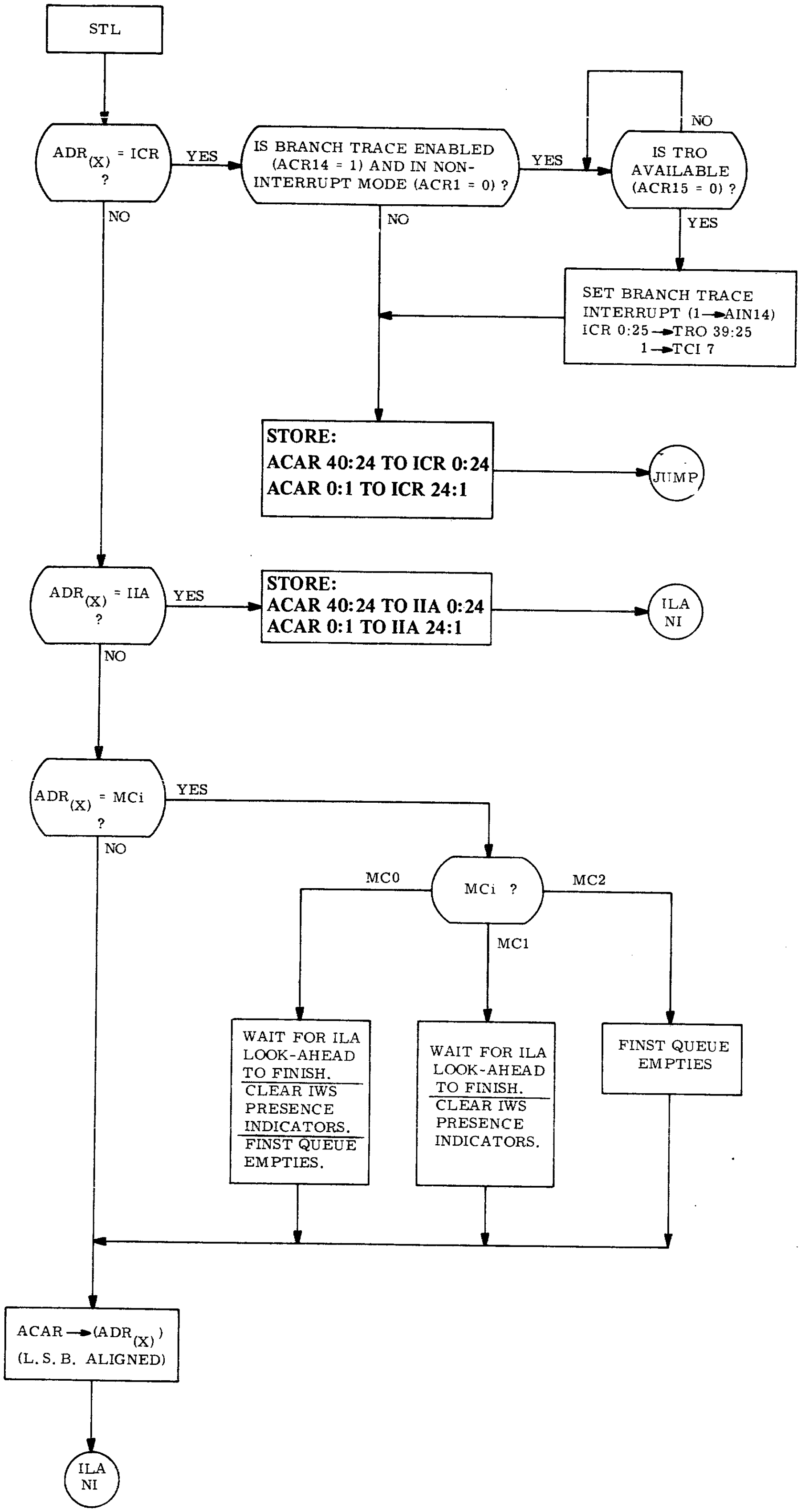
AIR:

04	ACARX	▨	ACAR	▨	—	04	ADR
0 4	5 7		16 17		19	20 23	24 31

DESCRIPTION: This instruction transfers the contents of the specified ACAR to the location specified by the local address field. The local address field is indexable. Only the following addresses are permitted: ADB 00-77, AIN, ALR, AMR, AC0-3, ICR, MC0-2, IIA, and TRO. With the exception of ICR and IIA, all of these addresses have their least significant bit aligned with the least significant bit of the ACAR. ICR or IIA has its second least significant bit aligned with bit 63 of the ACAR. The most significant bit of the ACAR is transferred to the least significant bit of ICR or IIA. When this instruction loads the ICR, normal ICR updating is inhibited. Normal instruction execution resumes at the new ICR location. Loading MC0 or MC1 causes the IWS to be cleared. Loading MC0 or MC2 causes FINQ to empty. All bits not replaced by local memory will be reset to zero.

If this instruction is executed and the ICR is updated and other branch trace conditions are met, then an interrupt will occur and program control will proceed to interrupt processing.

FLOW CHART: See next page.


STL
ADR$_{(X)}$ = ICR ?
YES
NO
IS BRANCH TRACE ENABLED (ACR14 = 1) AND IN NON-INTERRUPT MODE (ACR1 = 0) ?
YES
NO
IS TRO AVAILABLE (ACR15 = 0) ?
NO
YES
SET BRANCH TRACE INTERRUPT (1 → AIN14)
ICR 0:25 → TRO 39:25
1 → TCI 7
STORE:
ACAR 40:24 TO ICR 0:24
ACAR 0:1 TO ICR 24:1
JUMP
ADR$_{(X)}$ = IIA ?
YES
NO
STORE:
ACAR 40:24 TO IIA 0:24
ACAR 0:1 TO IIA 24:1
ILA NI
ADR$_{(X)}$ = MCi ?
YES
NO
MCi ?
MC0
MC1
MC2
WAIT FOR ILA LOOK-AHEAD TO FINISH.
CLEAR IWS PRESENCE INDICATORS.
FINST QUEUE EMPTIES.
WAIT FOR ILA LOOK-AHEAD TO FINISH.
CLEAR IWS PRESENCE INDICATORS.
FINST QUEUE EMPTIES
ACAR → (ADR$_{(X)}$)
(L.S.B. ALIGNED)
ILA NI

MNEMONIC CODE: STORE(X)

OPERATION: Store from Local Address into Specified PE Memory Location (RGX Indexed)

AIR:

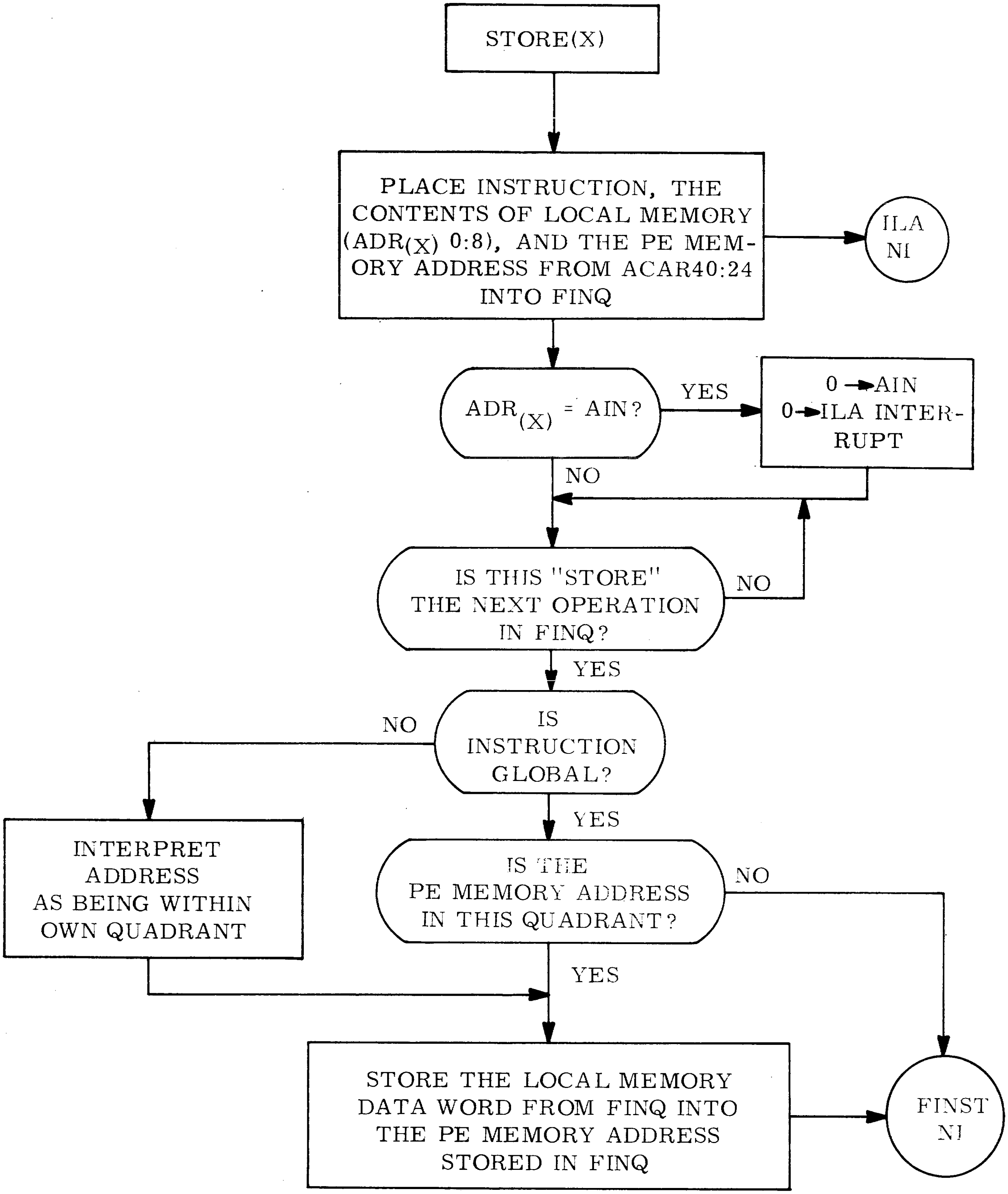
STORE	06	ACARX	▨	ACAR	G/L	—	02	ADR
	0 4	5 7		16 17	18	19	20 23	24 31
STOREX	06	ACARX	▨	ACAR	G/L	—	03	ADR
	0 4	5 7		16 17	18	19	20 23	24 31

DESCRIPTION: This instruction stores the operand specified by the local address into the PE memory location specified by the least significant 24 bits of the specified ACAR (indexed by PE register RGX if STOREX). The local address (ADR field) is indexable. Only the following addresses are permitted: ADB 00-77, ACR, AIN, ALR, AMR, AC0-3, ICR, MC0-2, IIA, TRO, TRI, ACU, and PEM (ARE). (See Table 5-1, page 5-6.) The recipient address is interpreted by ACAR 56:2 and the settings of MC0 and MC2.

The least significant bit of all operands, except ICR or IIA, is aligned with the least significant bit of the PE memory word. The second least significant bit of ICR or IIA is aligned with the least significant bit of the word to be stored; and the least significant bit of ICR or IIA is stored into the most significant bit of the word. All bits in the PE memory word that are not replaced by the CU local memory word are set to "zero".

The word is stored by the CU whose PE memory contains the specified address. This instruction is placed in FINQ to be executed in turn at FINST. The data from the specified local register is found in the FINQ slot next after the instruction.

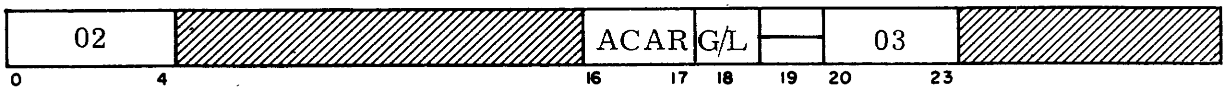
FLOW CHART: See next page.


STORE(X)
PLACE INSTRUCTION, THE CONTENTS OF LOCAL MEMORY (ADR(X) 0:8), AND THE PE MEMORY ADDRESS FROM ACAR40:24 INTO FINQ
ILA NI
ADR(X) = AIN?
YES
0 → AIN
0 → ILA INTERRUPT
NO
IS THIS "STORE" THE NEXT OPERATION IN FINQ?
NO
YES
IS INSTRUCTION GLOBAL?
NO
YES
INTERPRET ADDRESS AS BEING WITHIN OWN QUADRANT
IS THE PE MEMORY ADDRESS IN THIS QUADRANT?
NO
YES
STORE THE LOCAL MEMORY DATA WORD FROM FINQ INTO THE PE MEMORY ADDRESS STORED IN FINQ
FINST NI

MNEMONIC CODE: TCCW

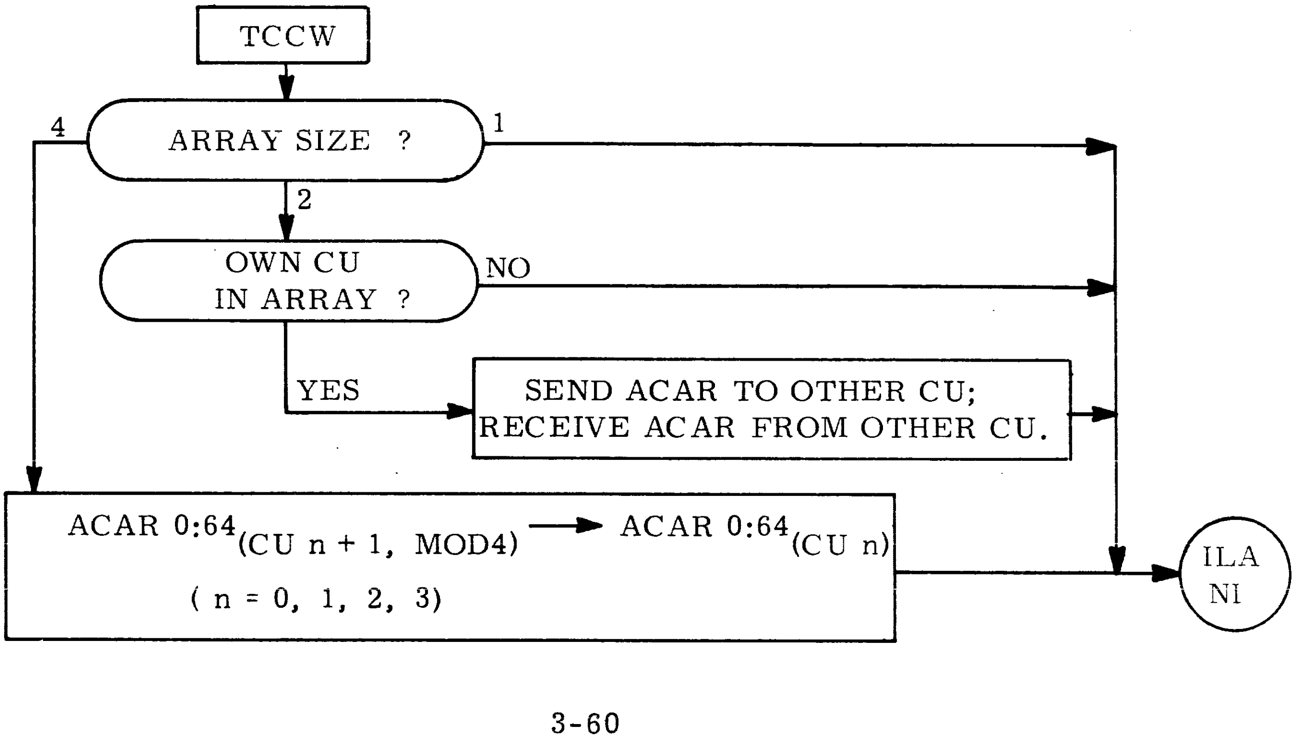
OPERATION: Transmit ACAR Counterclockwise (To Next Lower Numbered CU)

AIR:



DESCRIPTION: This instruction causes all CUs specified by MC0 to be synchronized at the beginning of the instruction. All CUs executing the instructions, as determined by MC2 relative to MC0, transmit the specified ACAR to the corresponding ACAR in the next lower numbered CU. Also, the ACAR of the lowest numbered CU is transmitted to the ACAR of the highest numbered CU. This instruction is a NO-OP in a single quadrant array.

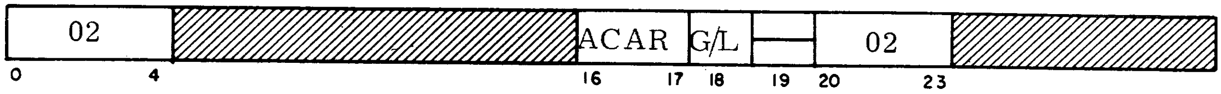
FLOW CHART:



MNEMONIC CODE: TCW

OPERATION: Transmit ACAR Clockwise (To Next Higher Numbered CU)

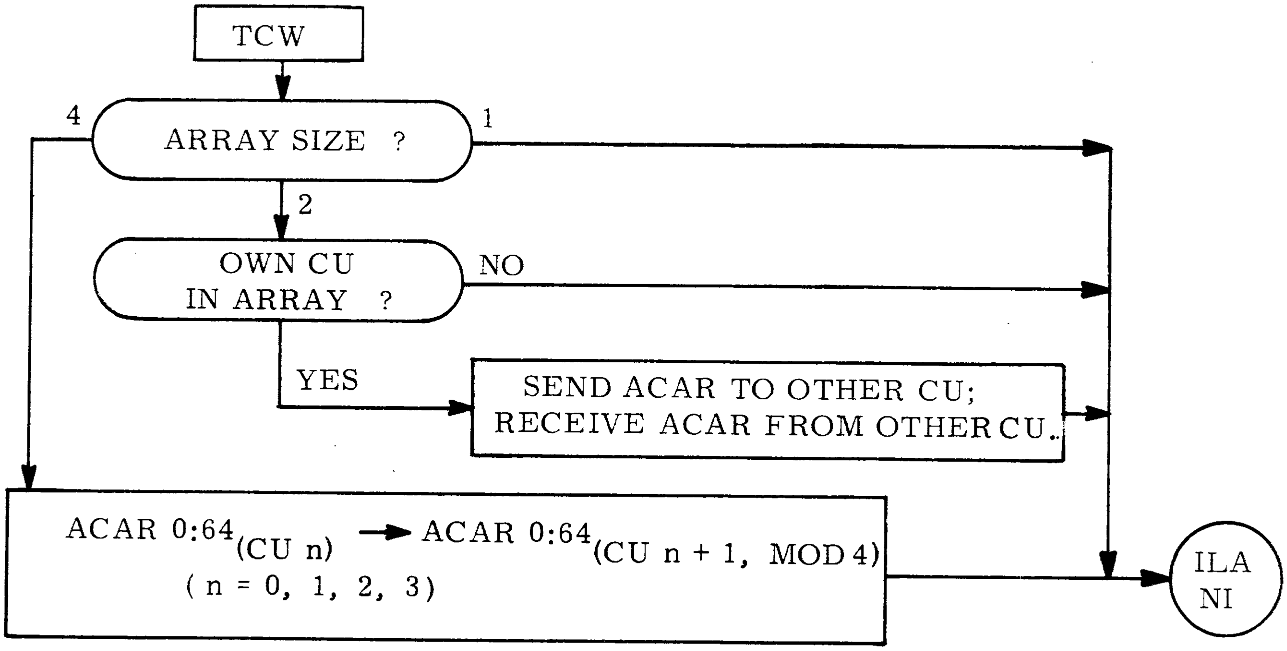
AIR:



DESCRIPTION: This instruction causes all CUs specified by MC0 to be synchronized at the beginning of the instruction. All CUs executing the instructions, as determined by MC2 relative to MC0, transmit the specified ACAR to the corresponding ACAR in the next higher numbered CU. Also, the ACAR of the highest number CU is transmitted to the ACAR of the lowest numbered CU.

In single quadrant array, this instruction is a NO-OP.

FLOW CHART:


TCW
ARRAY SIZE ?
4
1
2
OWN CU IN ARRAY ?
NO
YES
SEND ACAR TO OTHER CU; RECEIVE ACAR FROM OTHER CU.
ACAR 0:64(CU n) → ACAR 0:64(CU n + 1, MOD 4)
(n = 0, 1, 2, 3)
ILA NI

TEST - SKIP INSTRUCTIONS

MNEMONIC CODES: ________ T | F | A

OPERATION: Test and Skip Conditionally

DESCRIPTION: Each of the TEST-SKIP instructions that follow consists of four operation codes. The operation mnemonics are suffixed by the four combinations of T or F and A (that is, TA, T, FA, or F). The true-false flip-flops in the array are sampled resulting in four conditions: all true (TA), any true (T), all false (FA), and any false (F). At the completion of the TEST, the SKIP will be taken if the condition specified in the mnemonic is satisfied.

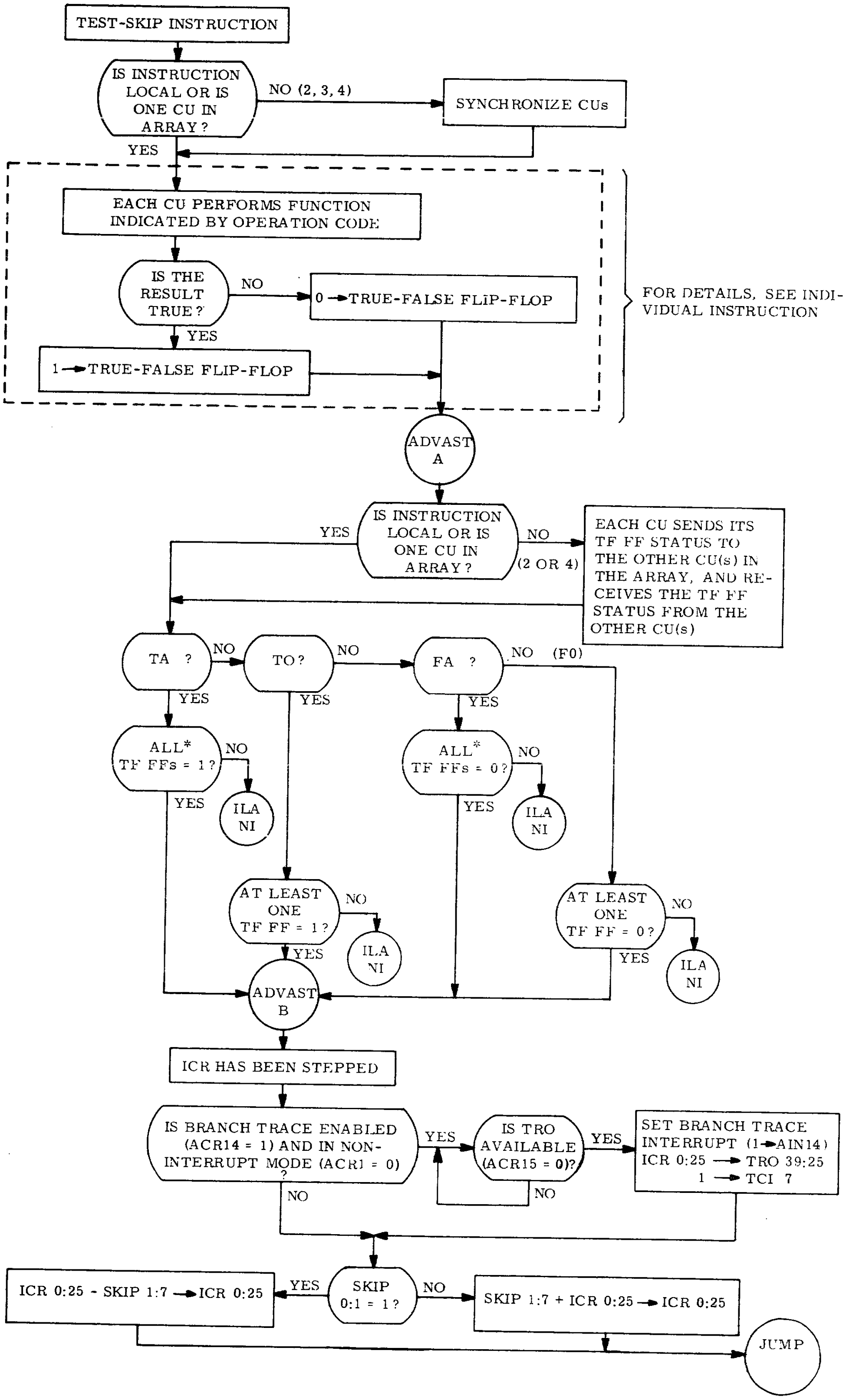
Each of the instructions causes all the CUs specified by MC0 to be synchronized at the beginning of the instruction. At the completion of the test each executing CU, as determined by MC2 relative to MC0, sets its TF flip-flop if the result is true, or resets it if the result is false. The CU sends the status of its TF flip-flop to the other CUs in the array and receives their TF flip-flops. The CU samples all the TF flip-flop lines, and if the condition specified in the op-code is satisfied then the jump is taken. Otherwise, the next instruction in sequence is executed. In single quadrant array, the CU uses its own TF flip-flop for the test. MC0 defines the array to be synchronized at the beginning of the instruction and the array executing the SKIP; MC2 relative to MC0 defines the TF flip-flops which are examined.

The jump address is derived by modifying the ICR with the contents of the SKIP field in the instruction (after stepping the ICR). Bit 0 of the SKIP field is the sign bit of the modifier ("1" means subtract, "0" means add) and bits 1:7 are the magnitude bits of the modifier.

If this instruction is executed and the ICR is updated and other branch trace conditions are met, then an interrupt will occur and program control will proceed to interrupt processing.

A general TEST-SKIP flow chart is shown on the next page. Subsequent pages describe the TEST-SKIP instructions, arranged in alphabetical sequence.

FLOW CHART: See next page.


TEST-SKIP INSTRUCTION
IS INSTRUCTION LOCAL OR IS ONE CU IN ARRAY ?
NO (2, 3, 4)
SYNCHRONIZE CUs
YES
EACH CU PERFORMS FUNCTION INDICATED BY OPERATION CODE
IS THE RESULT TRUE ?
NO
0 → TRUE-FALSE FLIP-FLOP
YES
1 → TRUE-FALSE FLIP-FLOP
FOR DETAILS, SEE INDIVIDUAL INSTRUCTION
ADVAST A
IS INSTRUCTION LOCAL OR IS ONE CU IN ARRAY ?
YES
NO
(2 OR 4)
EACH CU SENDS ITS TF FF STATUS TO THE OTHER CU(s) IN THE ARRAY, AND RECEIVES THE TF FF STATUS FROM THE OTHER CU(s)
TA ?
NO
TO?
NO
FA ?
NO (F0)
YES
YES
YES
ALL* TF FFs = 1?
NO
ILA NI
YES
ALL* TF FFs = 0?
NO
ILA NI
YES
AT LEAST ONE TF FF = 1?
NO
ILA NI
YES
AT LEAST ONE TF FF = 0?
NO
ILA NI
YES
ADVAST B
ICR HAS BEEN STEPPED
IS BRANCH TRACE ENABLED (ACR14 = 1) AND IN NON-INTERRUPT MODE (ACR1 = 0) ?
YES
IS TRO AVAILABLE (ACR15 = 0)?
YES
SET BRANCH TRACE INTERRUPT (1→AIN14) ICR 0:25 → TRO 39:25 1 → TCI 7
NO
NO
ICR 0:25 - SKIP 1:7 → ICR 0:25
YES
SKIP 0:1 = 1 ?
NO
SKIP 1:7 + ICR 0:25 → ICR 0:25
JUMP

*1, 2, or 4 CUs, depending on the array.

TEST - SKIP INSTRUCTIONS (Cont'd)

MNEMONIC CODE: EQLX_ _

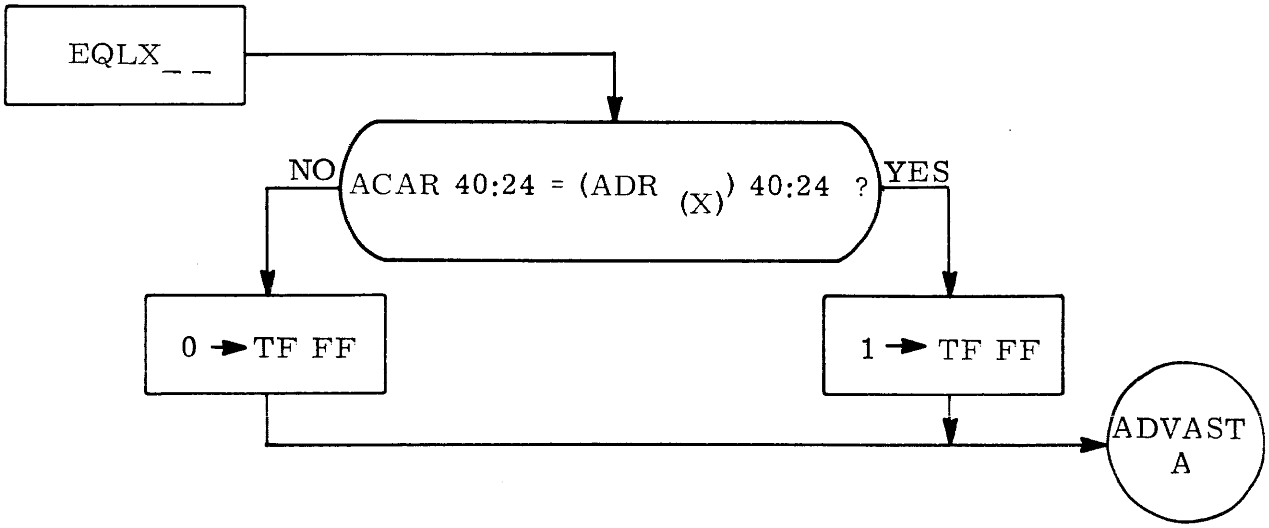
OPERATION: Skip if ACAR 40:24 are Equal to Bits 40:24 in the Operand

AIR:

EQLXTA	14	ACARX	SKIP	ACAR	G/L	—	14	ADR
	0 4	5 7	8 15	16 17	18	19	20–23	24 31
EQLXT	14	ACARX	SKIP	ACAR	G/L	—	15	ADR
	0 4	5 7	8 15	16 17	18	19	20–23	24 31
EQLXFA	14	ACARX	SKIP	ACAR	G/L	—	16	ADR
	0 4	5 7	8 15	16 17	18	19	20–23	24 31
EQLXF	14	ACARX	SKIP	ACAR	G/L	—	17	ADR
	0 4	5 7	8 15	16 17	18	19	20–23	24 31

DESCRIPTION: This instruction determines if bits 40:24 of the specified ACAR are equal to bits 40:24 of the operand in the local address. The local address field is indexable and addresses are limited to the ADB and the four ACARs. (Refer to TEST-SKIP for further details.)

FLOW CHART: See TEST-SKIP instruction.



p. 3-63

TEST - SKIP INSTRUCTIONS (Cont'd)

MNEMONIC CODE: GRTR_ _

OPERATION: Skip if ACAR 40:24 are Greater Than Bits 40:24 of the Operand

AIR:

GRTRTA

15	ACARX	SKIP	ACAR	G/L	—	00	ADR
0 4	5 7	8 15	16 17	18	19	20–23	24 31

GRTRT

15	ACARX	SKIP	ACAR	G/L	—	01	ADR
0 4	5 7	8 15	16 17	18	19	20–23	24 31

GRTRFA

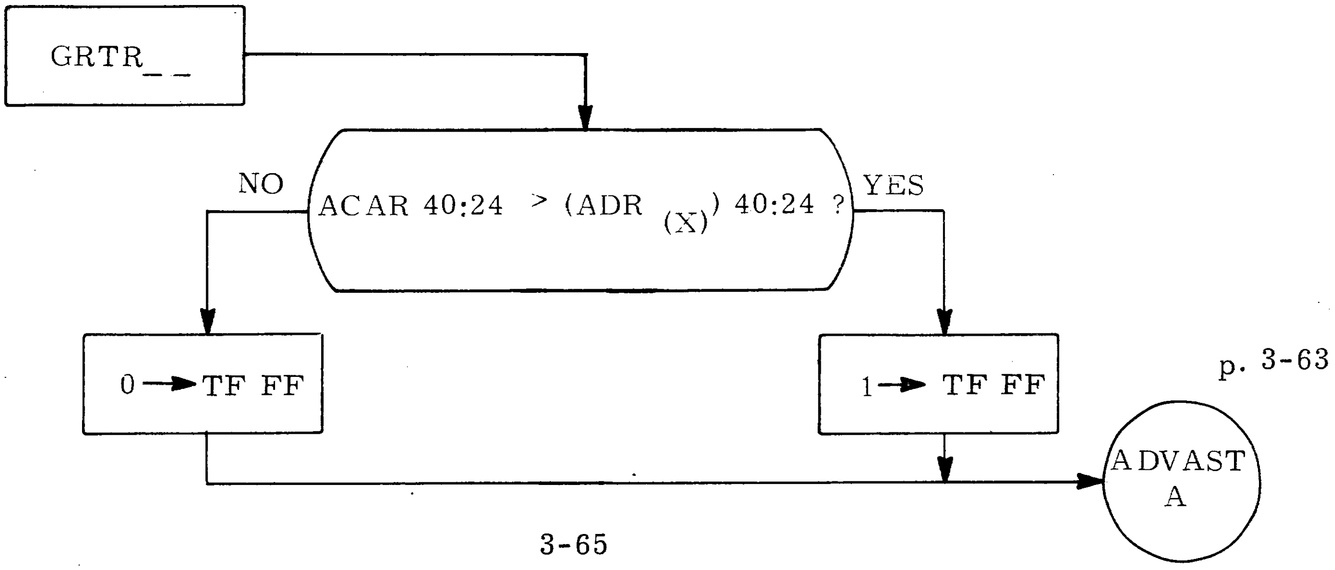
15	ACARX	SKIP	ACAR	G/L	—	02	ADR
0 4	5 7	8 15	16 17	18	19	20–23	24 31

GRTRF

15	ACARX	SKIP	ACAR	G/L	—	03	ADR
0 4	5 7	8 15	16 17	18	19	20–23	24 31

DESCRIPTION: This instruction determines if bits 40:24 of the specified ACAR are greater than bits 40:24 of the operand in the local address. The local address field is indexable and addresses are limited to the ADB and the four ACARs. (Refer to TEST-SKIP for further details.)

FLOW CHART: See TEST-SKIP instruction.



TEST - SKIP INSTRUCTIONS (Cont'd)

MNEMONIC CODE: LESS _ _

OPERATION: Skip if ACAR 40:24 are Less Than Bits 40:24 of the Operand

AIR:

LESSTA

15	ACARX	SKIP	ACAR	G/L	—	04	ADR
0 4	5 7	8 15	16 17	18	19	20–23	24 31

LESST

15	ACARX	SKIP	ACAR	G/L	—	05	ADR
0 4	5 7	8 15	16 17	18	19	20–23	24 31

LESSFA

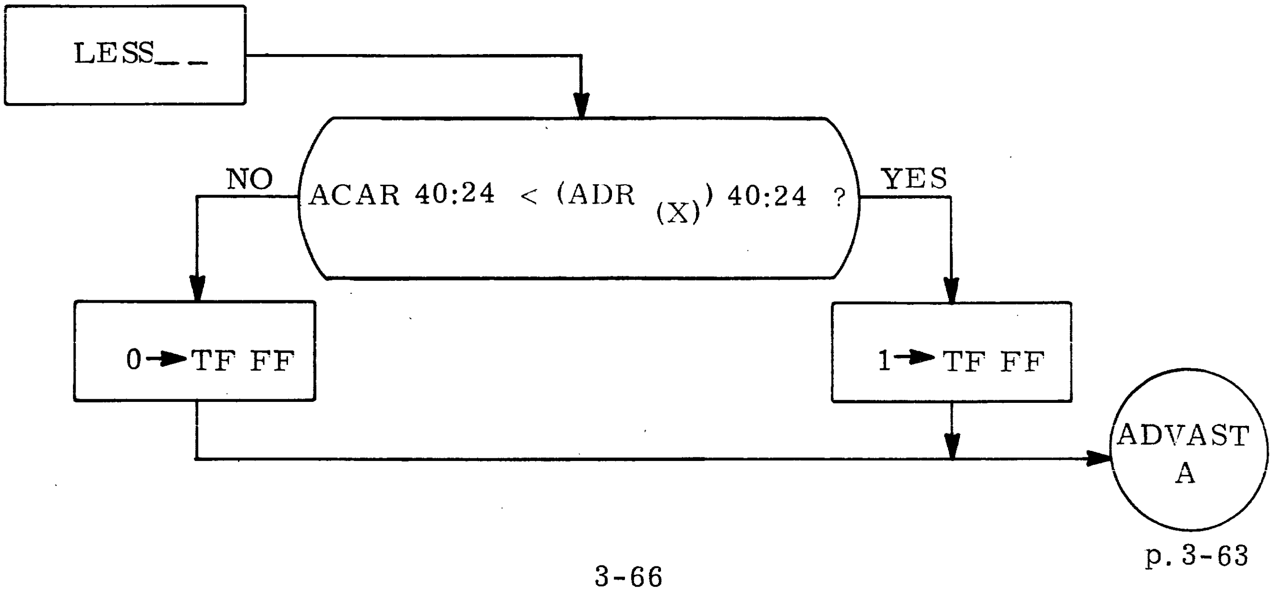
15	ACARX	SKIP	ACAR	G/L	—	06	ADR
0 4	5 7	8 15	16 17	18	19	20–23	24 31

LESSF

15	ACARX	SKIP	ACAR	G/L	—	07	ADR
0 4	5 7	8 15	16 17	18	19	20–23	24 31

DESCRIPTION: This instruction determines if bits 40:24 of the specified ACAR are less than bits 40:24 of the operand in the local address. The local address field is indexable and addresses are limited to the ADB and the four ACARs. (Refer to TEST-SKIP for further details.)

FLOW CHART: See TEST-SKIP instruction.



TEST - SKIP INSTRUCTIONS (Cont'd)

MNEMONIC CODE: ONES _ _

OPERATION: Skip if ACAR 0:64 are All Ones

AIR:

ONESTA

10	▨	SKIP	ACAR	G/L	—	04	▨
0	4	8	15 16 17	18	19	20—23	

ONEST

10	▨	SKIP	ACAR	G/L	—	05	▨
0	4	8	15 16 17	18	19	20—23	

ONESFA

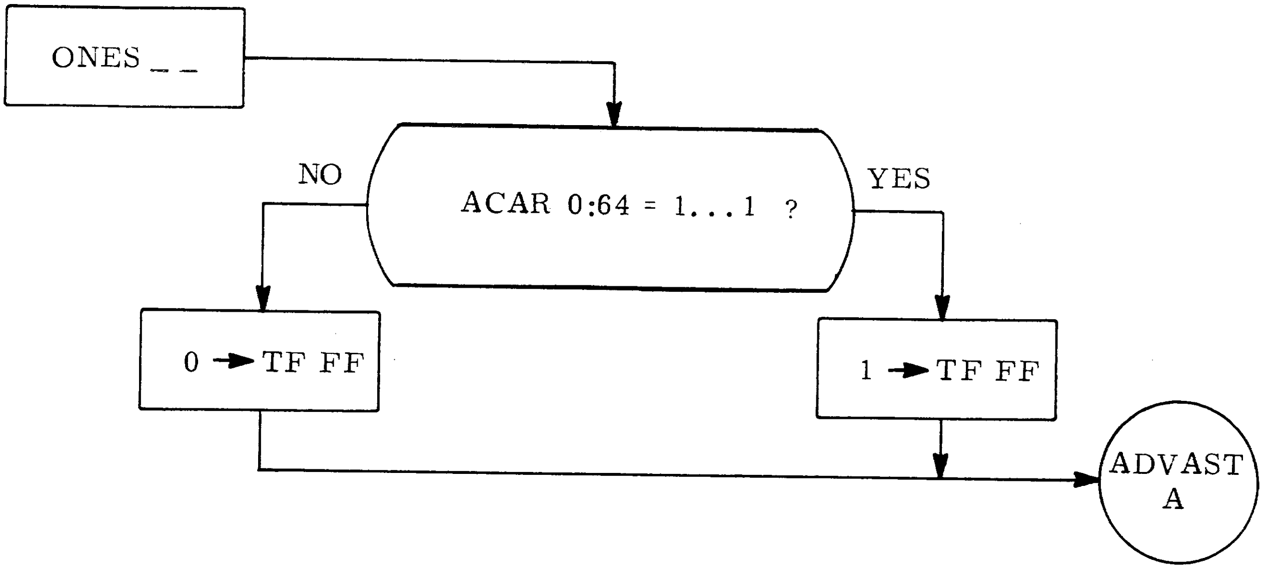
10	▨	SKIP	ACAR	G/L	—	06	▨
0	4	8	15 16 17	18	19	20—23	

ONESF

10	▨	SKIP	ACAR	G/L	—	07	▨
0	4	8	15 16 17	18	19	20—23	

DESCRIPTION: This instruction determines if bits 0:64 of the specified ACAR are all "ones". (Refer to TEST-SKIP for further details.)

FLOW CHART: See TEST-SKIP instruction.



TEST - SKIP INSTRUCTIONS (Cont'd)

MNEMONIC CODE: ONEX_ _

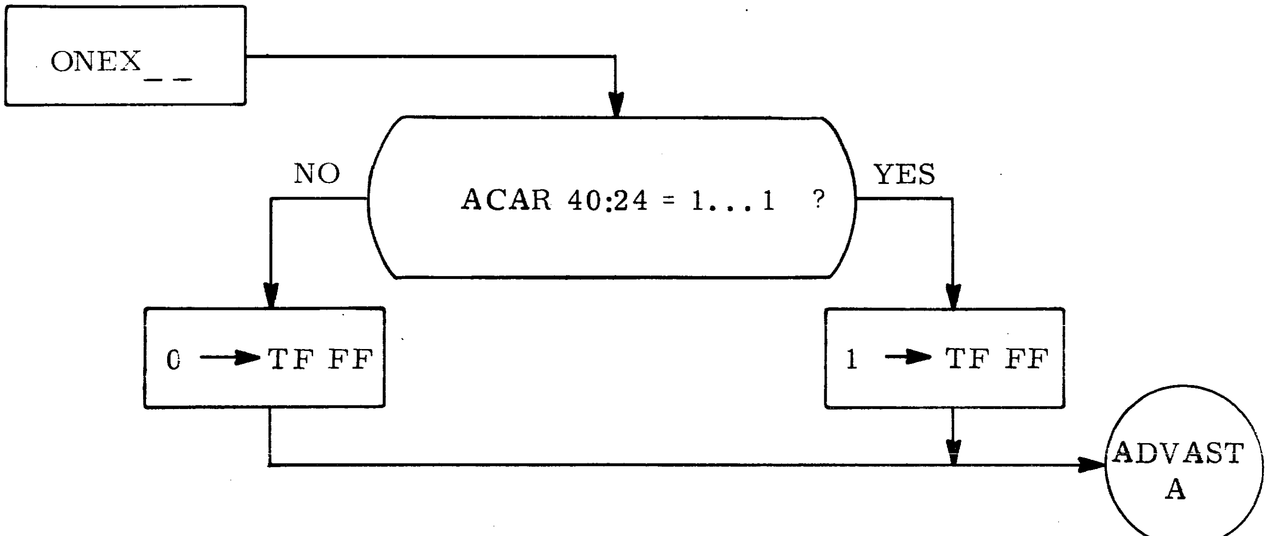
OPERATION: Skip if ACAR 40:24 are All Ones

AIR:

ONEXTA	10	▨	SKIP	ACAR	G/ L	—	14	▨
	0 4	8	15 16	17	18	19	20—23	
ONEXT	10	▨	SKIP	ACAR	G/ L	—	15	▨
	0 4	8	15 16	17	18	19	20-23	
ONEXFA	10	▨	SKIP	ACAR	G/ L	—	16	▨
	0 4	8	15 16	17	18	19	20-23	
ONEXF	10	▨	SKIP	ACAR	G/ L	—	17	▨
	0 4	8	15 16	17	18	19	20-23	

DESCRIPTION: This instruction determines if bits 40:24 of the specified ACAR are all "ones". (Refer to TEST-SKIP for further details.)

FLOW CHART: See TEST-SKIP instruction.



p. 3-63

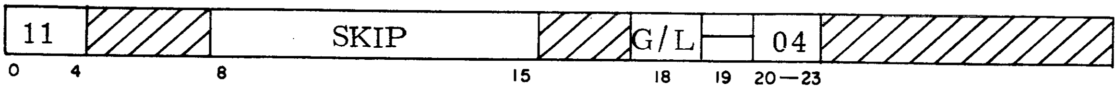
TEST - SKIP INSTRUCTIONS (Cont'd)

MNEMONIC CODE: SKIP_ _

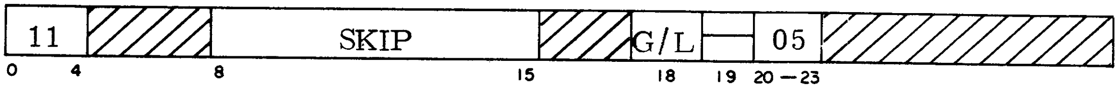
OPERATION: Skip Dependent Upon CU TF Flip-Flop

AIR:

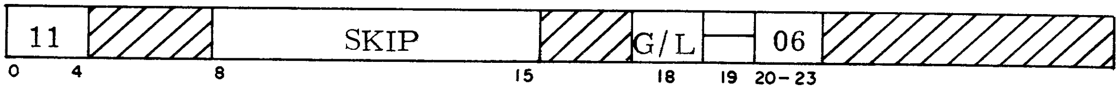
SKIPTA



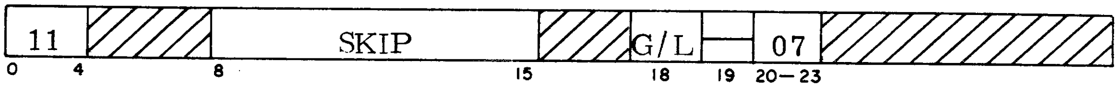
SKIPT



SKIPFA

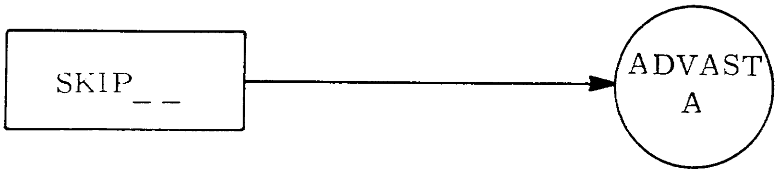


SKIPF



DESCRIPTION: This instruction is classified as a TEST and SKIP instruction. It differs from the preceding TEST and SKIP instructions in that no test is performed during the TEST portion of the instruction and, therefore, the TF flip-flop in each CU is sampled but not changed. The test of the flip-flops is made to determine if the SKIP is to be executed. (Refer to TEST-SKIP for further details.)

FLOW CHART: See TEST-SKIP instruction.



p. 3-63

TEST - SKIP INSTRUCTIONS (Cont'd)

MNEMONIC CODE: TXE_ _

OPERATION: Skip if ACAR 40:24 are Equal to Bits 16:24 in Local Memory

AIR:

TXETA

14	ACARX	SKIP	ACAR	G/L	—	10	ADR
0 4	5 7	8 15	16 17	18	19	20 – 23	24 31

TXET

14	ACARX	SKIP	ACAR	G/L	—	11	ADR
0 4	5 7	8 15	16 17	18	19	20–23	24 31

TXEFA

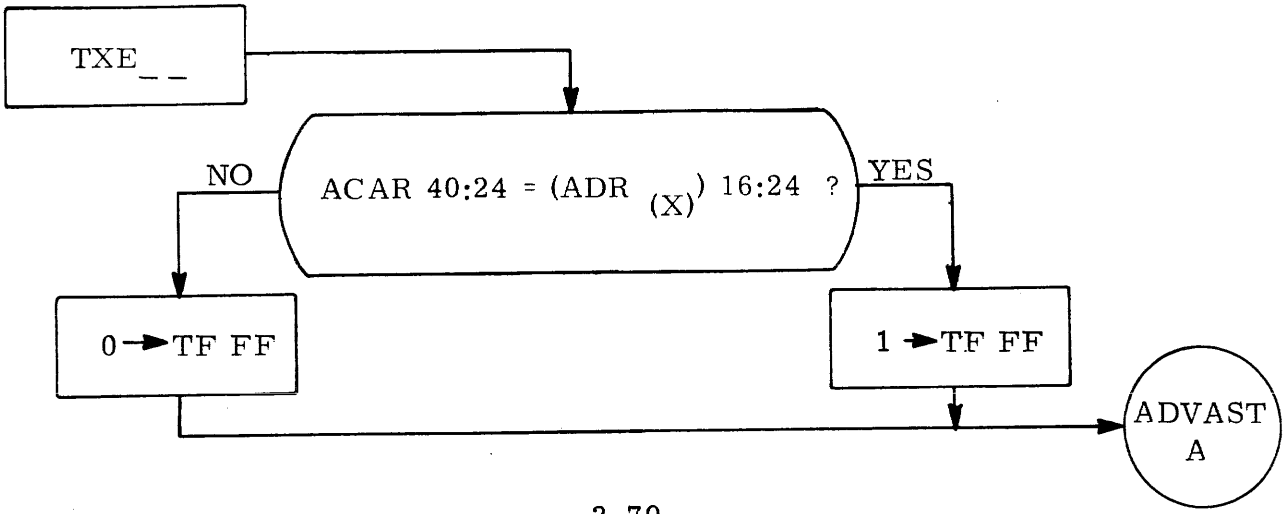
14	ACARX	SKIP	ACAR	G/L	—	12	ADR
0 4	5 7	8 15	16 17	18	19	20–23	24 31

TXEF

14	ACARX	SKIP	ACAR	G/L	—	13	ADR
0 4	5 7	8 15	16 17	18	19	20–23	24 31

DESCRIPTION: This instruction determines if bits 40:24 of the specified ACAR are equal to bits 16:24 of the operand in the local address. The local address field is indexable and addresses are limited to the ADB and the four ACARs. (Refer to TEST-SKIP for further details.)

FLOW CHART: See TEST-SKIP instruction.



TEST - SKIP INSTRUCTIONS (Cont'd)

MNEMONIC CODE: TXE_ _ M

OPERATION: Skip if ACAR 40:24 are Equal to Bits 16:24 (also, 40:24 are modified by 1:15) of Same ACAR

AIR:

TXETAM

12	▨	SKIP	ACAR	G/L	—	14	▨
0	4	8	15 16	17 18	19	20–23	

TXETM

12	▨	SKIP	ACAR	G/L	—	15	▨
0	4	8	15 16	17 18	19	20–23	

TXEFAM

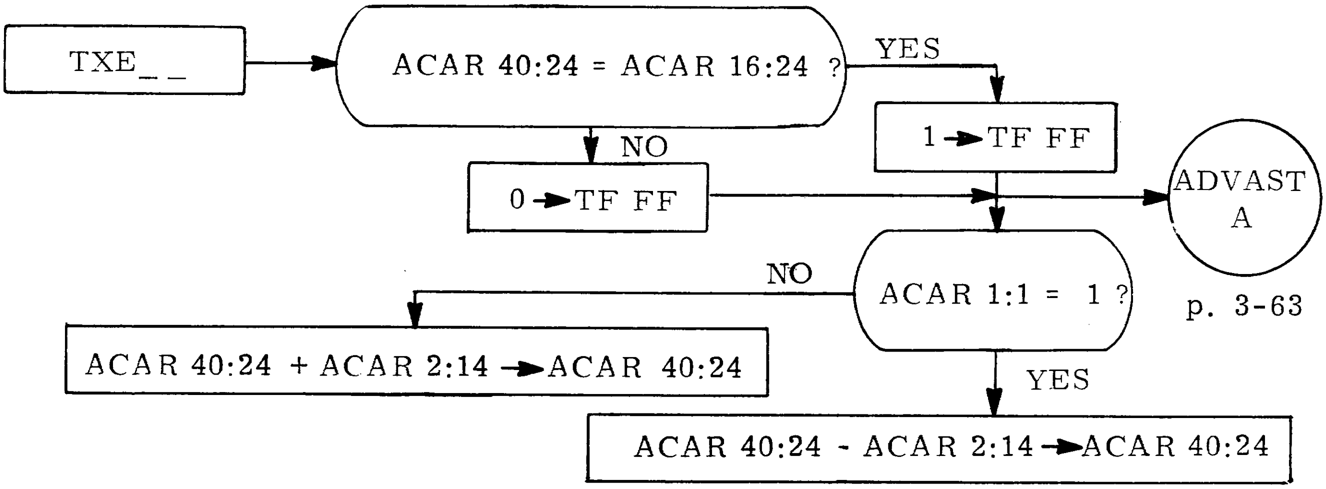
12	▨	SKIP	ACAR	G/L	—	16	▨
0	4	8	15 16	17 18	19	20–23	

TXEFM

12	▨	SKIP	ACAR	G/L	—	17	▨
0	4	8	15 16	17 18	19	20–23	

DESCRIPTION: This instruction determines if bits 40:24 of the specified ACAR are equal to bits 16:24 of the same ACAR. After the comparison, ACAR 40:24 are modified by ACAR 1:15. Bit 1 is the sign bit ("1" means subtract, "0" means add) and bits 2:14 are the magnitude of the modifier. (Refer to TEST-SKIP for further details.)

FLOW CHART: See TEST-SKIP instruction.


TXE_ _
ACAR 40:24 = ACAR 16:24 ?
YES
NO
1 → TF FF
0 → TF FF
ADVAST A
p. 3-63
NO
ACAR 1:1 = 1 ?
YES
ACAR 40:24 + ACAR 2:14 → ACAR 40:24
ACAR 40:24 - ACAR 2:14 → ACAR 40:24

TEST - SKIP INSTRUCTIONS (Cont'd)

MNEMONIC CODE: TXG _ _

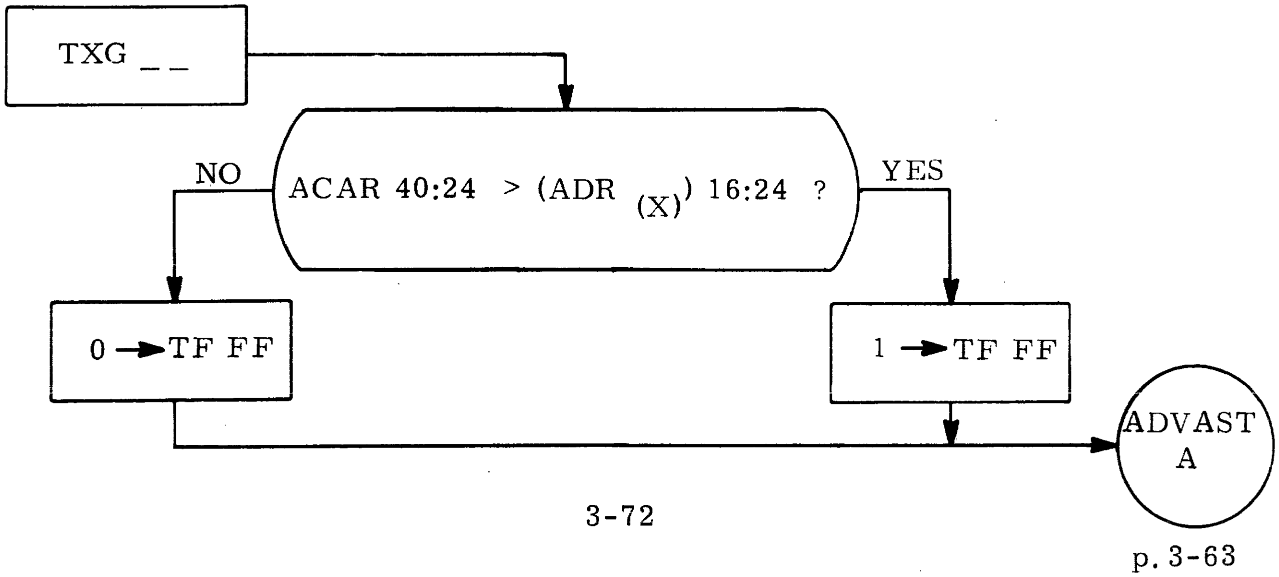
OPERATION: Skip if ACAR 40:24 are Greater Than Bits 16:24 in Local Memory

AIR:

	0 4	5 7	8 15	16 17	18	19	20–23	24 31
TXGTA	14	ACARX	SKIP	ACAR	G/L	—	00	ADR
TXGT	14	ACARX	SKIP	ACAR	G/L	—	01	ADR
TXGFA	14	ACARX	SKIP	ACAR	G/L	—	02	ADR
TXGF	14	ACARX	SKIP	ACAR	G/L	—	03	ADR

DESCRIPTION: This instruction determines if bits 40:24 of the specified ACAR are greater than bits 16:24 of the operand in the local address. The address field is indexable and addresses are limited to the ADB and the four ACARs. (Refer to TEST-SKIP for further details.)

FLOW CHART: See TEST-SKIP instruction.



TEST - SKIP INSTRUCTIONS (Cont'd)

MNEMONIC CODE: TXG _ _ M

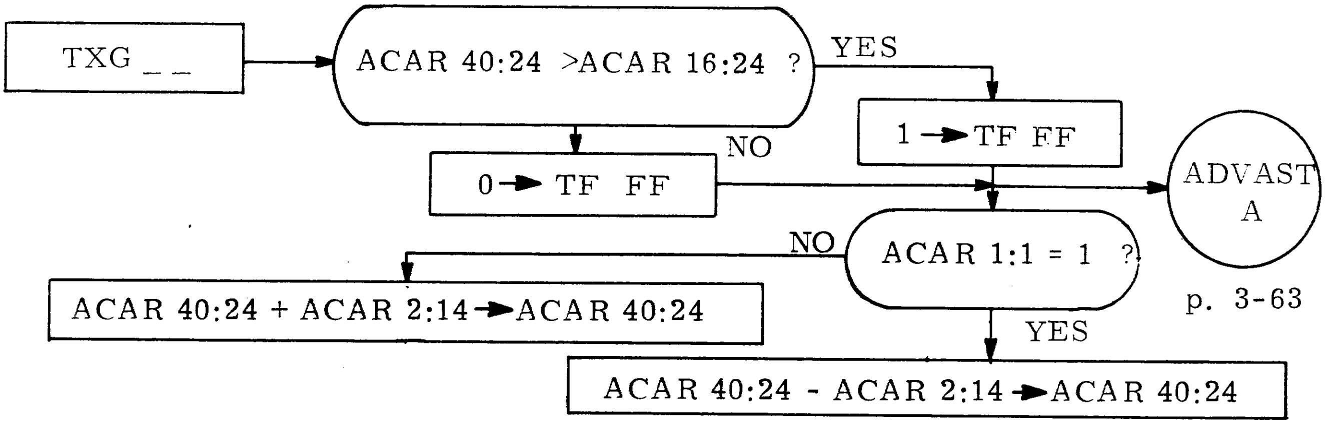
OPERATION: Skip if Bits 40:24 are Greater Than Bits 16:24 (also, 40:24 are modified by 1:15) of Same ACAR

AIR:

TXGTAM	13	▨	SKIP	ACAR	G/L	—	00	▨
	0	4	8	15 16	17	18 19	20–23	
TXGTM	13	▨	SKIP	ACAR	G/L	—	01	▨
	0	4	8	15 16	17	18 19	20–23	
TXGFAM	13	▨	SKIP	ACAR	G/L	—	02	▨
	0	4	8	15 16	17	18 19	20–23	
TXGFM	13	▨	SKIP	ACAR	G/L	—	03	▨
	0	4	8	15 16	17	18 19	20–23	

DESCRIPTION: This instruction determines if bits 40:24 of the specified ACAR are greater than bits 16:24 of the same ACAR. After the comparison, ACAR 40:24 are modified by ACAR 1:15. Bit 1 is the sign bit ("1" means subtract, "0" means add) and bits 2:14 are the magnitude of the modifier. (Refer to TEST-SKIP for further details.)

FLOW CHART: See TEST-SKIP instruction.



TEST - SKIP INSTRUCTIONS (Cont'd)

MNEMONIC CODE: TXL _ _

OPERATION: Skip if ACAR 40:24 are Less Than Bits 16:24 in Local Memory

AIR:

TXLTA	14	ACARX	SKIP	ACAR	G/L	—	04	ADR
	0 4	5 7	8 15	16 17	18	19	20–23	24 31

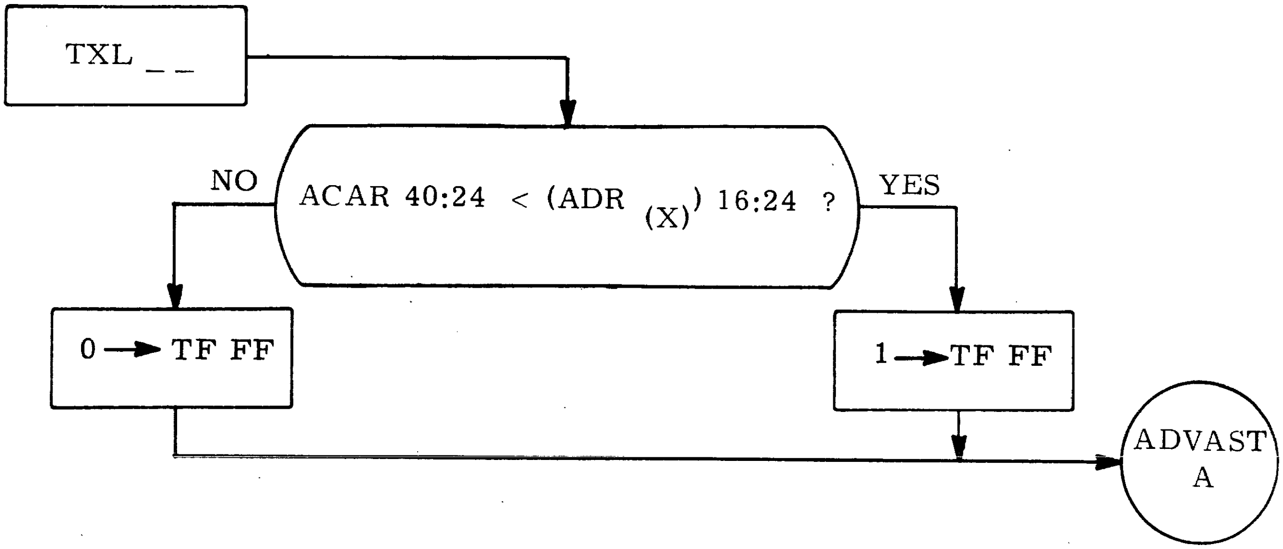
TXLT	14	ACARX	SKIP	ACAR	G/L	—	05	ADR
	0 4	5 7	8 15	16 17	18	19	20–23	24 31

TXLFA	14	ACARX	SKIP	ACAR	G/L	—	06	ADR
	0 4	5 7	8 15	16 17	18	19	20–23	24 31

TXLF	14	ACARX	SKIP	ACAR	G/L	—	07	ADR
	0 4	5 7	8 15	16 17	18	19	20–23	24 31

DESCRIPTION: This instruction determines if bits 40:24 of the specified ACAR are less than bits 16:24 of the operand in the local address. The local address field is indexable and addresses are limited to the ADB and the four ACARs. (Refer to TEST-SKIP for further details.)

FLOW CHART: See TEST-SKIP instruction.



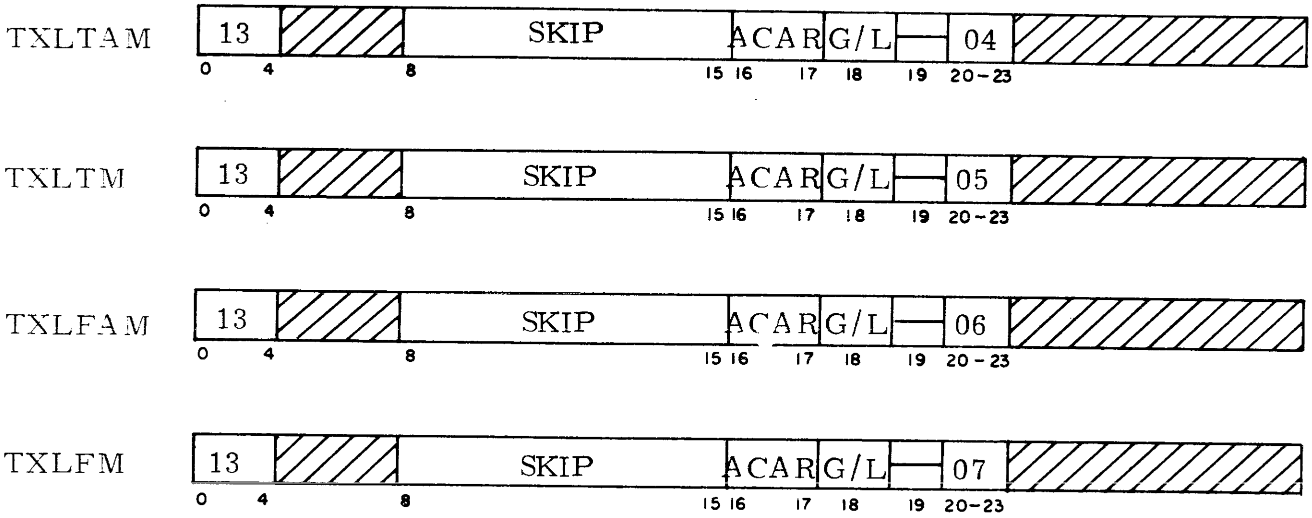
p. 3-63

TEST - SKIP INSTRUCTIONS (Cont'd)

MNEMONIC CODE: TXL _ _ M

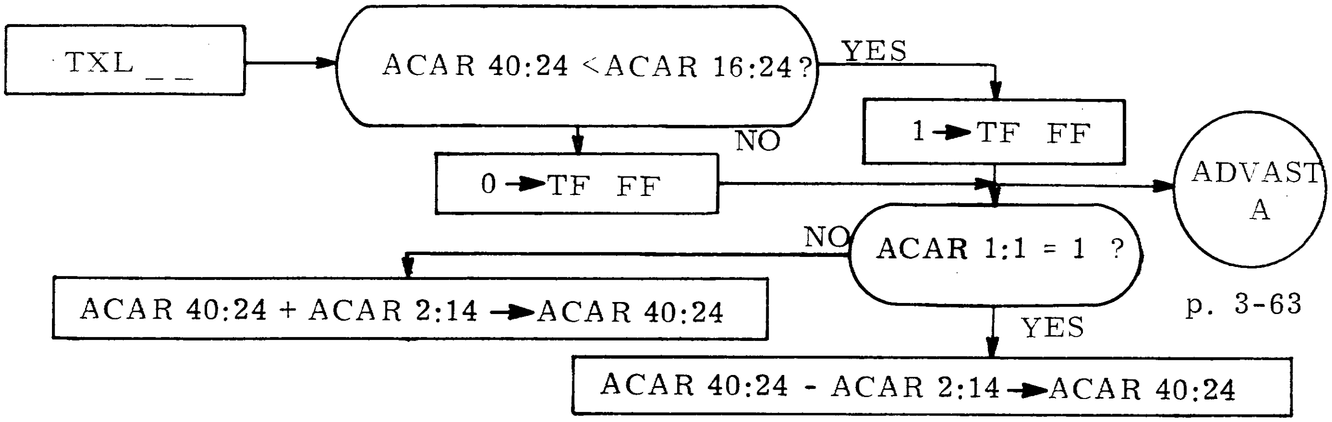
OPERATION: Skip if ACAR 40:24 are Less Than Bits 16:24 (then 40:24 is modified by 1:15) of Same ACAR

AIR:



DESCRIPTION: This instruction determines if bits 40:24 of the specified ACAR are less than bits 16:24 of the same ACAR. After the comparison, ACAR 40:24 are modified by ACAR 1:15. Bit 1 is the sign bit ("1" means subtract, "0" means add) and bits 2:14 are the magnitude of the modifier. (Refer to TEST-SKIP for further details.)

FLOW CHART: See TEST-SKIP instruction.

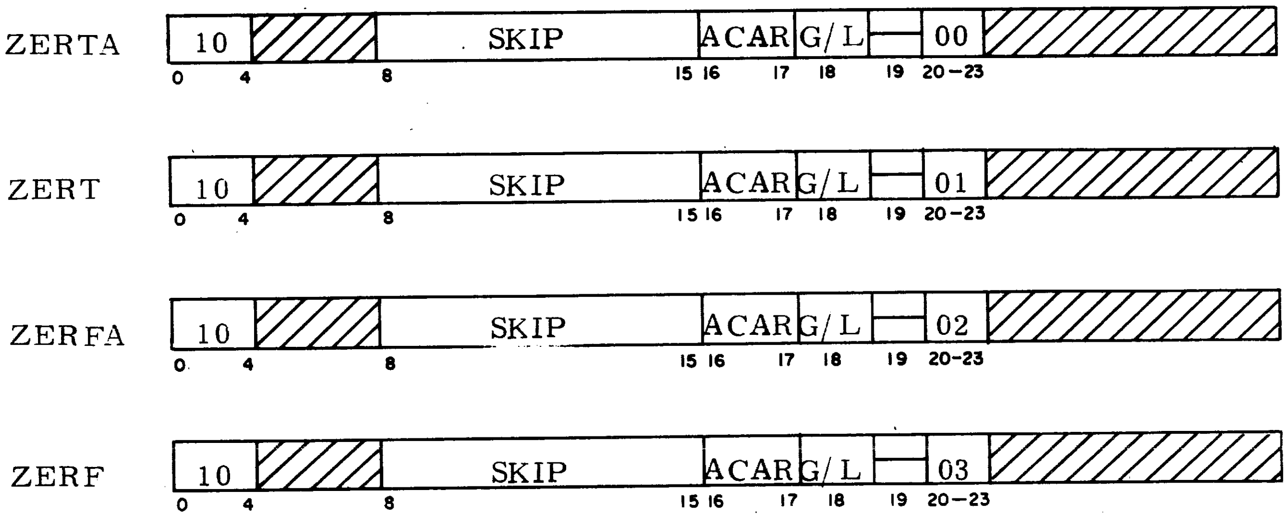


TEST - SKIP INSTRUCTIONS (Cont'd)

MNEMONIC CODE: ZER _ _

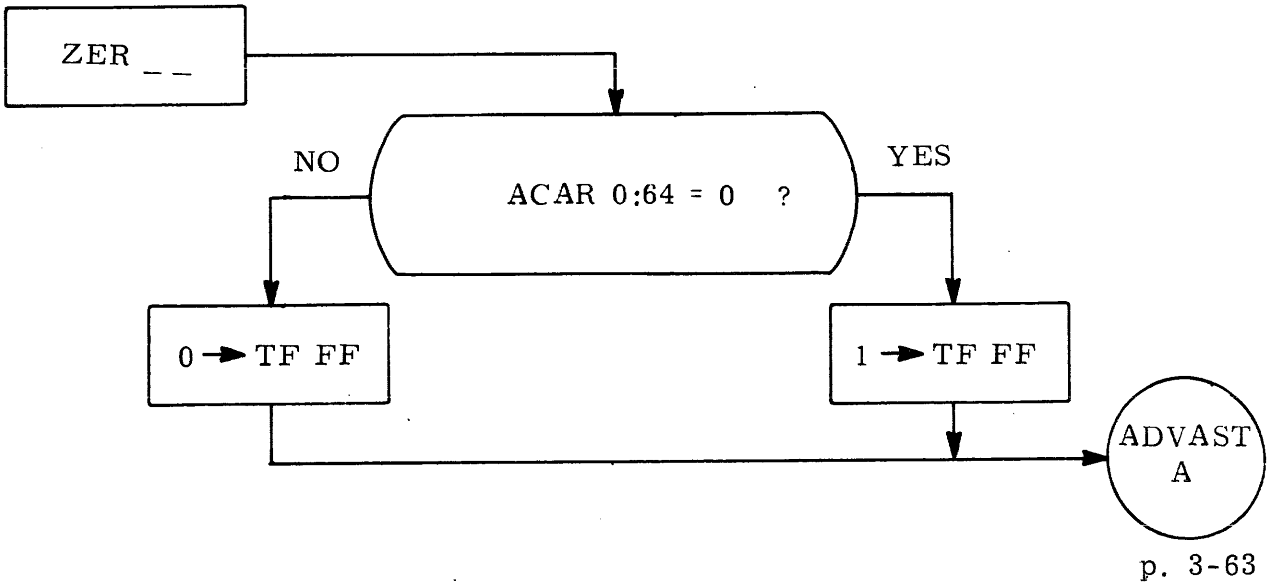
OPERATION: Skip if ACAR 0:64 are All Zeros

AIR:



DESCRIPTION: This instruction determines if bits 0:64 of the specified ACAR are all "zeros". (Refer to TEST-SKIP for further details.)

FLOW CHART: See TEST-SKIP instruction.



TEST - SKIP INSTRUCTIONS (Cont'd)

MNEMONIC CODE: ZERX _ _

OPERATION: Skip if ACAR 40:24 are All Zeros

AIR:

ZERXTA	10	▨	SKIP	ACAR	G/L	—	10	▨
	0 4	8	15	16 17	18	19	20-23	

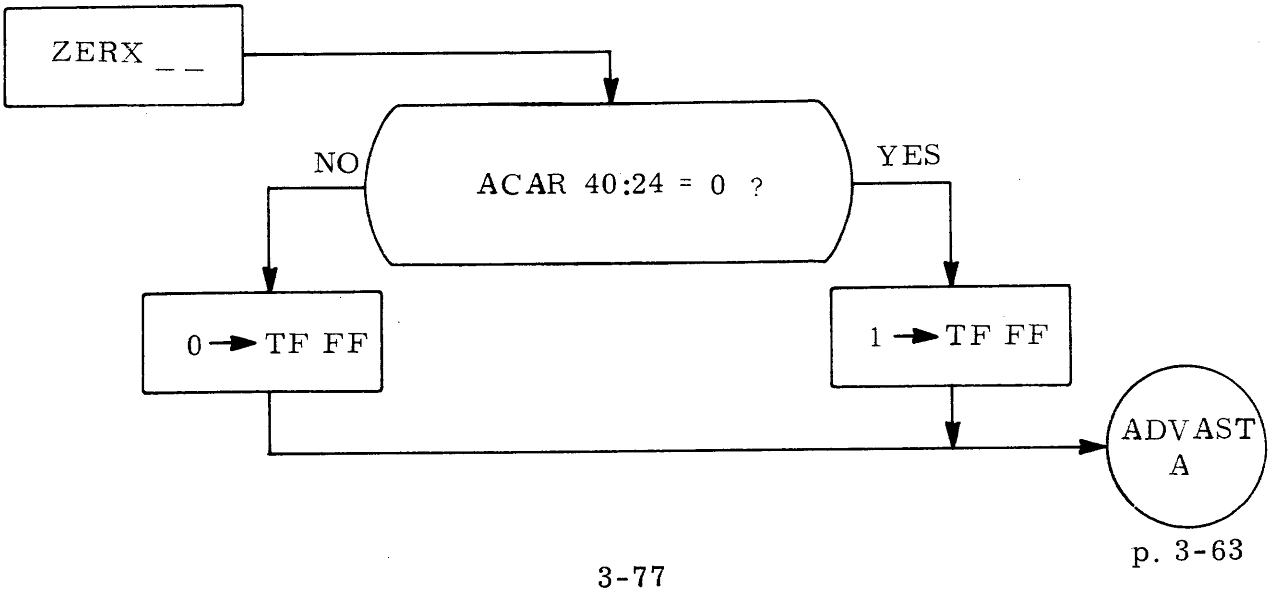
ZERXT	10	▨	SKIP	ACAR	G/L	—	11	▨
	0 4	8	15	16 17	18	19	20-23	

ZERXFA	10	▨	SKIP	ACAR	G/L	—	12	▨
	0 4	8	15	16 17	18	19	20-23	

ZERXF	10	▨	SKIP	ACAR	G/L	—	13	▨
	0 4	8	15	16 17	18	19	20-23	

DESCRIPTION: This instruction determines if bits 40:24 of the specified ACAR are all "zeros". (Refer to TEST-SKIP for further details.)

FLOW CHART: See TEST-SKIP instruction.



MNEMONIC CODE: WAIT

OPERATION: Synchronize All CUs in Array or Join all CUs specified by ADR 4:4

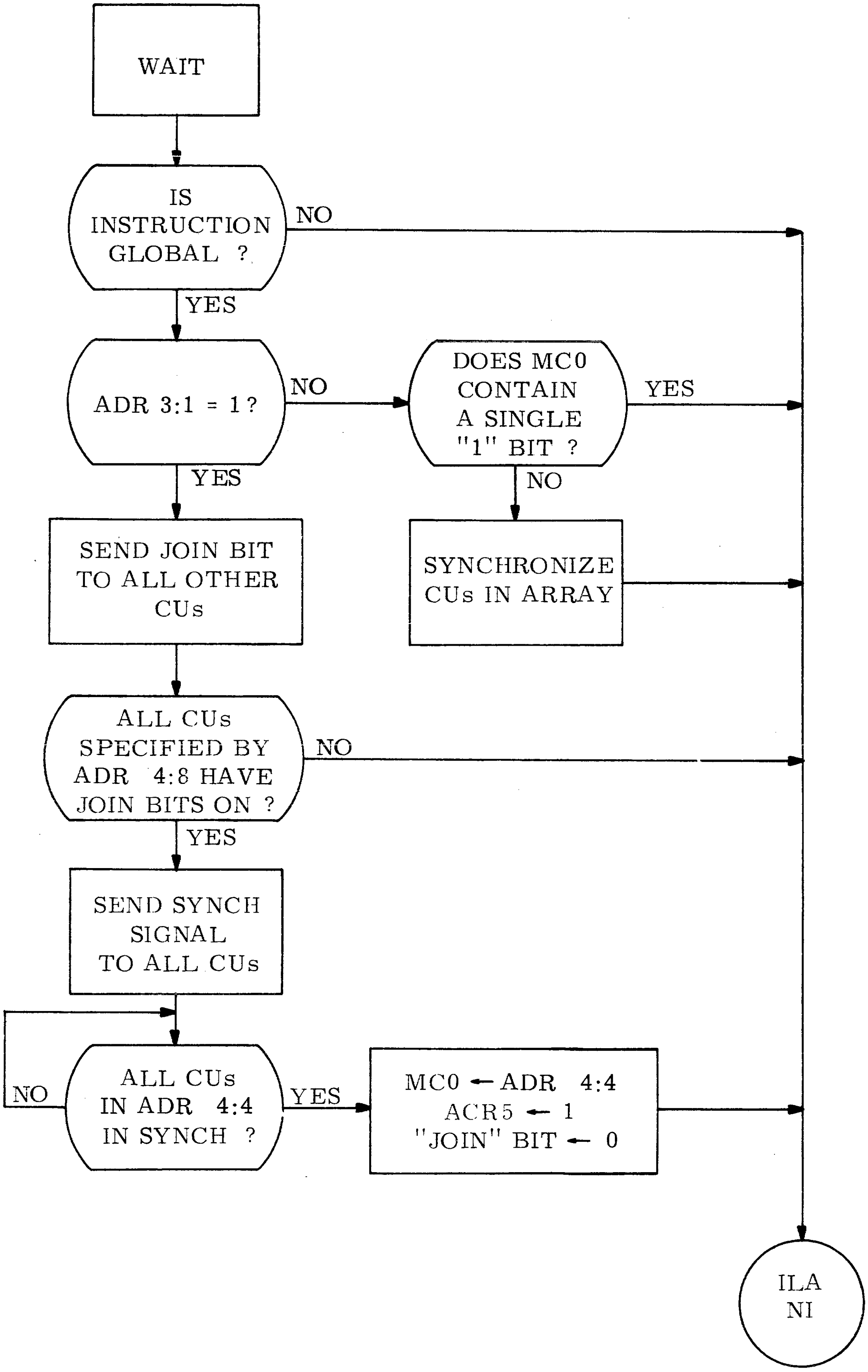
AIR:

02	ACARX	▨	G/L	—	06	ADR
0 4	5 7		18	19	20 23	24 31

DESCRIPTION: This instruction causes all CUs in the array to be synchronized at the beginning of the instruction. When ADR 3:1 is OFF, no other action is taken, and normal operation resumes when all CUs have synchronized. ADR 3:1 ON specifies that this CU is requesting other quadrants (CUs) to join it in multiquadrant array. After the CUs have joined, they will synchronize and load MC0 with the contents of ADR 4:4 and set ACR5. ACR5 is the indicator to the program that the CUs have joined. The desired array is specified by ADR 4:4.

> NOTE: The use of the ACARX field can cause this instruction to be modified from the normal to the special option and the reverse, by causing the setting and resetting of ADR 27.

FLOW CHART: See next page.


WAIT
IS INSTRUCTION GLOBAL ?
NO
YES
ADR 3:1 = 1 ?
NO
DOES MC0 CONTAIN A SINGLE "1" BIT ?
YES
NO
YES
SEND JOIN BIT TO ALL OTHER CUs
SYNCHRONIZE CUs IN ARRAY
ALL CUs SPECIFIED BY ADR 4:8 HAVE JOIN BITS ON ?
NO
YES
SEND SYNCH SIGNAL TO ALL CUs
NO
ALL CUs IN ADR 4:4 IN SYNCH ?
YES
MC0 ← ADR 4:4
ACR5 ← 1
"JOIN" BIT ← 0
ILA NI

CONTENTS

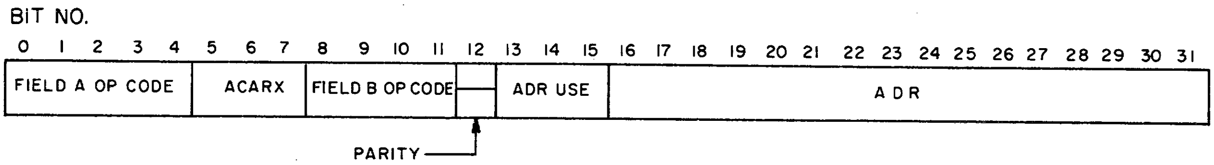
FINST/PE INSTRUCTION INDEX

Mnemonic Code	Octal Code	Ref. Page	Mnemonic Code	Octal Code	Ref. Page	Mnemonic Code	Octal Code	Ref. Page
AD	3504	4-17	IXL	2310	4-59	NORN	2307	4-31
ADA	3505	4-17	IXLD	2712	4-62	OFB	2506	4-76
ADB	2606	4-22	JAG	3715	4-52	OR	2304	4-31
ADD	2604	4-23	JAL	3717	4-52	ORN	2306	4-31
ADEX	2500	4-24	JB	3503	4-54	RAB	3701	4-36
ADM	3414	4-17	JLE	3517	4-55	RTAL	3513	4-87
ADMA	3415	4-17	JLG	3315	4-55	RTAR	3512	4-88
ADN	3404	4-17	JLL	3317	4-55	RTG	2413	4-77
ADNA	3405	4-17	JLO	3311	4-57	RTL	2412	4-77
ADR	3506	4-17	JLZ	3313	4-57	SAB	3702	4-36
ADRA	3507	4-17	JME	3515	4-55	SAN	3702	4-38
ADRN	3406	4-17	JMG	3115	4-55	SAP	3701	4-38
ADRNA	3407	4-17	JML	3117	4-55	SB	3704	4-79
AND	2704	4-27	JMO	3111	4-57	SBA	3705	4-79
ANDN	2706	4-27	JMZ	3113	4-57	SBB	2607	4-82
ASB	2507	4-26	JSE	2513	4-59	SBEX	2501	4-83
~~ASTRG~~	~~2417~~	~~4-26A~~	JSG	2113	4-59	SBM	3614	4-79
~~ASTRL~~	~~2416~~	~~4-26A~~	JSL	2313	4-59	SBMA	3615	4-79
CAB	3700	4-33	JSN	3503	4-54	SBN	3604	4-79
CHSA	3700	4-35	JXE	2511	4-59	SBNA	3605	4-79
CLRA	2411	4-39	JXG	2111	4-59	SBR	3706	4-79
COMPA	2211	4-40	JXGI	2711	4-61	SBRA	3707	4-79
DV	3304	4-41	JXL	2311	4-59	SBRN	3606	4-79
DVA	3305	4-41	JXLD	2713	4-62	SBRNA	3607	4-79
DVM	3214	4-41	LB	2107	4-63	SCM	2104	4-85
DVMA	3215	4-41	LDA	2617	4-104	SETE	2514	4-69
DVN	3204	4-41	LDB	2700	4-104	SETE1	2515	4-69
DVNA	3205	4-41	LDD	2212	4-104	SETF	2516	4-69
DVR	3306	4-41	LDE	2114	4-69	SETF1	2517	4-70
DVRA	3307	4-41	LDE1	2115	4-69	SETG	2714	4-70
DVRM	3216	4-41	LDEE1	2116	4-69	SETH	2715	4-70
DVRMA	3217	4-41	LDG	2314	4-69	SETI	2716	4-70
DVRN	3206	4-41	LDH	2315	4-69	SETJ	2717	4-70
DVRNA	3207	4-41	LDI	2316	4-69	SHABL	3711	4-89
EAD	2010	4-45	LDJ	2317	4-69	SHABML	3713	4-91
EOR	2505	4-29	LDR	2701	4-104	SHABMR	3712	4-92
EQV	2504	4-30	~~LDRAG~~	~~2415~~	~~4-106A~~	SHABR	3710	4-90
ESB	2410	4-48	~~LDRAL~~	~~2414~~	~~4-106A~~	SHAL	3501	4-93
GB	2106	4-50	LDS	2702	4-104	SHAML	3511	4-95
IAG	3714	4-52	LDX	2703	4-104	SHAMR	3510	4-96
IAL	3716	4-52	LEX	2117	4-64	SHAR	3500	4-94
IB	3502	4-54	ML	3104	4-65	STA	2612	4-97
ILE	3516	4-55	MLA	3105	4-65	STB	2613	4-97
ILG	3314	4-55	MLM	3014	4-65	STR	2614	4-97
ILL	3316	4-55	MLMA	3015	4-65	STS	2615	4-97
ILO	3310	4-57	MLN	3004	4-65	STX	2616	4-97
ILZ	3312	4-57	MLNA	3005	4-65	SUB	2605	4-99
IME	3514	4-55	MLR	3106	4-65	SWAP	3103	4-100
IMG	3114	4-55	MLRA	3107	4-65	SWAPA	3303	4-101
IML	3116	4-55	MLRM	3016	4-65	SWAPX	3703	4-102
IMO	3110	4-57	MLRMA	3017	4-65	T3A	2105	4-103
IMZ	3112	4-47	MLRN	3006	4-65	TCY	3100	---
ISE	2512	4-59	MLRNA	3007	4-65	TCYS	3101	---
ISG	2112	4-59	MULT	2213	4-72	TCYX	3102	---
ISL	2312	4-59	NAND	2705	4-27	XD	2503	4-107
ISN	3502	4-54	NANDN	2707	4-27	XI	2502	4-108
IXE	2510	4-59	NEB	2210	4-73			
IXG	2110	4-59	NOR	2305	4-31			
IXGI	2710	4-61	NORM	2013	4-74			

SECTION IV
FINST/PE INSTRUCTIONS

INSTRUCTION FORMAT AND FIELD USAGE

The format of FINST/PE instruction words is given below, followed by an explanation of field usage. Table 4-1 provides a complete listing of the FINST/PE operation codes.



Field

Description

FIELD A OP CODE

BITS 0:2, 2:3. First part of operation code (see Table 4-1). Bit 0 is always "one" for FINST/PE type instructions.

ACARX

BITS 5:1, 6:2. When bit 5 is "one", the contents of the ACAR specified by bits 6 and 7 are added to the ADR field. When bit 5 is "zero", the values in bits 6 and 7 are irrelevant.

Where a literal is being transmitted, the value received by the PE is as follows:

ACARX	
None	Bits 0:48 = 0; Bits 48:16 = ADR 0:16
Any	Bits 0:48 = ACAR 0:48; Bits 48:16 = ACAR 48:16 + ADR 0:16 (high-order carry is lost).

FIELD B OP CODE

BITS 8:1, 9:3. Second part of operation code.

PARITY

BIT 12:1. Odd parity bit.

ADR USE

BITS 13:3. These bits govern the use of the ADR field according to the following:

If BIT 15 is set, the ADR field is a PEM address, and the other bits have the following meaning:

Bits 13-14 reset: No PE indexing
BIT 13 set: Index by RGS
BIT 14 set: Index by RGX
Bits 13-14 set: Index by RGS

If BIT 15 is reset, then either the operand, the (shift or bit) value or the register code is being transmitted from the CU. (A bit value is considered a shift count.) The balance of the ADR USE field bits are defined as follows:

BIT 13 set: CU is transmitting a register code.
BIT 13 reset: CU is transmitting a literal.
BIT 14: Disregarded.

Where a register code is being transmitted, the following codes are used:

		Corresponding	
ADR Bit	Register	AIR Bit	ACn Bit
1	A	17	49
2	B	18	50
3	X	19	51
4	S	20	52
5	R	21	53
6	D	22	54

All register codes are allowed with every instruction except for the instructions LD (A | D| R| S |X).

ADR

BITS 16:16. Address field, designating, according to instruction type, one of the following: location of the operand, shift count or bit value, index amount, or routing distance.

ADR (Cont.)

Normally, the use of this field is defined according to the instruction type. For example, for the RT(G/L) instruction this field contains a routing distance and the source register address; for shift instructions, this field contains only a shift count. (The bit value is also considered a shift count.) However, for other instructions, the ADR field normally contains the source of the operand.

Table 4-1. FINST/PE Instruction OP Codes

INSTRUCTION BITS 8:4

INSTRUCTION BITS 0:5	00	01	02	03	04	05	06	07	10	11	12	13	14	15	16	17
20									EAD	*	*	NORM				
21					SCM	T3A	GB	LB	IXG	JXG	ISG	JSG	LDE	LDE1	LDEE1	LEX
22									NEB	COMPA	LDD	MULT				
23					OR	NOR	ORN	NORN	IXL	JXL	ISL	JSL	LDG	LDH	LDI	LDJ
24									ESB	CLRA	RTL	RTG	LDRAL	LDRAG	ASTRL	ASTRG
25	ADEX	SBEX	XI	XD	EQV	EOR	OFB	ASB	IXE	JXE	ISE	JSE	SETE	SETE1	SETF	SETF1
26	*	*	*	*	ADD	SUB	ADB	SBB	*	*	STA	STB	STR	STS	STX	LDA
27	LDB	LDR	LDS	LDX	AND	NAND	ANDN	NANDN	IXGI	JXGI	IXLD	JXLD	SETG	SETH	SETI	SETJ
30					MLN	MLNA	MLRN	MLRNA					MLM	MLMA	MLRM	MLRMA
31	TCY	TCYS	TCYX	SWAP	ML	MLA	MLR	MLRA	IMO	JMO	IMZ	JMZ	IMG	JMG	IML	JML
32					DVN	DVNA	DVRN	DVRNA					DVM	DVMA	DVRM	DVRMA
33				SWAPA	DV	DVA	DVR	DVRA	ILO	JLO	ILZ	JLZ	ILG	JLG	ILL	JLL
34					ADN	ADNA	ADRN	ADRNA					ADM	ADMA	**	**
35	SHAR	SHAL	IB/ISN	JB/JSN	AD	ADA	ADR	ADRA	SHAMR	SHAML	RTAR	RTAL	IME	JME	ILE	JLE
36					SBN	SBNA	SBRN	SBRNA					SBM	SBMA	**	**
37	CAB/ CHSA	RAB/ SAP	SAB/ SAN	SWAPX	SB	SBA	SBR	SBRA	SHABR	SHABL	SHABMR	SHABML	IAG	JAG	IAL	JAL

(AD, SB, ML, and DV)	Suffix	Meaning
	A	Unsigned
	M	Fixed Point
	N	Normalized
	R	Rounded

* These op codes are used by ADVAST instructions
** These op codes are undefined variants of AD and SB.
BLANK: Illegal op codes

FINST/PE INSTRUCTION REPERTOIRE

The next several pages provide a listing of the instructions that comprise the PE/FINST repertoire. They are arranged in alphabetical order according to mnemonic and functional group, and in the same order as the instruction descriptions which comprise the remainder of this subsection.

Mnemonic Code	Octal Op Code Field A 0:5	 Field B 8:4	Operation
AD	34	04	Add (ADR) to RGA. Variants are: Suffix — Meaning A — Unsigned M — Fixed point N — Normalized floating R — Rounded
ADA	34	05	
ADM	34	14	
ADMA	34	15	
ADN	35	04	
ADNA	35	05	
ADR	34	06	
ADRA	34	07	
ADRN	35	06	
ADRNA	35	07	
ADB	26	06	Add (ADR) to RGA in 8-bit bytes
ADD	26	04	Add 64-bit unsigned fixed-point numbers (ADR) to RGA
ADEX	25	00	Add (ADR) exponent field(s) to RGA exponents
ASB	25	07	Place the sign(s) of RGA into the sign(s) of RGB
Boolean Operations:			Place the result of the specified logical function of RGA with (ADR) into RGA:
AND	27	04	Logical AND of RGA with (ADR)
ANDN	27	06	Logical AND of RGA with complement of (ADR)
EOR	25	05	Logical EXCLUSIVE-OR of RGA with (ADR)
EQV	25	04	Logical EQUIVALENCE of RGA with (ADR)
NAND	27	05	Logical AND of complement of RGA with (ADR)
NANDN	27	07	Logical AND of complement of RGA with complement of (ADR)
NOR	23	05	Logical OR of complement of RGA with (ADR)
NORN	23	07	Logical OR of complement of RGA with complement of (ADR)
OR	23	04	Logical OR of RGA with (ADR)
ORN	23	06	Logical OR of RGA with complement of (ADR)

Mnemonic	Octal Op Code		Operation
	Field A	Field B	
Code	0:5	8:4	
Change RGA Bit:			Perform the indicated operation on the specified RGA bit:
CAB	37	00	Complement bit(s) in RGA
CHSA	37	00	Change sign bit(s) in RGA
RAB	37	01	Reset bit(s) in RGA
SAB	37	02	Set bit(s) in RGA
SAP	37	01	Reset sign bit(s) in RGA
SAN	37	02	Set sign bit(s) in RGA
CLRA	24	11	Clear RGA
COMPA	22	11	Complement RGA
DV	32	04	Divide RGA and RGB, double-length mantissa, by (ADR). Variants are: Suffix — Meaning A — Unsigned M — Fixed point N — Normalized R — Rounded
DVA	32	05	
DVM	32	14	
DVMA	32	15	
DVN	33	04	
DVNA	33	05	
DVR	32	06	
DVRA	32	07	
DVRM	32	16	
DVRMA	32	17	
DVRN	33	06	
DVRNA	33	07	
EAD	20	10	Recover extended precision after floating-point add
ESB	24	10	Recover extended precision after floating-point subtract
GB	21	06	Test for RGA greater than (ADR) in 8-bit bytes.
(I|J) A (G| L)			RGA arithmetic test to mode bit (for 32-bit mode, result also goes to G or to H):
IAG	37	14	Place result of test for RGA arithmetically greater than (ADR) into I (and G)
IAL	37	16	Place result of test for RGA arithmetically less than (ADR) into I (and G)
JAG	37	15	Place result of test for RGA arithmetically greater than (ADR) into J (and H)
JAL	37	17	Place result of test for RGA arithmetically less than (ADR) into J (and H)

<table>
<tr><th></th><th colspan="2">Octal Op Code</th><th></th></tr>
<tr><th>Mnemonic Code</th><th>Field A 0:5</th><th>Field B 8:4</th><th>Operation</th></tr>
<tr><td>(I | J)(B | SN)</td><td></td><td></td><td>Move RGA bit to mode bit:</td></tr>
<tr><td>IB</td><td>35</td><td>02</td><td>Transfer RGA bit(s) to I (and G)</td></tr>
<tr><td>ISN</td><td>35</td><td>02</td><td>Transfer RGA sign(s) to I (and G)</td></tr>
<tr><td>JB</td><td>35</td><td>03</td><td>Transfer RGA bit(s) to J (and H)</td></tr>
<tr><td>JSN</td><td>35</td><td>03</td><td>Transfer RGA sign(s) to J (and H)</td></tr>
<tr><td>(I | J) (L | M) (E | G | L)</td><td></td><td></td><td>RGA logical test to mode bit (for 32-bit mode, results go to I and G or to J and H):</td></tr>
<tr><td>ILE</td><td>35</td><td>16</td><td>Place result of test for RGA logically equal to (ADR) into I</td></tr>
<tr><td>ILG</td><td>33</td><td>14</td><td>Place result of test for RGA logically greater than (ADR) into I</td></tr>
<tr><td>ILL</td><td>33</td><td>16</td><td>Place result of test for RGA logically less than (ADR) into I</td></tr>
<tr><td>IME</td><td>35</td><td>14</td><td>Place result of test for RGA mantissa logically equal to (ADR) mantissa into I</td></tr>
<tr><td>IMG</td><td>31</td><td>14</td><td>Place result of test for RGA mantissa logically greater than (ADR) mantissa into I</td></tr>
<tr><td>IML</td><td>31</td><td>16</td><td>Place result of test for RGA mantissa logically less than (ADR) mantissa into I</td></tr>
<tr><td>JLE</td><td>35</td><td>17</td><td>Place result of test for RGA logically equal to (ADR) into J</td></tr>
<tr><td>JLG</td><td>33</td><td>15</td><td>Place result of test for RGA logically greater than (ADR) into J</td></tr>
<tr><td>JLL</td><td>33</td><td>17</td><td>Place result of test for RGA logically less than (ADR) into J</td></tr>
<tr><td>JME</td><td>35</td><td>15</td><td>Place result of test for RGA mantissa logically equal to (ADR) mantissa into J</td></tr>
<tr><td>JMG</td><td>31</td><td>15</td><td>Place result of test for RGA mantissa logically greater than (ADR) mantissa into J</td></tr>
<tr><td>JML</td><td>31</td><td>17</td><td>Place result of test for RGA mantissa logically less than (ADR) mantissa into J</td></tr>
<tr><td>(I | J) (L | M) (O | Z)</td><td></td><td></td><td>RGA zeros or ones test to mode bit (for 32-bit mode, results also go into G or H):</td></tr>
<tr><td>ILO</td><td>33</td><td>10</td><td>Place result of test for RGA logically equal to all "ones" into I</td></tr>
<tr><td>ILZ</td><td>33</td><td>12</td><td>Place result of test for RGA logically equal to zero into I</td></tr>
<tr><td>IMO</td><td>31</td><td>10</td><td>Place result of test for RGA mantissa logically equal to all "ones" into I</td></tr>
<tr><td>IMZ</td><td>31</td><td>12</td><td>Place result of test for RGA mantissa logically equal to zero into I</td></tr>
</table>

	Octal Op Code		
Mnemonic	Field A	Field B	
Code	0:5	8:4	Operation
JLO	33	11	Place result of test for RGA logically equal to all "ones" into J
JLZ	33	13	Place result of test for RGA logically equal to zero into J
JMO	31	11	Place result of test for RGA mantissa logically equal to all "ones" into J
JMZ	31	13	Place result of test for RGA mantissa logically equal to zero into J
(I | J) (S | X) (E | G | L)			Index test to mode bit:
ISE	25	12	Place result of test for (RGS) arithmetically equal to (ADR) into I
ISG	21	12	Place result of test for (RGS) arithmetically greater than (ADR) into I
ISL	23	12	Place result of test for (RGS) arithmetically less than (ADR) into I
IXE	25	10	Place result of test for (RGX) arithmetically equal to (ADR) into I
IXG	21	10	Place result of test for (RGX) arithmetically greater than (ADR) into I
IXL	23	10	Place result of test for (RGX) arithmetically less than (ADR) into I
JSE	25	13	Place result of test for (RGS) arithmetically equal to (ADR) into J
JSG	21	13	Place result of test for (RGS) arithmetically greater than (ADR) into J
JSL	23	13	Place result of test for (RGS) arithmetically less than (ADR) into J
JXE	25	11	Place result of test for (RGX) arithmetically equal to (ADR) into J
JXG	21	11	Place result of test for (RGX) arithmetically greater than (ADR) into J
JXL	23	11	Place result of test for (RGX) arithmetically less than (ADR) into J
(I | J) XGI			Index add overflow to mode bit:
IXGI	27	10	Add (ADR) 48:16 to RGX; store overflow in mode register bit I
JXGI	27	11	Add (ADR) 48:16 to RGX; store overflow in mode register bit J
(I | J) XLD			Index subtract underflow to mode bit:
IXLD	27	12	Subtract (ADR) 48:16 from RGX; store complement of overflow in I
JXLD	27	13	Subtract (ADR) 48:16 from RGX; store complement of overflow in J

	Octal Op Code		
Mnemonic Code	Field A 0:5	Field B 8:4	Operation
LB	21	07	Test for RGA less than (ADR) in 8-bit bytes
LDRA (G/L)	24	14	Fetch to RGR from memory, starting at given PEM number, and align (route), first word to PE 0
LEX	21	17	Load RGA exponent(s) from (ADR) exponent(s)
ML	30	04	Multiply RGA by (ADR). Variants are:
MLA	30	05	
MLM	30	14	
MLMA	30	15	
MLN	31	04	
MLNA	31	05	
MLR	30	06	
MLRA	30	07	
MLRM	30	16	
MLRMA	30	17	
MLRN	31	06	
MLRNA	31	07	
Mode Register Instructions:			
LDE	21	14	Load mode register bit from ACAR
LDE1	21	15	
LDEE1	21	16	
LDG	23	14	
LDH	23	15	
LDI	23	16	
LDJ	23	17	
SETE	25	14	Set mode register bit with result of logical function specified in instruction address field
SETE1	25	15	
SETF	25	16	
SETF1	25	17	
SETG	27	14	
SETH	27	15	
SETI	27	16	
SETJ	27	17	
MULT	22	13	For 32-bit mode, both halves enabled, multiply RGA by ADR contents; leave inner double-length product mantissa in RGA, outer in RGB
NEB	22	10	Test for RGA not equal to (ADR) in 8-bit bytes

Suffix	Meaning
A	Unsigned
M	Fixed point
N	Normalized
R	Rounded

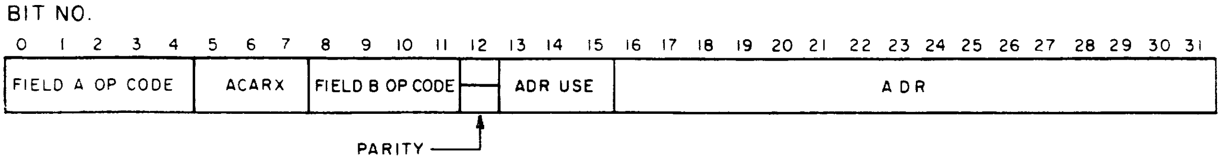
Mnemonic Code	Octal Op Code Field A 0:5	 Field B 8:4	Operation	
NORM	20	13	Normalize	
OFB	25	06	Overflow bits of previous 8-bit byte instruction are transmitted from RGC to RGB	
RT (G| L)			Route:	
RTG	24	13	Transmit register (Y) of every PE to RGR of PE number (N+D) modulo a, where Y = a specified PE register, N = initial PE No., D = routing distance, and a = number of PEs in array (64/128/256)	
RTL	24	12	Same as above, except single quadrant (a = 64)	
SB	36	04	Subtract (ADR) from RGA. Variants are:	
SBA	36	05	Suffix	Meaning
SBM	36	14	A	Unsigned
SBMA	36	15	M	Fixed point
SBN	37	04	N	Normalized
SBNA	37	05	R	Rounded
SBR	36	06		
SBRA	36	07		
SBRN	37	06		
SBRNA	37	07		
SBB	26	07	Subtract (ADR) from RGA in 8-bit bytes	
SBEX	25	01	Subtract exponent(s) of (ADR) from RGA exponent(s)	
SCM	21	04	Execute one cycle of a multiplication	
Shift Instructions:			Shift Variants are:	
RTAL	35	13	Variant	Meaning
RTAR	35	12		
SHABL	37	11	SH | RT	Shift | rotate
SHABR	37	10	A | AB	RGA | RGA + RGB (single | double)
SHABML	37	13		
SHABMR	37	12	_ | M	Full register | mantissa
SHAL	35	01	L | R	Left | right
SHAR	35	00		
SHAML	35	11		
SHAMR	35	10		

	Octal Op Code		
Mnemonic	Field A	Field B	
Code	0:5	8:4	
STA	26	12	Store from RGA to memory
STB	26	13	Store from RGB to memory
STR	26	14	Store from RGR to memory
STS	26	15	Store from RGS to memory
STX	26	16	Store from RGX to memory
SUB	26	05	Subtract 64-bit unsigned fixed point number (ADR) from RGA
SWAP	31	03	Interchange (RGA) and (RGB)
SWAPA	33	03	Interchange the inner and outer operands in RGA
SWAPX	37	03	Interchange the outer operand of RGA and the inner operand of RGB
T3A	21	05	Transfer contents of C register (RGC) to RGA
TCY*	31	00	Transfer data from CDB to MAR
TCYS*	31	01	Add RGS to CDB and store in MAR
TCYX*	31	02	Add RGX to CDB and store in MAR
Transmit Instructions:			Transmit source data to register indicated in op code. (Source is specified in bits 13:3 and 16:16.)
LDA	26	17	Transmit to RGA
LDB	27	00	Transmit to RGB
LDD	22	12	Transmit to RGD
LDR	27	01	Transmit to RGR
LDS	27	02	Transmit to RGS
LDX	27	03	Transmit to RGX
XD	25	03	Subtract (ADR) 48:16 from RGX
XI	25	02	Add (ADR) 48:16 to RGX

* These instructions, while they have separate operation codes, are actually portions of those instructions which reference main memory. As free-standing instructions, they set a value into the MAR, but the MAR contents are not accessible to the program.

FINST/PE INSTRUCTION DESCRIPTIONS

The remainder of this section consists of descriptions of the various FINST/PE instructions. These are arranged alphabetically according to instruction mnemonic, in the same order as presented in the instruction repertoire previously listed. Each description includes the mnemonic code, the operation performed, the bit contents for the specific instruction, a brief functional description and a flow chart of major operations performed during instruction execution. The word format for these instructions follows:



The general format used in the instruction descriptions is as shown below. Shaded fields are used to indicate irrelevant fields for specific instructions.

XX	ACARX	XX		ADR USE	ADR

The following notes generally apply to most FINST/PE instructions. The reader should familiarize himself with these notations before proceeding to the instruction descriptions.

1. Addressing and Indexing: Addressing and indexing, where applicable, are effected by the CU and PE in accordance with the settings of the ACARX, ADR USE, and ADR fields. Following is a simplified description of the addressing and indexing logic.

If bit 5 of the instruction (i. e., the first bit of the ACARX field) is set, the CU adds the contents of the ADR field to the ACAR which is specified in bits 6:2 of the PE instruction. If bit 5 is reset, no such add takes place. In either case, the resultant 16 bits (i. e. (ACAR (48:16)) + ADR (0:16) or ADR (0:16)) are loaded into the data queue (FDQ) and the proper times are transmitted to the PE via the CDB into the Address Adder (ADA). At the same time inputs are enabled into the ADA from RGS or RGX, as specified by the ADR USE field. The result is enabled into the Shift Count Register (SCR), Memory Address Register (MAR), or the RGX, depending on which instruction is being operated. Finally, the CU enables an LDB (LDR for the DV instruction) to fetch the operand, if the result was enabled into the MAR.

Relative to the use of these three fields (ACARX, ADR USE, and ADR), there are five classes of instructions, as follows:

Class 1: This class uses no addressing or indexing at all, since the source/destination/bit address is implied within the instruction. Instructions within this class are as follows: ASB, CLRA, COMPA, (I | J) (L | M) (O | Z), NORM, OFB, and SWAP (__| A | X).

Class 2: In this class of instruction, the source is a specified ACAR, and the destination and bit are implied by the FIELD A OP CODE and FIELD B OP CODE. Therefore the ACARX field is used, but ADR USE is disregarded (all bits are assumed to be off), and ADR is a DATA field. The instructions belonging to this class are LD (E|E1|EE1|G|H|I|J) and the special case of SET (E|E1|F|F1|G|H|I|J), where no bits are set in the B2 field. (See page 4-70 for an explanation of the LD_ and SET_ instructions.)

Class 3: The only instruction in this class is the RT(G | L). The ADR field contains the routing distance, and can be indexed by an ACAR. However, indexing by RGS or RGX is not permitted, and the ADR USE field is disregarded.

Class 4: This class includes the bit-oriented instructions (C|R|S) AB, (I|J) B, the rotate (RT) and shift (SH) instructions, and the PE store instructions. In these instructions the ADR field contains a bit number or shift count. ACAR, RGS, and RGX indexing is permitted (that is, bit 15 of the instruction word, which is the third bit of the ADR USE field, is ignored and assumed to be on).

Class 5: In the balance of the instruction set, normal addressing and indexing logic is used. The result is enabled into the MAR, RGB, or RGR.

2. E Bits: Mode register bits E and E1 in each PE are used to protect the contents of PE registers RGA and RGS and the PE fault bits F and F1 respectively, and also to prevent the PE from writing data into the PEM. PE register RGX contents are also protected by the E bit. In 32-bit mode, the E bit protects the outer word, and the E1 bit protects the inner word; either bit or both may be disabled. In 64-bit mode, since both the E and E1 bits act on only their respective half-words, it is advisable to set both bits to the same state, by programming, When E ≠ E1, all the logic for instruction execution operates as if E E1 = 11, but for the results, only certain bytes are loaded into RGA. (This could cause undefined results due to incorrect partial stores of intermediate results.) The E bit controls bytes 1, 6, 7, and 8 (the outer word) and E1 controls bytes 2, 3, 4, and 5 (the inner word). The following table shows applicable settings for the various data word formats.

64-Bit Mode:

E	E1	Full Word
0	0	Disabled
0	1	Partial word results; should generally be avoided
1	0	
1	1	Enabled

32-Bit Mode:

E	E1	Outer Word	Inner Word
0	0	Disabled	Disabled
0	1	Disabled	Enabled
1	0	Enabled	Disabled
1	1	Enabled	Enabled

The E and/or E1 bits are set/reset by the LD_ or SET_ instructions.

Note: The E and E1 bits do not protect PEM from transfers via the DC.

3. Zero: When arithmetic operations produce a zero result (i.e., as a result of exponent underflow, or, in normalized operation, if the resultant mantissa equals zero) it is always effected by setting all 64 (or 32) bits to zero. This is interpreted mathematically as a +0 mantissa and an exponent of $-(2^{14})$. In 32-bit mode the exponent is interpreted as $-(2^{6})$.

4. F Bits: Mode register bits F and F1 (fault bits) in each PE are used to indicate the following conditions:

   a. Exponent overflow.

   b. Overflow of the mantissa magnitude, in mantissa-sized fixed-point arithmetic operations.

   c. Zero or unnormalized divisor.

   d. Exponent underflow, if ACR bit 9 (Exponent Underflow Inhibit bit) is reset, and if, in normalized operations, the resultant mantissa is non-zero.

The F bit indicates a fault in 64-bit mode, or a fault for the outer half-word in 32-bit mode, while the F1 bit indicates a fault for the inner half-word in 32-bit mode.

Setting of the F and F1 bits as a result of one of the above-mentioned conditions is dependent on the setting of the E and E1 bits as stated in (2) above, and these conditions can cause only the setting of the appropriate bit(s), but not the resetting of the bit(s). However, the F and F1 bits can be set and reset programmatically by the SETF and SETF1 instructions, which are not E-bit sensitive.

5. Exponent Overflow: When exponent overflow occurs as a result of an arithmetic operation, the resultant exponent is the true exponent modulo 2^{14} (64-bit mode) or 2^{6} (32-bit mode). The resultant mantissa will be correct unless the exponent overflow occurred in response to normalizing a mantissa overflow.

6. Exponent Underflow: When exponent underflow occurs as the result of an arithmetic operation, all bits of the underflowed resultant word (half-word) are set to zero. (See (3) above for the mathematical interpretation of zero.)

7. Mantissa Overflow: If neither the M nor N variants are used on arithmetic instructions (unnormalized floating point), a mantissa overflow is adjusted by shifting the mantissa right one place and adding one to the exponent. Rounding, if specified, is not affected by this operation.

8. Fixed-point Operands: When using the M (fixed-point) variant on arithmetic instructions, the exponent field in both operands is ignored. Therefore the exponent field of the result will be unchanged.

9. Normalizing: When using the N (normalized) variant on arithmetic instructions, the input operands are assumed to be normalized for ML and DV, although they need not be for AD or SB. Inputs to MULT are also assumed normalized.

MNEMONIC CODE: AD

OPERATION: Add (ADR) to RGA (Additional op codes allow certain variants:
A for unsigned, where the sign of RGA is unchanged by the operation,
M for fixed point, N for normalized floating, R for rounded)

INSTRUCTION WORDS:

AD	34	ACARX	04	—	ADR USE	ADR
	0 4	5 7	9 11	12	13 15	16 31
ADA	34	ACARX	05	—	ADR USE	ADR
ADM	34	ACARX	14	—	ADR USE	ADR
ADMA	34	ACARX	15	—	ADR USE	ADR
ADN	35	ACARX	04	—	ADR USE	ADR
ADNA	35	ACARX	05	—	ADR USE	ADR
ADR	34	ACARX	06	—	ADR USE	ADR
ADRA	34	ACARX	07	—	ADR USE	ADR
ADRN	35	ACARX	06	—	ADR USE	ADR
ADRNA	35	ACARX	07	—	ADR USE	ADR

DESCRIPTION: This instruction adds (ADR) to RGA. The result of the addition remains in RGA. RGB will contain (ADR) either unmodified, or modified by the mantissa portion(s) that were shifted to align with RGA. RGC and RGR are not used.

This instruction may cause the F bit(s) to be set. (Refer to pages 4-15, 4-16, items 4, 5, and 6.)

The following variants are permitted:

No suffix - Both operands are treated as floating point, the result is not normalized

A - Both operands are treated as unsigned values;

M - Both operands are treated as fixed-point and the result is in fixed-point;

N - Result is normalized (after rounding, if specified);

R - Result is rounded in RGA.

If neither M nor N is specified, the operation is an unnormalized floating point operation. In this case, exponent and mantissa will be adjusted for any mantissa overflow, but leading zeros in the result mantissa will not be disturbed.

If both E bits are disabled, RGA will remain unchanged. If E ≠ E1 for 64-bit mode, the results, for purposes of this manual, are undefined.

In 32-bit mode, there is no loss of accuracy because each half-word is aligned independently of the other.

Note: The addition is effected as follows (assuming that the E bits are enabled):

1. The exponent of the result is determined as the larger of the two operand exponents.

2. The mantissa of the operand with the smaller exponent is shifted right end-off until it is aligned with the mantissa of the operand with the greater exponent. Thus, if the difference between the two exponents is greater than 47, the smaller mantissa will be set to zero, and the result of the add will exactly equal the larger value.

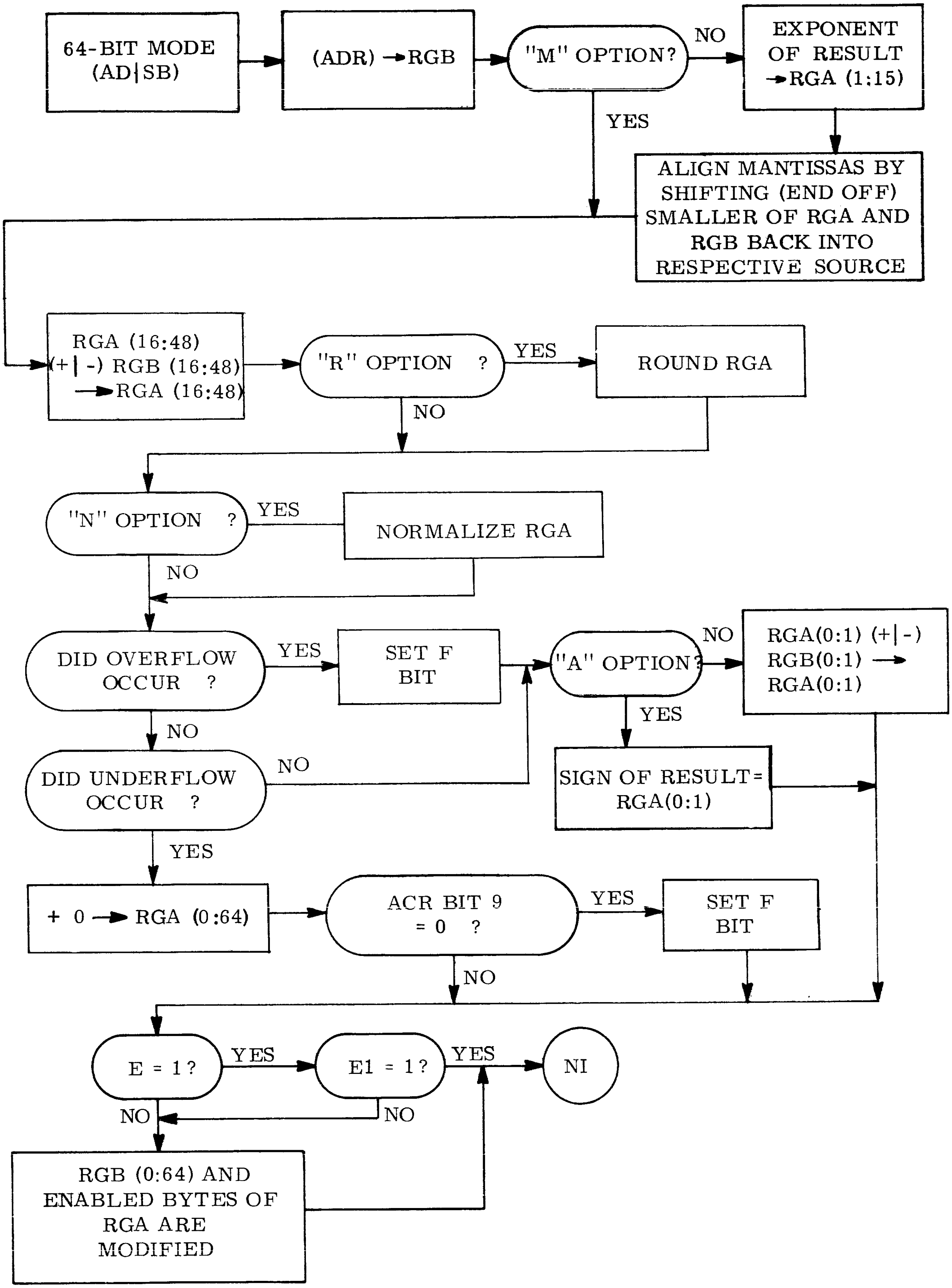
3. The aligned mantissa is returned to its source register (RGA or RGB).

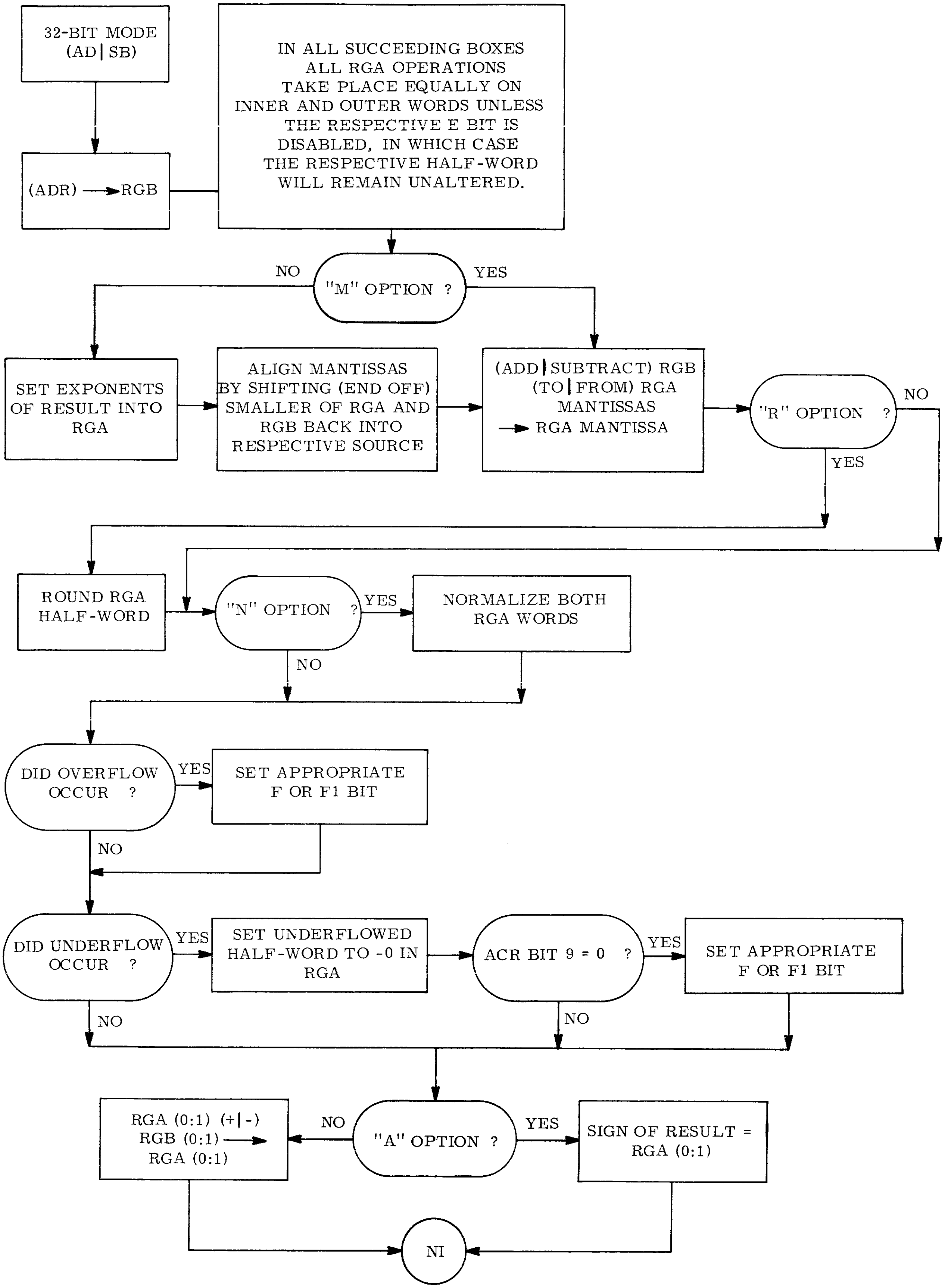
4. The mantissas are added, and the result stored in RGA.

5. The exponent portion of RGB will be set to the exponent correction whether the normalized variant (N) is requested or not.

6. The mantissa portion of RGB will be unchanged unless the RGB exponent is smaller than the RGA exponent. (Refer to paragraphs 2 and 3 above.)

FLOW CHARTS: See the next two pages for 64- and 32-bit modes respectively.


64-BIT MODE (AD|SB)
(ADR) → RGB
"M" OPTION?
NO
YES
EXPONENT OF RESULT → RGA (1:15)
ALIGN MANTISSAS BY SHIFTING (END OFF) SMALLER OF RGA AND RGB BACK INTO RESPECTIVE SOURCE
RGA (16:48) (+|-) RGB (16:48) → RGA (16:48)
"R" OPTION ?
YES
ROUND RGA
NO
"N" OPTION ?
YES
NORMALIZE RGA
NO
DID OVERFLOW OCCUR ?
YES
SET F BIT
"A" OPTION?
NO
RGA(0:1) (+|-) RGB(0:1) → RGA(0:1)
YES
SIGN OF RESULT= RGA(0:1)
NO
DID UNDERFLOW OCCUR ?
NO
YES
+ 0 → RGA (0:64)
ACR BIT 9 = 0 ?
YES
SET F BIT
NO
E = 1?
YES
E1 = 1?
YES
NI
NO
NO
RGB (0:64) AND ENABLED BYTES OF RGA ARE MODIFIED


32-BIT MODE (AD | SB)
(ADR) → RGB
IN ALL SUCCEEDING BOXES ALL RGA OPERATIONS TAKE PLACE EQUALLY ON INNER AND OUTER WORDS UNLESS THE RESPECTIVE E BIT IS DISABLED, IN WHICH CASE THE RESPECTIVE HALF-WORD WILL REMAIN UNALTERED.
"M" OPTION ?
NO
YES
SET EXPONENTS OF RESULT INTO RGA
ALIGN MANTISSAS BY SHIFTING (END OFF) SMALLER OF RGA AND RGB BACK INTO RESPECTIVE SOURCE
(ADD | SUBTRACT) RGB (TO | FROM) RGA MANTISSAS → RGA MANTISSA
"R" OPTION ?
NO
YES
ROUND RGA HALF-WORD
"N" OPTION ?
YES
NO
NORMALIZE BOTH RGA WORDS
DID OVERFLOW OCCUR ?
YES
NO
SET APPROPRIATE F OR F1 BIT
DID UNDERFLOW OCCUR ?
YES
NO
SET UNDERFLOWED HALF-WORD TO -0 IN RGA
ACR BIT 9 = 0 ?
YES
NO
SET APPROPRIATE F OR F1 BIT
RGA (0:1) (+ | -) RGB (0:1) → RGA (0:1)
NO
"A" OPTION ?
YES
SIGN OF RESULT = RGA (0:1)
NI

MNEMONIC CODE: ADB

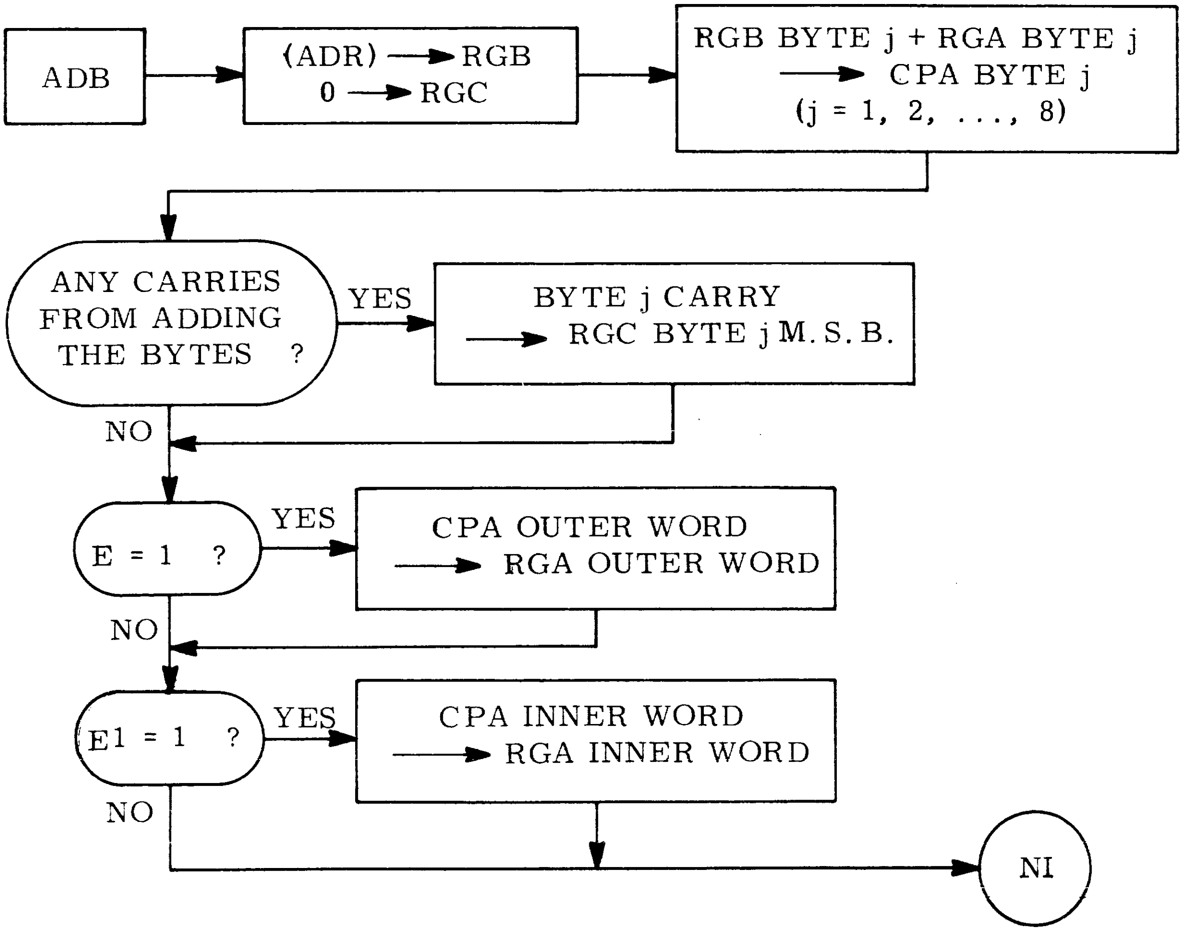
OPERATION: Add (ADR) to RGA in 8-bit bytes

INSTRUCTION WORD:

26	ACARX	06	—	ADR USE	ADR
0 4	5 7	8 11	12	13 15	16 31

DESCRIPTION: This instruction adds (ADR) to RGA in 8-bit bytes. Each group of eight bits is an unsigned fixed-point number. The result of the addition is placed in RGA. The carries out of the 8-bit bytes are stored in RGC. (ADR) remains in RGB. When the E bits are disabled, RGA is unchanged but RGC contains the carries. Execution of this instruction is the same for 64- and 32-bit modes.

FLOW CHART:


ADB
(ADR) ⟶ RGB
0 ⟶ RGC
RGB BYTE j + RGA BYTE j ⟶ CPA BYTE j (j = 1, 2, . . . , 8)
ANY CARRIES FROM ADDING THE BYTES ?
YES
BYTE j CARRY ⟶ RGC BYTE j M.S.B.
NO
E = 1 ?
YES
CPA OUTER WORD ⟶ RGA OUTER WORD
NO
E1 = 1 ?
YES
CPA INNER WORD ⟶ RGA INNER WORD
NO
NI

MNEMONIC CODE: ADD

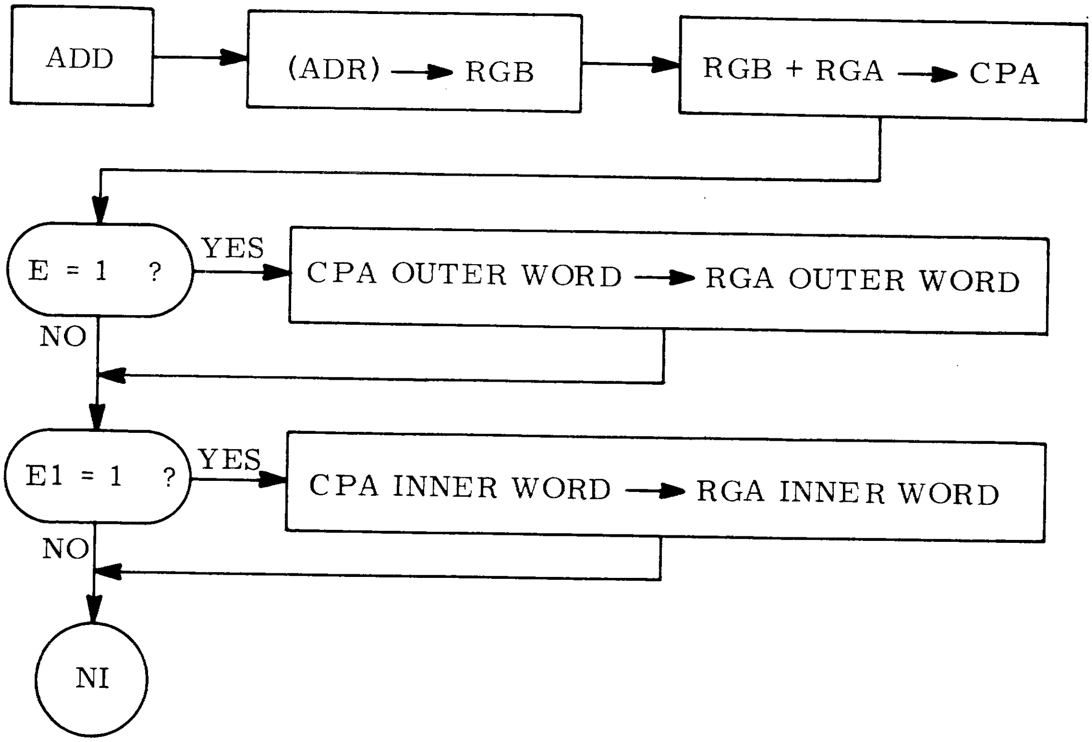
OPERATION: Add 64-bit unsigned fixed-point numbers (ADR) to RGA.

INSTRUCTION WORD:

26	ACARX	04		ADR USE	ADR
0 4	5 7	8 11	12	13 15	16 31

DESCRIPTION: This instruction adds 64-bit unsigned fixed-point numbers (ADR) and RGA. The result of the addition is placed in RGA if the E bits permit. Overflow generates an end-around carry, but does not set the F bit. (ADR) remains in RGB at the completion of instruction execution. When the E bits are disabled, RGA is unchanged. When operating in 32-bit mode, this instruction operates as if in 64-bit mode.

FLOW CHART:


ADD
(ADR) → RGB
RGB + RGA → CPA
E = 1 ?
YES
CPA OUTER WORD → RGA OUTER WORD
NO
E1 = 1 ?
YES
CPA INNER WORD → RGA INNER WORD
NO
NI

MNEMONIC CODE: ADEX

OPERATION: Add (ADR) exponent field(s) to RGA exponent(s)

INSTRUCTION WORD:

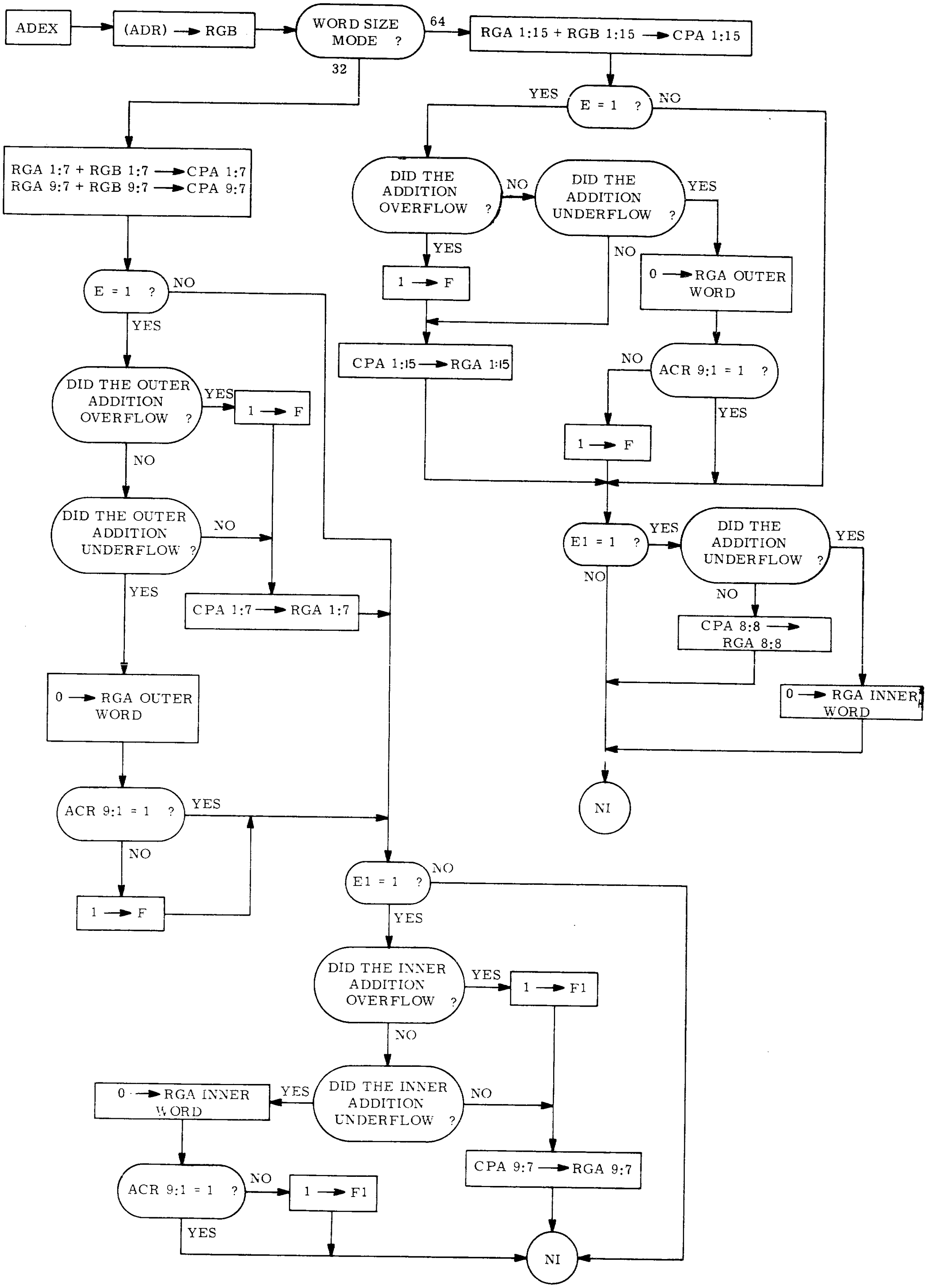
25	ACARX	00	—	ADR USE	ADR
0 4	5 7	8 11	12	13 15	16 31

DESCRIPTION: In 64-bit mode this instruction adds the exponent field of (ADR) to the exponent of RGA. This addition treats these quantities as exponents, not as binary numbers. The sign bit and mantissa field are unchanged unless underflow occurs, in which case RGA is cleared to all zeros. When the E and E1 bits are disabled, RGA is unchanged.

In 32-bit mode the instruction is performed on the inner and outer exponents independently.

Exponent overflow or underflow may cause the setting of the F bits (see pages 4-15, 4-16, items 4, 5, and 6.).

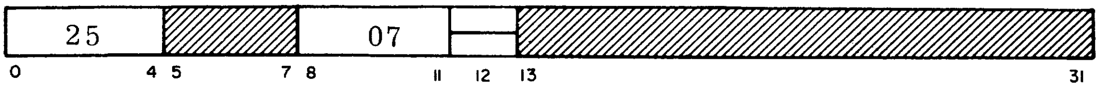
FLOW CHART: See next page.


ADEX
(ADR) → RGB
WORD SIZE MODE ?
64
32
RGA 1:15 + RGB 1:15 → CPA 1:15
E = 1 ?
YES
NO
DID THE ADDITION OVERFLOW ?
NO
DID THE ADDITION UNDERFLOW ?
YES
YES
1 → F
NO
0 → RGA OUTER WORD
CPA 1:15 → RGA 1:15
NO
ACR 9:1 = 1 ?
YES
1 → F
E1 = 1 ?
YES
DID THE ADDITION UNDERFLOW ?
YES
NO
NO
CPA 8:8 → RGA 8:8
0 → RGA INNER WORD
NI
RGA 1:7 + RGB 1:7 → CPA 1:7
RGA 9:7 + RGB 9:7 → CPA 9:7
E = 1 ?
NO
YES
DID THE OUTER ADDITION OVERFLOW ?
YES
1 → F
NO
DID THE OUTER ADDITION UNDERFLOW ?
NO
YES
CPA 1:7 → RGA 1:7
0 → RGA OUTER WORD
ACR 9:1 = 1 ?
YES
NO
1 → F
E1 = 1 ?
NO
YES
DID THE INNER ADDITION OVERFLOW ?
YES
1 → F1
NO
DID THE INNER ADDITION UNDERFLOW ?
YES
NO
0 → RGA INNER WORD
CPA 9:7 → RGA 9:7
ACR 9:1 = 1 ?
NO
1 → F1
YES
NI

MNEMONIC CODE: ASB

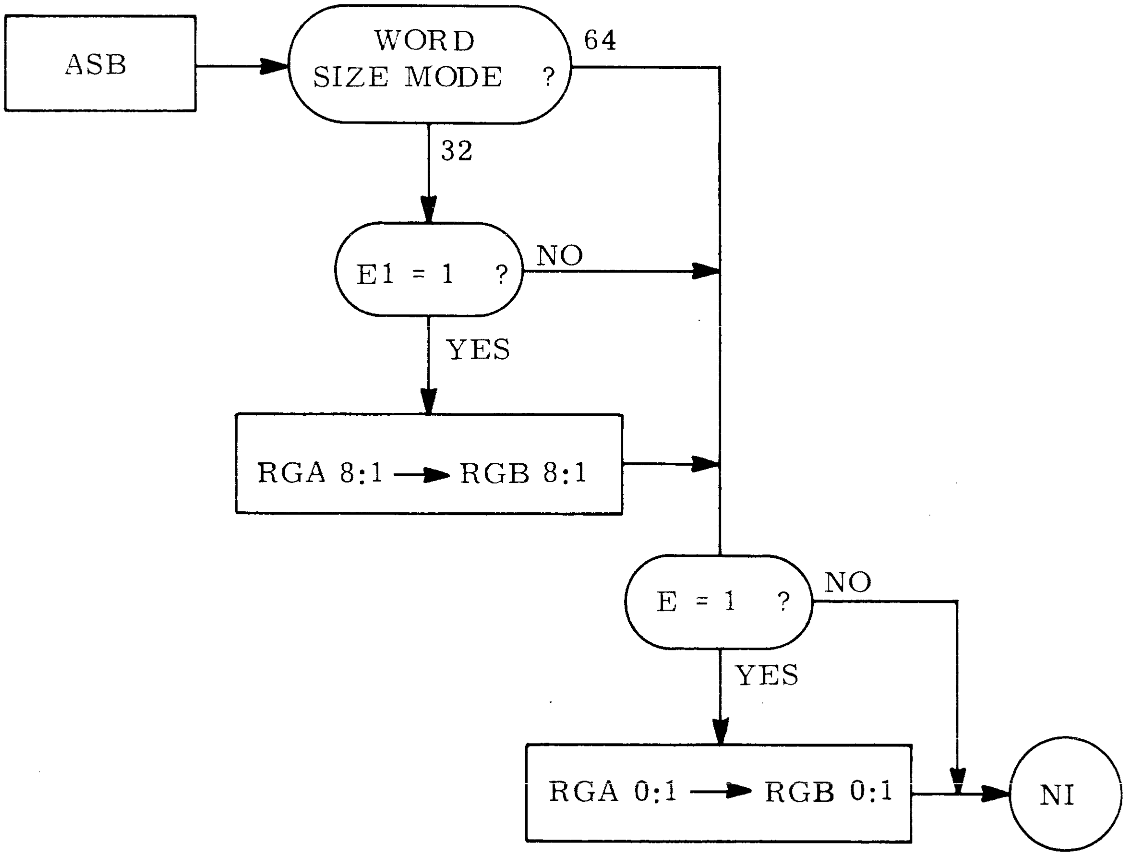
OPERATION: Place the sign(s) of RGA into the sign(s) of RGB.

INSTRUCTION WORD:



DESCRIPTION: This instruction places the mantissa sign bit(s) of RGA into the sign bit(s) of RGB.

FLOW CHART:



BOOLEAN OPERATIONS

MNEMONIC CODE: AND

OPERATIONS: Logical AND of (ADR) with RGA | $\overline{\text{(ADR)}}$ with RGA | ADR with $\overline{\text{RGA}}$ | $\overline{\text{(ADR)}}$ with $\overline{\text{RGA}}$.

INSTRUCTION WORDS:

AND	27	ACARX	04	—	ADR USE	ADR
	0 4	5 7	8 11	12	13 15	16 31
ANDN	27	ACARX	06	—	ADR USE	ADR
NAND	27	ACARX	05	—	ADR USE	ADR
NANDN	27	ACARX	07	—	ADR USE	ADR

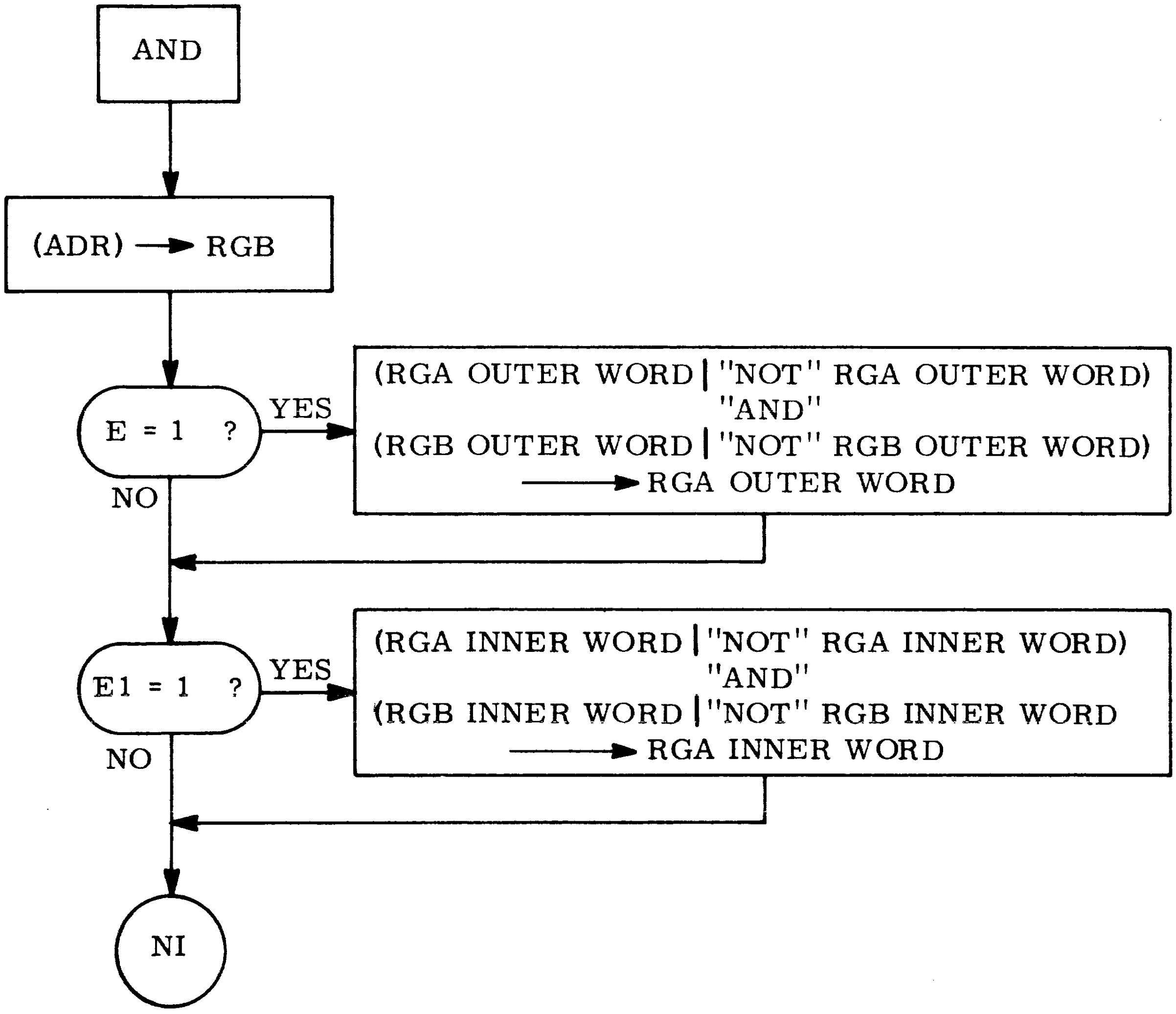
DESCRIPTION: These instructions perform the logical AND of (ADR) with RGA; either operand may be true, or complemented (shown by "N" as prefix and/or suffix in mnemonic code), giving four combinations for this function. The result of the indicated operation is placed in RGA. The (ADR) is first fetched to RGB. The bit-by-bit determination of the result is in accordance with the following table:

Result of ("A" | NOT "A") "AND" ("B" | NOT "B")

"A" Operand Bit	"B" Operand Bit	AND	ANDN	NAND	NANDN
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

This operation is the same for both the 64-bit and 32-bit modes. (ADR) remains in RGB. When the E bits are disabled, RGA is unchanged.

FLOW CHART:


AND
(ADR) → RGB
E = 1 ?
YES
NO
(RGA OUTER WORD | "NOT" RGA OUTER WORD)
"AND"
(RGB OUTER WORD | "NOT" RGB OUTER WORD)
→ RGA OUTER WORD
E1 = 1 ?
YES
NO
(RGA INNER WORD | "NOT" RGA INNER WORD)
"AND"
(RGB INNER WORD | "NOT" RGB INNER WORD
→ RGA INNER WORD
NI

BOOLEAN OPERATIONS (Continued)

MNEMONIC CODE: EOR

OPERATION: Logical exclusive-OR of RGA with (ADR) into RGA

INSTRUCTION WORD:

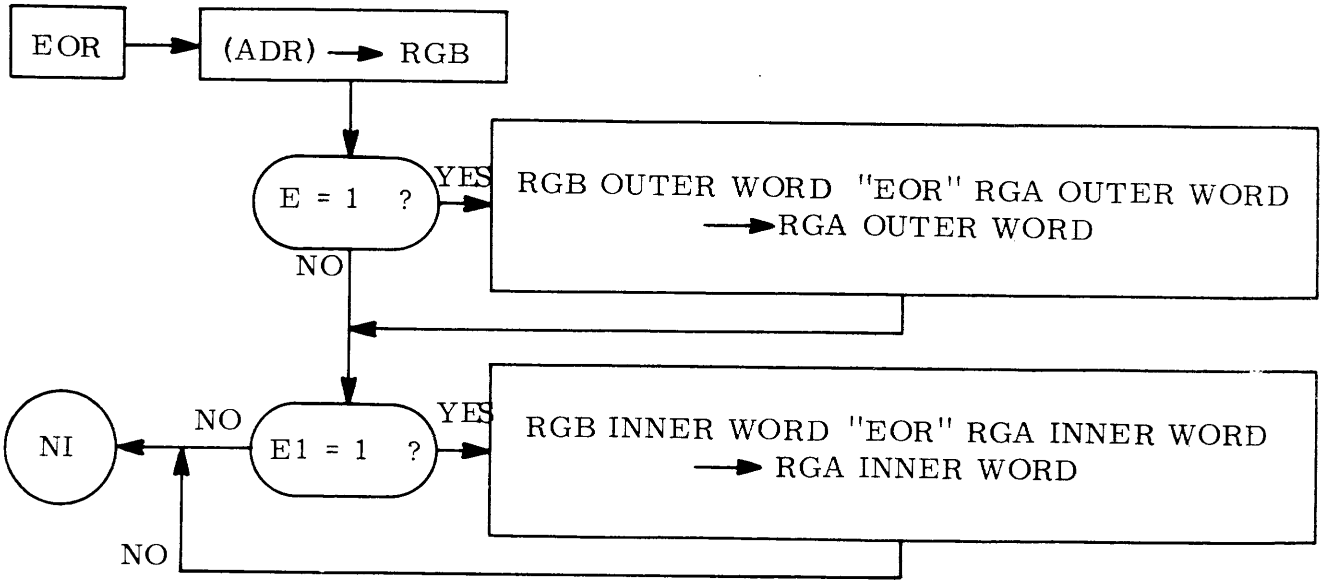
25	ACARX	05		ADR USE	ADR
0 4	5 7	8 11	12	13 15	16 31

DESCRIPTION: This instruction performs the logical exclusive-OR of (ADR) with RGA. The (ADR) is first fetched to RGB. The 64-bit adder is used. The result is loaded into RGA, as determined by the rule shown in the truth table below. (ADR) remains in RGB. This instruction is the same for 64- and 32-bit modes. When the E bits are disabled, RGA is unchanged.

TRUTH TABLE

"B" Operand Bit \ "A" Operand Bit	0	1
0	0	1
1	1	0

FLOW CHART:


EOR
(ADR) → RGB
E = 1 ?
YES
NO
RGB OUTER WORD "EOR" RGA OUTER WORD → RGA OUTER WORD
NI
NO
E1 = 1 ?
YES
RGB INNER WORD "EOR" RGA INNER WORD → RGA INNER WORD
NO

BOOLEAN OPERATIONS (Continued)

MNEMONIC CODE: EQV

OPERATION: Logical equivalence of (ADR) with RGA into RGA

INSTRUCTION WORD:

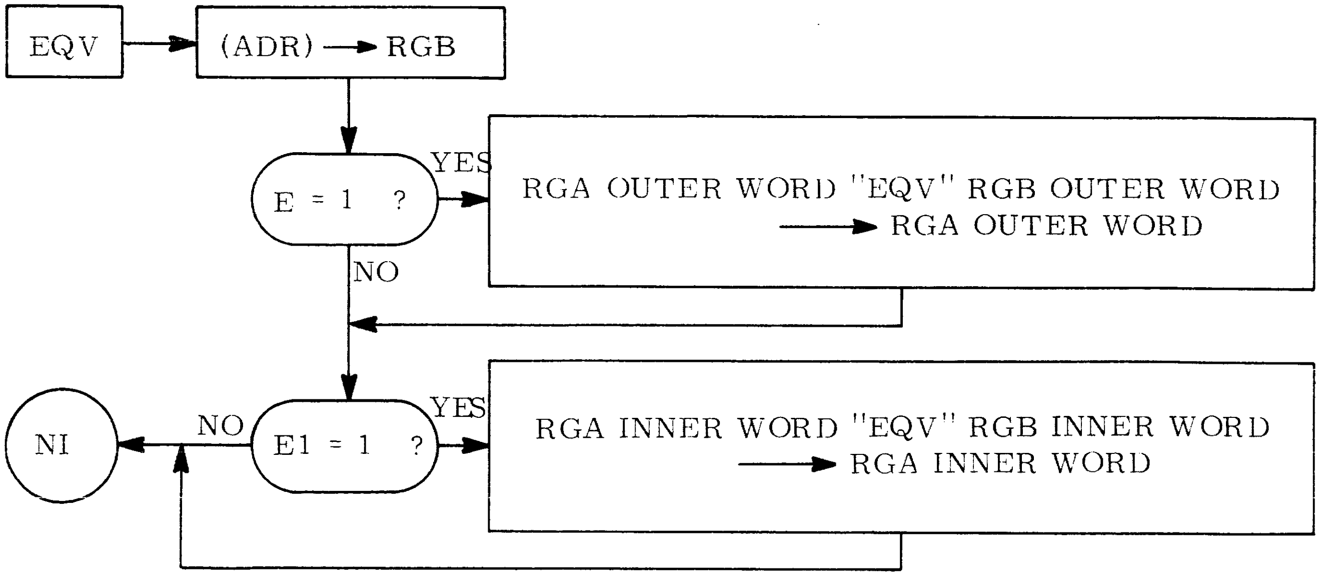
25	ACARX	04		ADR USE	ADR
0 4	5 7	8 11	12	13 15	16 31

DESCRIPTION: This instruction performs the logical equivalence of (ADR) with RGA. The (ADR) is fetched to RGB. The result of the operation is placed in RGA, as determined by the rule shown in the truth table below. (ADR) remains in RGB. EQV is the same for 64- and 32-bit modes. When the E bits are disabled, RGA is unchanged.

TRUTH TABLE

"B" Operand Bit	"A" Operand Bit 0	"A" Operand Bit 1
0	1	0
1	0	1

FLOW CHART:


EQV
(ADR) → RGB
E = 1 ?
YES
RGA OUTER WORD "EQV" RGB OUTER WORD → RGA OUTER WORD
NO
E1 = 1 ?
YES
RGA INNER WORD "EQV" RGB INNER WORD → RGA INNER WORD
NO
NI

BOOLEAN OPERATIONS (Continued)

MNEMONIC CODE: OR

OPERATION: Logical OR of (ADR) with RGA | $\overline{\text{RGA}}$ with (ADR) |
$\overline{\text{(ADR)}}$ with RGA | $\overline{\text{(ADR)}}$ with $\overline{\text{RGA}}$

INSTRUCTION WORDS:

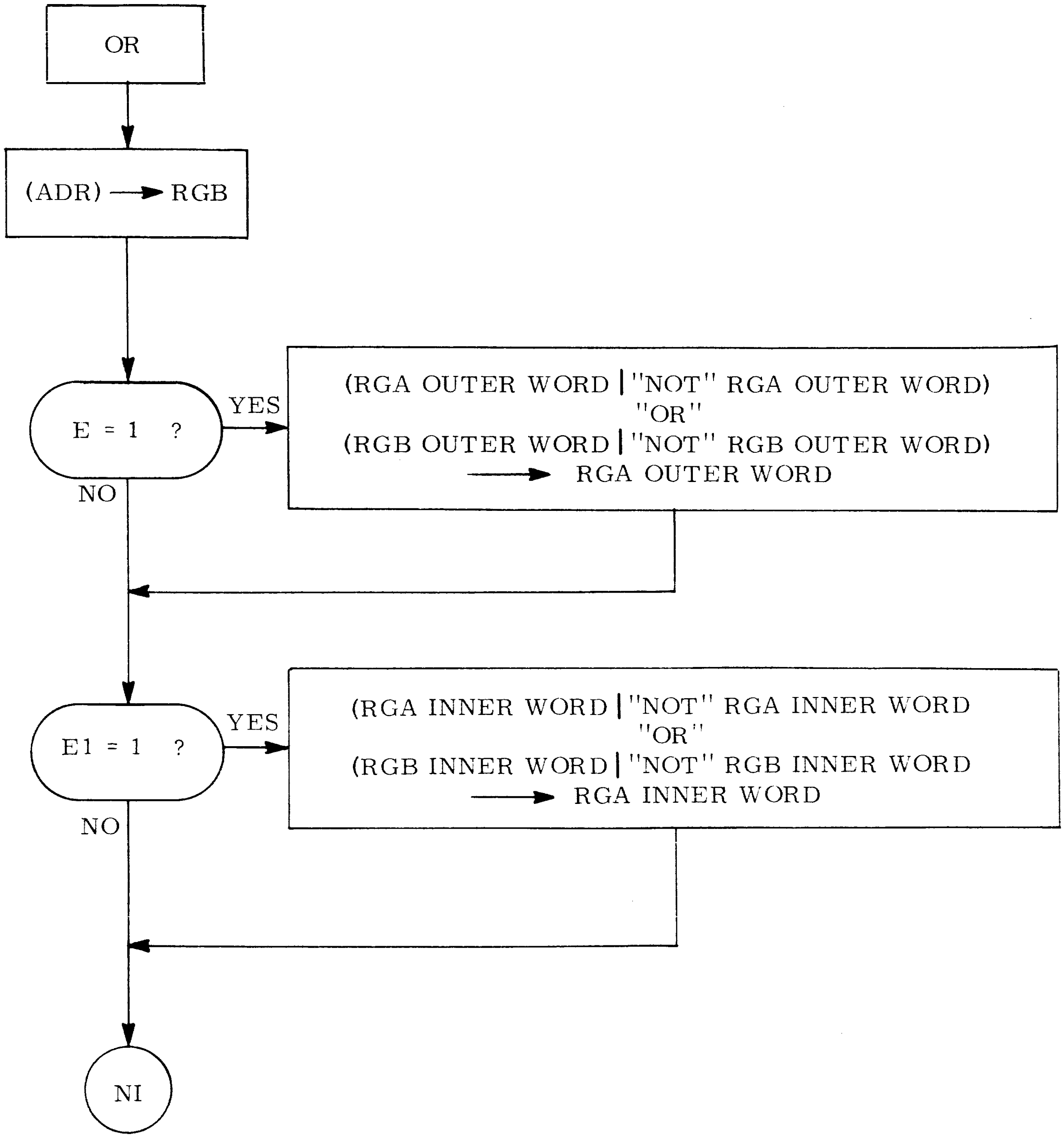
NOR	23	ACARX	05		ADR USE	ADR
	0 4	5 7	8 11	12	13 15	16 31
NORN	23	ACARX	07		ADR USE	ADR
OR	23	ACARX	04		ADR USE	ADR
ORN	23	ACARX	06		ADR USE	ADR

DESCRIPTION: This instruction is the same for 32- and 64-bit modes and performs the logical OR of (ADR) with RGA. Either operand may be true, or complemented ("N"), as indicated by the op code, giving four combinations for this Boolean function. The bit-by-bit determination of the result is in accordance with the following table.

Result of ("A" | NOT "A") OR ("B" | NOT "B")

"A" Operand Bit	"B" Operand Bit	NOR	NORN	OR	ORN
0	0	1	1	0	1
0	1	1	1	1	0
1	0	0	1	1	1
1	1	1	0	1	1

FLOW CHART: See next page.


OR
(ADR) ⟶ RGB
E = 1 ?
YES
NO
(RGA OUTER WORD | "NOT" RGA OUTER WORD)
"OR"
(RGB OUTER WORD | "NOT" RGB OUTER WORD)
⟶ RGA OUTER WORD
E1 = 1 ?
YES
NO
(RGA INNER WORD | "NOT" RGA INNER WORD
"OR"
(RGB INNER WORD | "NOT" RGB INNER WORD
⟶ RGA INNER WORD
NI

CHANGE RGA BIT(S)

MNEMONIC CODE: CAB

OPERATION: Complement RGA bit(s) in position(s) specified by address field

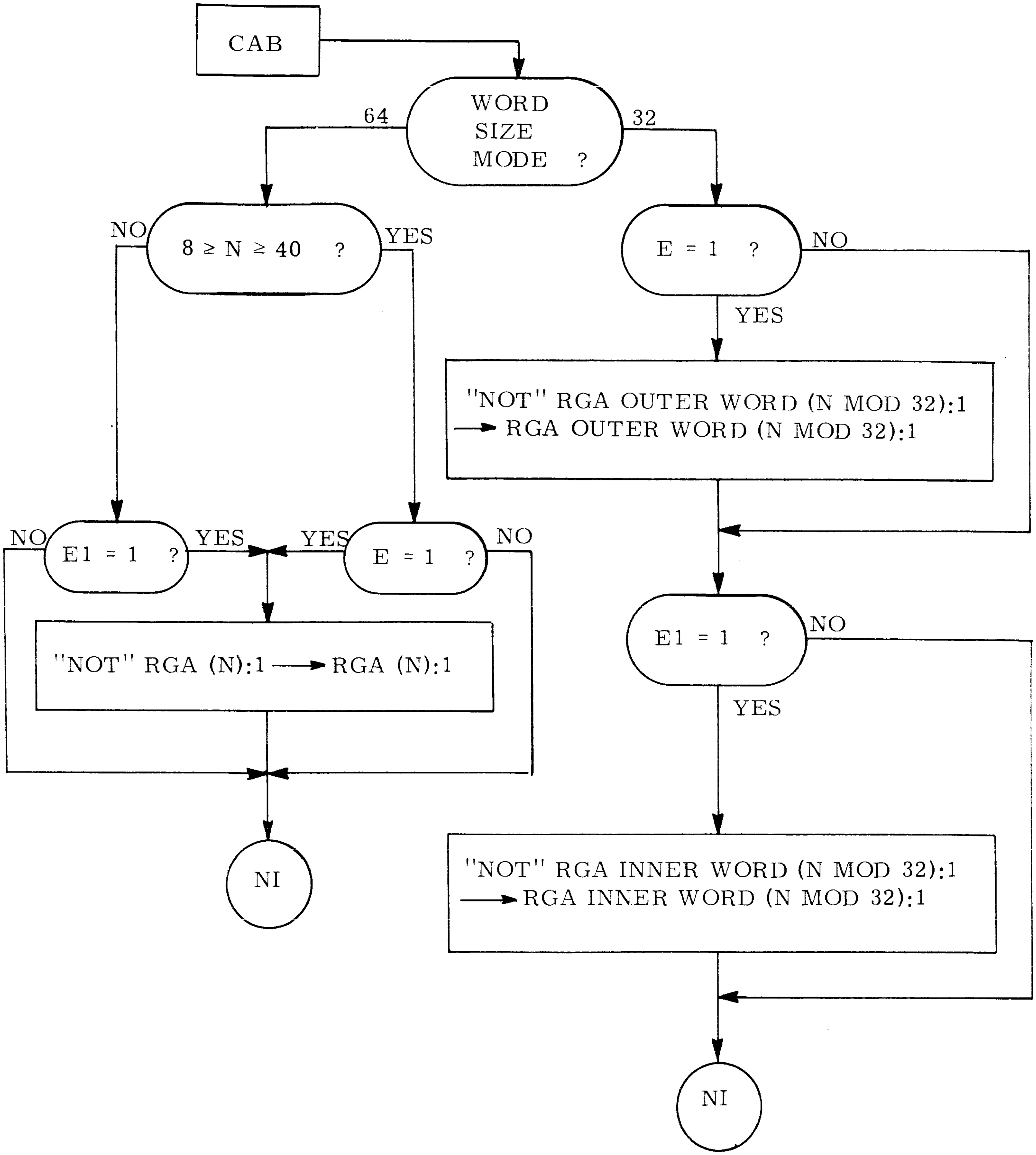
INSTRUCTION WORD:

37	ACARX	00	—	ADR USE	ADR
0 4	5 7	8 11	12	13 15	16 31

DESCRIPTION: In 64-bit mode, one bit in RGA is complemented; in 32-bit mode, two bits are complemented. The bit position is specified in the address field (indexed if desired), and the resultant six least significant bits are treated as a 6-bit coded number, N. In 32-bit mode, the corresponding bits in both the inner and outer words are changed.

When "N plus index" is greater than 63, the bit position is modulo 64 for 64-bit mode; in 32-bit mode, the bit position is modulo 32. A mask, with a "one" in the affected bit(s) and "zeros" in the remaining bits is retained in RGB at the end of instruction execution. When E bits are disabled, RGA is unchanged.

FLOW CHART: See next page.


CAB
WORD SIZE MODE ?
64
32
8 ≥ N ≥ 40 ?
NO
YES
E = 1 ?
NO
YES
"NOT" RGA OUTER WORD (N MOD 32):1 ⟶ RGA OUTER WORD (N MOD 32):1
E1 = 1 ?
NO
YES
YES
E = 1 ?
NO
"NOT" RGA (N):1 ⟶ RGA (N):1
NI
E1 = 1 ?
NO
YES
"NOT" RGA INNER WORD (N MOD 32):1 ⟶ RGA INNER WORD (N MOD 32):1
NI

CHANGE RGA BIT(S): (Continued)

MNEMONIC CODE: CHSA

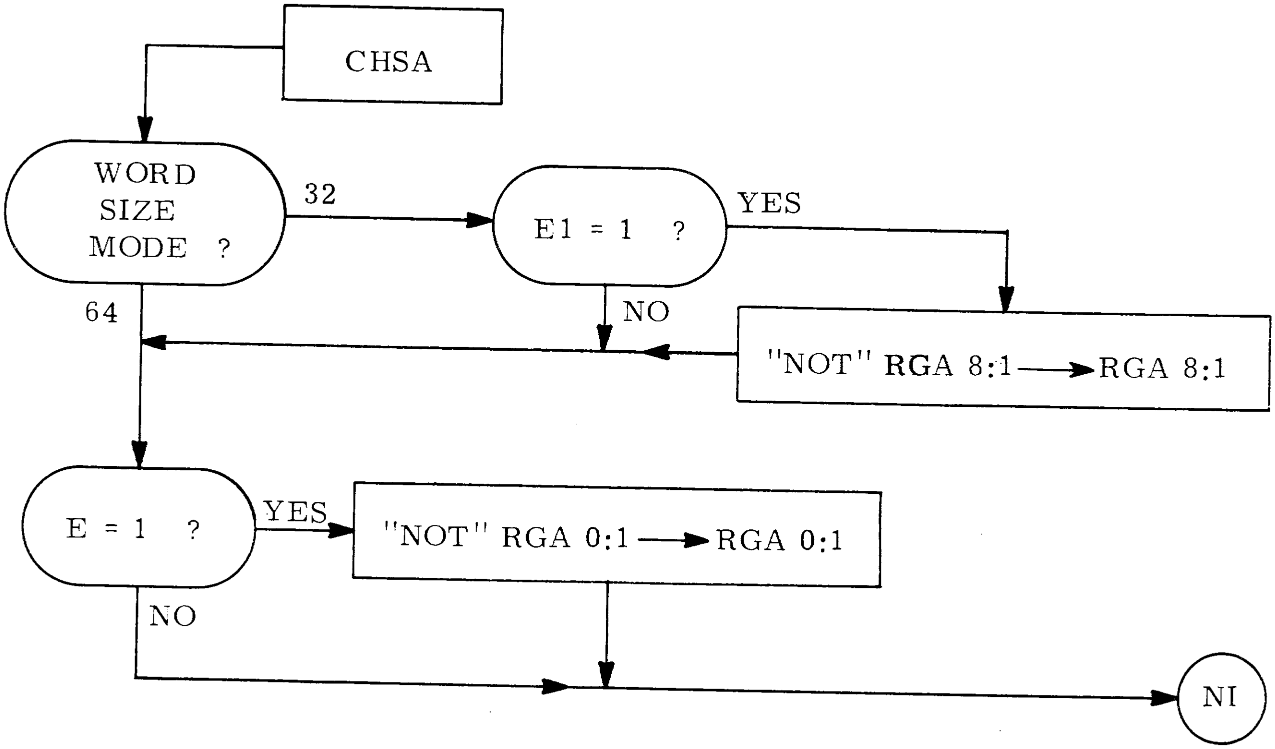
OPERATION: Change sign(s) in RGA

INSTRUCTION WORD:

0 4	5 7	8 11	12	13 31
37	0——0	00		0——0

DESCRIPTION: In 64-bit mode, RGA 0:1 is complemented. In 32-bit mode, the signs of both inner and outer words are complemented. When the E bits are disabled, the appropriate section of RGA remains unchanged. This is the CAB instruction, with the value of N equal to zero.

FLOW CHART:



CHANGE RGA BIT(S) (Continued)

MNEMONIC CODE: (R | S) AB

OPERATIONS: Reset | Set RGA bit(s) in position(s) specified by address field

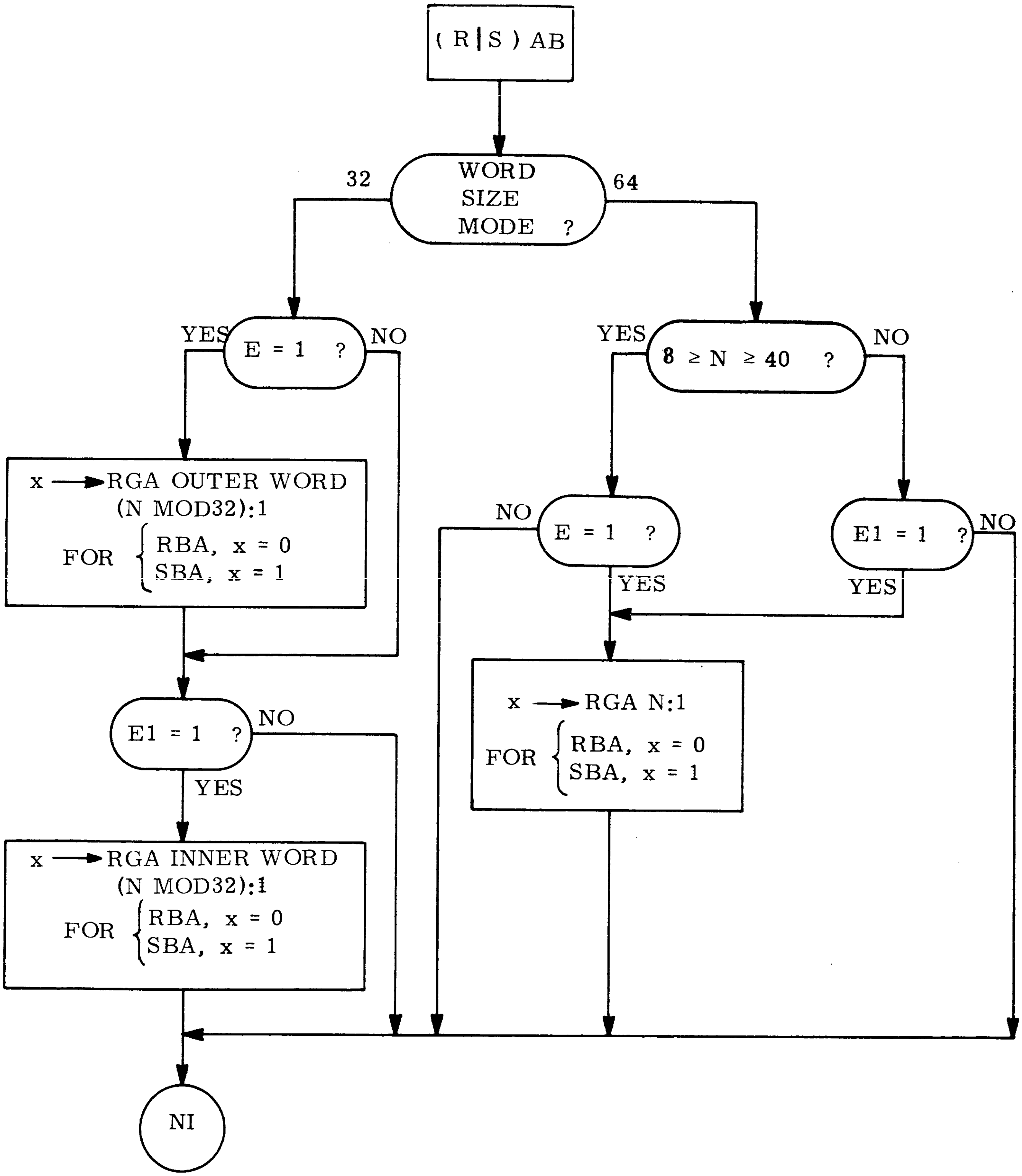
INSTRUCTION WORDS:

RAB	37	ACARX	01		ADR USE	ADR
	0 4	5 7	8 11	12	13 15	16 31
SAB	37	ACARX	02		ADR USE	ADR

DESCRIPTION: One bit in RGA is either reset (R) or set (S) in 64-bit mode; two bits are affected in 32-bit mode. The bit position is specified in the address field (indexed if desired) and is treated as a 6-bit coded number, N. In 32-bit mode, the corresponding bits in both the inner and outer words are affected.

When "N plus index" is greater than 63, the affected bit position is modulo 64; in 32-bit mode, it is modulo 32. A mask is then formed in RGB by inserting a "one" into the specified bit position(s). For SAB, the mask is ORed with RGA. For RAB, the complement of the mask is ANDed with RGA. In 32-bit mode, the mask has a "one" in both inner and outer words. The mask remains in RCB. When the E bits are disabled, RGA is unchanged.

FLOW CHART: See next page.


(R | S) AB
WORD SIZE MODE ?
32
64
E = 1 ?
YES
NO
8 ≥ N ≥ 40 ?
YES
NO
x ⟶ RGA OUTER WORD (N MOD32):1
FOR { RBA, x = 0 / SBA, x = 1
E = 1 ?
NO
YES
E1 = 1 ?
NO
YES
x ⟶ RGA N:1
FOR { RBA, x = 0 / SBA, x = 1
E1 = 1 ?
NO
YES
x ⟶ RGA INNER WORD (N MOD32):1
FOR { RBA, x = 0 / SBA, x = 1
NI

CHANGE RGA BIT(S) (Continued)

MNEMONIC CODE: SA (N | P)

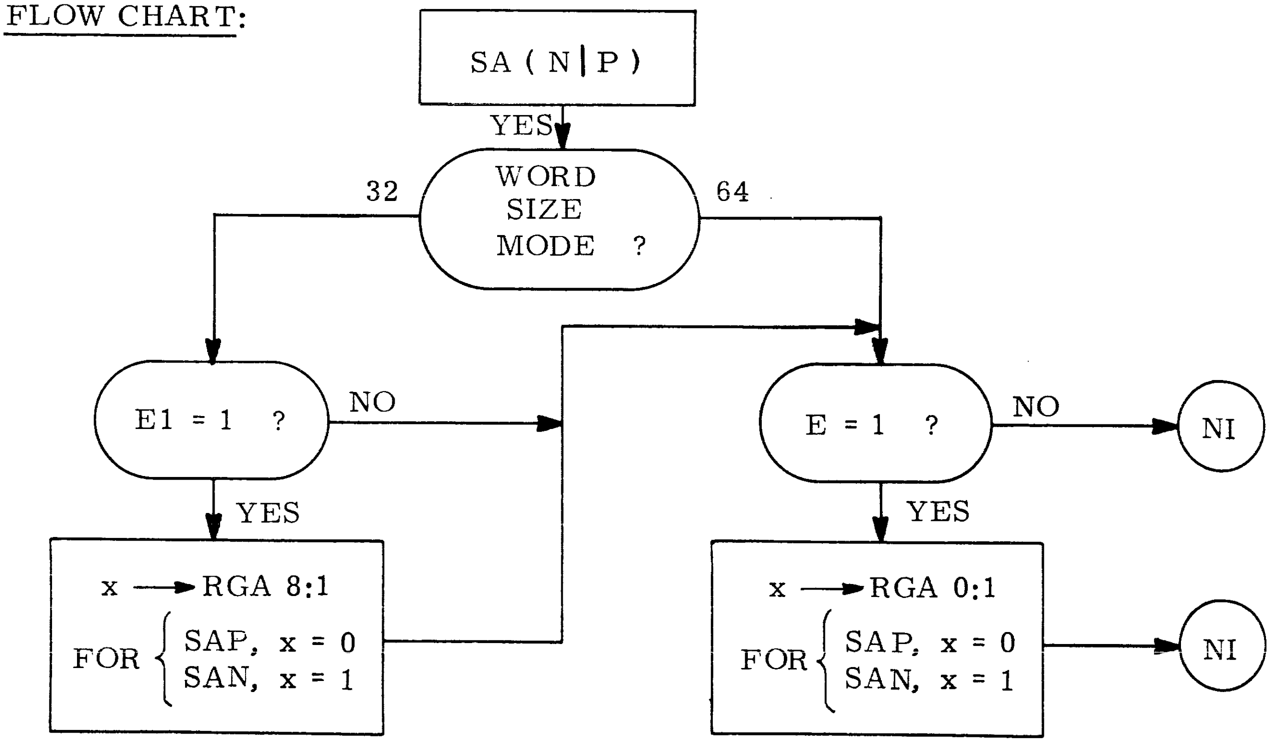
OPERATION: Reset | Set RGA sign bit(s)

INSTRUCTION WORDS:

	0–4	5–7	8–11	12	13–31
SAP	37	0——0	01	——	0——0
SAN	37	0——0	02	——	0——0

DESCRIPTION: In order to set (SAN) or reset (SAP) the sign bit, the SAB or RAB instruction is used, with the value of N equal to zero. This sets | resets RGA 0:1 in 64-bit mode; both RGA 0:1 and RGA 8:1 are affected in 32-bit mode.

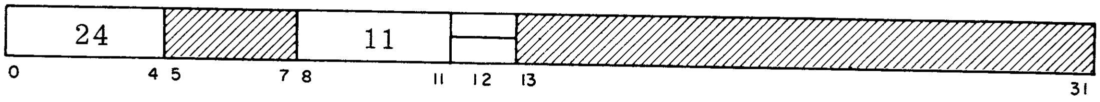
FLOW CHART:



MNEMONIC CODE: CLRA

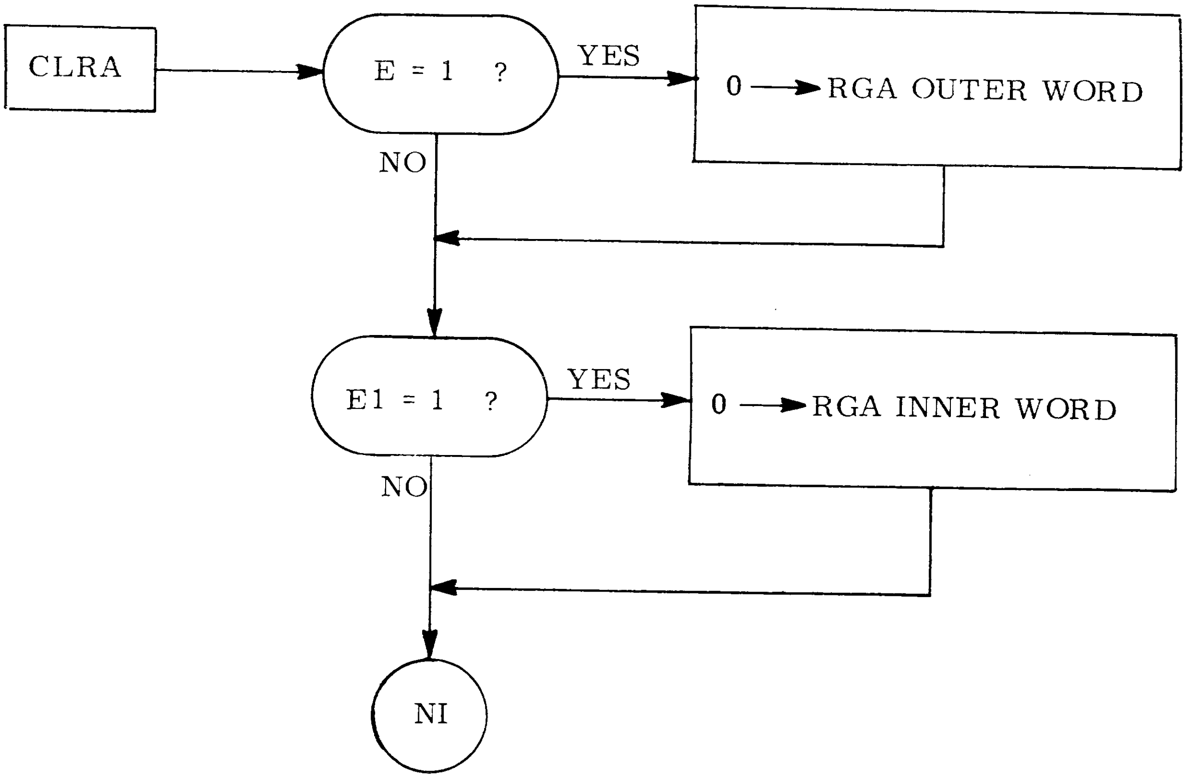
OPERATION: Clear RGA

INSTRUCTION WORD:



DESCRIPTION: This instruction clears RGA. It is the same for 64- and 32-bit modes. Disabled E bits prevent the clearing of RGA.

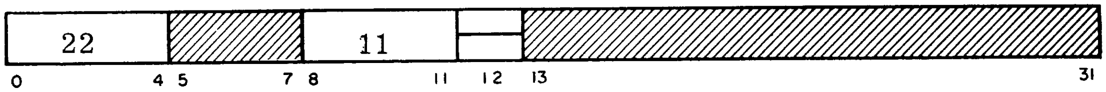
FLOW CHART:



MNEMONIC CODE: COMPA

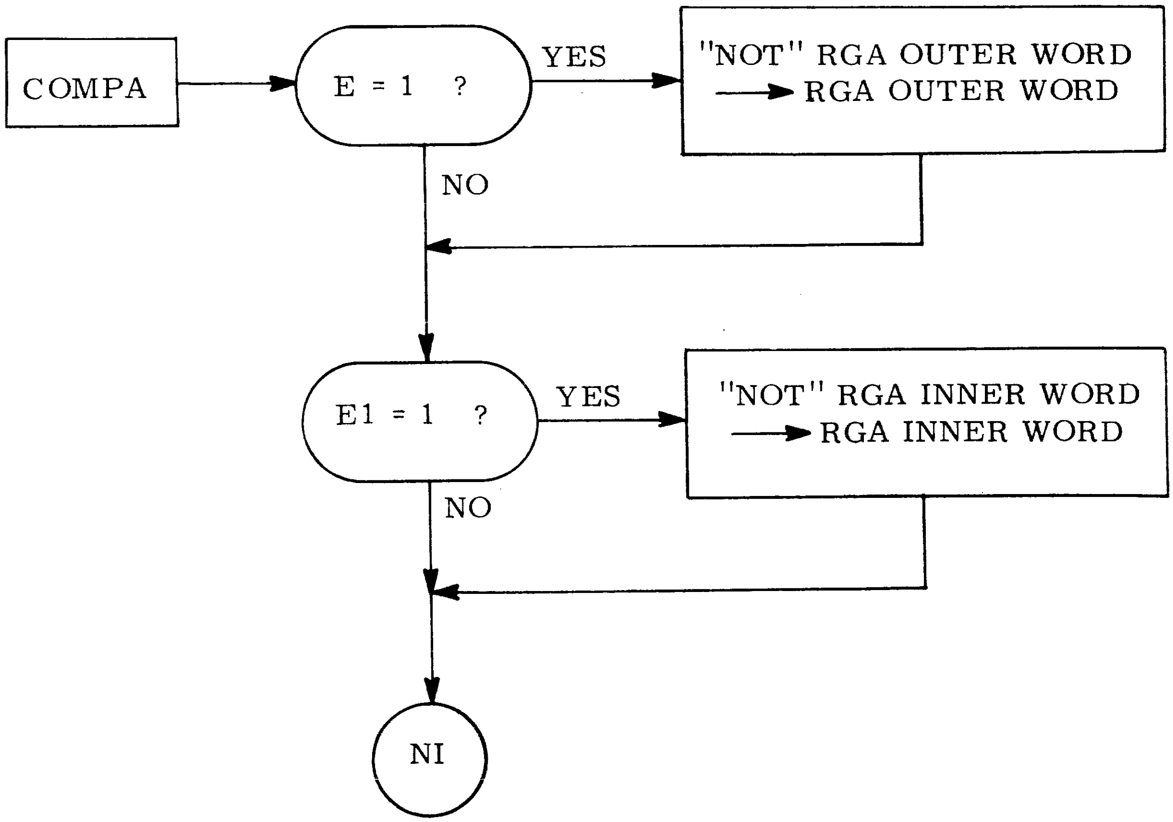
OPERATION: Complement RGA

INSTRUCTION WORD:



DESCRIPTION: This instruction complements each bit in RGA. When the E bits are disabled, RGA is unchanged.

FLOW CHART:


COMPA
E = 1 ?
YES
"NOT" RGA OUTER WORD → RGA OUTER WORD
NO
E1 = 1 ?
YES
"NOT" RGA INNER WORD → RGA INNER WORD
NO
NI

MNEMONIC CODE: DV

OPERATION: Divide RGA and RGB, double-length mantissa, by (ADR).

INSTRUCTION WORDS:

	0 4	5 7	8 11	12	13 15	16 31
DV	32	ACARX	04	—	ADR USE	ADR
DVA	32	ACARX	05	—	ADR USE	ADR
DVM	32	ACARX	14	—	ADR USE	ADR
DVMA	32	ACARX	15	—	ADR USE	ADR
DVN	33	ACARX	04	—	ADR USE	ADR
DVNA	33	ACARX	05	—	ADR USE	ADR
DVR	32	ACARX	06	—	ADR USE	ADR
DVRA	32	ACARX	07	—	ADR USE	ADR
DVRM	32	ACARX	16	—	ADR USE	ADR
DVRMA	32	ACARX	17	—	ADR USE	ADR
DVRN	33	ACARX	06	—	ADR USE	ADR
DVRNA	33	ACARX	07	—	ADR USE	ADR

DESCRIPTION: In 64-bit mode, this instruction divides RGA and RGB by (ADR). RGB is considered the low-order extension of RGA, and must have been loaded prior to the execution of the DV instruction. The 48-bit quotient will be placed into RGA, while the 48-bit remainder will be placed into RGB. RGC will contain minus zero (-0). RGR will contain the divisor.

This instruction may cause the F bit(s) to be set. (See pages 4-15, 4-16, items 4, 5, and 6.)

The divisor is always assumed to be normalized. The remainder in RGB will be meaningless if the "R" variant was specified, or if RGA was larger than the divisor.

The following variants are permitted:

NO SUFFIX - BOTH OPERANDS ARE TREATED AS FLOATING POINT, AND THE RESULT IS NOT NORMALIZED

A - Both RGA and (ADR) are treated as unsigned values;

M - Both values are treated as fixed-point and the result is in fixed-point;

N - Result is normalized;

R - Quotient is rounded in RGA (contents of RGB are meaningless).

If both E bits are disabled, RGA will remain unchanged, and RGB will be undefined. For purposes of this manual, the results of this instruction are undefined when E ≠ E1.

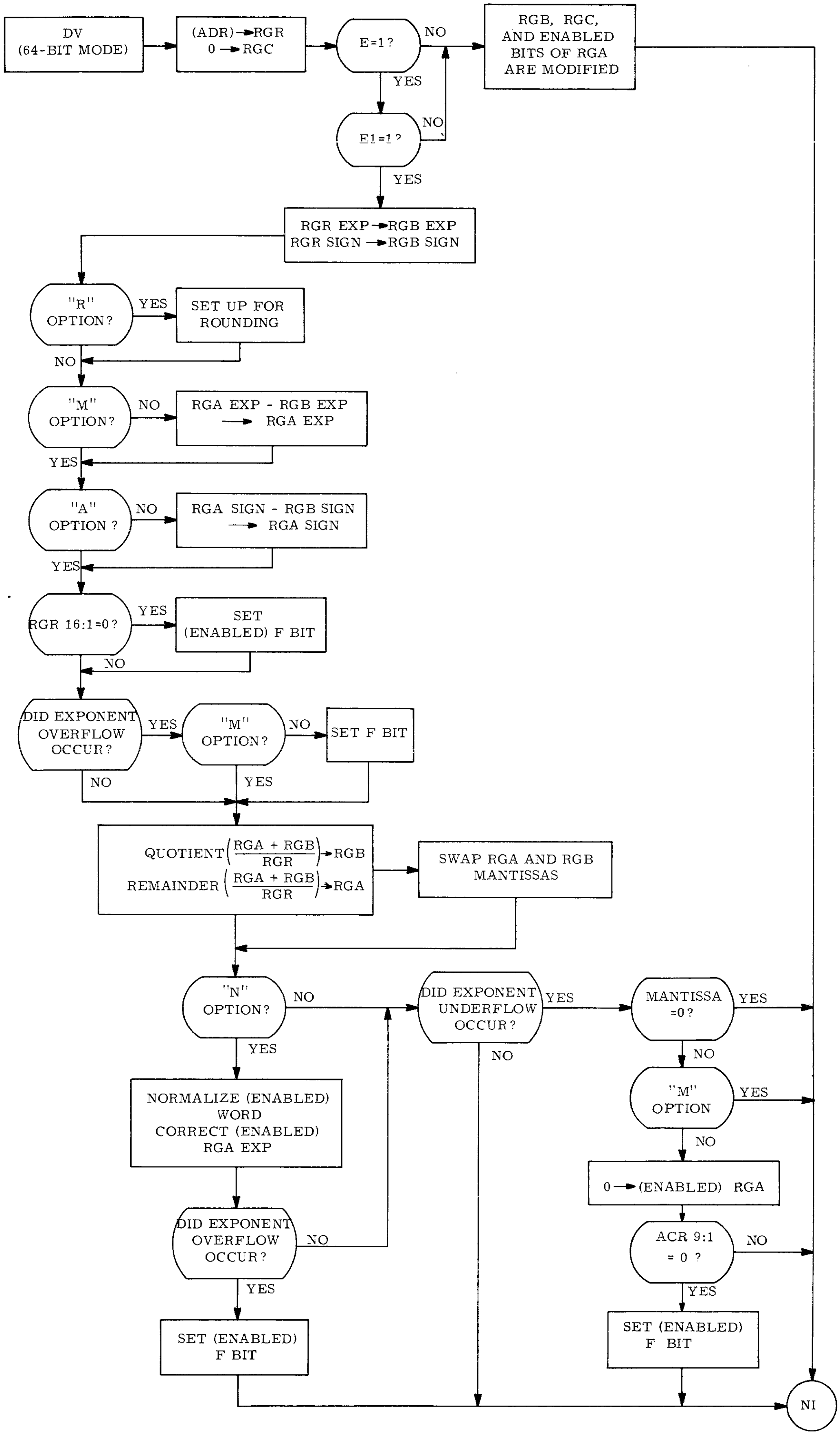
In 32-bit mode, the execution of the DV instruction is the same as for 64-bit mode, with the following modifications:

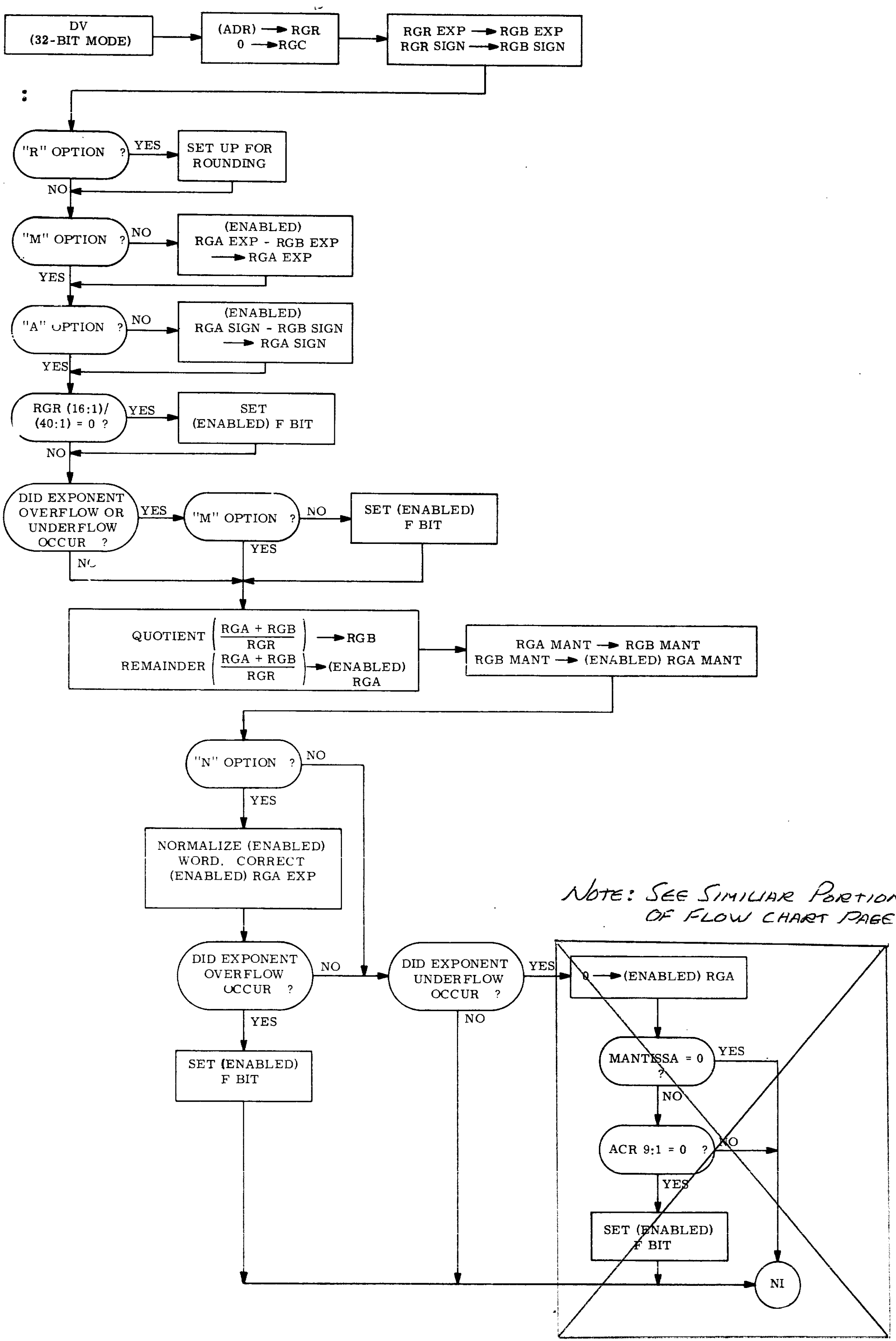
1. If both E bits are enabled, RGA will contain two 24-bit quotients and RGB will contain two 24-bit remainders.

2. RGR will be modified by a swap of outer mantissa for inner mantissa.

3. If either E bit is disabled, the results in the normally protected half of RGA are undefined.

FLOW CHARTS: See next two pages for 64- and 32-bit modes respectively.


DV
(64-BIT MODE)
(ADR)→RGR
0 → RGC
E=1?
NO
YES
RGB, RGC,
AND ENABLED
BITS OF RGA
ARE MODIFIED
E1=1 ?
NO
YES
RGR EXP→RGB EXP
RGR SIGN →RGB SIGN
"R" OPTION?
YES
SET UP FOR ROUNDING
NO
"M" OPTION?
NO
RGA EXP - RGB EXP → RGA EXP
YES
"A" OPTION ?
NO
RGA SIGN - RGB SIGN → RGA SIGN
YES
RGR 16:1=0?
YES
SET (ENABLED) F BIT
NO
DID EXPONENT OVERFLOW OCCUR?
YES
"M" OPTION?
NO
SET F BIT
NO
YES
QUOTIENT (RGA + RGB / RGR) →RGB
REMAINDER (RGA + RGB / RGR) →RGA
SWAP RGA AND RGB MANTISSAS
"N" OPTION?
NO
YES
NORMALIZE (ENABLED) WORD
CORRECT (ENABLED) RGA EXP
DID EXPONENT OVERFLOW OCCUR?
NO
YES
SET (ENABLED) F BIT
DID EXPONENT UNDERFLOW OCCUR?
YES
NO
MANTISSA =0 ?
YES
NO
"M" OPTION
YES
NO
0→(ENABLED) RGA
ACR 9:1 = 0 ?
NO
YES
SET (ENABLED) F BIT
NI



Note: See Similiar Portion of Flow Chart Page 4

MNEMONIC CODE: EAD

OPERATION: Extended precision add (ADR) to RGA

INSTRUCTION WORD:

20	ACARX	10	—	ADR USE	ADR
0 4	5 7	8 11	12	13 15	16 31

DESCRIPTION: The same floating point single-length result that would be produced by floating-point add is left in RGB. That portion of the augend, or the addend, which was shifted off to the right to allow alignment, plus any bit of significance shifted off of the single-length floating point result due to normalization of mantissa overflow, are added together. The result is placed in RGA as a floating point number, with proper sign and exponent.

The F bit may be set in case of exponent overflow. In case of exponent underflow in RGA, all 64 bits of RGA will be reset to zero and the F bit will be set, conditional on ACR(09) (exponent underflow inhibit) being zero.

The unnormalized variant of the add (AD) instruction is assumed (that is, the result, both in RGA and RGB, will not necessarily be normalized). At the conclusion of this instruction, RGR will contain a copy of RGA. If not in 64-bit mode or if both E bits are not enabled, the results, for purposes of this manual, are undefined.

Note: The addition is effected as follows (assuming that the E bits are enabled):

1. The exponent of the result is determined as the larger of the two operand exponents.

2. The operand with the smaller exponent is stored in RGR.

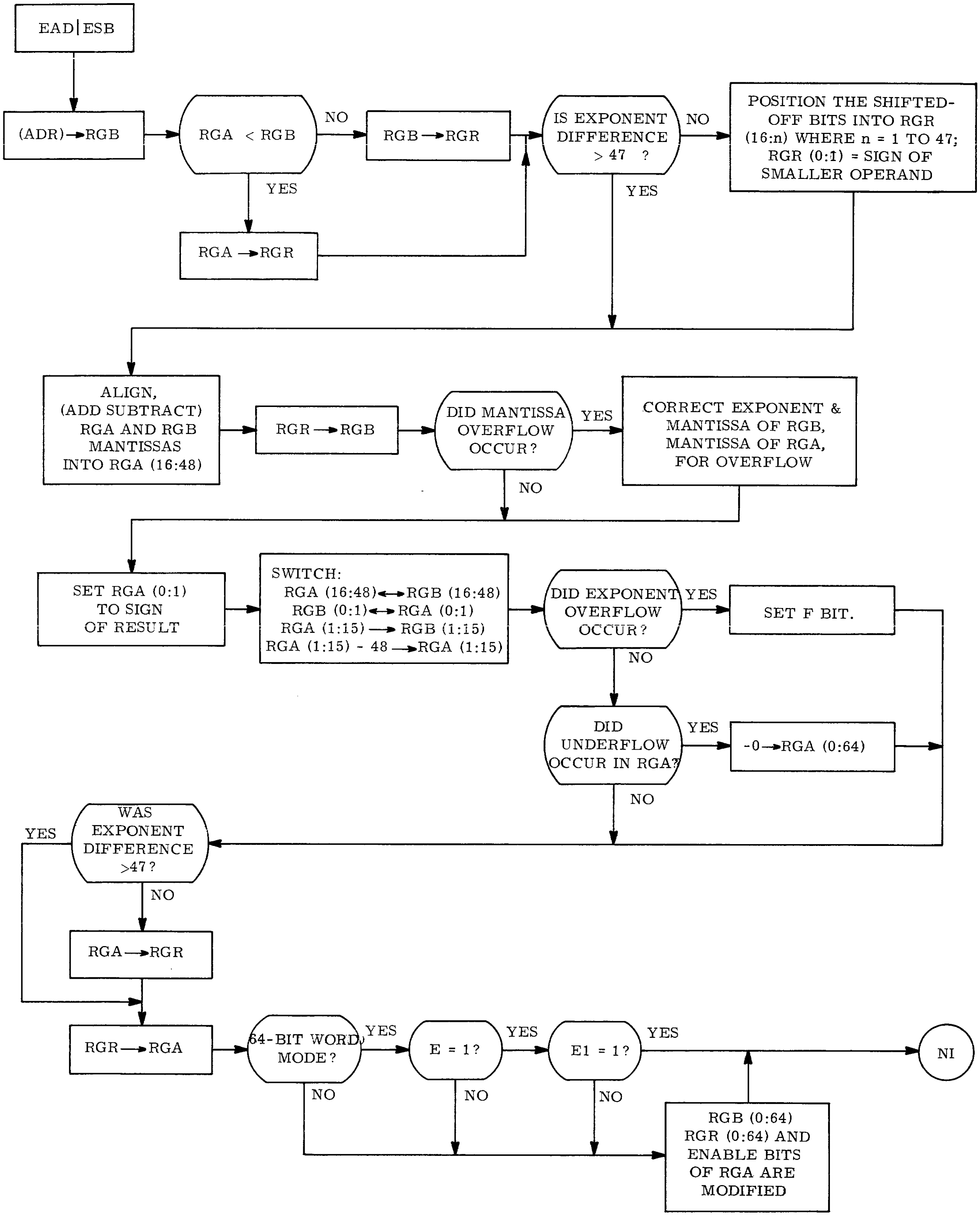
3. If the difference between the two exponents is greater than 47, then the smaller operand is placed into RGA, the larger operand is placed into RGB, and the instruction is completed. (Note that the signs may be different.)

4. Otherwise, the mantissa of the operand with the smaller exponent is shifted right end-off until it is aligned with the mantissa of the operand with the greater exponent. The shifted-off bits are stored into the RGR, while the aligned mantissa is returned to its source register (RGA or RGB).

5. The mantissas are added, and the result stored in RGB.

6. The shifted-off bits in RGR are stored into RGA with an exponent equal to the exponent of RGB, minus 48 (i.e., RGB (1:15) - 48). The sign of RGA will be the sign of the original operand with the smaller exponent. (Note that the signs of RGA and RGB may be different.)

FLOW CHART: See next page. (Note that the flow chart is also used for instruction ESB.)


EAD|ESB
(ADR)→RGB
RGA < RGB
NO
YES
RGB→RGR
RGA→RGR
IS EXPONENT DIFFERENCE > 47 ?
NO
YES
POSITION THE SHIFTED-OFF BITS INTO RGR (16:n) WHERE n = 1 TO 47; RGR (0:1) = SIGN OF SMALLER OPERAND
ALIGN, (ADD SUBTRACT) RGA AND RGB MANTISSAS INTO RGA (16:48)
RGR→RGB
DID MANTISSA OVERFLOW OCCUR?
YES
NO
CORRECT EXPONENT & MANTISSA OF RGB, MANTISSA OF RGA, FOR OVERFLOW
SET RGA (0:1) TO SIGN OF RESULT
SWITCH:
RGA (16:48)↔RGB (16:48)
RGB (0:1)↔RGA (0:1)
RGA (1:15)→RGB (1:15)
RGA (1:15) - 48→RGA (1:15)
DID EXPONENT OVERFLOW OCCUR?
YES
NO
SET F BIT.
DID UNDERFLOW OCCUR IN RGA?
YES
NO
-0→RGA (0:64)
WAS EXPONENT DIFFERENCE >47?
YES
NO
RGA→RGR
RGR→RGA
64-BIT WORD MODE?
YES
NO
E = 1?
YES
NO
E1 = 1?
YES
NO
NI
RGB (0:64) RGR (0:64) AND ENABLE BITS OF RGA ARE MODIFIED

MNEMONIC CODE: ESB

OPERATION: Extended precision subtract (ADR) from RGA

INSTRUCTION WORD:

24	ACARX	10	—	ADR USE	ADR
0 4	5 7	8 11	12	13 15	16 31

DESCRIPTION: (Same as EAD, except the mantissas are subtracted.)

The same floating point single-length result that would be produced by floating-point subtract is left in RGB. That portion of the minuend, or the subtrahend, which was shifted off to the right to allow alignment, plus any bit of significance shifted off the single-length floating point result due to normalization of mantissa overflow, are added together. The result is placed in RGA as a floating point number, with proper sign and exponent.

The F bit may be set in case of exponent overflow. In case of exponent underflow in RGA, all 64 bits of RGA will be reset to zero and the F bit will be set, conditional on ACR(09) (exponent underflow inhibit) being zero.

The unnormalized variant of the add (AD) instruction is assumed (that is, the result, both in RGA and RGB, will not necessarily be normalized). At the conclusion of this instruction, RGR will contain a copy of RGA. If not in 64-bit mode or if both E bits are not enabled, the results, for purposes of this manual, are undefined.

Note: The subtraction is effected as follows (assuming that the E bits are enabled):

1. The exponent of the result is determined as the larger of the two operand exponents.

2. The operand with the smaller exponent is stored in RGR.

3. If the difference between the two exponents is greater than 47, then the smaller operand is placed into RGA, the larger operand is placed into RGB, and the instruction is completed. (Note that the signs may be different.)

4. Otherwise, the mantissa of the operand with the smaller exponent is shifted right end-off until it is aligned with the mantissa of the operand with the greater exponent. The shifted-off bits are stored into the RGR, while the aligned mantissa is returned to its source register (RGA or RGB).

5. The mantissas are subtracted, and the result stored in RGB.

6. The shifted-off bits in RGR are stored into RGA with an exponent equal to the exponent of RGB, minus 48 (i.e., RGB (1:15) - 48).

7. The sign of RGA will be the sign of the original operand with the smaller exponent, if that operand was the minuend. Otherwise, the sign will be the complement of the sign of the original operand (subtrahend). (Note that the signs of RGA and RGB may be different.)

FLOW CHART: See EAD instruction for combined flow chart.

MNEMONIC CODE: GB

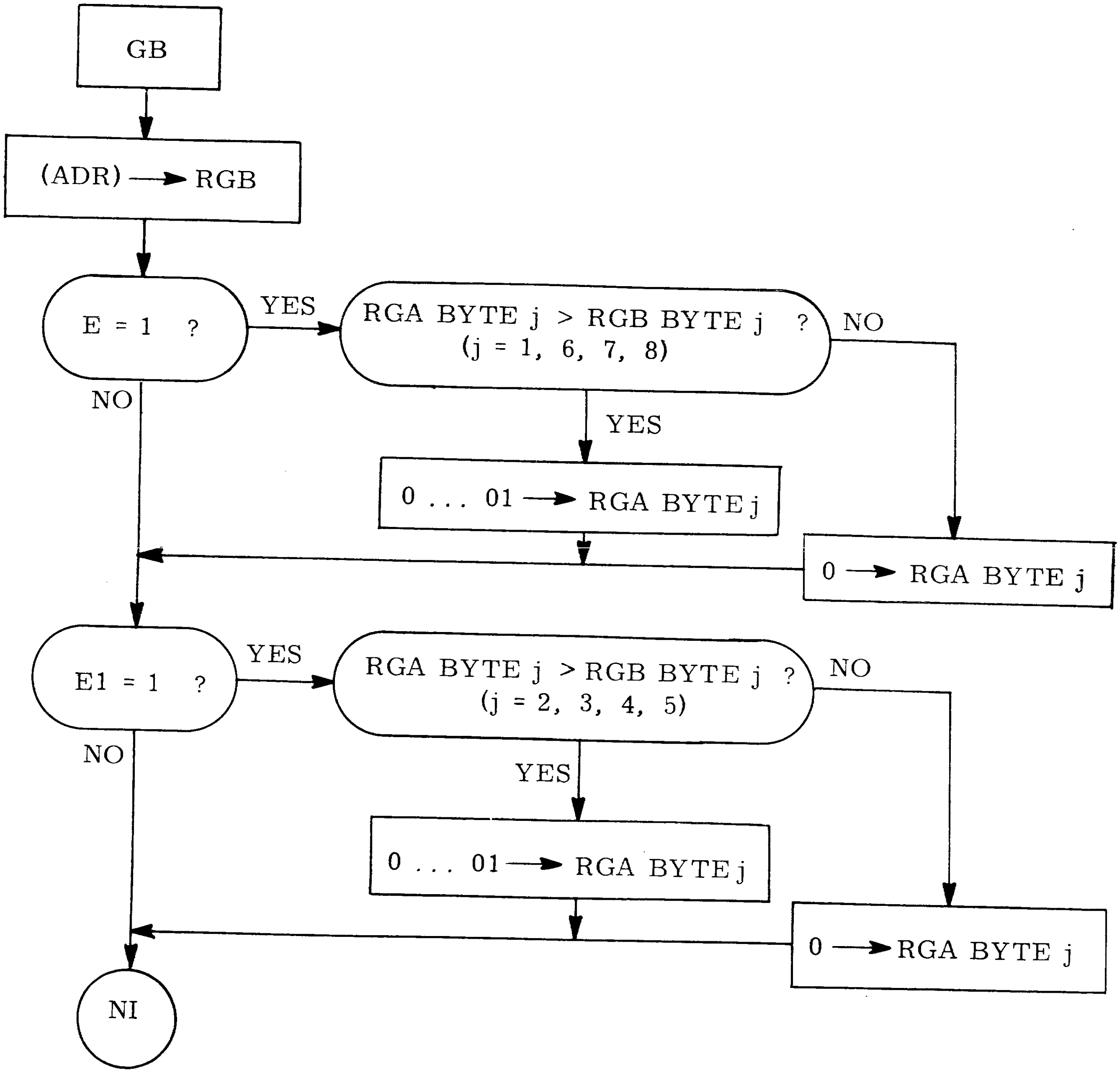
OPERATION: Test for RGA greater than (ADR) in 8-bit bytes.

INSTRUCTION WORD:

21	ACARX	06		ADR USE	ADR
0 4	5 7	8 11	12	13 15	16 31

DESCRIPTION: This instruction tests for RGA greater than (ADR), in 8-bit bytes. The result is stored in the least significant bit of each byte in RGA, "1" for true and "0" for false; the other bits of RGA are set to zero. (ADR) is first fetched to RGB. This instruction uses the 64-bit adder, CPA. The true of RGA and the complement of RGB are enabled into CPA. The bit carries between 8-bit bytes are disabled. The carries out of the CPA are then stored in RGC; these overflow carries are the test results. (ADR) remains in RGB. When the E bits are disabled, RGA is unchanged. This instruction is the same for 64- and 32-bit modes.

FLOW CHART: See next page.


GB
(ADR) → RGB
E = 1 ?
YES
NO
RGA BYTE j > RGB BYTE j ?
(j = 1, 6, 7, 8)
YES
NO
0 . . . 01 → RGA BYTE j
0 → RGA BYTE j
E1 = 1 ?
YES
NO
RGA BYTE j > RGB BYTE j ?
(j = 2, 3, 4, 5)
YES
NO
0 . . . 01 → RGA BYTE j
0 → RGA BYTE j
NI

MNEMONIC CODE: (I | J) A (G | L)

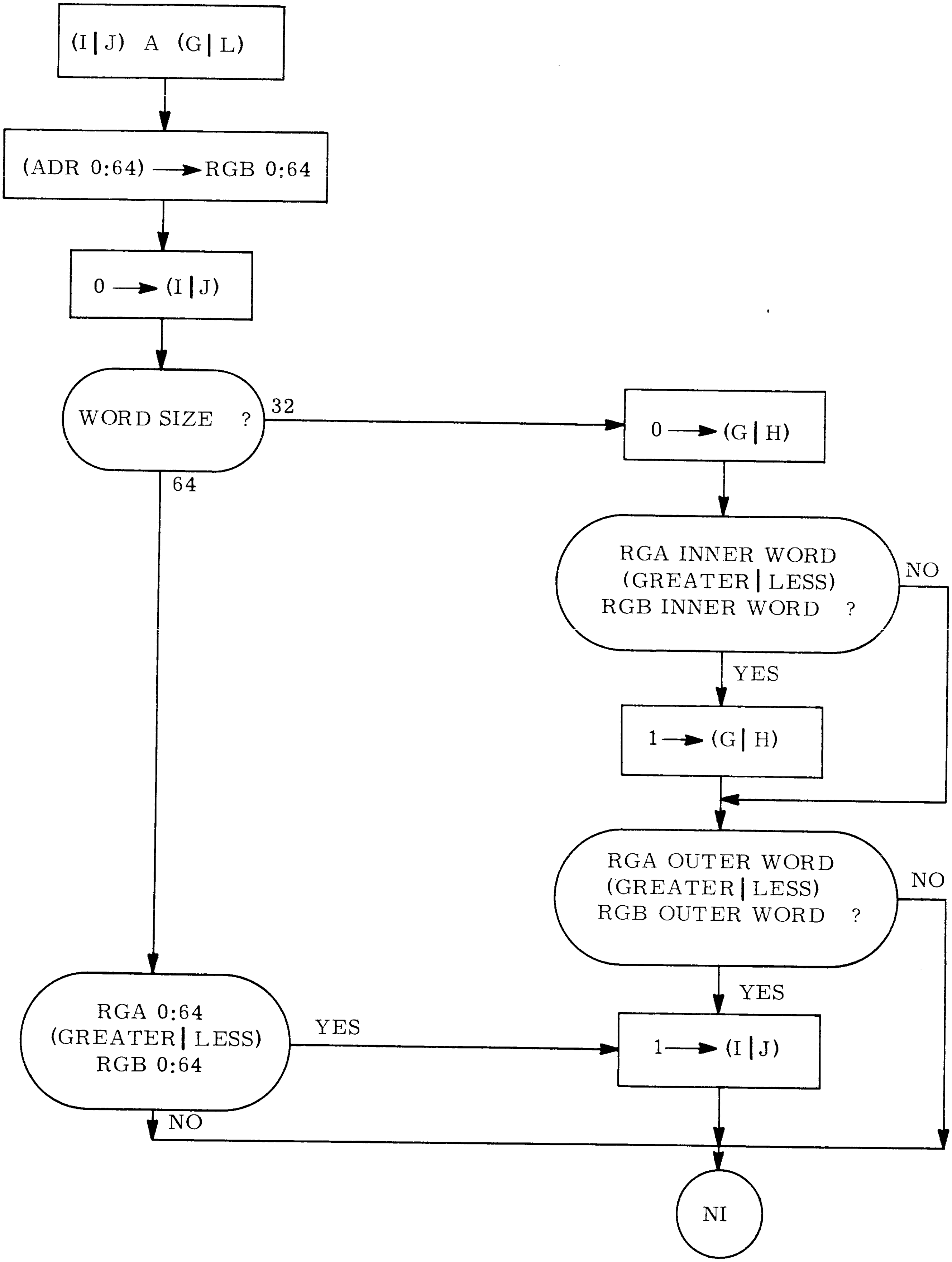
OPERATION: If (RGA) is "greater than" | "less than" (ADR), set mode register bit I | J

INSTRUCTION WORDS:

	0 4	5 7	8 11	12	13 15	16 31
IAG	37	ACARX	14	—	ADR USE	ADR
IAL	37	ACARX	16	—	ADR USE	ADR
JAG	37	ACARX	15	—	ADR USE	ADR
JAL	37	ACARX	17	—	ADR USE	ADR

DESCRIPTION: This is a set of four instructions, each of which is a test to determine if (RGA) is arithmetically "greater than" or "less than" (ADR). In 64-bit mode the test result is stored in the I | J bit of the mode register. In 32-bit mode the test result for the outer word is stored in I | J and in G | H for the test result of the inner word.

FLOW CHART: See next page.


(I | J) A (G | L)
(ADR 0:64) ⟶ RGB 0:64
0 ⟶ (I | J)
WORD SIZE ?
32
64
0 ⟶ (G | H)
RGA INNER WORD (GREATER | LESS) RGB INNER WORD ?
NO
YES
1 ⟶ (G | H)
RGA OUTER WORD (GREATER | LESS) RGB OUTER WORD ?
NO
YES
RGA 0:64 (GREATER | LESS) RGB 0:64
YES
NO
1 ⟶ (I | J)
NI

MNEMONIC CODE: (I|J) B

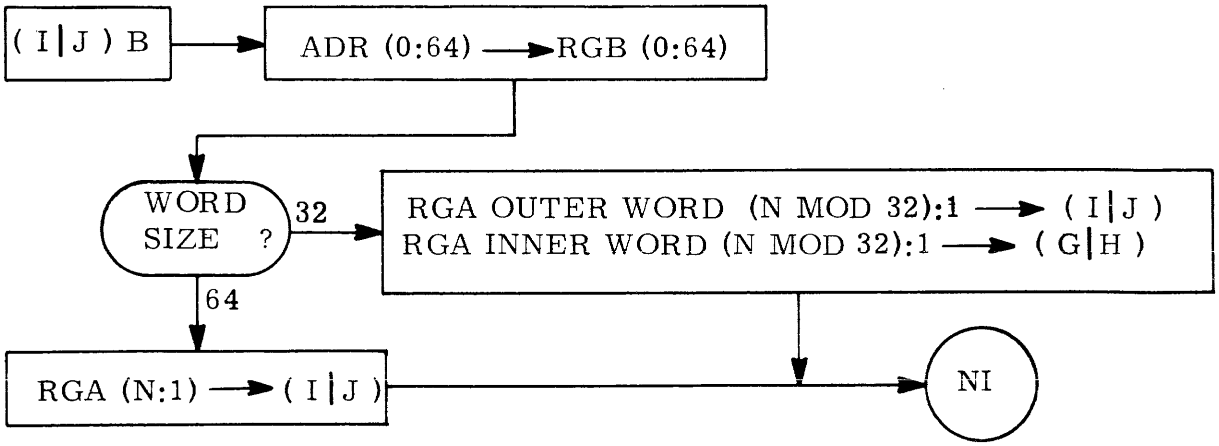
OPERATION: Transfer bit(s) from RGA to mode register bits I|J and G|H

INSTRUCTION WORDS:

IB	35	ACARX	02	—	ADR USE	ADR
	0 4	5 7	8 11	12	13 15	16 31
ISN	35	0——0	02	—	0	——0
JB	35	ACARX	03	—	ADR USE	ADR
JSN	35	0——0	03	—	0	——0

DESCRIPTION: A selected bit of RGA is transmitted to the I|J bit of the mode register. In 32-bit mode, two bits are transmitted: one bit to I|J for the outer word and the other to G|H for the inner word. The bit is specified in the address field and is received over the CDB by the PE as a 6-bit coded number (N) which may be indexed. The ISN | JSN instructions are the same except that the ADR, ADR USE, and ACARX fields are zero.

FLOW CHART:



(I|J) B
ADR (0:64) ⟶ RGB (0:64)
WORD SIZE ?
32
64
RGA OUTER WORD (N MOD 32):1 ⟶ (I|J)
RGA INNER WORD (N MOD 32):1 ⟶ (G|H)
RGA (N:1) ⟶ (I|J)
NI

MNEMONIC CODE: (I | J) (L | M) (E | G | L)

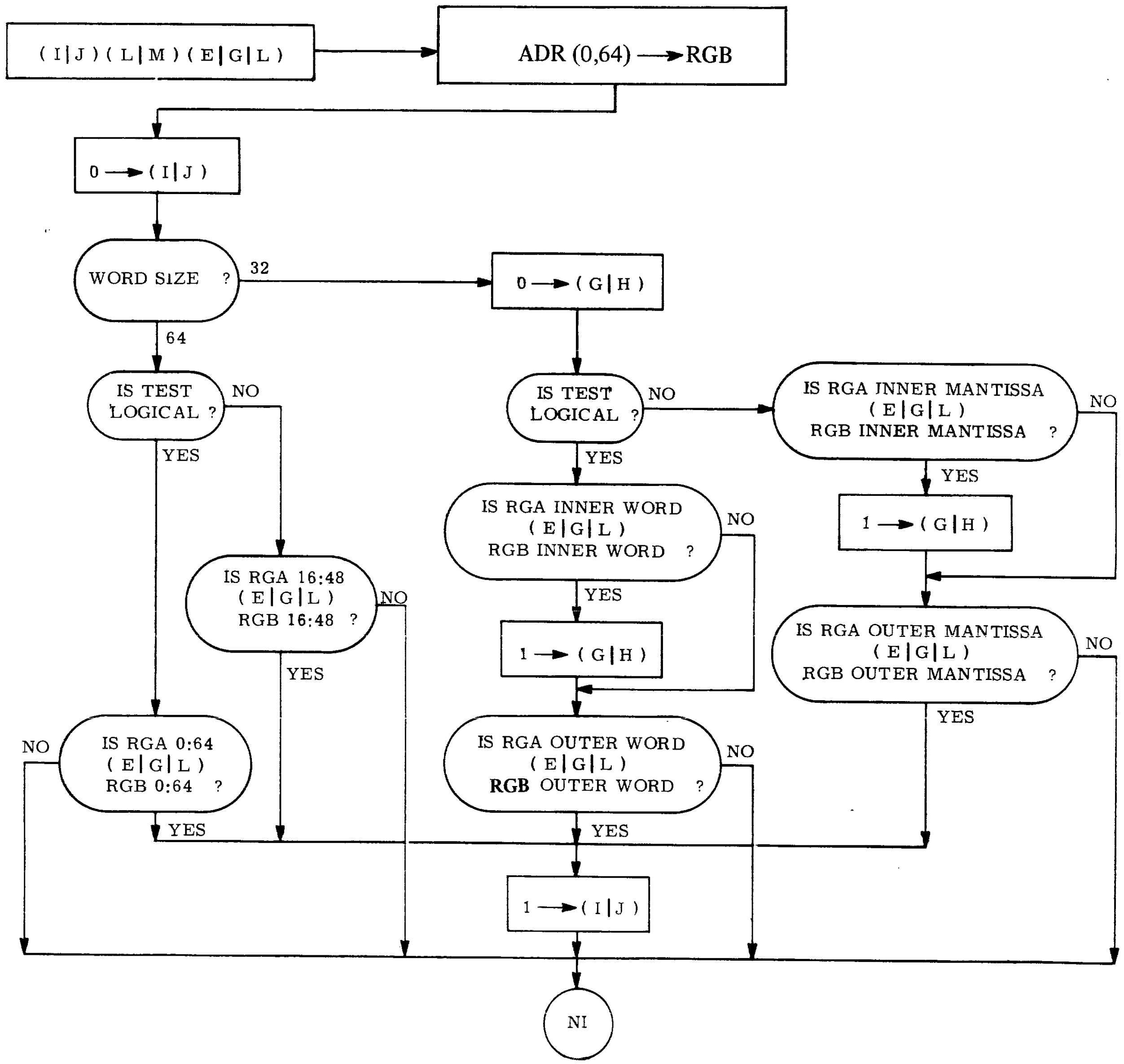
OPERATION: If RGA (either logical word or mantissa portion alone) is "equal to" | "greater than" | "less than" the corresponding portion of (ADR), set mode register bit I | J

INSTRUCTION WORDS:

	0 4	5 7	8 11	12	13 15	16 31
ILE	35	ACARX	16	—	ADR USE	ADR
ILG	33	ACARX	14	—	ADR USE	ADR
ILL	33	ACARX	16	—	ADR USE	ADR
IME	35	ACARX	14	—	ADR USE	ADR
IMG	31	ACARX	14	—	ADR USE	ADR
IML	31	ACARX	16	—	ADR USE	ADR
JLE	35	ACARX	17	—	ADR USE	ADR
JLG	33	ACARX	15	—	ADR USE	ADR
JLL	33	ACARX	17	—	ADR USE	ADR
JME	35	ACARX	15	—	ADR USE	ADR
JMG	31	ACARX	15	—	ADR USE	ADR
JML	31	ACARX	17	—	ADR USE	ADR

DESCRIPTION: These instructions test to determine if (RGA) is logically "greater than", "less than", or "equal to" (ADR). The tests are either on the full 64 bits or on the mantissa fields only. In 64-bit mode, the test result is stored in either the I or J bit of the mode register. In 32-bit mode, the test result for the outer word is stored in the I or J mode bit and in G or H for the inner word.

FLOW CHART:


(I | J) (L | M) (E | G | L)
ADR (0,64) ⟶ RGB
0 ⟶ (I | J)
WORD SIZE ?
32
64
0 ⟶ (G | H)
IS TEST LOGICAL ?
NO
YES
IS RGA 16:48 (E | G | L) RGB 16:48 ?
IS RGA 0:64 (E | G | L) RGB 0:64 ?
IS RGA INNER WORD (E | G | L) RGB INNER WORD ?
1 ⟶ (G | H)
IS RGA OUTER WORD (E | G | L) RGB OUTER WORD ?
IS RGA INNER MANTISSA (E | G | L) RGB INNER MANTISSA ?
IS RGA OUTER MANTISSA (E | G | L) RGB OUTER MANTISSA ?
1 ⟶ (I | J)
NI

MNEMONIC CODE: (I | J) (L | M) (O | Z)

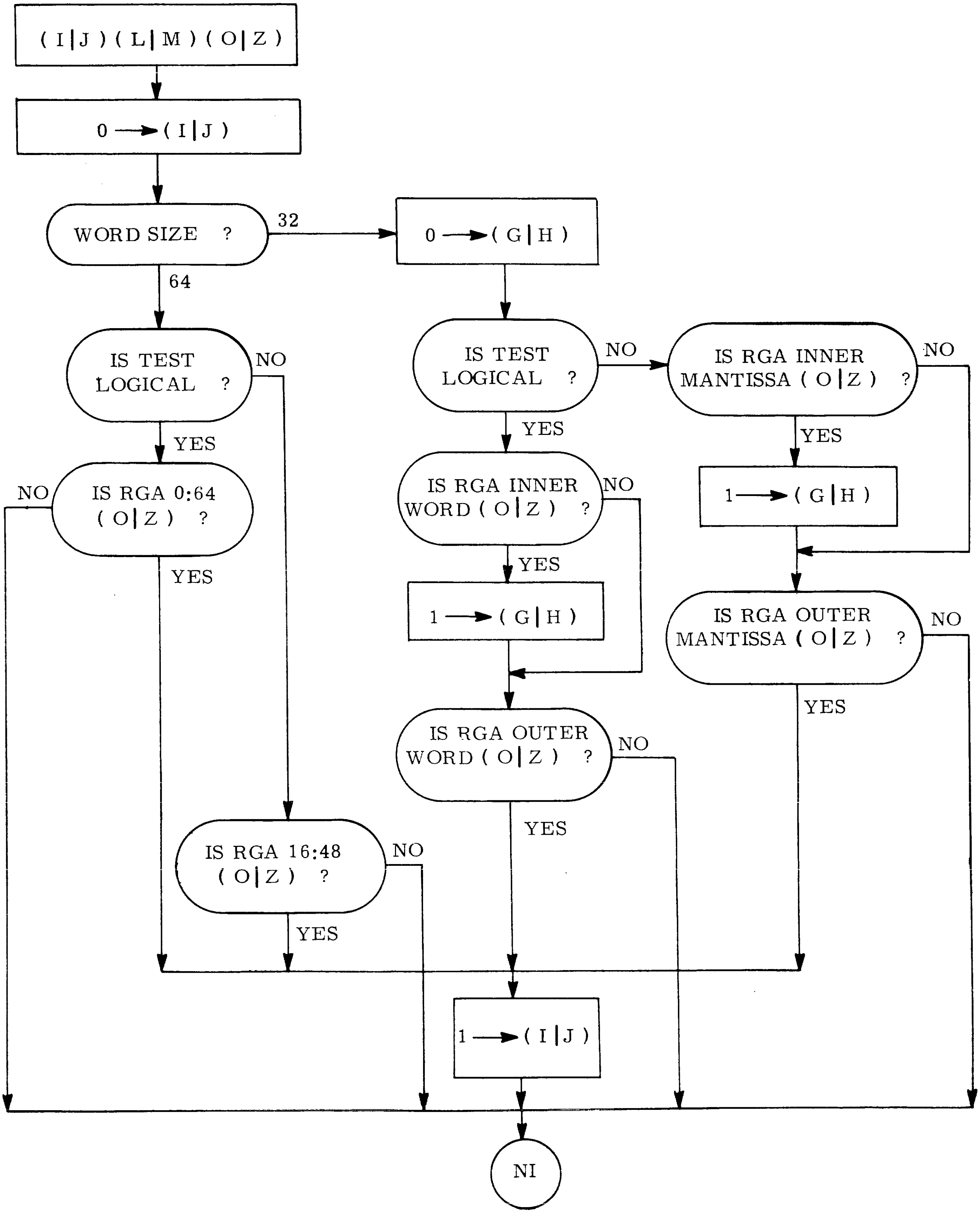
OPERATION: If RGA (either logical word or mantissa portion alone) is "all ones" | "all zeros", set mode register bit I | J

INSTRUCTION WORDS:

	0 4	5 7	8 11	12 31
ILO	33	▨	10	
ILZ	33	▨	12	
IMO	31	▨	10	
IMZ	31	▨	12	
JLO	33	▨	11	
JLZ	33	▨	13	
JMO	31	▨	11	
JMZ	31	▨	13	

DESCRIPTION: These instructions test to determine if RGA is logically equal to zero (Z) or all ones (O). In 64-bit mode, the tests are either on the full 64 bits or on the mantissa field only (48 bits). The results are stored in the I|J bit of the mode register. In 32-bit mode, the results are stored in the I|J mode bit for the outer word and in G|H for the inner word.

FLOW CHART: See next page.


(I | J) (L | M) (O | Z)
0 ⟶ (I | J)
WORD SIZE ?
32
0 ⟶ (G | H)
64
IS TEST LOGICAL ?
NO
YES
IS RGA 0:64 (O | Z) ?
NO
YES
IS TEST LOGICAL ?
NO
YES
IS RGA INNER MANTISSA (O | Z) ?
NO
YES
1 ⟶ (G | H)
IS RGA INNER WORD (O | Z) ?
NO
YES
1 ⟶ (G | H)
IS RGA OUTER MANTISSA (O | Z) ?
NO
YES
IS RGA OUTER WORD (O | Z) ?
NO
YES
IS RGA 16:48 (O | Z) ?
NO
YES
1 ⟶ (I | J)
NI

MNEMONIC CODE: (I | J) (S | X) (E | G | L)

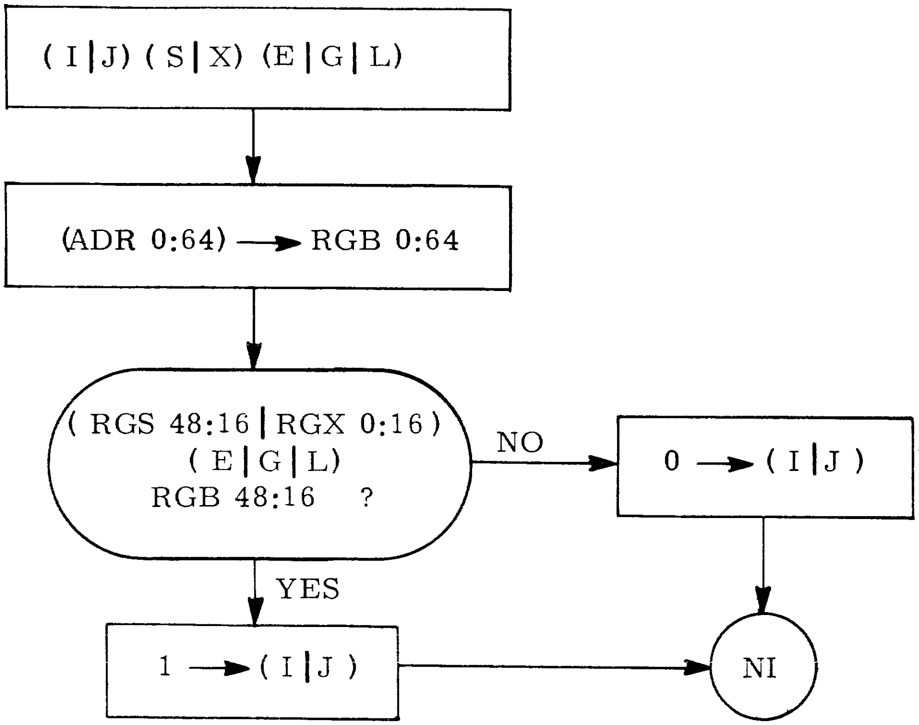
OPERATION: If RGS | RGX is "equal to" | "greater than" | "less than" (ADR), set mode register bit I | J

INSTRUCTION WORDS:

ISE	25	ACARX	12	—	ADR USE	ADR
	0 4	5 7	8 11	12	13 15	16 31
ISG	21	ACARX	12	—	ADR USE	ADR
ISL	23	ACARX	12	—	ADR USE	ADR
IXE	25	ACARX	10	—	ADR USE	ADR
IXG	21	ACARX	10	—	ADR USE	ADR
IXL	23	ACARX	10	—	ADR USE	ADR
JSE	25	ACARX	13	—	ADR USE	ADR
JSG	21	ACARX	13	—	ADR USE	ADR
JSL	23	ACARX	13	—	ADR USE	ADR
JXE	25	ACARX	11	—	ADR USE	ADR
JXG	21	ACARX	11	—	ADR USE	ADR
JXL	23	ACARX	11	—	ADR USE	ADR

DESCRIPTION: These instructions test to determine if (RGX) or the 16 low-order bits of (RGS) is logically "greater than", "less than", or "equal to" the 16 low-order bits of (ADR). The test result is stored in either the I or J bit of the mode register.

FLOW CHART:



MNEMONIC CODE: (I|J) XGI

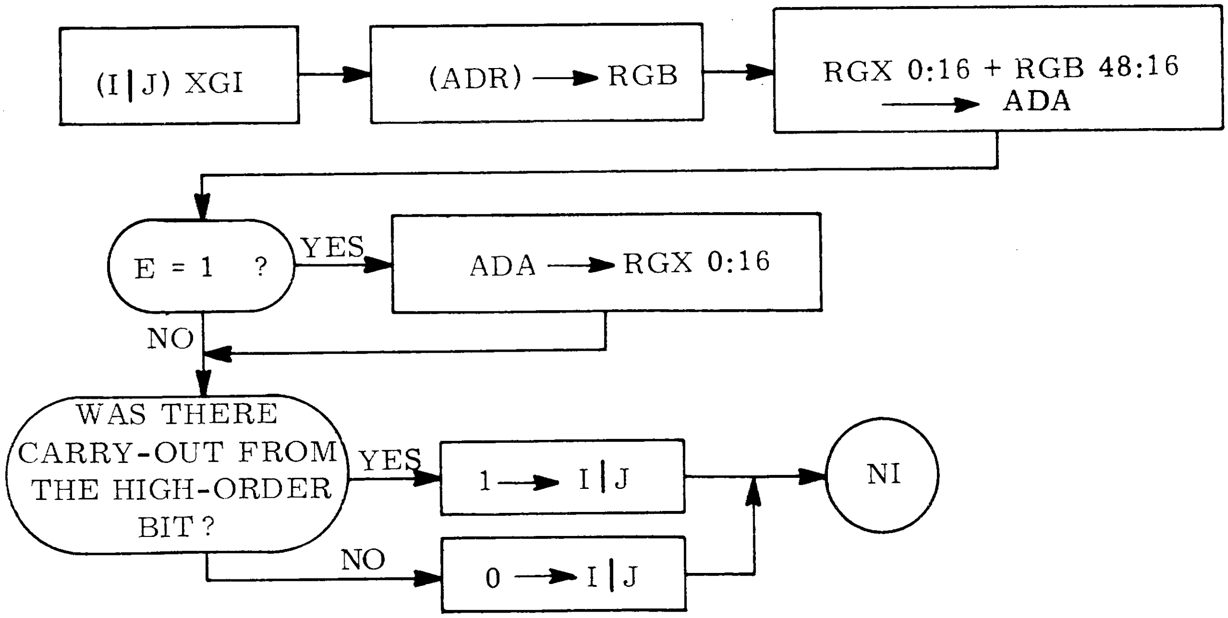
OPERATION: Add (ADR) 48:16 to RGX; store carry-out in mode register bit I|J

INSTRUCTION WORDS:

	0 4	5 7	8 11	12	13 15	16 31
IXGI	27	ACARX	10	—	ADR USE	ADR
JXGI	27	ACARX	11	—	ADR USE	ADR

DESCRIPTION: These are the same as the XI instruction, but additionally, the carry-out is stored in the I | J bit of the mode register. This instruction is the same for 64- and 32-bit modes. When the E bit is disabled RGX is unchanged, but the carry-out (CO = 1 for sum $\geq 2^{16}$) is registered in the I | J bit of the mode register.

FLOW CHART:


(I|J) XGI
(ADR) → RGB
RGX 0:16 + RGB 48:16 → ADA
E = 1 ?
YES
ADA → RGX 0:16
NO
WAS THERE CARRY-OUT FROM THE HIGH-ORDER BIT ?
YES
1 → I|J
NO
0 → I|J
NI

MNEMONIC CODE: (I | J) XLD

OPERATION: Subtract (ADR) 48:16 from RGX; store complement of carry-out in mode register bit I|J

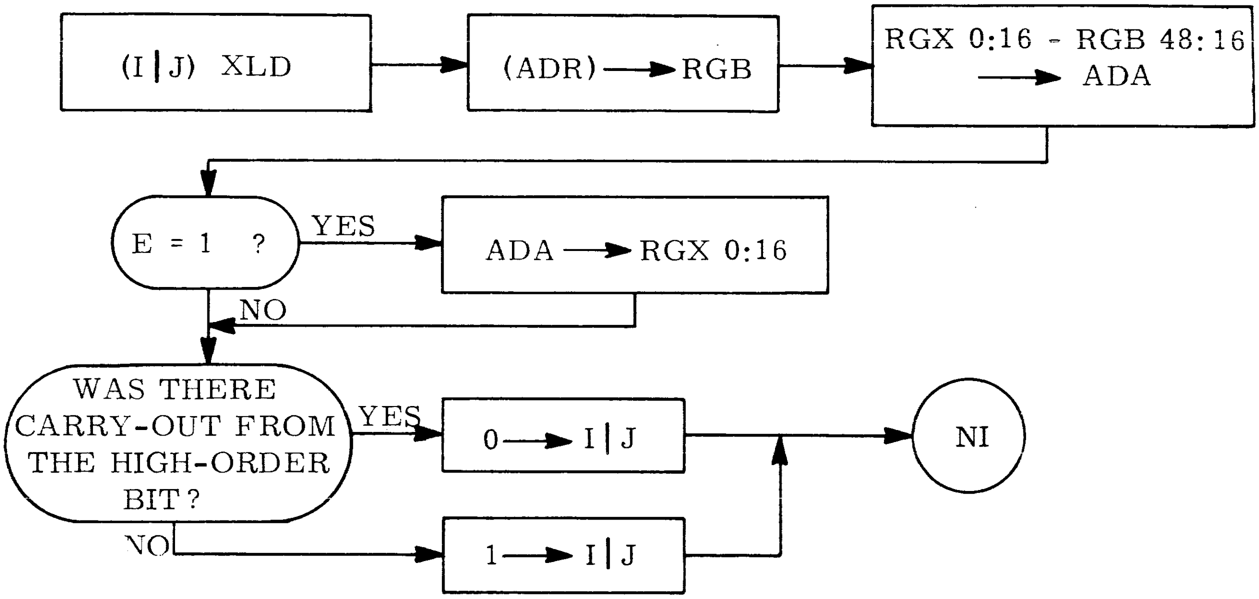
INSTRUCTION WORDS:

IXLD	27	ACARX	12	—	ADR USE	ADR
	0 4	5 7	8 11	12	13 15	16 31

JXLD	27	ACARX	13	—	ADR USE	ADR

DESCRIPTION: These two instructions are the same as the XD instruction, but additionally, the complement of the carry-out is stored in the I|J bit of the mode register. This operation is the same for both 64-bit and 32-bit modes. When the E bit is disabled RGX is unchanged, but the complement of the carry-out is registered in the I|J bit of the mode register.

FLOW CHART:


(I|J) XLD
(ADR) → RGB
RGX 0:16 - RGB 48:16 → ADA
E = 1 ?
YES
ADA → RGX 0:16
NO
WAS THERE CARRY-OUT FROM THE HIGH-ORDER BIT?
YES
0 → I|J
NO
1 → I|J
NI

MNEMONIC CODE: LB

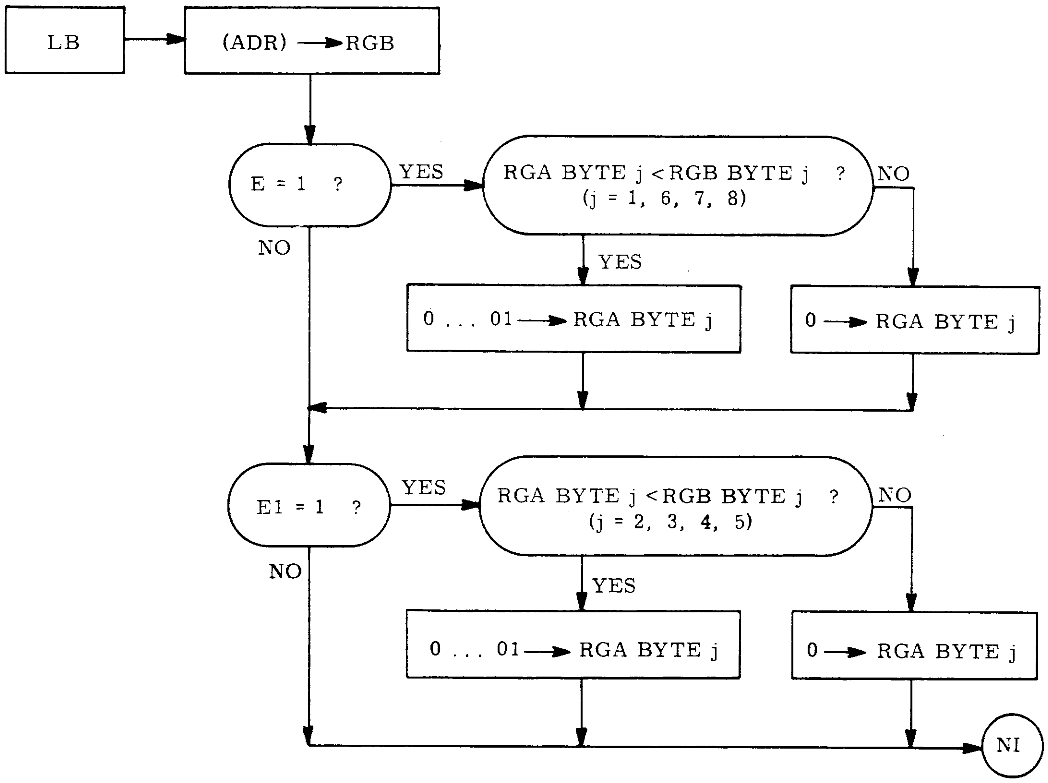
OPERATION: Test for RGA less than (ADR) in 8-bit bytes

INSTRUCTION WORD:

21	ACARX	07	—	ADR USE	ADR
0 4	5 7	8 11	12	13 15	16 31

DESCRIPTION: This instruction tests for (RGA) less than (ADR), in 8-bit bytes. The result is stored in the least significant bit of the corresponding byte in RGA; the remaining bits of RGA are zero. (ADR) remains in RGB. When the E bits are disabled, RGA is unchanged.

FLOW CHART:



MNEMONIC CODE: LEX

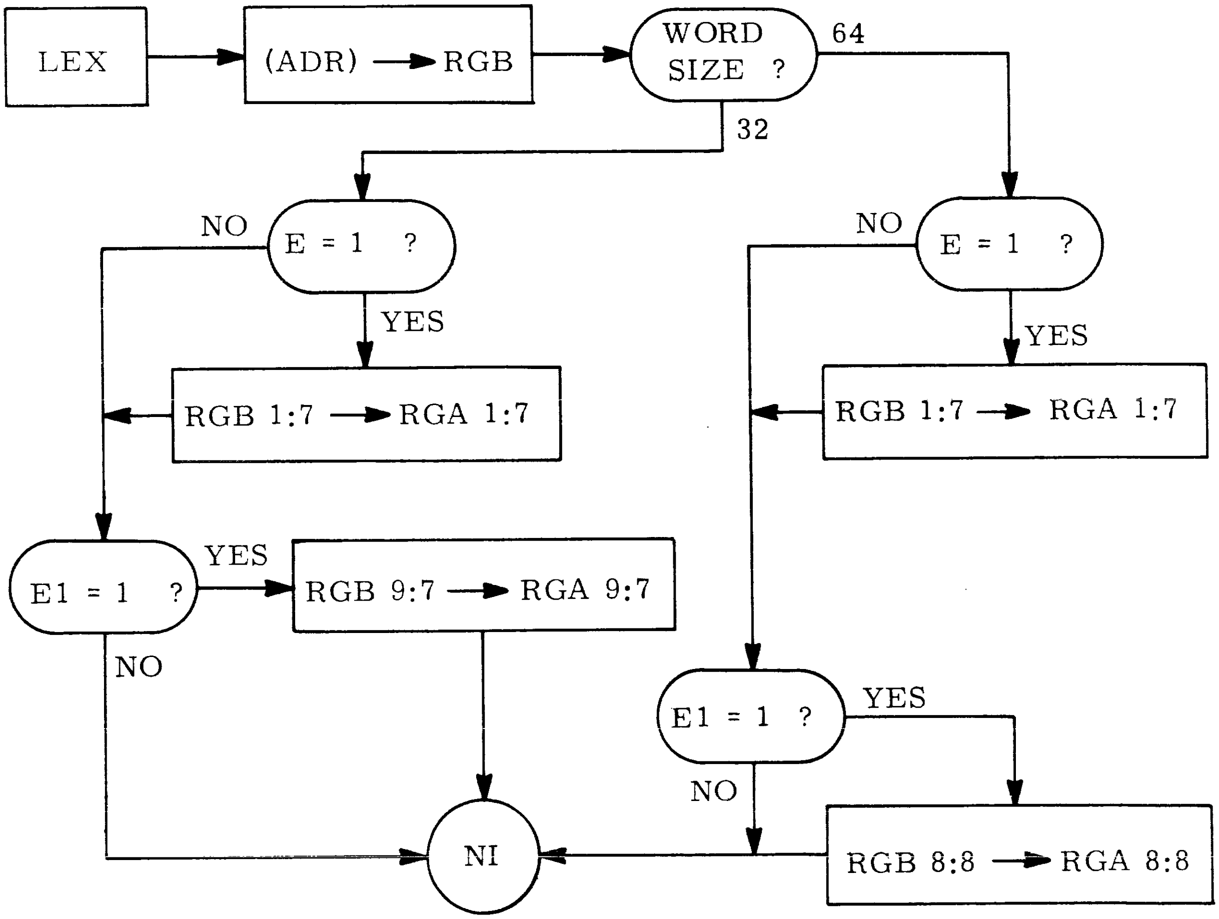
OPERATION: Load the exponent field(s) of RGA with the exponent(s) from (ADR)

INSTRUCTION WORD:

21	ACARX	17	—	ADR USE	ADR
0 4	5 7	8 11	12	13 15	16 31

DESCRIPTION: This instruction loads the exponent field(s) of RGA with the exponent(s) from (ADR). The sign(s) and mantissa field(s) are left unchanged. The RGA exponent is cleared and loaded with the new exponent. (ADR) remains in RGB. When the E bits are disabled, RGA remains unchanged.

FLOW CHART:



MNEMONIC CODE: ML

OPERATION: Multiply RGA by (ADR)

INSTRUCTION WORDS:

ML	30	ACARX	04	—	ADR USE	ADR
	0 4	5 7	8 11	12	13 15	16 31
MLA	30	ACARX	05	—	ADR USE	ADR
MLM	30	ACARX	14	—	ADR USE	ADR
MLMA	30	ACARX	15	—	ADR USE	ADR
MLN	31	ACARX	04	—	ADR USE	ADR
MLNA	31	ACARX	05	—	ADR USE	ADR
MLR	30	ACARX	06	—	ADR USE	ADR
MLRA	30	ACARX	07	—	ADR USE	ADR
MLRM	30	ACARX	16	—	ADR USE	ADR
MLRMA	30	ACARX	17	—	ADR USE	ADR
MLRN	31	ACARX	06	—	ADR USE	ADR
MLRNA	31	ACARX	07	—	ADR USE	ADR

DESCRIPTION: In 64-bit mode, this instruction multiplies RGA by (ADR). RGA will contain the sign, exponent, and 48 high-order bits of the result, while the 48 low-order bits of the result will be placed in the mantissa field of the RGB. Bytes 1 and 2 of RGB will contain 00 111 111 00 111 111, which represents an exponent of minus one (-1) for 32-bit mode. RGC will contain the last subtotal carry.

This instruction may cause the F bit(s) to be set. (See pages 4-15, 4-16, items 4, 5, and 6.)

The following variants are permitted:

NO SUFFIX - BOTH OPERANDS ARE TREATED AS FLOATING POINT, AND THE RESULT IS NOT NORMALIZED

A - Both RGA and (ADR) are treated as unsigned values;

M - Both values are treated as fixed-point and the result is in fixed-point;

N - Result is normalized (after rounding, if specified);

R - Result is rounded in RGA; RGB (bytes 3-8) will be cleared.

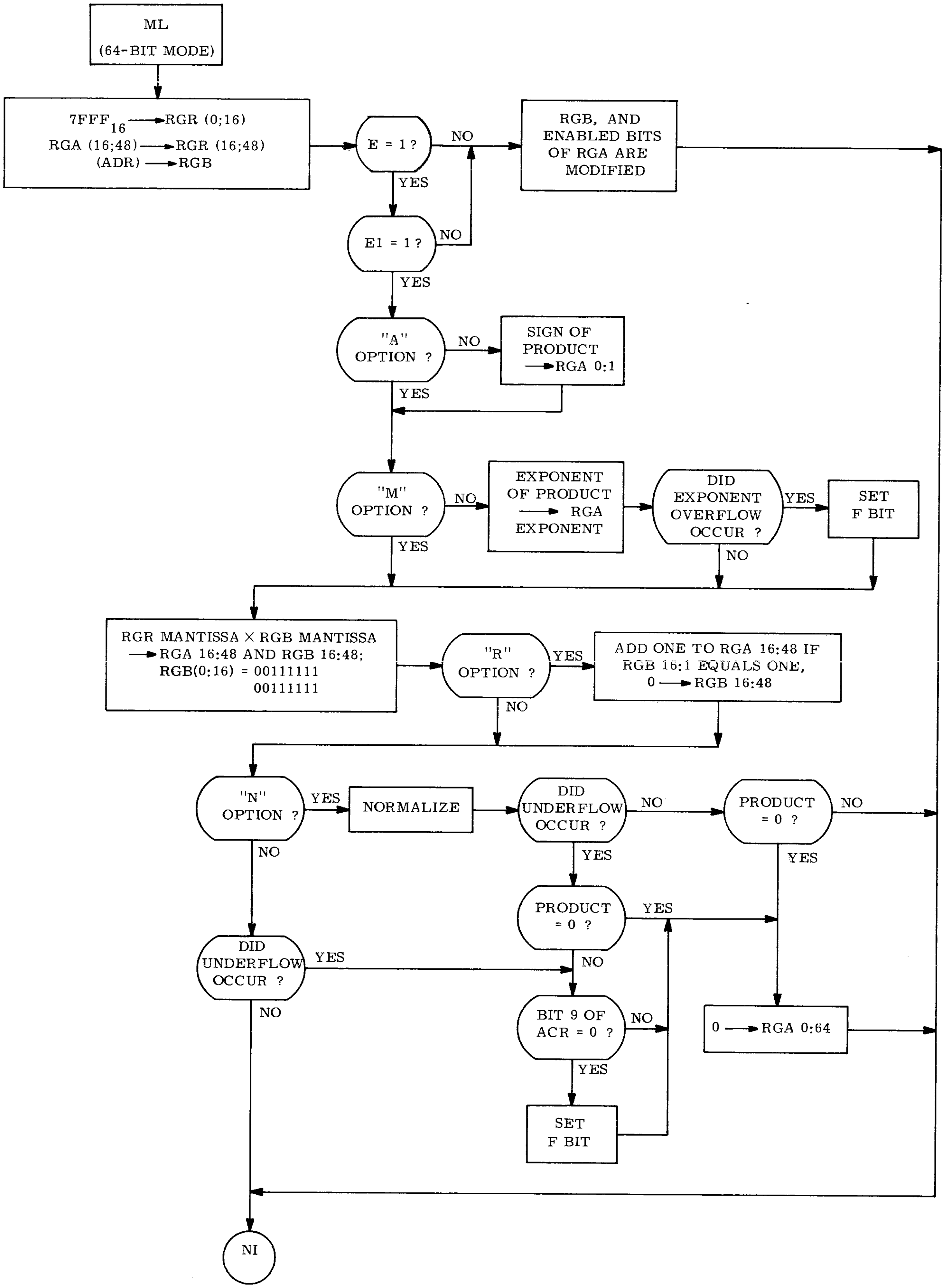
If both E bits are disabled, RGA will remain unchanged, and RGB and RGC will be undefined. For purposes of this manual, the results of this instruction are undefined when E $\neq$ E1.

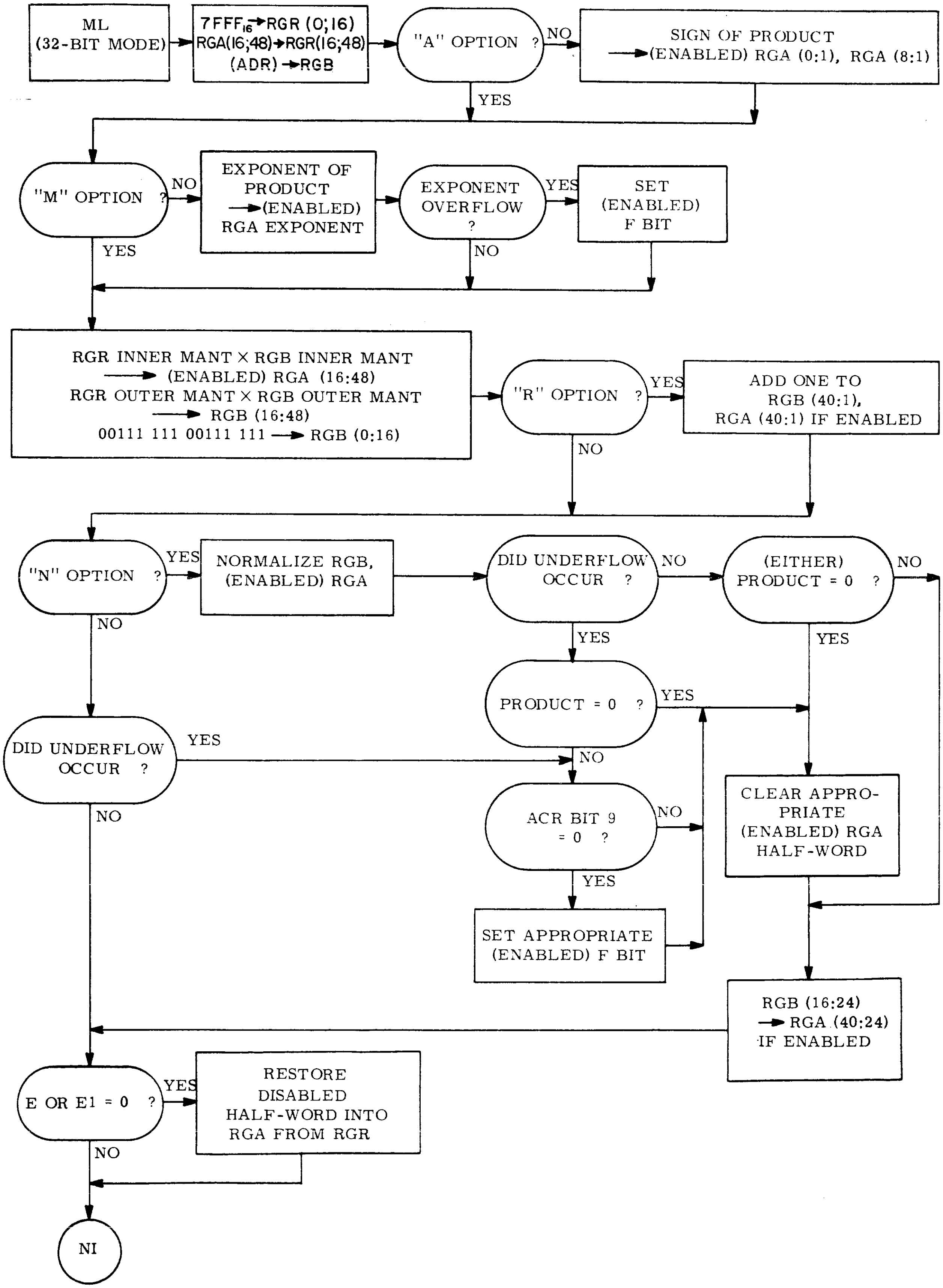
In 32-bit mode, the execution of the ML instruction is the same as for 64-bit mode, with the following modifications:

1. If both E bits are enabled, RGA will contain both 24-bit products, while RGB will contain the 48-bit product of the outer word. Bytes 1 and 2 of RGB still contain the minus one (-1) exponent.

2. If either or both E bits are disabled, the RGA half-word will be properly restored; however, RGB will always contain the 48-bit outer-word product.

FLOW CHARTS: See next two pages for 64- and 32-bit modes respectively.


ML
(64-BIT MODE)
7FFF16 → RGR (0;16)
RGA (16;48) → RGR (16;48)
(ADR) → RGB
E = 1 ?
NO
YES
RGB, AND ENABLED BITS OF RGA ARE MODIFIED
E1 = 1 ?
NO
YES
"A" OPTION ?
NO
SIGN OF PRODUCT → RGA 0:1
YES
"M" OPTION ?
NO
EXPONENT OF PRODUCT → RGA EXPONENT
DID EXPONENT OVERFLOW OCCUR ?
YES
SET F BIT
NO
YES
RGR MANTISSA × RGB MANTISSA → RGA 16:48 AND RGB 16:48; RGB(0:16) = 00111111 00111111
"R" OPTION ?
YES
ADD ONE TO RGA 16:48 IF RGB 16:1 EQUALS ONE, 0 → RGB 16:48
NO
"N" OPTION ?
YES
NORMALIZE
DID UNDERFLOW OCCUR ?
NO
PRODUCT = 0 ?
NO
YES
YES
PRODUCT = 0 ?
YES
NO
NO
DID UNDERFLOW OCCUR ?
YES
NO
BIT 9 OF ACR = 0 ?
NO
YES
0 → RGA 0:64
SET F BIT
NI


ML
(32-BIT MODE)
7FFF$_{16}$→RGR (0;16)
RGA(16;48)→RGR(16;48)
(ADR)→RGB
"A" OPTION ?
NO
SIGN OF PRODUCT
→(ENABLED) RGA (0:1), RGA (8:1)
YES
"M" OPTION ?
NO
EXPONENT OF PRODUCT
→(ENABLED) RGA EXPONENT
EXPONENT OVERFLOW ?
YES
SET (ENABLED) F BIT
NO
YES
RGR INNER MANT × RGB INNER MANT
→ (ENABLED) RGA (16:48)
RGR OUTER MANT × RGB OUTER MANT
→ RGB (16:48)
00111 111 00111 111 → RGB (0:16)
"R" OPTION ?
YES
ADD ONE TO
RGB (40:1),
RGA (40:1) IF ENABLED
NO
"N" OPTION ?
YES
NORMALIZE RGB,
(ENABLED) RGA
DID UNDERFLOW OCCUR ?
NO
(EITHER) PRODUCT = 0 ?
NO
YES
YES
NO
PRODUCT = 0 ?
YES
DID UNDERFLOW OCCUR ?
YES
NO
ACR BIT 9 = 0 ?
NO
CLEAR APPROPRIATE (ENABLED) RGA HALF-WORD
YES
SET APPROPRIATE (ENABLED) F BIT
NO
RGB (16:24)
→ RGA (40:24)
IF ENABLED
E OR E1 = 0 ?
YES
RESTORE DISABLED HALF-WORD INTO RGA FROM RGR
NO
NI

MODE REGISTER INSTRUCTIONS

MNEMONIC CODE: LD___; SET __

OPERATION: LD E | E1 | EE1 | G | H | I | J causes the bit(s) in the mode register specified in the mnemonic to be loaded by a bit in the ACAR; SET E| E1 | F | F1 | G | H | I | J sets the bit in the mode register specified in the mnemonic by the result of a logical function.

INSTRUCTION WORDS:

	0 4	5 7	8 11	12	13 15	16 31
LDE	21	ACARX	14	—	▨	DATA
LDE1	21	ACARX	15	—	▨	DATA
LDEE1	21	ACARX	16	—	▨	DATA
LDG	23	ACARX	14	—	▨	DATA
LDH	23	ACARX	15	—	▨	DATA
LDI	23	ACARX	16	—	▨	DATA
LDJ	23	ACARX	17	—	▨	DATA

	0 4	5 7	8 11	12	13 15	16 19	20 23	24 31
SETE	25	ACARX	14	—	▨	LOG FUNC	B2	B1
SETE1	25	ACARX	15	—	▨	LOG FUNC	B2	B1
SETF	25	ACARX	16	—	▨	LOG FUNC	B2	B1

	0 4	5 7	8 11	12	13 15	16 19	20 23	24 31
SETF1	25	ACARX	17	—	▨	LOG FUNC	B2	B1
SETG	27	ACARX	14	—	▨	LOG FUNC	B2	B1
SETH	27	ACARX	15	—	▨	LOG FUNC	B2	B1
SETI	27	ACARX	16	—	▨	LOG FUNC	B2	B1
SETJ	27	ACARX	17	—	▨	LOG FUNC	B2	B1

DESCRIPTION:

LD___ This instruction causes the mode register (RGD) bit(s) as indicated in the mnemonic to be loaded by a bit of the DATA field. The first 48 bits of the DATA field are implied ZEROes; the last 16 are bits 16:16 of the instruction. ACAR indexing may be used normally, so that when the DATA field of the instruction is all ZEROes, each bit of the ACAR can be sent to the indicated mode bit of the corresponding PE. The ADR USE field is ignored.

SET___ This instruction sets the mode register bit as indicated in the mnemonic with the result of a logic function of two bits specified in the ADR field of the instruction word. The first bit (B1) is designated by one of eight bits in the instruction word, as follows:

Mode Bit B1	Instruction Word Bit Number
H	24
G	25
J	26
I	27
E1	28
E	29
F1	30
F	31

The second bit (B2) is used as indicated by one of four bits in the instruction word, as follows:

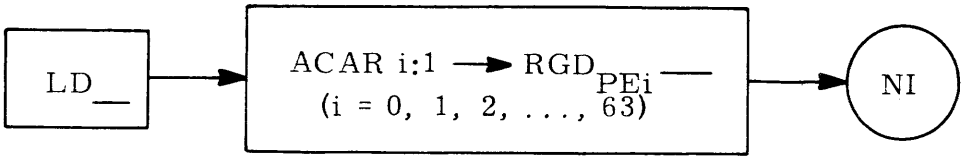
Mode Bit B2	Instruction Word Bit Number
$\overline{E1}$	20
E1	21
$\overline{E}$	22
E	23

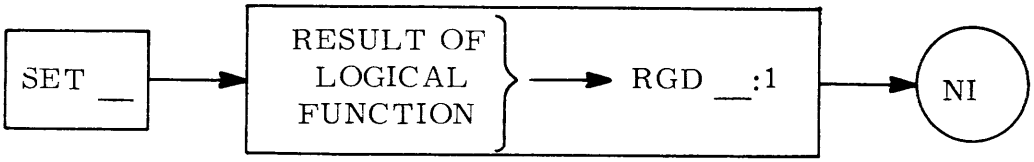
The logical function of mode bits B1 and B2 is specified in one of four bits in the instruction word, as follows:

Logical Function	Instruction Word Bit Number
B1 OR B2	16
$\overline{B1}$ OR B2	17
B1 AND B2	18
$\overline{B1}$ AND B2	19

Note: If multiple ONEs are found in the B1 field, B2 field, or "Logical Function" field, the results are undefined for purposes of this manual. If no ONEs are found in the "Logical Function" field, the function is B1 OR B2. If no ONEs are found in the B2 field, B2 is ZERO.

FLOW CHARTS:





MNEMONIC CODE: MULT

OPERATION: If in 32-bit mode, multiply RGA by (ADR); leave inner double-length product mantissa in RGA, outer in RGB.

In 64-bit mode, this instruction operates as if in 32-bit mode.

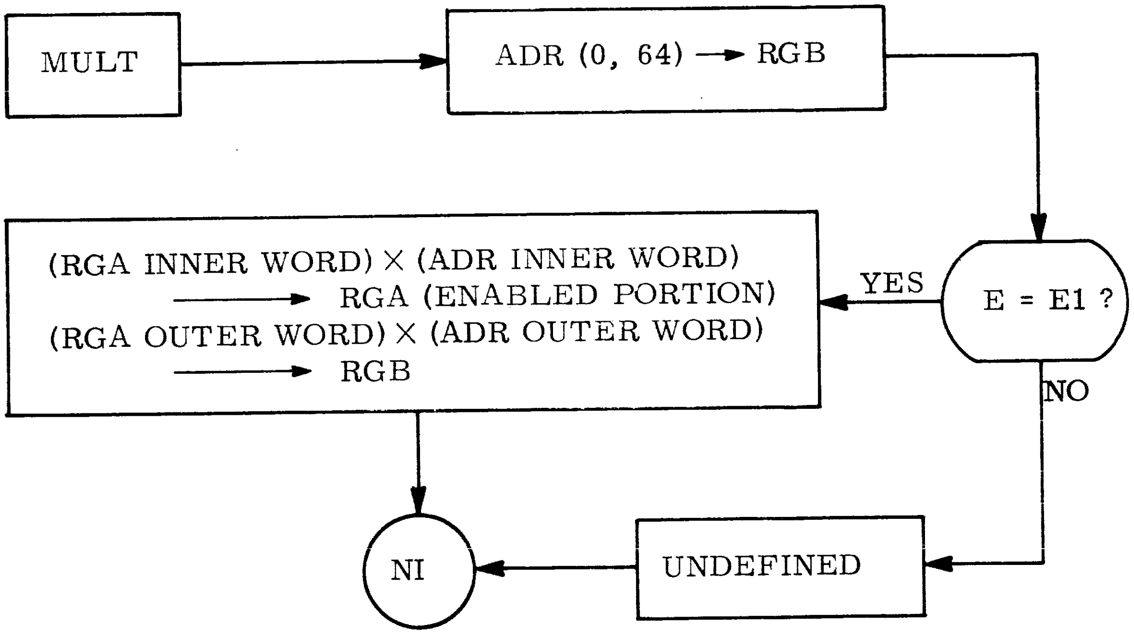
INSTRUCTION WORD:

22	ACARX	13	—	ADR USE	ADR
0 4	5 7	8 11	12	13 15	16 31

DESCRIPTION: Multiply RGA by (ADR). If both E bits are enabled, leave the inner double-length product mantissa in RGA and the outer mantissa in RGB. The exponent of the product of the inner operands is left in RGA and the exponent of the product of the outer operands is left in RGB, both in the position occupied by the exponent of the inner word. The sign bit is in the sign bit position of the outer word. The unused sign and exponent bits are ZERO. If both E bits are disabled, RGA will remain unchanged and RGB will contain the correct outer product. If E ≠ E1, RGA results are undefined for purposes of this manual.

No variants are permitted. Both operands will be treated as floating-point signed values, and the results will be unrounded and normalized.

FLOW CHART:



MNEMONIC CODE: NEB

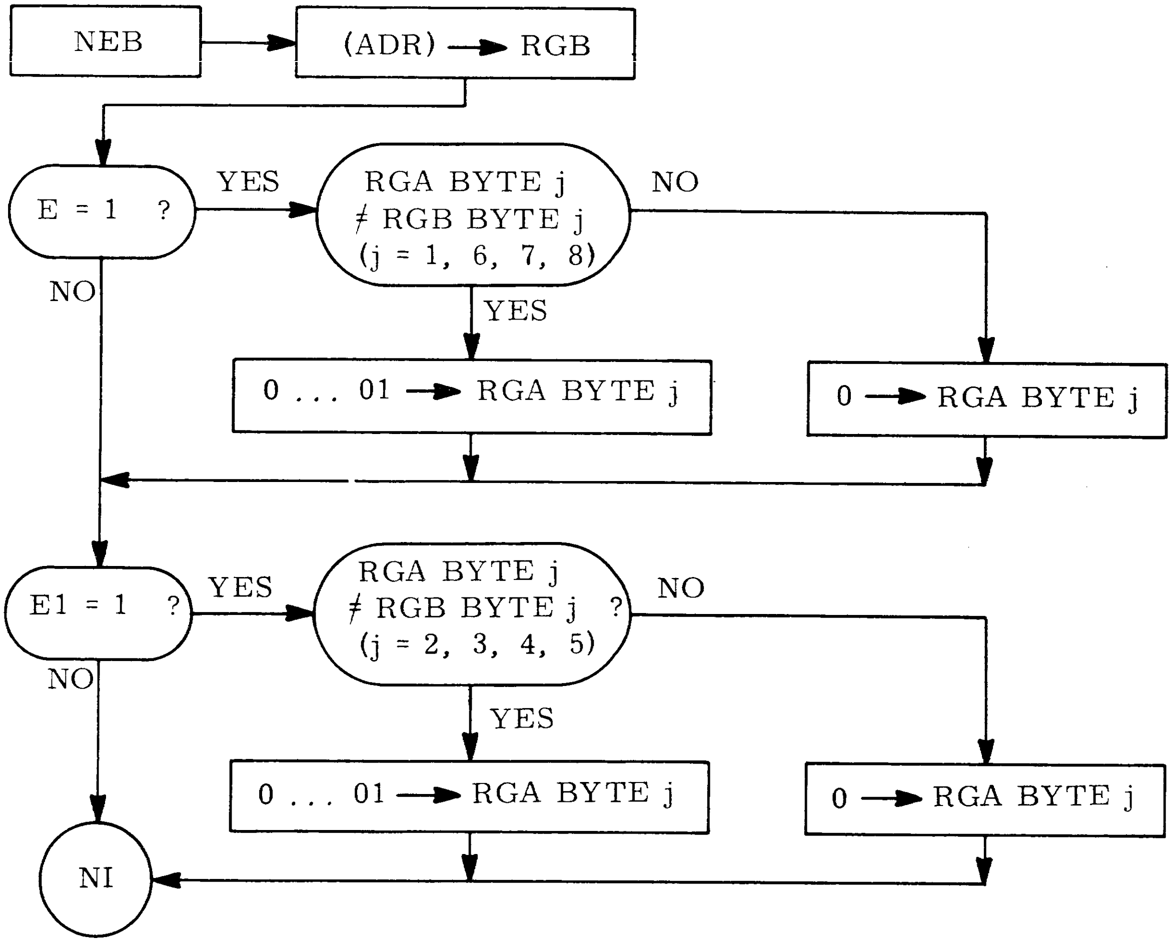
OPERATION: Test for RGA not equal to (ADR) in 8-bit bytes

INSTRUCTION WORD:

22	ACARX	10		ADR USE	ADR
0 4	5 7	8 11	12	13 15	16 31

DESCRIPTION: This instruction tests for RGA not equal to (ADR), in 8-bit bytes. Results are stored in the least significant bit of each byte in RGA; the other bits of RGA are set to zero. (ADR) remains in RGB. The OR of the carries from the "greater than" and "less than" tests remain in RGC. When the E bits are disabled, RGA is unchanged. The instruction is the same for 64- and 32-bit modes.

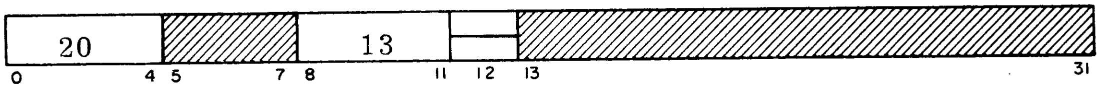
FLOW CHART:



MNEMONIC CODE: NORM

OPERATION: Normalize

INSTRUCTION WORD:

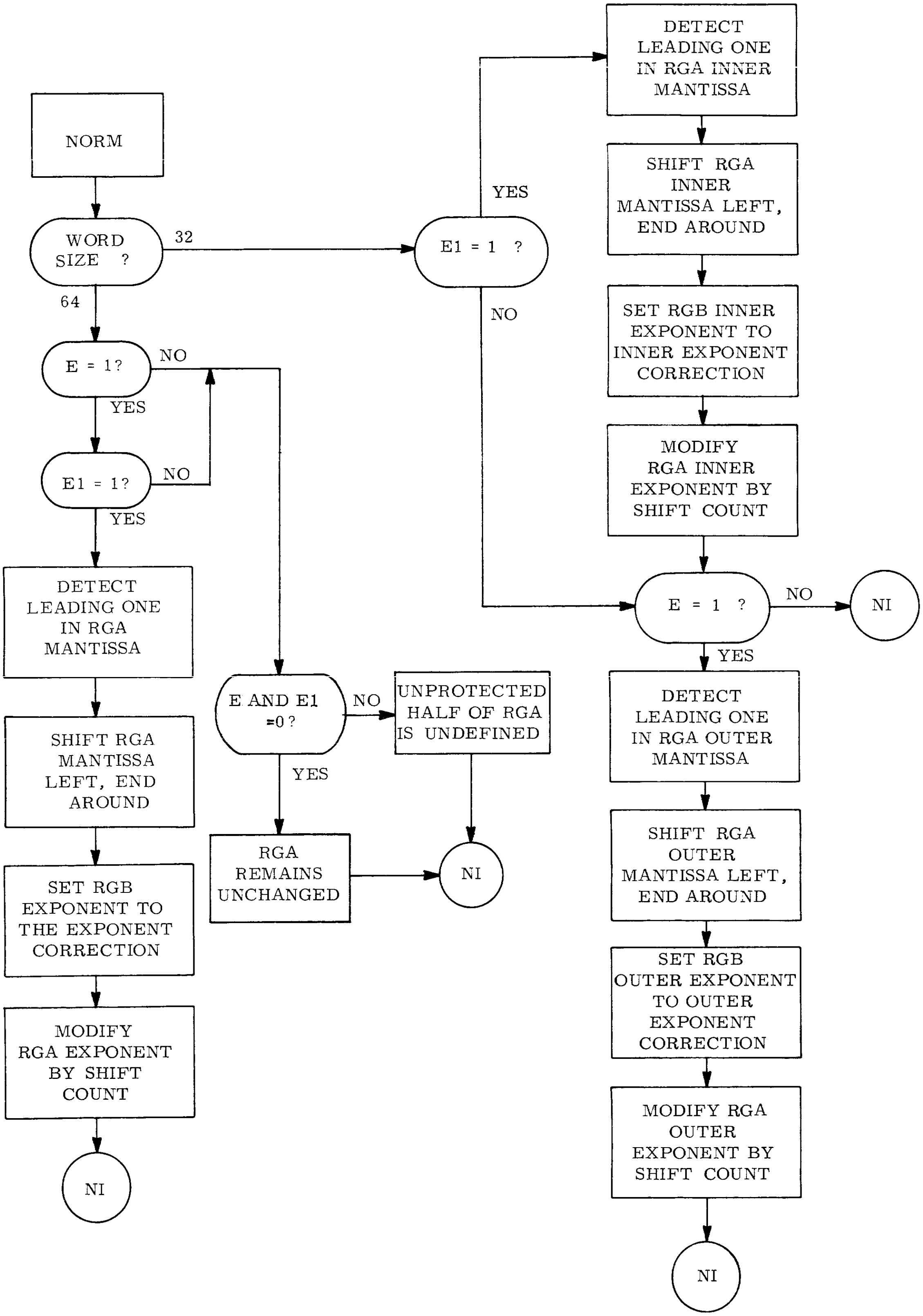


DESCRIPTION: This instruction will shift the RGA mantissa left, end-around, until a "one" bit is detected. In 64-bit mode, the mantissa of RGA is shifted, and the exponent is adjusted. In 32-bit mode, the inner mantissa is acted upon first.

If both E and E1 are zero, the contents of RGA will remain unchanged.

Note: This instruction may cause the F bit(s) to be set.
See pages 4-15, 4-16, items 4, 5, and 6.

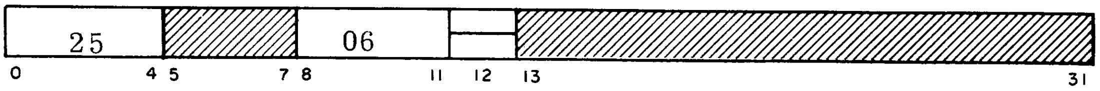
FLOW CHART: See next page.


NORM
WORD SIZE ?
32
64
E = 1?
NO
YES
E1 = 1?
NO
YES
DETECT LEADING ONE IN RGA MANTISSA
SHIFT RGA MANTISSA LEFT, END AROUND
SET RGB EXPONENT TO THE EXPONENT CORRECTION
MODIFY RGA EXPONENT BY SHIFT COUNT
NI
E AND E1 =0 ?
NO
YES
UNPROTECTED HALF OF RGA IS UNDEFINED
RGA REMAINS UNCHANGED
NI
E1 = 1 ?
YES
NO
DETECT LEADING ONE IN RGA INNER MANTISSA
SHIFT RGA INNER MANTISSA LEFT, END AROUND
SET RGB INNER EXPONENT TO INNER EXPONENT CORRECTION
MODIFY RGA INNER EXPONENT BY SHIFT COUNT
E = 1 ?
NO
NI
YES
DETECT LEADING ONE IN RGA OUTER MANTISSA
SHIFT RGA OUTER MANTISSA LEFT, END AROUND
SET RGB OUTER EXPONENT TO OUTER EXPONENT CORRECTION
MODIFY RGA OUTER EXPONENT BY SHIFT COUNT
NI

MNEMONIC CODE: OFB

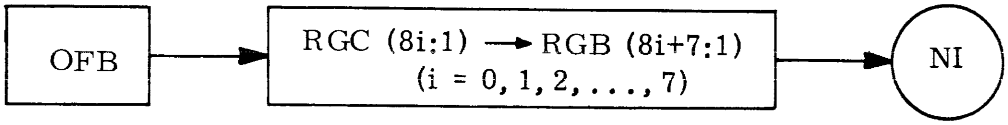
OPERATION: Overflow bits of previous 8-bit byte instruction are transmitted from RGC to RGB

INSTRUCTION WORD:



DESCRIPTION: Transmit overflow bits of previous 8-bit byte instruction from RGC to RGB. RGC is unchanged. This instruction is the same for 64- and 32-bit modes.

FLOW CHART:



MNEMONIC CODE: RTG; RTL

OPERATION: RTG - Transmit register (Y) of every PE to RGR of PE number (N + D) modulo a, where

Y = a specified PE register
N = initial PE No.
D = routing distance
a = number of PEs in array (64/128/256)

RTL - Same as RTG, except for single quadrant (a = 64)

INSTRUCTION WORDS:

	0 4	5 7	8 11	12	13	14	15	16	17 21	22 31
RTG	24	ACARX	13	—	1	▨	0	▨	Y	D
RTL	24	ACARX	12	—	1	▨	0	▨	Y	D

DESCRIPTION:

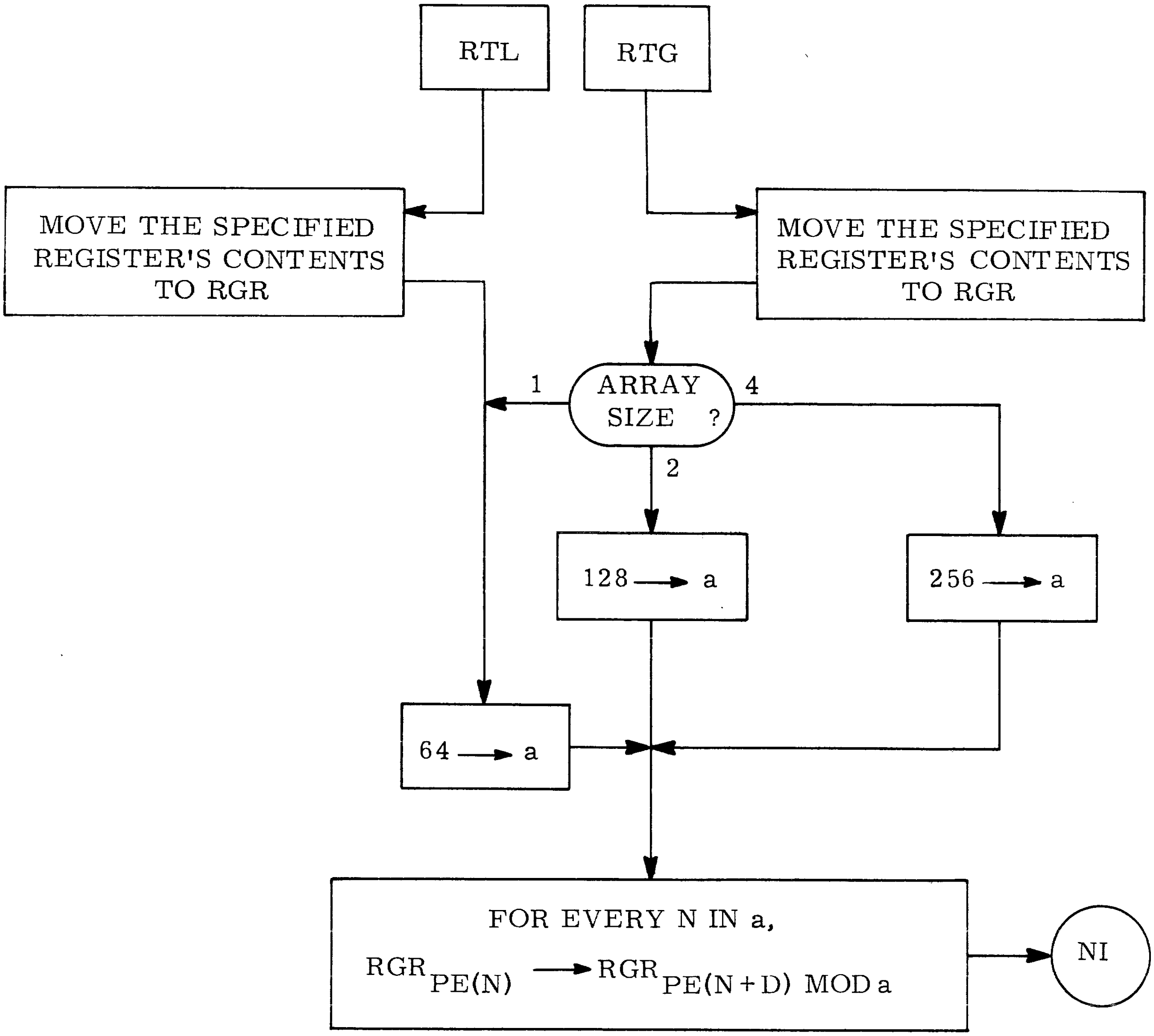
RTG Transmit the data found in the PE register (specified in bits 17-21 of the instruction) of every PE to the RGR of every PE. Data initially found in PE number N is left in PE number (N + D) modulo a (where "D" is the routing distance specified in the ADR field of the instruction and "a" is the number of PEs in the array, whether 64, 128, or 256). "D" and "Y" are indexable by a selected ACAR but not by any RGX or RGS. The array is defined by the contents of MC2 relative to MC0. The register address bits are as follows:

Register	Address Bit
RGA	17
RGB	18
RGX	19
RGS	20
RGR	21

RTL Same as RTG, except that this instruction is for single quadrant only. Data originally stored in the specified register of PE number N is left in the RGR of PE number (N+D) modulo 64.

Note: RGD cannot be transmitted. Also, care should be taken that indexing should not inadvertently change the specified register address (Y) or the routing distance (D).

FLOW CHART:



RTL
RTG
MOVE THE SPECIFIED REGISTER'S CONTENTS TO RGR
MOVE THE SPECIFIED REGISTER'S CONTENTS TO RGR
ARRAY SIZE ?
1
2
4
128 → a
256 → a
64 → a
FOR EVERY N IN a,
RGR PE(N) → RGR PE(N+D) MOD a
NI

MNEMONIC CODE: SB

OPERATION: Subtract (ADR) from RGA (Additional op codes allow certain variants: A for unsigned, M for fixed point, N for normalized floating, R for rounded)

INSTRUCTION WORDS:

SB	36	ACARX	04	—	ADR USE	ADR
	0 4	5 7	8 11	12	13 15	16 31
SBA	36	ACARX	05	—	ADR USE	ADR
SBM	36	ACARX	14	—	ADR USE	ADR
SBMA	36	ACARX	15	—	ADR USE	ADR
SBN	37	ACARX	04	—	ADR USE	ADR
SBNA	37	ACARX	05	—	ADR USE	ADR
SBR	36	ACARX	06	—	ADR USE	ADR
SBRA	36	ACARX	07	—	ADR USE	ADR
SBRN	37	ACARX	06	—	ADR USE	ADR
SBRNA	37	ACARX	07	—	ADR USE	ADR

DESCRIPTION: This instruction subtracts (ADR) from RGA. The result of the subtraction remains in RGA. RGB will contain (ADR) either unmodified, or modified by the mantissa portion(s) that were shifted to align with RGA. RGC and RGR are not used.

This instruction may cause the F bit(s) to be set. (See page 4-15, 4-16, items 4, 5, and 6.)

The following variants are permitted:

A - Both operands are treated as unsigned values;

M - Both operands are treated as fixed-point and the result is in fixed-point;

N - Result is normalized (after rounding, if specified);

R - Result is rounded in RGA.

If both E bits are disabled, RGA will remain unchanged. If E ≠ E1 for 64-bit mode, the results, for purposes of this manual, are undefined.

In 32-bit mode, there is no loss of accuracy because each half-word is aligned independently of the other.

Note: The subtraction is effected as follows (assuming that the E bits are enabled):

1. The exponent of the result is determined as the larger of the two operand exponents.

2. The mantissa of the operand with the smaller exponent is shifted right end-off until it is aligned with the mantissa of the operand with the greater exponent. Thus, if the difference between the two exponents is greater than 47, the smaller mantissa will be set to zero, and the result of the subtract will exactly equal the larger value.

3. The aligned mantissa is returned to its source register (RGA or RGB).

4. The mantissas are subtracted, and the result stored in RGA.

5. The exponent portion of RGB will not be changed except when the normalized variant (N) of the instruction is requested. The RGB exponent will then be set to the exponent correction.

6. The mantissa portion of RGB will be unchanged unless the RGB exponent is smaller than the RGA exponent. (Refer to paragraphs 2 and 3 above.)

FLOW CHART: See AD instruction for combined flow chart.

MNEMONIC CODE: SBB

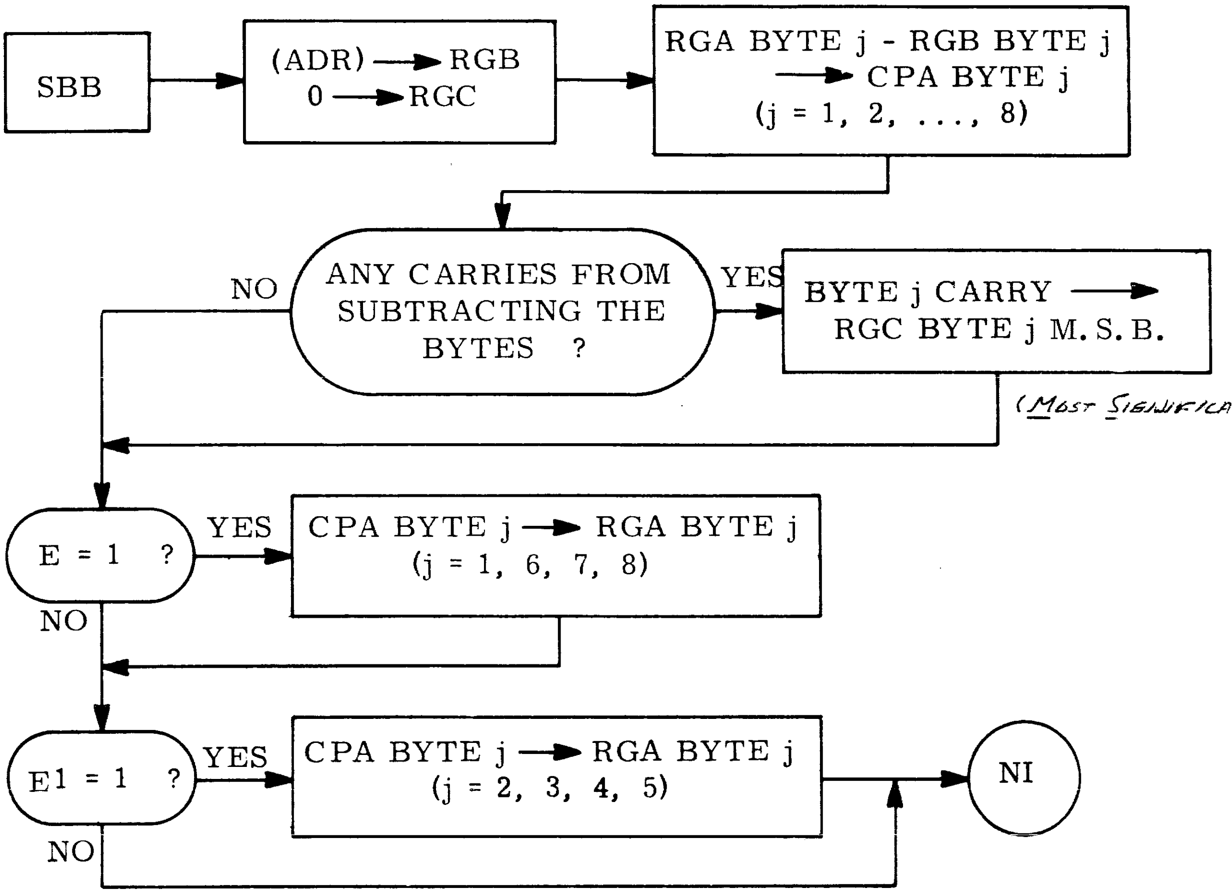
OPERATION: Subtract (ADR) from RGA in 8-bit bytes

INSTRUCTION WORD:

26	ACARX	07	—	ADR USE	ADR
0 4	5 7	8 11	12	13 15	16 31

DESCRIPTION: This instruction subtracts (ADR) from RGA in 8-bit bytes. The result in RGA will be in one's complement form when no overflow occurs. When overflow occurs, the carries are stored in RGC. (ADR) remains in RGB. When the E bits are disabled, RGA is unchanged but RGC contains the carries. Execution of this instruction is the same for 64- and 32-bit modes.

FLOW CHART:



(MOST SIGNIFICAN

MNEMONIC CODE: SBEX

OPERATION: Subtract the exponent field(s) of (ADR) from the exponent(s) of RGA

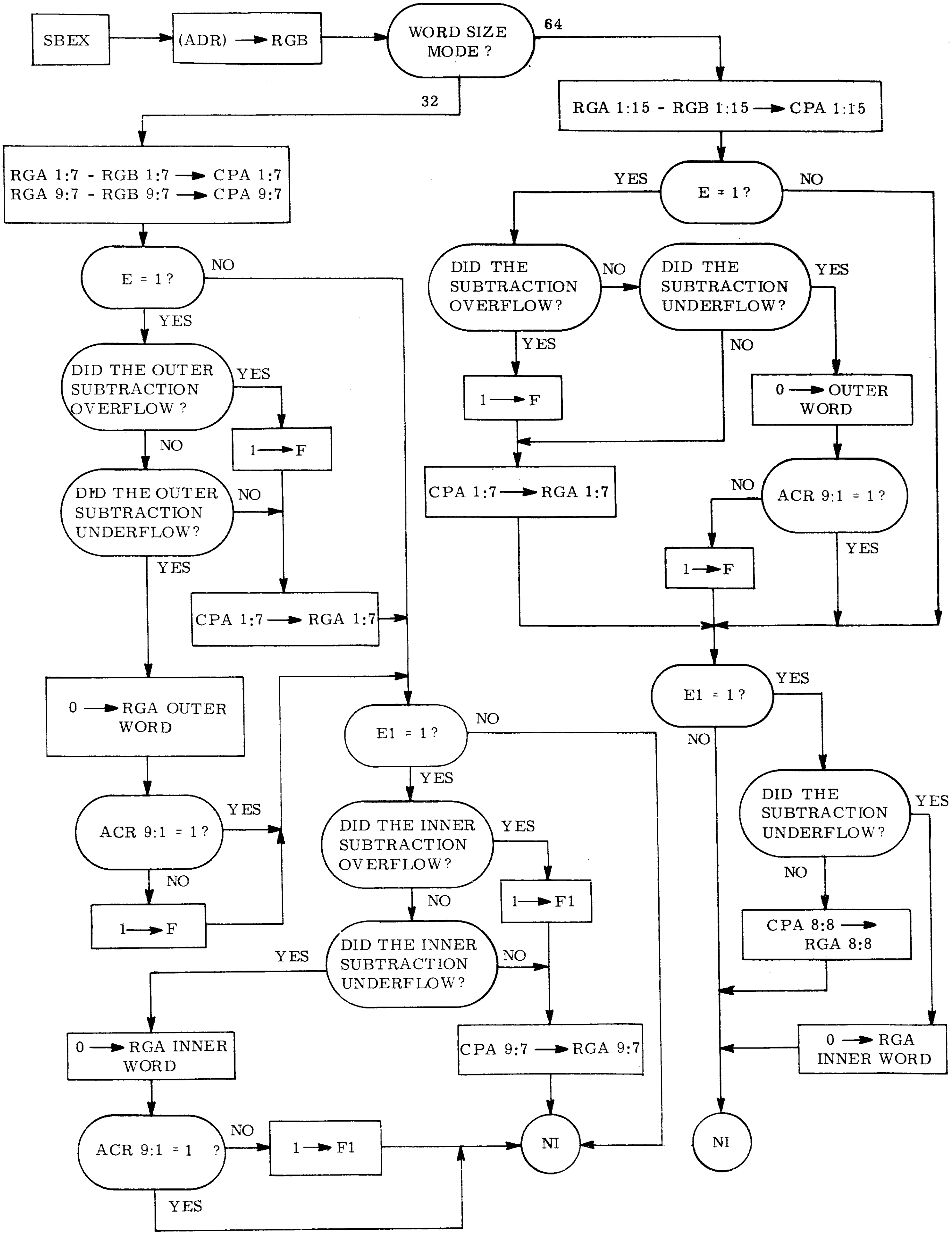
INSTRUCTION WORD:

25	ACARX	01		ADR USE	ADR
0 4	5 7	8 11	12	13 15	16 31

DESCRIPTION: This instruction subtracts the exponent of (ADR) from the exponent of RGA. In 32-bit mode the inner and outer exponents are subtracted independently. When the E bits are disabled, RGA is unchanged and F bits cannot be set.

This instruction may cause the F bit(s) to be set. See pages 4-15, 4-16, items 4, 5, and 6.

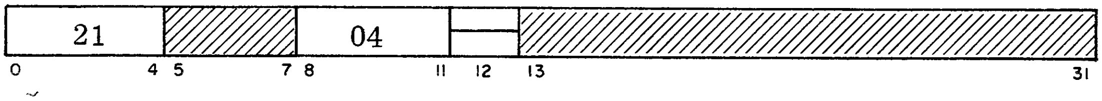
FLOW CHART: See next page.


SBEX
(ADR) → RGB
WORD SIZE MODE ?
64
32
RGA 1:15 - RGB 1:15 → CPA 1:15
RGA 1:7 - RGB 1:7 → CPA 1:7
RGA 9:7 - RGB 9:7 → CPA 9:7
E = 1?
YES
NO
E = 1?
NO
YES
DID THE SUBTRACTION OVERFLOW?
NO
DID THE SUBTRACTION UNDERFLOW?
YES
YES
NO
1 → F
0 → OUTER WORD
CPA 1:7 → RGA 1:7
NO
ACR 9:1 = 1?
YES
1 → F
DID THE OUTER SUBTRACTION OVERFLOW ?
YES
NO
1 → F
DID THE OUTER SUBTRACTION UNDERFLOW?
NO
YES
CPA 1:7 → RGA 1:7
0 → RGA OUTER WORD
E1 = 1?
YES
NO
E1 = 1?
NO
YES
ACR 9:1 = 1?
YES
NO
1 → F
DID THE INNER SUBTRACTION OVERFLOW?
YES
NO
1 → F1
DID THE SUBTRACTION UNDERFLOW?
YES
NO
CPA 8:8 → RGA 8:8
DID THE INNER SUBTRACTION UNDERFLOW?
YES
NO
0 → RGA INNER WORD
CPA 9:7 → RGA 9:7
0 → RGA INNER WORD
ACR 9:1 = 1 ?
NO
1 → F1
YES
NI
NI

MNEMONIC CODE: SCM

OPERATION: Execute one iterative cycle of a multiplication

INSTRUCTION WORD:

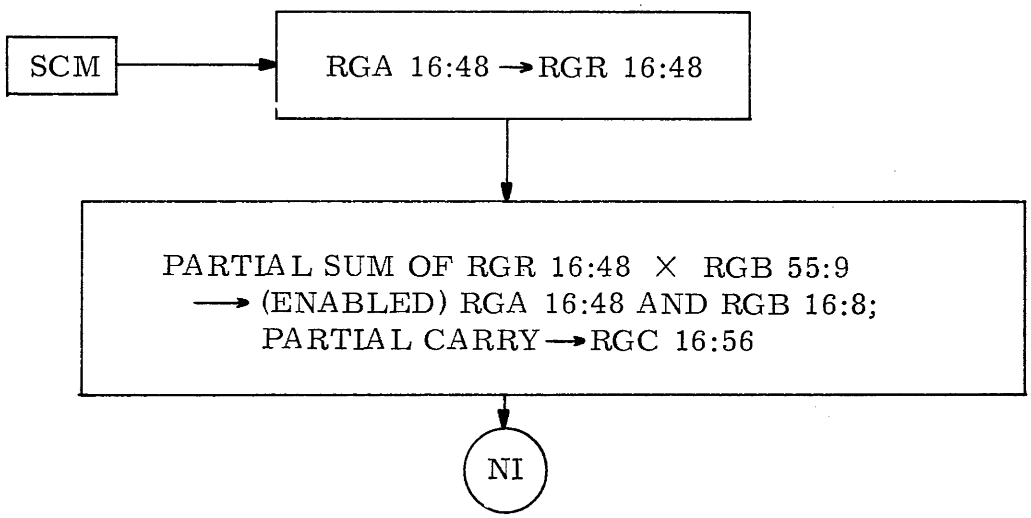


DESCRIPTION: This instruction multiplies RGA mantissa by the nine least significant bits of RGB mantissa, leaving the product in two parts with unassimilated carries. The 48 most significant bits of the "partial sum" will be placed into RGA mantissa, and the eight least significant bits of the "partial sum" will be placed in the eight most significant bits of RGB mantissa, while the next 16 bits of RGB mantissa are set to ONEs. RGC will contain the 56 bits of the "partial carry".

If both E bits are disabled, RGA mantissa will be unchanged. However, RGB and RGC contents will be set as defined above. If E≠E1, the disabled portion of RGA will remain unchanged. The enabled portion will be set to the same contents as if both E bits were enabled. RGB and RGC contents will be as defined above.

This instruction is independent of word size, and therefore results are the same in 32-bit and 64-bit modes.

FLOW CHART:


SCM
RGA 16:48 → RGR 16:48
PARTIAL SUM OF RGR 16:48 × RGB 55:9
→ (ENABLED) RGA 16:48 AND RGB 16:8;
PARTIAL CARRY → RGC 16:56
NI

SHIFT INSTRUCTIONS

The next ten instructions are variations of a basic shift instruction. For brevity, the most frequently used options are described below.

Shift Count with Indexing - The shift count N is a sum of the contents of ADR plus the contents of ACAR (if specified) plus the contents of RGX or RGS (if either is specified). These sums are taken modulo 64 or 32 depending on the word size. Numerically, these sums take on the following values:

for all shifts: N = ADR + ACAR* + RGX (or RGS)*

E Bits Disabled - When the E bits are disabled, the disabled part of RGA is unchanged by the shift instructions. In double-length shifts, RGB is modified as though the E bits were enabled.

End-Around Shifts - When an end-around shift occurs, bits shifted out of the end of a register reappear at the opposite end of the register.

End-Off Shifts - When an end-off shift occurs, the bits at the end of the register opposite from the shift direction are filled with "zeros" as the register is shifted.

Mantissa Shifts - In a mantissa shift, only the bits which constitute the mantissa are acted upon. In 64-bit mode, bits (16:48) will be affected. In 32-bit mode, PE mode bit E controls the outer mantissa (bits 40:24), and PE mode bit E1 controls the inner mantissa (bits 16:24).

Logical Shifts - In a logical shift in 64-bit mode, all of the bits of a word are acted upon. In 32-bit mode, each half-word is acted upon separately.

Double-Length Shifts - For a double-length shift, the two 64-bit registers (RGA and RGB) are effectively acted upon as one 128-bit register. In 32-bit mode, double-length shifts give results which are undefined for purposes of this manual.

Note: For purposes of this manual, the results of any of the shift instructions are undefined when, in 64-bit mode, E ≠ E1.

*If not specified, these terms are zero.

SHIFT INSTRUCTIONS (Continued)

MNEMONIC CODE: RTAL

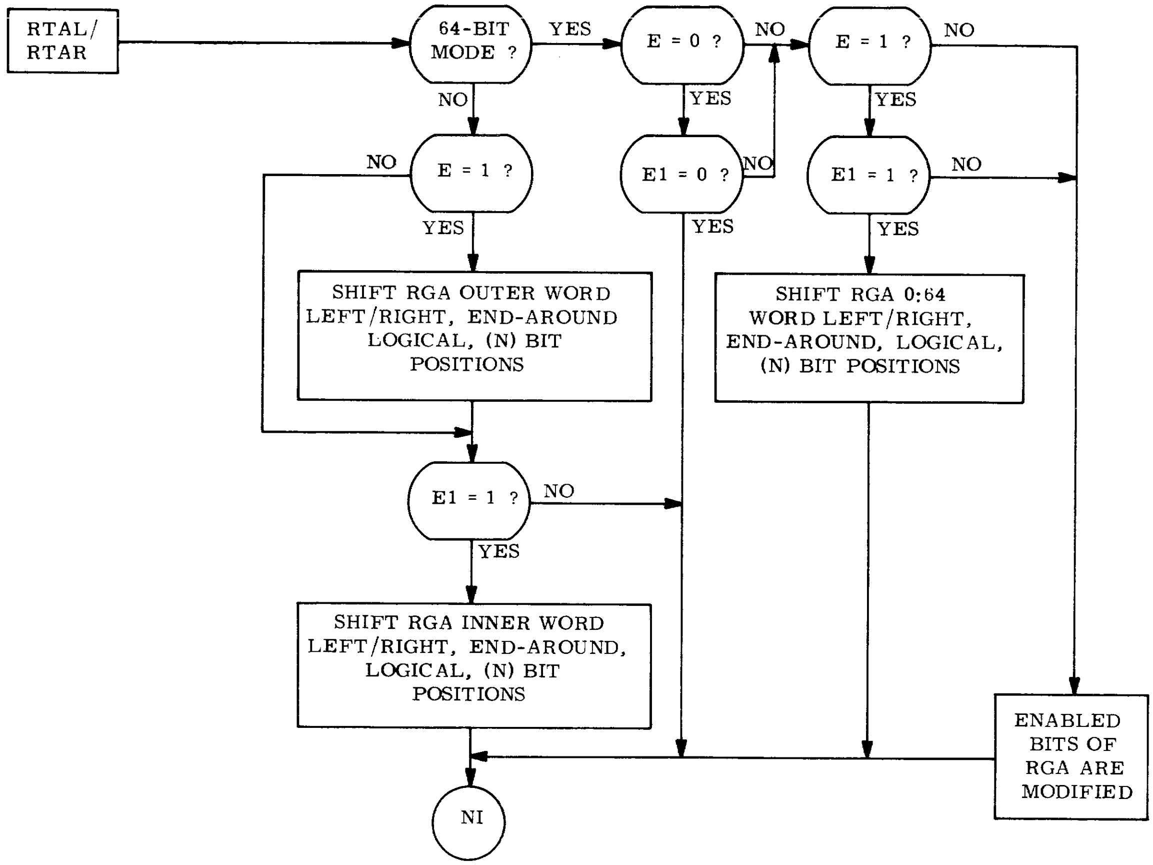
OPERATION: Shift left, end-around, logical, single-length

INSTRUCTION WORD:

35	ACARX	13	—	ADR USE	ADR
0 4	5 7	8 11	12	13 15	16 31

DESCRIPTION: The contents of RGA are shifted and returned to (enabled portions of) RGA. If the shift is being done in 32-bit mode, the inner and outer words are acted upon separately. In 32-bit mode, the effective shift amount is the shift count modulo 32.

FLOW CHART:


RTAL/ RTAR
64-BIT MODE ?
YES
NO
E = 0 ?
NO
YES
E = 1 ?
NO
YES
E = 1 ?
NO
YES
E1 = 0 ?
NO
YES
E1 = 1 ?
NO
YES
SHIFT RGA OUTER WORD LEFT/RIGHT, END-AROUND LOGICAL, (N) BIT POSITIONS
SHIFT RGA 0:64 WORD LEFT/RIGHT, END-AROUND, LOGICAL, (N) BIT POSITIONS
E1 = 1 ?
NO
YES
SHIFT RGA INNER WORD LEFT/RIGHT, END-AROUND, LOGICAL, (N) BIT POSITIONS
ENABLED BITS OF RGA ARE MODIFIED
NI

SHIFT INSTRUCTIONS (Continued)

MNEMONIC CODE: RTAR

OPERATION: Shift right, end-around, logical, single-length

INSTRUCTION WORD:

35	ACARX	12	—	ADR USE	ADR
0 4	5 7	8 11	12	13 15	16 31

DESCRIPTION: This instruction is the same as RTAL, except that the shift is to the right.

FLOW CHART: See RTAL for combined flow chart.

SHIFT INSTRUCTIONS (Continued)

MNEMONIC CODE: SHABL

OPERATION: Shift left, end-off, logical, double-length

INSTRUCTION WORD:

37	ACARX	11	—	ADR USE	ADR
0 4	5 7	8 11	12	13 15	16 31

DESCRIPTION: The contents of RGA are shifted left end-off, and returned to (enabled portions of) RGA. Next, RGB is shifted right end-off, "ORed" with RGA, and restored into (enabled portions of) RGA. RGB is then shifted left end-off and returned to RGB.

The effective result of the shift is as follows:

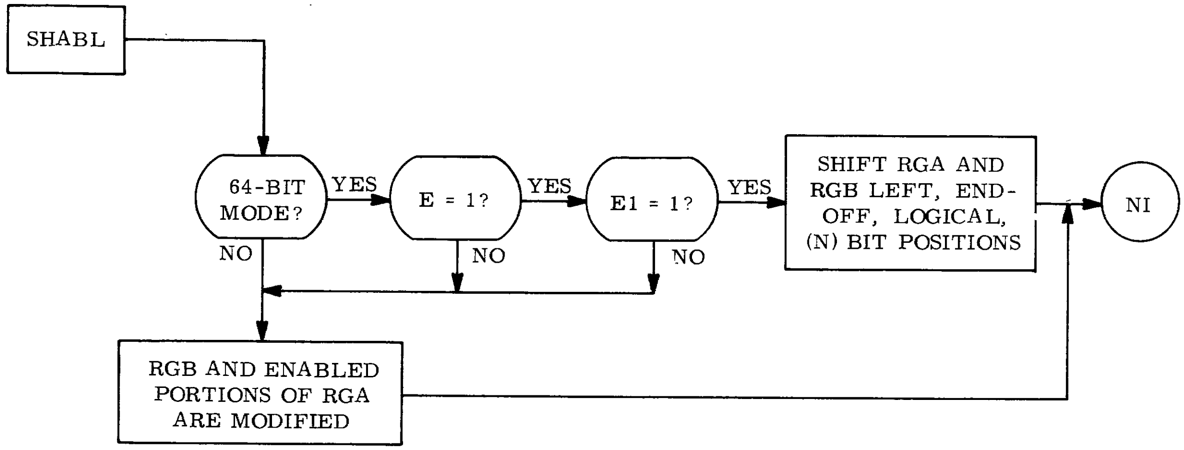
The (N) high-order bits of RGA were deleted;

The (128 - N) remaining bits of RGA and RGB were shifted left to bit 0 of RGA;

The (N) low-order bits of RGB became zeros.

For purposes of this manual the results of this instruction are defined only for 64-bit mode, and when E = E1.

FLOW CHART:


SHABL
64-BIT MODE?
YES
NO
E = 1?
YES
NO
E1 = 1?
YES
NO
SHIFT RGA AND RGB LEFT, END-OFF, LOGICAL, (N) BIT POSITIONS
NI
RGB AND ENABLED PORTIONS OF RGA ARE MODIFIED

SHIFT INSTRUCTIONS (Continued)

MNEMONIC CODE: SHABR

OPERATION: Shift right, end-off, logical, double-length

INSTRUCTION WORD:

37	ACARX	10		ADR USE	ADR
0 4	5 7	8 11	12	13 15	16 31

DESCRIPTION: The contents of RGB are shifted right end-off, and returned to RGB. Next, the contents of RGA are shifted left end-off, "ORed" with RGB, and restored into RGB. The contents of RGA are then shifted right end-off and returned to (the enabled portions of) RGA.

The effective result of the shift is as follows:

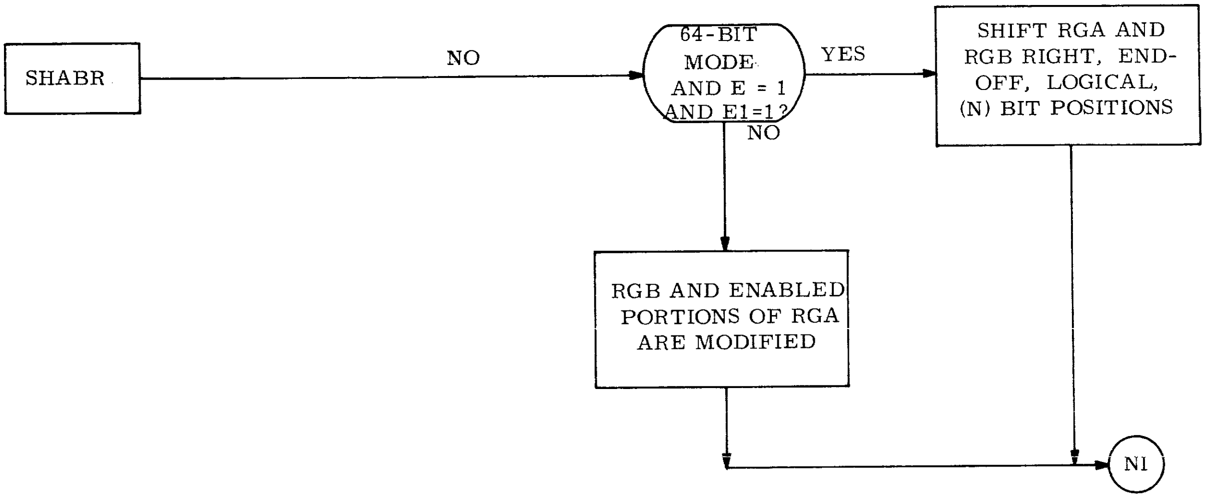
The (N) low-order bits of RGB were deleted;

The remaining (128 - N) bits of RGA and RGB were shifted right to bit 63 of RGB;

The (N) high-order bits of RGA became zeros.

For purposes of this manual, the results of this instruction are defined only for 64-bit mode, and when E = E1.

FLOW CHART:


SHABR
NO
64-BIT MODE AND E = 1 AND E1=1?
YES
SHIFT RGA AND RGB RIGHT, END-OFF, LOGICAL, (N) BIT POSITIONS
NO
RGB AND ENABLED PORTIONS OF RGA ARE MODIFIED
N1

SHIFT INSTRUCTIONS (Continued)

MNEMONIC CODE: SHABML

OPERATION: Shift left, end-off, mantissa only, double-length

INSTRUCTION WORD:

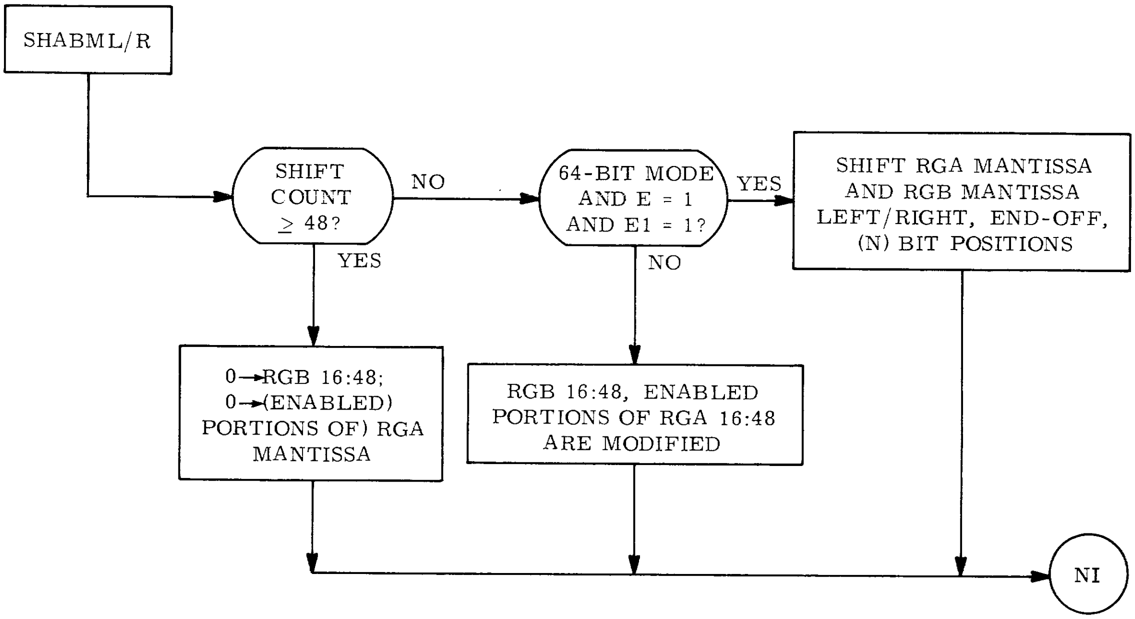
37	ACARX	13	—	ADR USE	ADR
0 4	5 7	8 11	12	13 15	16 31

DESCRIPTION: This is the same as the SHABL instruction, with the following exceptions:

1. Instead of the entire RGA/RGB being shifted, only the mantissa portions are used;

2. There is an added constraint, that if the shift count ≥ 48, modulo 64 the mantissa portion of RGB and (the enabled mantissa portions of) RGA will be set by zero.

FLOW CHART:


SHABML/R
SHIFT COUNT ≥ 48?
NO
YES
64-BIT MODE AND E = 1 AND E1 = 1?
YES
NO
SHIFT RGA MANTISSA AND RGB MANTISSA LEFT/RIGHT, END-OFF, (N) BIT POSITIONS
0→RGB 16:48; 0→(ENABLED) PORTIONS OF) RGA MANTISSA
RGB 16:48, ENABLED PORTIONS OF RGA 16:48 ARE MODIFIED
NI

SHIFT INSTRUCTIONS (Continued)

MNEMONIC CODE: SHABMR

OPERATION: Shift right, end-off, mantissa only, double-length

INSTRUCTION WORD:

37	ACARX	12	—	ADR USE	ADR
0 4	5 7	8 11	12	13 15	16 31

DESCRIPTION: This is the same as the SHABR instruction, with the following exceptions:

1. Instead of the entire RGA/RGB being shifted, only the mantissa portions are used;

2. There is an added constraint, that if the shift count $\geq$ 48 modulo 64 the mantissa portion of the RGB and (the enabled mantissa portions of) RGA will be set to zero.

FLOW CHART: See the SHABML instruction for the combined flow chart.

SHIFT INSTRUCTIONS (Continued)

MNEMONIC CODE: SHAL

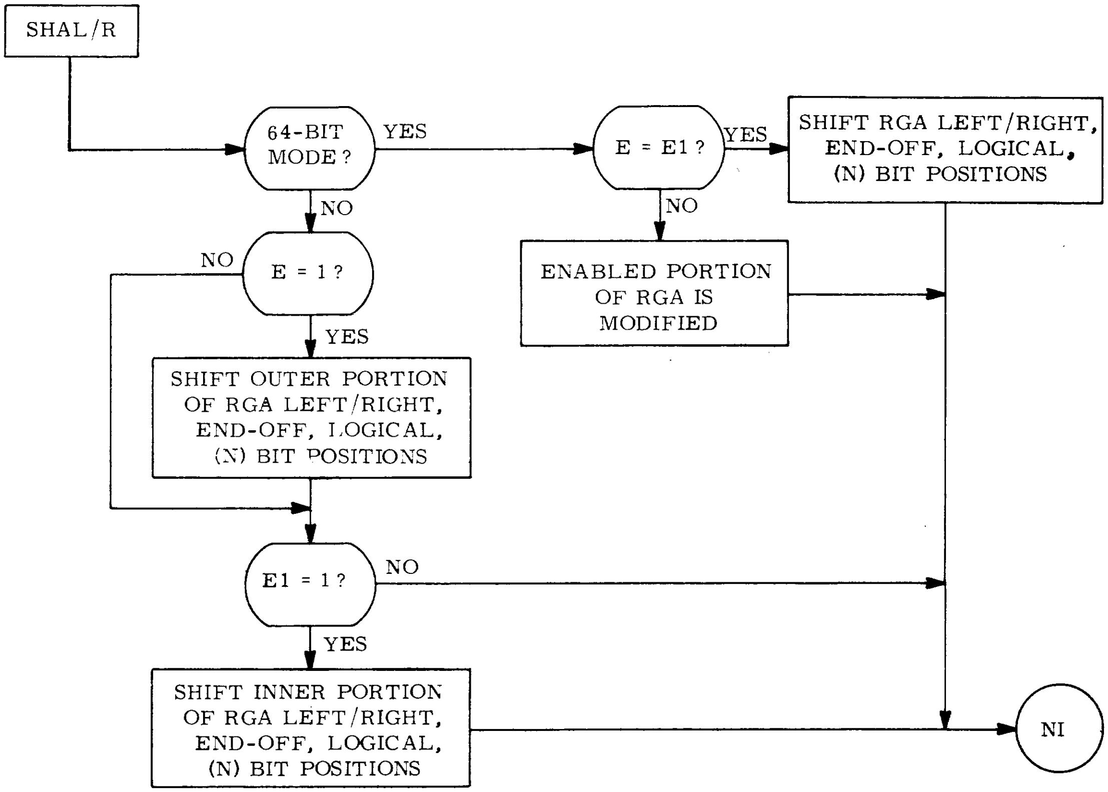
OPERATION: Shift left, end-off, logical, single-length

INSTRUCTION WORD:

35	ACARX	01	—	ADR USE	ADR
0 4	5 7	8 11	12	13 15	16 31

DESCRIPTION: The contents of RGA are shifted and returned to RGA. If the shift is being done in 32-bit mode, the inner and outer words are acted upon separately.

FLOW CHART:



SHAL/R
64-BIT MODE ?
YES
NO
E = E1 ?
YES
NO
SHIFT RGA LEFT/RIGHT, END-OFF, LOGICAL, (N) BIT POSITIONS
ENABLED PORTION OF RGA IS MODIFIED
E = 1 ?
NO
YES
SHIFT OUTER PORTION OF RGA LEFT/RIGHT, END-OFF, LOGICAL, (N) BIT POSITIONS
E1 = 1 ?
NO
YES
SHIFT INNER PORTION OF RGA LEFT/RIGHT, END-OFF, LOGICAL, (N) BIT POSITIONS
NI

SHIFT INSTRUCTIONS (Continued)

MNEMONIC CODE: SHAR

OPERATION: Shift right, end-off, logical, single-length

INSTRUCTION WORD:

35	ACARX	00	—	ADR USE	ADR
0 4	5 7	8 11	12	13 15	16 31

DESCRIPTION: This instruction is the same as SHAL, except that the shift is to the right.

FLOW CHART: See SHAL instruction for combined flow chart.

SHIFT INSTRUCTIONS (Continued)

MNEMONIC CODE: SHAML

OPERATION: Shift left,end-off, mantissa only, single-length

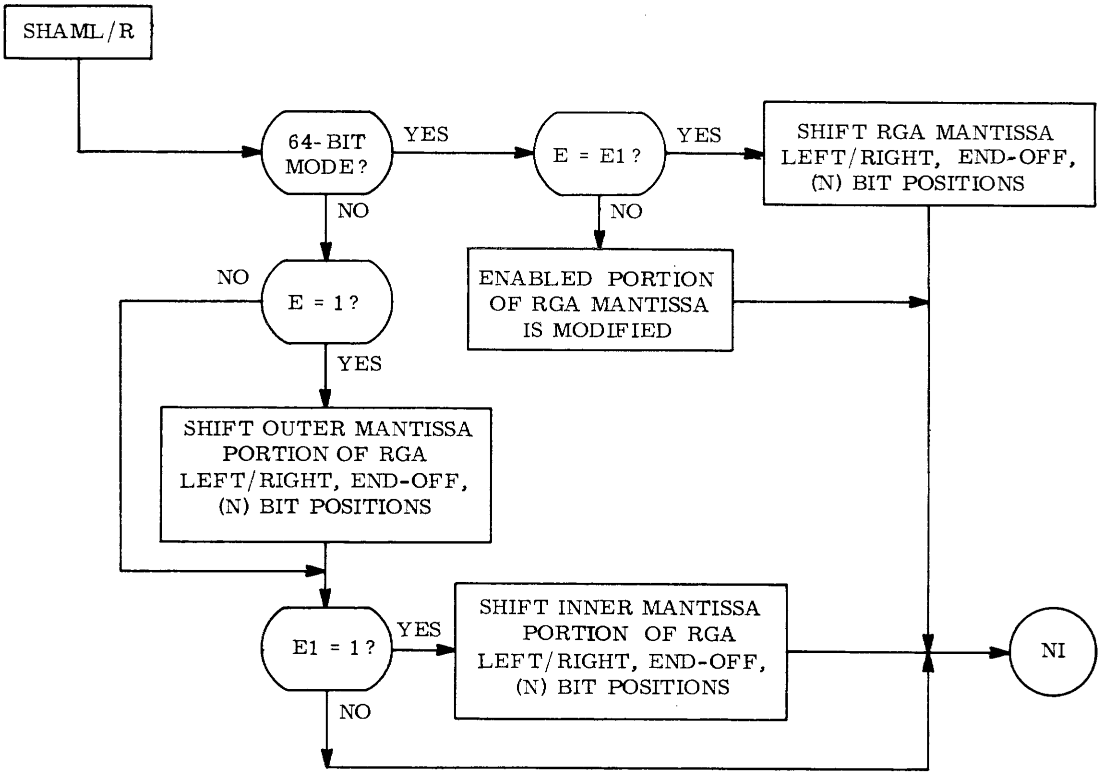
INSTRUCTION WORD:

35	ACARX	11	—	ADR USE	ADR
0 4	5 7	8 11	12	13 15	16 31

DESCRIPTION: The contents of RGA mantissa are shifted left and returned to (enabled mantissa portion of) RGA. If the shift is being done in 32-bit mode, the inner and outer words are acted upon separately.

> Note: If, in 64-bit mode, the shift count $\geq$ 48, modulo 64 (24, modulo 32 for 32-bit mode), the enabled portions of the) RGA mantissa will be set to 0.

FLOW CHART:



SHAML/R
64-BIT MODE?
YES
NO
E = E1 ?
YES
NO
SHIFT RGA MANTISSA LEFT/RIGHT, END-OFF, (N) BIT POSITIONS
ENABLED PORTION OF RGA MANTISSA IS MODIFIED
E = 1 ?
NO
YES
SHIFT OUTER MANTISSA PORTION OF RGA LEFT/RIGHT, END-OFF, (N) BIT POSITIONS
E1 = 1 ?
YES
NO
SHIFT INNER MANTISSA PORTION OF RGA LEFT/RIGHT, END-OFF, (N) BIT POSITIONS
NI

SHIFT INSTRUCTIONS (Continued)

MNEMONIC CODE: SHAMR

OPERATION: Shift right, end-off, mantissa only, single-length

INSTRUCTION WORD:

35	ACARX	10	—	ADR USE	ADR
0 4	5 7	8 11	12	13 15	16 31

DESCRIPTION: This instruction is the same as SHAML, except that the shift is to the right.

FLOW CHART: See SHAML instruction for combined flow chart.

MNEMONIC CODE: ST (A | B | R | S | X)

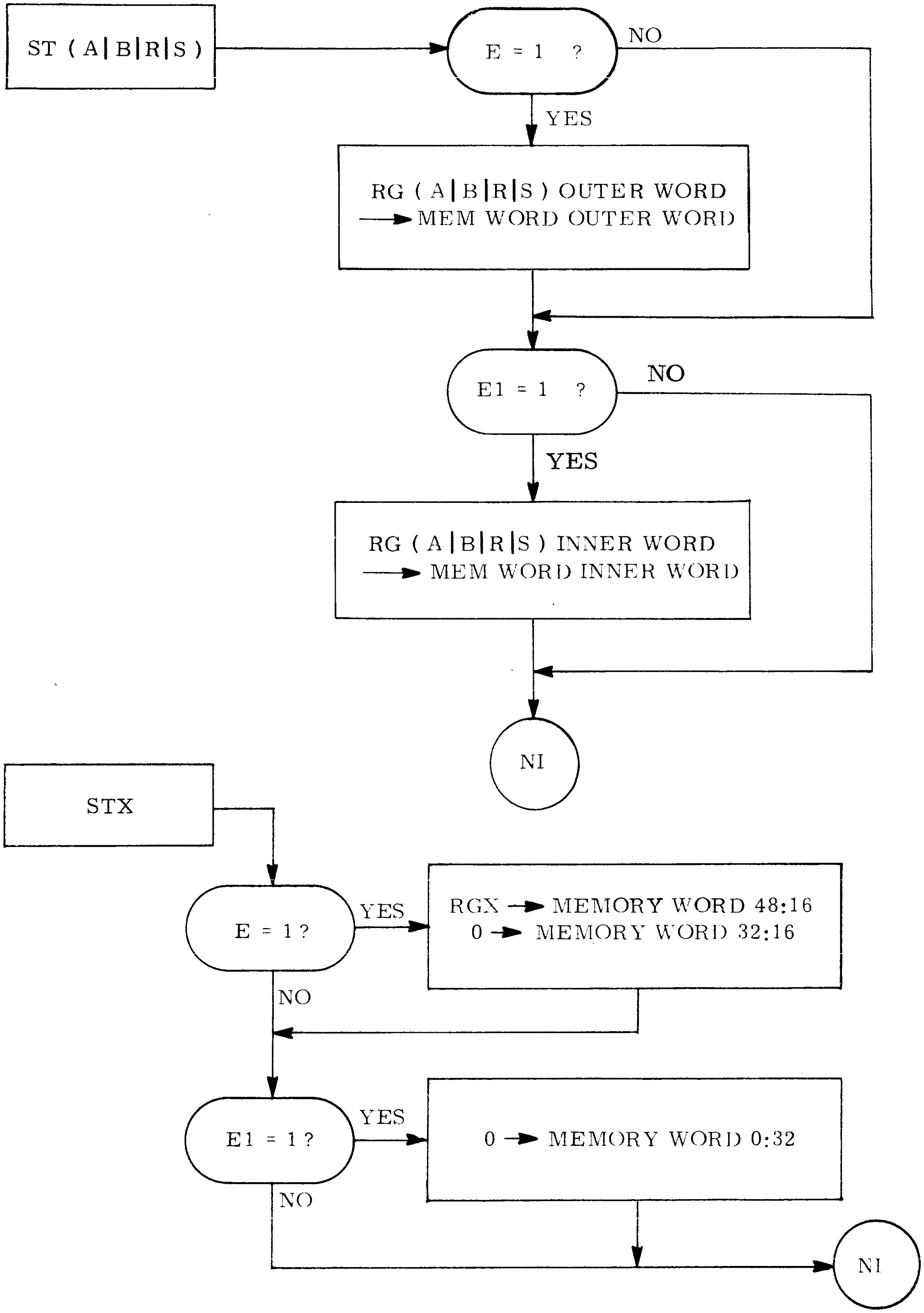
OPERATION: Store from RG (A | B | R | S | X) to memory

INSTRUCTION WORDS:

STA	26	ACARX	12	—	ADR USE	ADR
	0 4	5 7	8 11	12	13 15	16 31
STB	26	ACARX	13	—	ADR USE	ADR
STR	26	ACARX	14	—	ADR USE	ADR
STS	26	ACARX	15	—	ADR USE	ADR
STX	26	ACARX	16	—	ADR USE	ADR

DESCRIPTION: These are five store instructions. The ADR field specifies where the indicated data is stored in memory. These instructions are the same for 64- and 32-bit modes. RGX data goes to memory bit locations (48:16). Disabled E bits prevent changing of the data in memory.

FLOW CHART: See next page.


ST (A | B | R | S)
E = 1 ?
NO
YES
RG (A | B | R | S) OUTER WORD → MEM WORD OUTER WORD
E1 = 1 ?
NO
YES
RG (A | B | R | S) INNER WORD → MEM WORD INNER WORD
NI
STX
E = 1 ?
YES
RGX → MEMORY WORD 48:16
0 → MEMORY WORD 32:16
NO
E1 = 1 ?
YES
0 → MEMORY WORD 0:32
NO
NI

MNEMONIC CODE: SUB

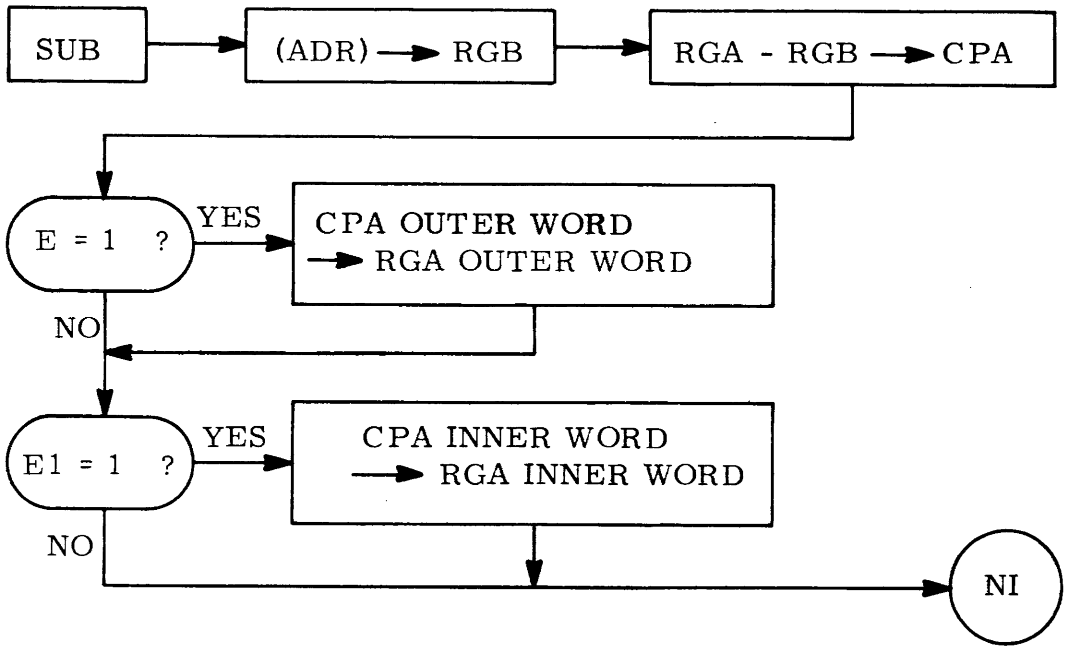
OPERATION: Subtract 64-bit unsigned fixed point number (ADR) from RGA

INSTRUCTION WORD:

26	ACARX	05		ADR USE	ADR
0 4	5 7	8 11	12	13 15	16 31

DESCRIPTION: This instruction subtracts a 64-bit unsigned fixed-point number (ADR) from RGA; the result is placed in RGA if the E bits permit. (ADR) is first fetched to RGB. Overflow generates an end-around-carry, but does not set the F bit. (ADR) remains in RGB. When the E bits are disabled, RGA is unchanged. When in 32-bit mode, this instruction will operate as if in 64-bit mode.

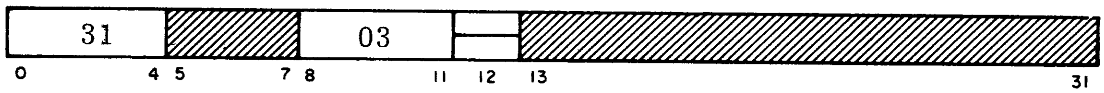
FLOW CHART:


SUB
(ADR) → RGB
RGA - RGB → CPA
E = 1 ?
YES
CPA OUTER WORD → RGA OUTER WORD
NO
E1 = 1 ?
YES
CPA INNER WORD → RGA INNER WORD
NO
NI

MNEMONIC CODE: SWAP

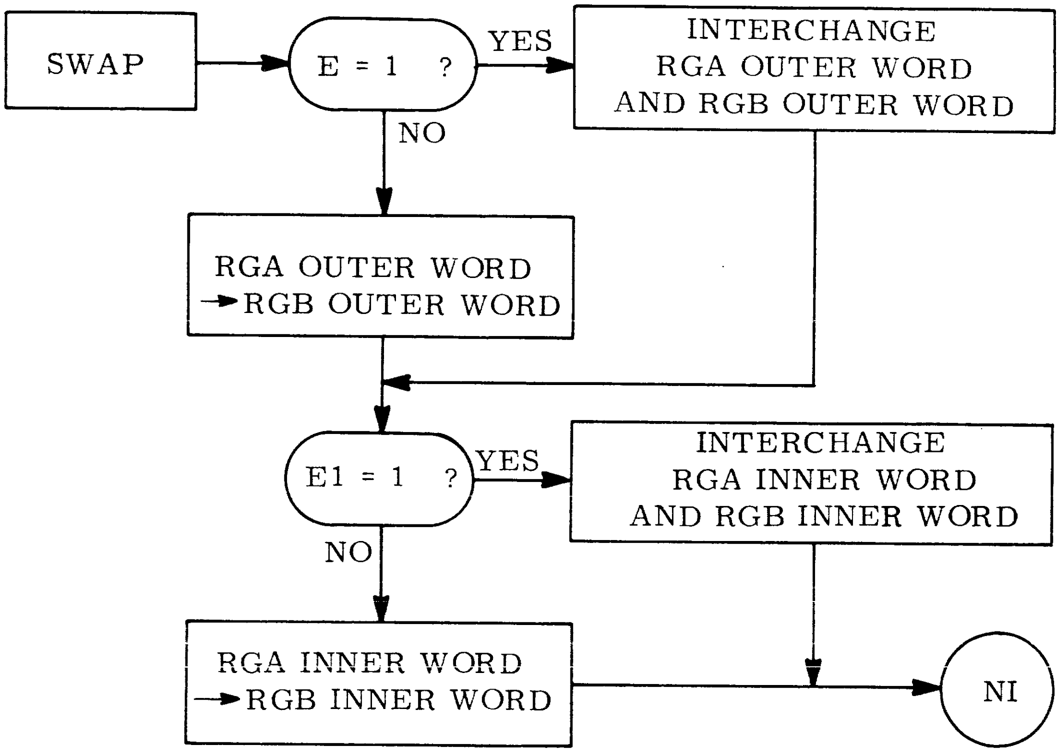
OPERATION: Interchange the contents of RGA and RGB

INSTRUCTION WORD:



DESCRIPTION: This instruction interchanges the contents of RGA with the contents of RGB. When the E bits are disabled, RGA is not changed. SWAP can be used in either 64-bit or 32-bit mode.

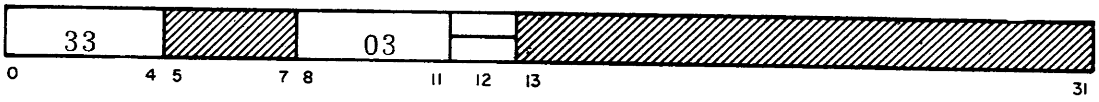
FLOW CHART:


SWAP
E = 1 ?
YES
INTERCHANGE RGA OUTER WORD AND RGB OUTER WORD
NO
RGA OUTER WORD → RGB OUTER WORD
E1 = 1 ?
YES
INTERCHANGE RGA INNER WORD AND RGB INNER WORD
NO
RGA INNER WORD → RGB INNER WORD
NI

MNEMONIC CODE: SWAPA

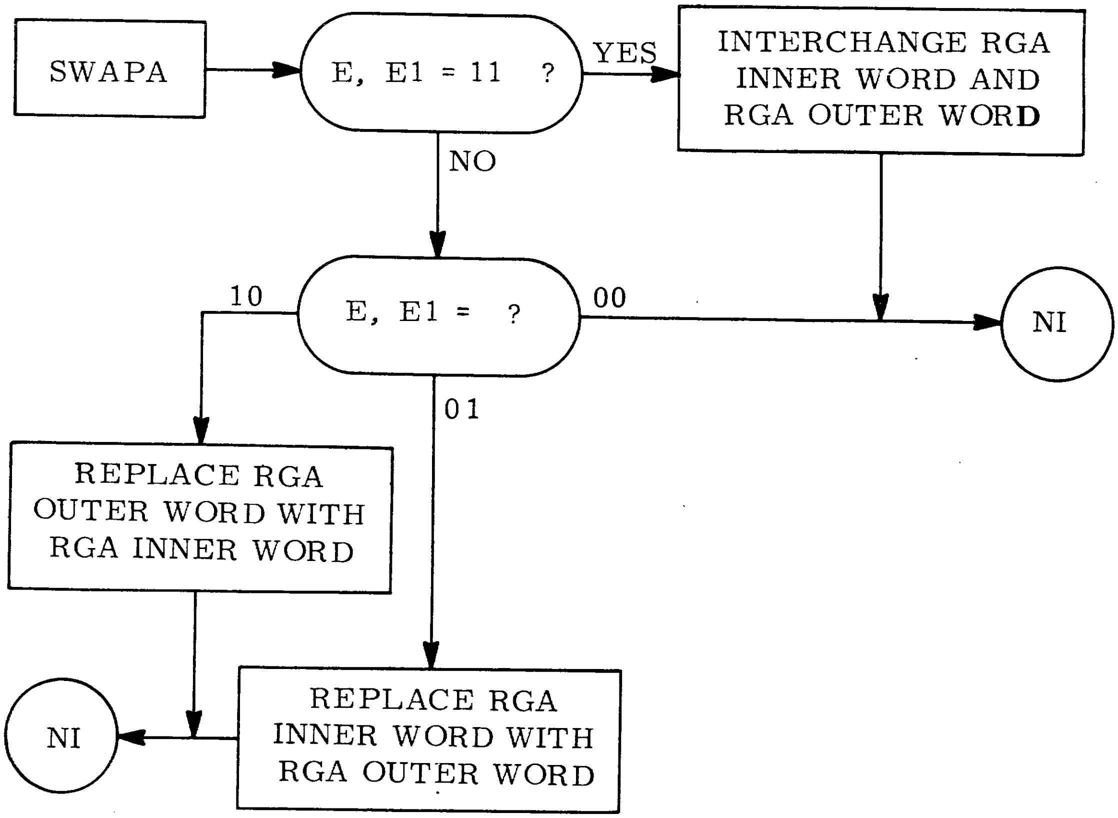
OPERATION: Interchange the inner and the outer operands in RGA

INSTRUCTION WORD:



DESCRIPTION: This instruction interchanges the inner and outer words of RGA. When the E bits are disabled, RGA remains unchanged.

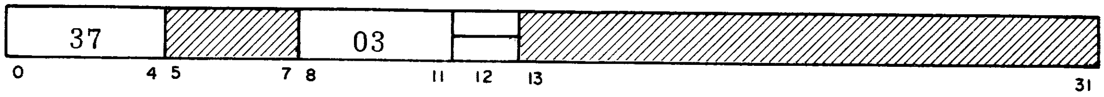
FLOW CHART:


SWAPA
E, E1 = 11 ?
YES
INTERCHANGE RGA INNER WORD AND RGA OUTER WORD
NO
10
E, E1 = ?
00
NI
01
REPLACE RGA OUTER WORD WITH RGA INNER WORD
REPLACE RGA INNER WORD WITH RGA OUTER WORD
NI

MNEMONIC CODE: SWAPX

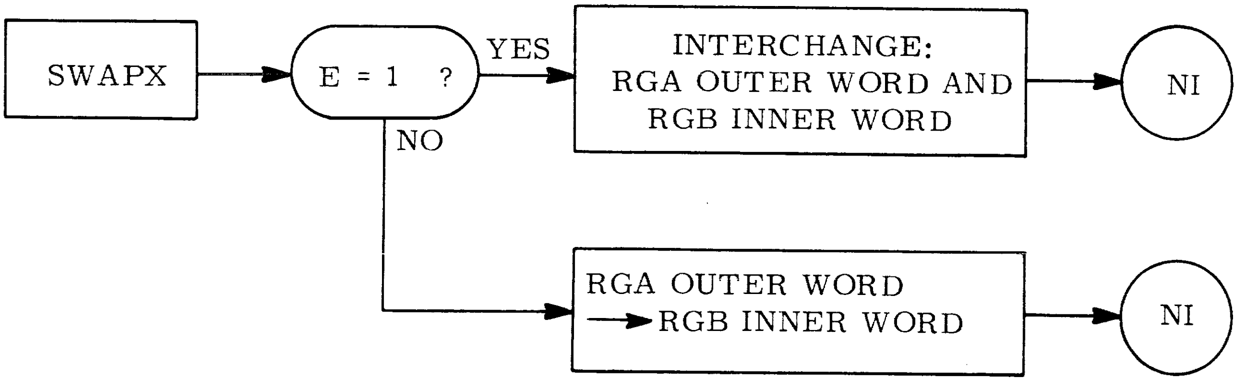
OPERATION: Interchange the outer operand of RGA and the inner operand of RGB

INSTRUCTION WORD:



DESCRIPTION: This instruction interchanges the outer word of RGA and the inner word of RGB. When the E bit is disabled, RGA is not changed; however, the outer word of RGA is copied into the inner word of RGB.

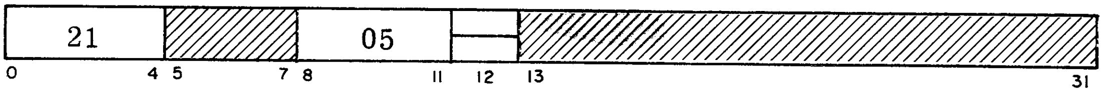
FLOW CHART:



MNEMONIC CODE: T3A

OPERATION: Transfer contents of RGC to RGA

INSTRUCTION WORD:

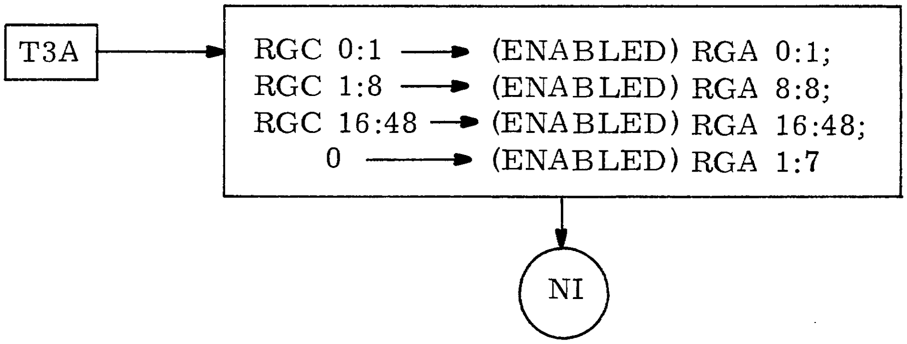


DESCRIPTION: This instruction transfers the contents of RGC to RGA, as follows:

RGC 0:1 is transferred to RGA 0:1
RGC 1:8 are transferred to RGA 8:8
RGC 16:48 are transferred to RGA 16:48
RGC 9:7 are not transferred; RGA 1:7 are set to zeroes

When either or both E bits are disabled, the corresponding portion of RGA is unchanged.

FLOW CHART:


T3A
RGC 0:1 → (ENABLED) RGA 0:1;
RGC 1:8 → (ENABLED) RGA 8:8;
RGC 16:48 → (ENABLED) RGA 16:48;
0 → (ENABLED) RGA 1:7
NI

TRANSMIT INSTRUCTIONS:

MNEMONIC CODE: LD (A | B | D | R | S | X)

OPERATION: Transmit source data to register indicated in op code (Source is specified in instruction word bits 5:3, 13:3, and 16:16.)

INSTRUCTION WORDS:

	0 4	5 7	8 11	12	13 15	16 31
LDA	26	ACARX	17	—	ADR USE	ADR
LDB	27	ACARX	00	—	ADR USE	ADR
LDD	22	ACARX	12	—	ADR USE	ADR
LDR	27	ACARX	01	—	ADR USE	ADR
LDS	27	ACARX	02	—	ADR USE	ADR
LDX	27	ACARX	03	—	ADR USE	ADR

DESCRIPTION: The permissible applications of the transmit instructions are shown in the accompanying table. These instructions are performed by enabling the source data through a path in the PE to the input of the destination register, and then clearing and loading the destination register. All destination registers are 64 bits in length except for the mode register (RGD) and the index register (RGX). The instruction LDD is not E-bit sensitive.

RGD is an 8-bit register. Transfers to RGD are from bits 0:8 in the source register. Transfers from RGD are to bits 0:8 in the destination register; the remaining bits in the destination register are undefined.

The mode bit locations within the RGD are defined as follows:

Bit Location	Mode Bit
0	E
1	E1
2	F
3	F1
4	I
5	G
6	J
7	H

RGX is a 16-bit register. Transfers to RGX are from bits 48:16 of the source register. Transfers from RGX are to bits 48:16 of the destination register; the most significant 48 bits of the destination register are cleared.

The transmit instructions are the same for both 64- and 32-bit modes. When the E bits are disabled, RGA, RGS, and RGX cannot be changed.

Variations of Transmit Instruction

Source of Data	Address Bit	Destination Register					
		RGA	RGB	RGD	RGR	RGS	RGX
RGA	17	****	LDB	*	LDR	LDS	*
RGB	18	LDA	****	LDD	LDR	LDS	LDX
RGD	22	---	LDB	****	---	---	*
RGR	21	LDA	LDB	---	****	LDS	LDX
RGS	20	LDA	LDB	---	LDR	****	LDX
RGX	19	LDA	LDB	---	LDR	LDS	****
MEM	**	LDA	LDB	---	LDR	LDS	LDX
Literal	***	LDA	LDB	*	LDR	LDS	LDX

* No direct path available.

** ADR USE field BIT 15 set; ADR contains memory address; RGS and RGX indexing is permitted.

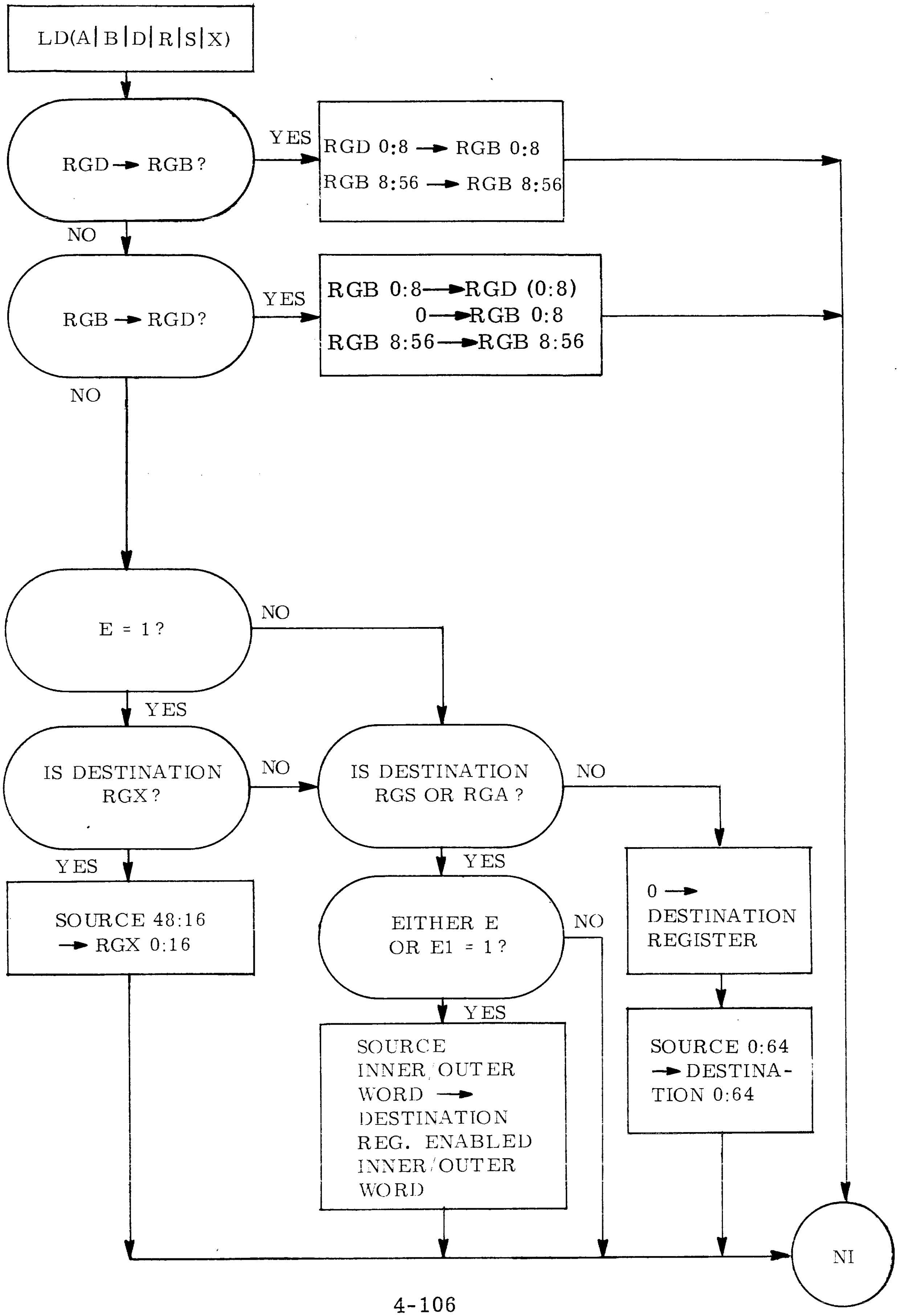
*** ADR USE field BITS 13-15 reset; ADR plus ACAR equal the literal; RGS and RGX indexing is not permitted.

**** Illegal instruction

--- Not used.

Note: In all cases, except where the source of data is memory or a literal, bit 13 of the ADR USE field is set, bit 15 is reset, and RGS and RGX indexing is not permitted. ACAR indexing is permitted, however.

FLOW CHART: See next page.


LD(A|B|D|R|S|X)
RGD → RGB?
YES
RGD 0:8 → RGB 0:8
RGB 8:56 → RGB 8:56
NO
RGB → RGD?
YES
RGB 0:8 → RGD (0:8)
0 → RGB 0:8
RGB 8:56 → RGB 8:56
NO
E = 1?
NO
YES
IS DESTINATION RGX?
NO
IS DESTINATION RGS OR RGA?
NO
YES
YES
SOURCE 48:16 → RGX 0:16
EITHER E OR E1 = 1?
NO
0 → DESTINATION REGISTER
YES
SOURCE INNER/OUTER WORD → DESTINATION REG. ENABLED INNER/OUTER WORD
SOURCE 0:64 → DESTINATION 0:64
NI

MNEMONIC CODE: XD

OPERATION: Subtract (ADR) (48:16) from RGX

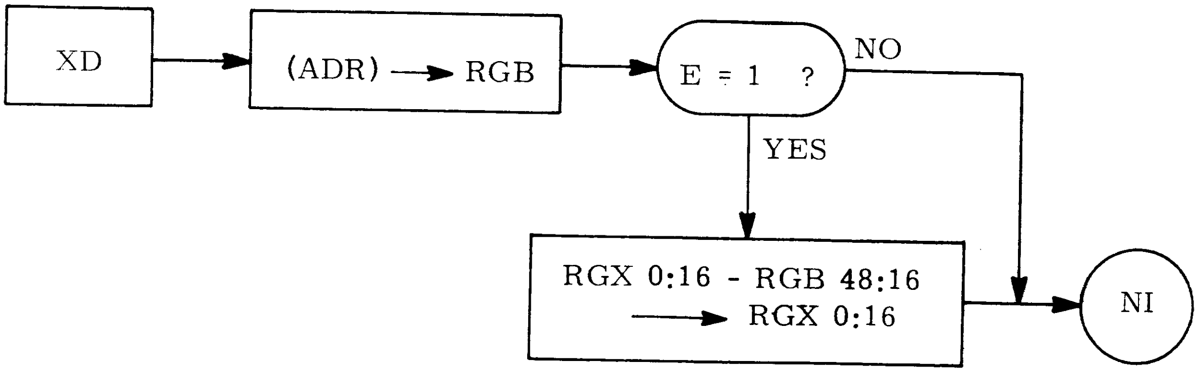
INSTRUCTION WORD:

25	ACARX	03		ADR USE	ADR
0 4	5 7	8 11	12	13 15	16 31

DESCRIPTION: This instruction modifies the index value by subtracting (ADR) from the contents of RGX. The result is returned to RGX. If overflow occurred, the result is modulo 16 bits. The instruction is defined for both 64- and 32-bit modes. When the E bit is disabled, RGX is unchanged.

The subtract is effected by taking the 2's complement (two's complement = one's complement plus 1) of (ADR 48:16) and adding to the contents of RGX. No end-around-carry is generated.

FLOW CHART:



MNEMONIC CODE: XI

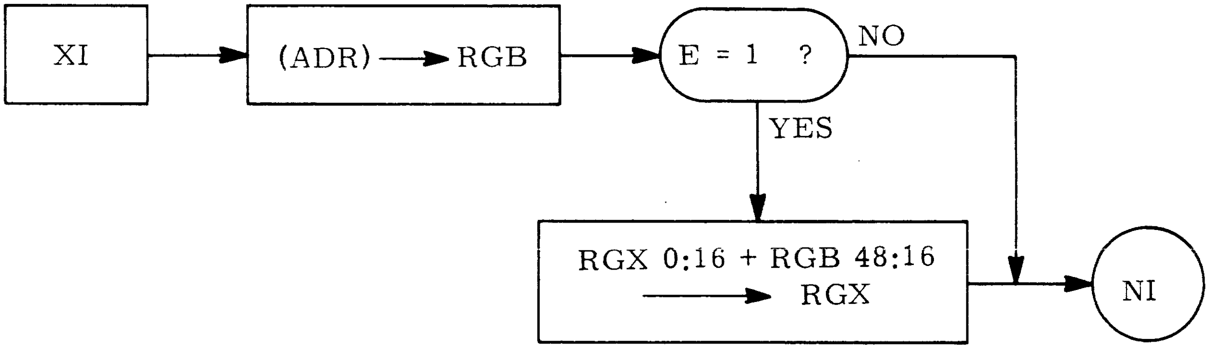
OPERATION: Add (ADR) (48:16) to RGX

INSTRUCTION WORD:

25	ACARX	02		ADR USE	ADR
0 4	5 7	8 11	12	13 15	16 31

DESCRIPTION: This instruction modifies the index value by adding (ADR) to the contents of RGX. The result, in RGX, is modulo 16 bits. Instruction XI is defined for both 64- and 32-bit modes. When the E bit is disabled, RGX is unchanged. No end-around-carry is generated.

FLOW CHART:



CONTENTS

(See Index on Reverse Side)

TMU INSTRUCTION INDEX

Mnemonic Code	Octal Code	Reference Page
EFA	160	5-16
EFF	164	5-18
LICR	041	5-20
LISR	040	5-21
RPT	001	5-22
RUN	020	5-23
SA	007	5-24
SAT	047	5-25
SIS	120	5-26
SIV	100	5-27
SL	006	5-24
SLT	046	5-25
SOC	011	5-30
SOD	010	5-32
SR	005	5-24
SRT	045	5-25
TIC	121	5-33
TOC	002	5-34
WIS	044	5-35

SECTION V
TEST MAINTENANCE UNIT

The Test Maintenance Unit (TMU) is a functional component of the Control Unit (CU). It serves three principal purposes: as the control information input-output interface between an ILLIAC IV quadrant and the B6700 system; as the controller for the other subunits within the CU; and as the medium by which manual, semiautomatic, and automatic testing of the system may be accomplished. Figure 5-1 is a block diagram of the TMU.

Functionally, the TMU acts much like the control panel of a conventional system. The pushbuttons on such a system actually constitute instructions with address either implied or set into panel switches that cause a specific command to be performed as, for example, loading the instruction counter. In the TMU this implied structure is mechanized so that control pushbuttons on the TMU Panel are actually encoded into a command register and then executed. The B6700 has access to this command register – via the Descriptor Controller (DC) —and thus can simulate manual manipulation of system controls. The TMU also functions as the window through which operation of the CU may be monitored.

There are two input and two output ports to the TMU (excluding CU interfaces). Input may be from the TMU Maintenance Panel, the B6700, or both. Output from the system may be observed on a CRT display of register names and formatted octal values, or the same information may be accessed by the B6700 for display.

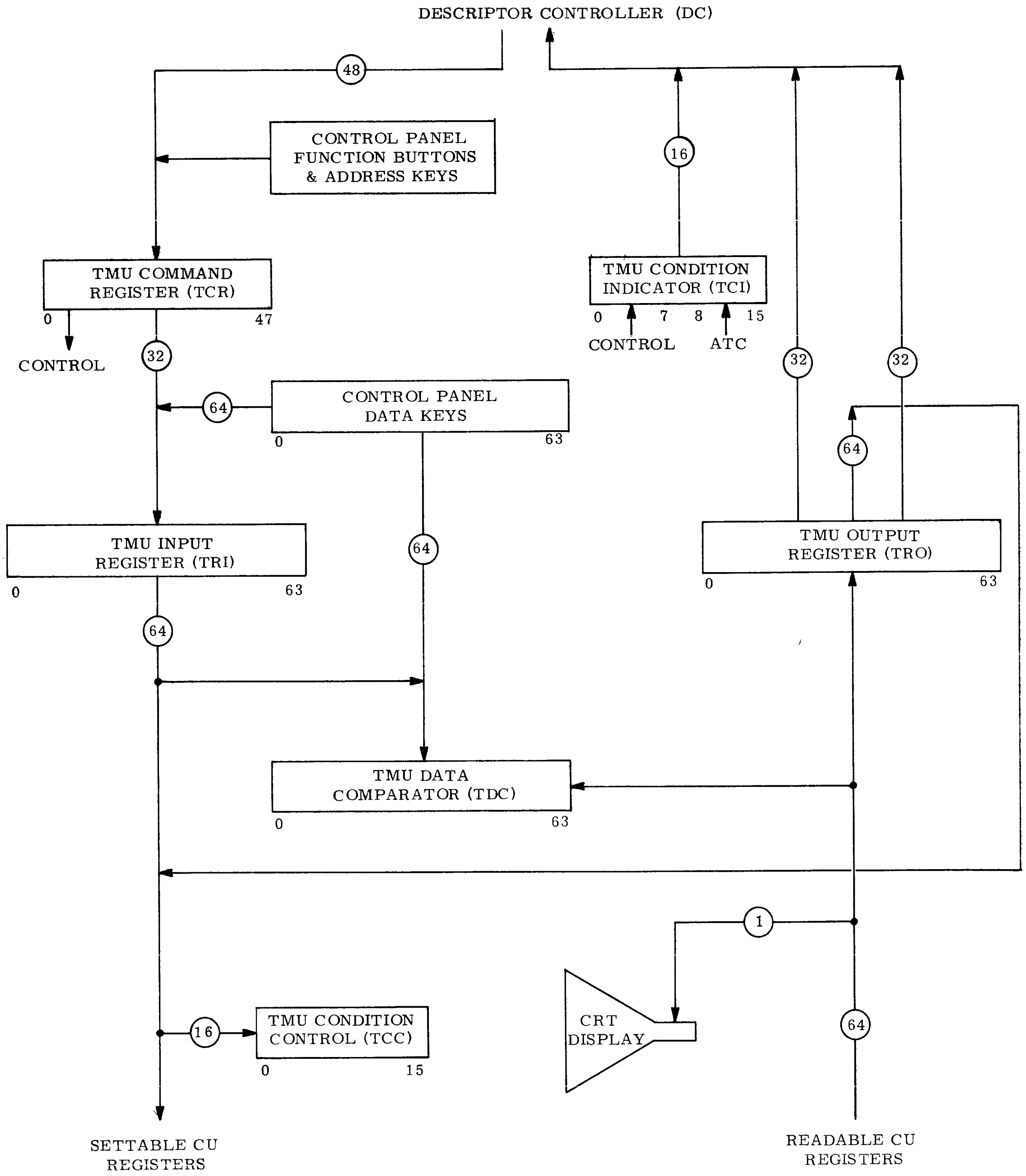

DESCRIPTOR CONTROLLER (DC)
48
CONTROL PANEL FUNCTION BUTTONS & ADDRESS KEYS
16
TMU COMMAND REGISTER (TCR)
0
47
CONTROL
32
TMU CONDITION INDICATOR (TCI)
0
7
8
15
CONTROL
ATC
32
32
CONTROL PANEL DATA KEYS
64
0
63
64
64
TMU INPUT REGISTER (TRI)
0
63
TMU OUTPUT REGISTER (TRO)
0
63
64
TMU DATA COMPARATOR (TDC)
0
63
1
CRT DISPLAY
16
TMU CONDITION CONTROL (TCC)
0
15
64
SETTABLE CU REGISTERS
READABLE CU REGISTERS

Figure 5-1. Test Maintenance Unit

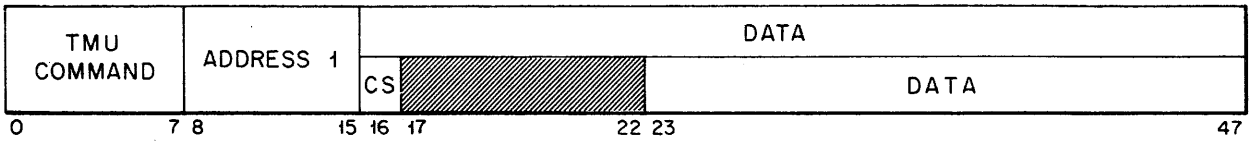
Data written into the TMU is in the form of instructions to be executed; that is, address, variants, and data are all included within the contents of the instruction word. In some cases, the instruction may simply be routed through the TMU for subsequent execution in ADVAST or FINST.

The DC initiates the data write operation by addressing the desired CU. The TMU will not accept the data until it has completed any operations that are currently in progress. When the TMU is, or becomes "not busy", access is granted to the DC, which then reads the instruction into the command register (TCR). At this time, the TMU "busy" flag is set and remains set until that portion of the instruction execution sequence involving the TMU is completed. Next, the TMU requests access to the instruction look-ahead (ILA) section of the CU for further instruction processing. Before proceeding, however, the contents of the ADVAST instruction timer are copied into a holding register to permit the CU to return to its current status following execution of the TMU command. The instruction timer is then reset and execution of the instruction held in the TCR proceeds. Following completion of the instruction, the ADVAST instruction timer is restored to its original status and the CU resumes operation at the place where it was interrupted. Should the repeat latch be set, the instruction in TCR will be repeated before the timer is restored.

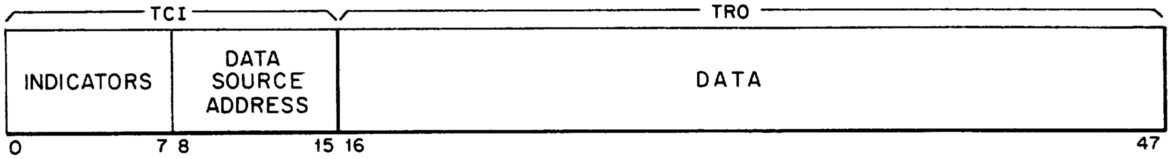
WORD FORMATS

All data transmitted from the B6700 system to the TMU is via the TMU command register (TCR) in the form of a TMU command. The information content of a command word, which is 48 bits long, is shown in the upper format on the next page. Words that are accessed from the TMU by the B6700 are also 48 bits in length. These words comprise the content of the TMU condition indicator register (TCI) and either the left or right 32 bits of the TMU output register (TRO). The format of an output word is shown on the lower portion of the next page.

TMU INSTRUCTION WORD FORMAT



Bits	Field	Function
0-7	TMU Command	TMU instruction.
8-15	Address 1	Register designation.
16	Comparison Selector	Comparison selector (SOC).
16-47 or 23-47	Data	Contains a literal value or control information.



Bit	Field	Function
0	Null	None (contains zero).
1	Illegal	Illegal TMU instruction or address (this includes illegal instructions or addresses for ADVAST or FINST via EFA or EFF instructions). See page 2-21. Illegal CU Addresses.
2	TCL Equal	Set by an SOC command if the comparison result is true.
3	SOC Interrupt	SOC interrupt if comparison result matches the setting of TCC. Comparison may be specified for either equal or unequal.
4	TRO Loaded	Indicates that CU has interrupted the B6700 under program control.
5	CU Halted	Indicates that CU has come to a halted condition.
6, 7	Left/Right Half Loaded	Indicates the section of the TRO from which the data field came. The bit 6, 7 configuration has following significance: 00 - No valid data 01 - Right half (TRO 32:32) 10 - Left half (TRO 0:32) 11 - Left half sent, right waiting
8-15	Data Source Address	Indicates the CU register address from which the TRO was loaded.
16-47	Data	Contains 32 bits of the TRO as indicated in bit 6 of this word.

B6700-TMU COMMUNICATION

The B6700 treats a TMU much like a typewriter inquiry station in that it can expect inputs from it at undefined intervals and that transmissions to or from it are issued by descriptors which cause the exchange of blocks of data. The TMU descriptors are processed by the DC. All connected TMUs are accessible either singly or in a group. When the TMUs are accessed as a group, a block of data may be sent to all TMUs simultaneously – each TMU receiving every word sent – or a block of data may be exchanged with the TMUs accepting or transmitting a word of the data block in round-robin fashion.

From the DC there is only one 48-bit data path to all connected TMUs and thus only one information transfer at a time is possible. Each TMU has its own set of control lines connecting it with the DC. By using these lines a TMU can cause an interrupt signal to be sent to the B6700. The DC may be interrupted for any of the following reasons:

1. If the CU detects an illegal instruction or address as received from the DC (TCI 01).
2. If a data comparison made in the TMU produces the desired result (TCI 03).
3. If the TRO has been loaded with data that must be sent to the B6700 controlling program, such data having been loaded under program control, presumably the executive. TCI 04 is not set in response to branch trace.
4. If the CU has halted owing to an ADVAST halt instruction, CU stalled, breakpoint reached, an interrupt while ACR(01) is set, or a result of certain TMU instructions (TCI 05).
5. If a CU interrupt is attempted while ACR 01 is set (error interrupt).

Upon the occurrence of an interrupt, the TMU will expect the issuance of a read command by the B6700 controlling program requesting additional information regarding the reason for the interrupt. For example, the controlling program might have to examine the contents of certain CU registers as, for instance, the AIN, in order to ascertain the full significance of the interrupt. However, although the read is expected, the TMU is not dependent upon it and will function in the normal manner without its issuance.

Table 5-1. Address Codes for CU Registers[1]

Octal Code	Register Mnemonic	Octal Code	Register Mnemonic	Octal Code	Register Mnemonic
000 to 077	ADB[2]	156	TRO[2]	224	FRR
		157	ACU	225	FRT
		200	ADV[2, 3]	226	FPS[4]
100	AC0[2]	201	FIN[2, 3]	230	IAR
101	AC1[2]	202	ILA[2, 3]	231	IBL
102	AC2[2]	203	MSU[2, 3]	232	ICT
103	AC3[2]	204	TMU[3]	233	IRT
104	ICR	205	FR0	234	ISR
105	IIA[2]	206	FR1	235	IWR
140	ACR	207	FR2	236	IWL
141	ADC	210	FR3	240	MTA
142	AIN[2]	211	FR4	241	MTB
143	AIR[2]	212	FCC	242	MTC
144	ALR	213	FDQ[2]	243	MA
145	AMR[2]	214	FIQ[2]	244	MSR
146	AWR	215	FIR	245	IBR
147	AFR	216	FOR	250	TCC[2]
151	MC0[2]	217	FLP	251	TCI
152	MC1[2]	220	FBZ	252	TCR
153	MC2[2]	221	FRP	253	TIT (AIT)[2]
154	PEM[5]	222	FTC	254	FDR
155	TRI[2]	223	FOQ	255	FQR

[1] All registers are accessible via SOC and SOD instructions, except FPS.

[2] Register has write capability via set-transmit instructions (SAT, SLT, SRT).

[3] Not a physical register, that is, storage elements included in this category are distributed throughout the major functional area noted.

[4] Register FPS has write capability (via set-transmit instructions) but is not accessible for read operations.

[5] PEM represents the ADVAST receivers ARE.

The DC uses the control lines to select a TMU and to request the TMU to perform one of three functions as follows:

1. Accept 48 bits of data into its TCR;
2. Transmit 48 bits of data from its TCI and TRO; or
3. Stop all CU functions and initialize the TMU.

The B6700 can send the following descriptors to the DC for communication with the TMU(s).

1. Read N words from the selected TMU(s);
2. Write N words to the selected TMU(s);
3. Write each word of a block to the selected TMU(s), immediately replacing each word written from B6700 storage with a word (or two words) read from the TMU(s);
4. Stop the selected TMU(s) and the respective quadrant(s).

REGISTER ADDRESS CODES AND ACCESSIBILITY

The CU registers and their address codes which may be read from the TMU are listed in Table 5-1. Registers which may be written into are denoted by the superscript "2".

OPERATION OF THE TMU

The TMU command register (TCR) is always available to accept information from the DC unless it has not completed processing the previously received instruction which may have been received from the DC the Test Maintenance Panel, or th϶ Test Maintenance Display. Execution of a command will take place from the TCR whenever the ADVAST instruction register (AIR) is between instructions or when ADVAST clocks are stopped, or when the instruction is TIC. which assumes stopped clocks.

Output from the TMU is not initiated by TCR commands, but is controlled directly by the DC It is possible for the DC to execute read commands

Table 5-2. Bit Configurations of TCC Register and SRT Instruction Data Field for Diagnostic Usage

Control	TCC Reg. Bit No.	SRT Instr. Data Field Bit	Function
Spare	0	48	----
Initiate	1	49	Used when initializing display memory (Refer to Automatic Initialization)
Lock ICR	2	50	Holds ICR value until bit is reset
Repeat	3	51	Causes next instruction received to be repeated
Interrupt ≠	4	52	Causes DC to be interrupted should an unequal comparison be made by an SOC instruction
Interrupt =	5	53	Causes DC to be interrupted should an equal comparison be made by an SOC instruction
ILA Hold	6	54	Temporarily inhibits clock to ILA
ILA Lock	7	55	Inhibits clock to ILA until restarted by B6700
MSU Hold	8	56	Temporarily inhibits clock to MSU
MSU Lock	9	57	Inhibits clock to MSU until restarted by B6700
ADVAST Hold	10	58	Temporarily inhibits clock to ADVAST
ADVAST Lock	11	59	Inhibits clock to ADVAST until restarted by B6700
FINST Hold	12	60	Temporarily inhibits clock to FINST
FINST Lock	13	61	Inhibits clock to FINST until restarted by B6700
ARRAY Hold	14	62	Temporarily inhibits clock to all PEs in quadrant
ARRAY Lock	15	63	Inhibits clock to all PEs in array until bit is reset

even though the TRO register has not been loaded; this condition will be flagged in the TCI information that is accessed with each word transferred.

It is not necessary for information to be passed directly from DC into registers in the other CU subunits, although this is possible. The TMU input register (TRI) can be loaded with 64 bits of information which is accessible using the normal ADVAST instruction set. It is also possible to set a maskable interrupt bit to notify ADVAST that the TRI contains information.

DIAGNOSTIC FEATURES

To facilitate both automatic and manual diagnostic and maintenance activities, several features have been incorporated in the TMU. The TMU condition control register (TCC) permits the major sections of the Control Unit, including ILA, MSU, ADVAST, and FINST, to be decoupled from the rest of the system to minimize side effects when debugging is in progress. This is accomplished by setting the Hold or Lock bit for a particular unit using the set-transmit instructions and specifying the unit. The Lock bit can only be changed by use of another set-transmit instruction. The Hold bit can be changed by either another set-transmit instruction or by the logic of the TMU which will temporarily turn on the clock for the execution of an instruction sent by the DC, and then turn off the clock. A repeat control is also included. The TCC can also enable the issuance of interrupts to the B6700 dependent upon the comparison between a test value and the value obtained from a CU register. The comparison is mechanized in the TMU data comparator (TDC), which can be used to set a value in the TMU condition indicator register (TCI). It is also possible to freeze the instruction counter for diagnostic purposes. The TCC is loaded by means of a Set Right Transmit (SRT) instruction. Setting the address 1 portion of the instruction word to the address of the TCC causes the least significant 16 bits of the data field to be loaded into the TCC. The bit configuration is described in Table 5-2.

TMU DISPLAY

The TMU display provides the means by which the contents of certain CU registers, controls, and data buffer locations may be monitored. Operation of the TMU may be accomplished automatically under program control, or may be done manually using the TMU control panel and keyboard. Normally, requests for the display of multiple CU registers are handled programmatically, with manual initiations being limited to those required during detailed debugging or diagnostic operations.

The display will present the contents of all registers established by the operator or by program control. The updating of display data occurs at operator request or upon the completion of any instruction causing the CU to be left in a new static state. The display capability includes approximately 50 registers, 64 locations of the ADVAST data buffer (ADB), and various controls dispersed throughout the five major functional areas of the CU, as listed in Table 5-1 (page 5-6). Hereinafter no distinction is made between the types of logical elements displayed, all of them being referred to as CU registers.

The display logic requests register data from the CU by inserting an instruction in the TMU's command register (TCR). The instruction is a Scan Out Data (SOD) with the proper address inserted in TCR 8:8. The display interface card (T-DISP) in the TMU formats the data as required and transfers it, bit serial, to the display where it is stored in core memory. This data is then used to refresh the operator's CRT display. Any data available to the B6700 is also available to the display in this manner.

The CRT screen accommodates a 40-line display of 53 characters per line. This provides for the display of two full 64-bit registers in octal format on the same line as follows:

MMMXXXXXXXXXXXXXXXXXXXXXX   NNNXXXXXXXXXXXXXXXXXXXXXX

where "MMM" and "NNN" represent the mnemonics of the registers being displayed and "X...X" represent the 22 octal characters for each of the registers.

Provisions are made for a read-back of register address information so that the operator can at any time check the source of the displayed data. To do this, he presses the VIEW pushbutton on the control panel. The address information is then displayed in place of the data display. The presentation for the above 64-bit registers would appear as follows:

MMM0000000000000000DDDCCB   NNN0000000000000000DDDCCB

where   MMM and NNN are the register mnemonics as before,
        DDD is the octal encoded contents of the address field,
        CC is the octal encoded contents of the format field,
        B is the binary/octal control bit.

Note that this format contains the same number of characters as the display it replaces. Zeros appear in the appropriate number of positions to the left of the leading character of the octal address.

The display will accommodate any mix of allowable register lengths and will be in exact multiples of 3-bit groups for octal displays or in the exact register bit length for binary displays. Further, any mix of binary and octal displays is allowed at the same time, at the discretion of the initiator (programmer or operator). The maximum register length that can be displayed is 64 bits binary (22 octal characters) and the minimum length is 6 bits if the display is binary or 18 bits (6 octal characters) if the display is octal. Displays for registers having a length shorter than the minimum will have zeros inserted in the appropriate number of left-hand bit positions. The leading bit position of the register contents is displayed left-justified (to the last filler zero if used), the only variations being produced by the optional formatting characters that may precede register mnemonics.

The display is free form in that registers are displayed in the same order in which they are initiated, without regard to size or address. No protection is afforded against a register being split, such that part of its contents may be displayed at the end of one line and the remainder at the beginning of the next. However, the initialization procedure is operable at any time, so that additions or corrections may be made to the list of displayed registers. Thus, the operator can, whenever he chooses, insert a carriage return before the mnemonic of a split register to eliminate the carry-over.

INITILIZATION

In order to initialize or set up a register for future display, the register mnemonic must be entered in display memory. To cause the display to be in binary form instead of the standard octal, the mnemonic is preceded by a comma.

The TMU keyboard utilizes a system of coding such that a three-character keyboard entry fully defines the CU register to be accessed for a display of its contents. For purposes of this display system, the keyboard tabs are assigned octal positions in an 8 X 8 matrix such that any two octal numbers define a unique position on the keyboard. (See Figure 5-2.) The first digit of a two-digit number refers to one of rows 0 through 7 proceeding from top to bottom of the matrix; the second digit, which defines the position within a row, refers to one of columns 0 through 7 proceeding from left to right through the matrix. For example, the octal number "32" would refer to the tab at the unique position established at the fourth row down, third column in from the left side.

	0	1	2	3	4	5	6	7
0	0	1	2	.3	4	5	6	7
1	8	9	*	;	$	(	=	'
2		A	B	C	D	E	F	G
3	H	I	+	•	)	]		%
4	—	J	K	L	M	N	Ø	P
5	Q	R				[		
6	SPACE	/	S	T	U	V	W	X
7	Y	Z	≠	,			"	

Figure 5-2. TMU Display Keyboard Character Set

To initialize the TMU display manually, the operator uses the following procedure:

1. Depress FIELD key;

2. Enter 3-character mnemonic on keyboard (preceding comma optional);

3. Enter any desired formatting (e.g., space, new line, etc.);

4. Repeat (2) and (3) for each register display desired.

The mnemonic will always be displayed ahead of the register contents. After the register mnemonic is entered, the display automatically generates the necessary load signals to signify the end of that register. All data entered from the keyboard is stored in memory, as it is typed, after the field identifier.

TMU INSTRUCTION SET

This section presents descriptions of the instruction set for the TMU. Each flow chart ends with the notation "O. C. ", which signifies "operation complete". This means that the TMU is free to accept a new command from the DC as soon as the TMU has accomplished its function. Thus, instructions such as RUN, which requires that processing in the rest of the CU be initiated in the TMU, are ready for "complete" as soon as the TMU portion of the operation has been completed. The instruction repertoire for the TMU is presented below, followed by the instruction descriptions, which appear in the same order as listed.

Mnemonic Op Code	Octal Op Code TCR 0:8	Operation
EFA	160	Execute from ADVAST instruction register
EFF	164	Execute from FINST
LICR	041	Load instruction counter
LISR	040	Load breakpoint register
RPT	001	Repeat latch set (TCC3)
RUN	020	Run
SA	007	Set all (TRI)
SL	006	Set left (TRI)
SR	005	Set right (TRI)
SAT	047	Set all transmit (TRI)
SLT	046	Set left transmit (TRI)
SRT	045	Set right transmit (TRI)
SIS	120	Single step
SIV	100	Set to initialize value
SOC	011	Scan out compare
SOD	010	Scan out data
TIC	121	Trigger I clocks
TOC	002	Timing oscillator connect
WIS	044	Write instruction storage

MNEMONIC CODE: EFA

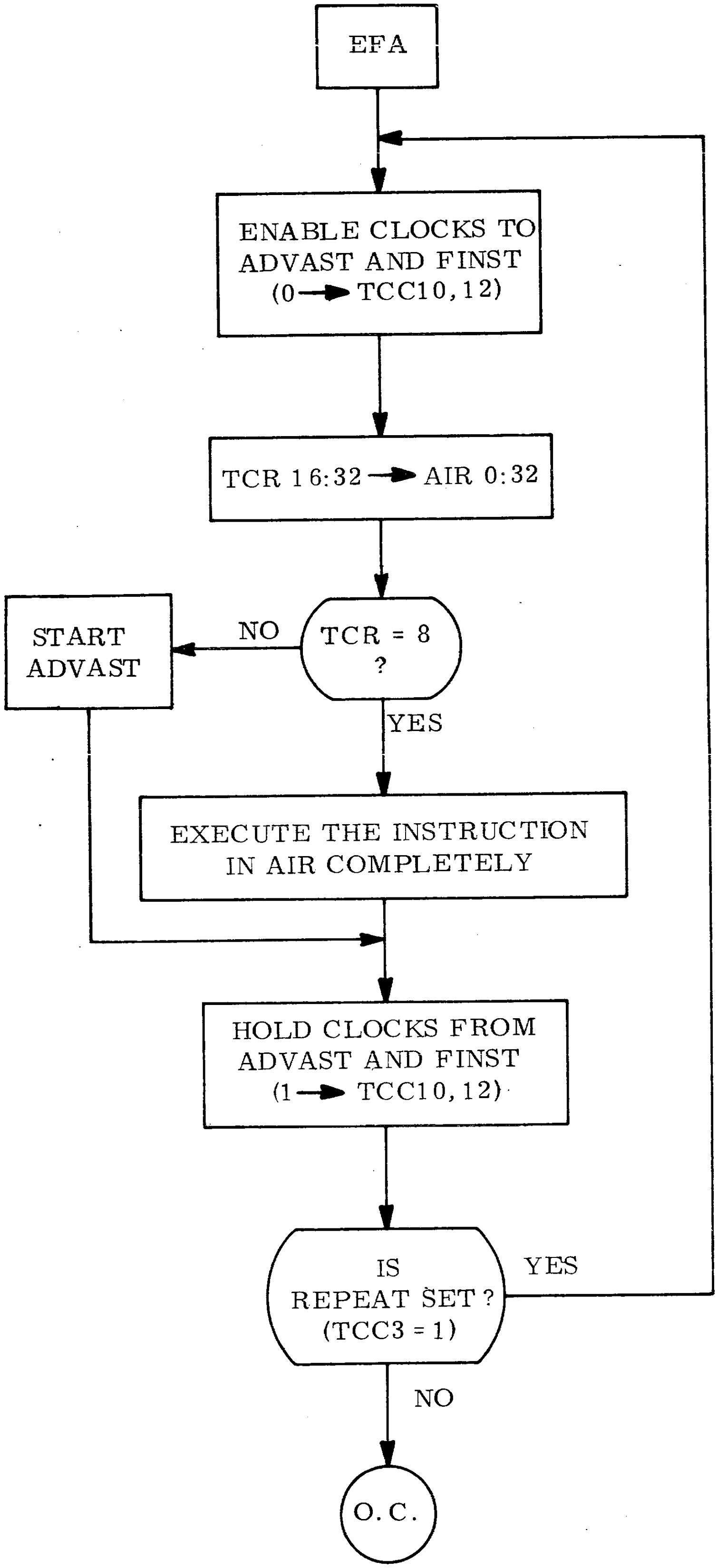
OPERATION: Execute from ADVAST Instruction Register

TCR:

160	▨	PE OR ADVAST INSTRUCTION
0 7	8	16 47

DESCRIPTION: The instruction contained in the data field is sent to the ADVAST instruction register (AIR) in the CU from where it is executed. The CU stops when processing of the instruction has been completed at all applicable CU subunits. Parity is not checked on the instruction. The instruction assumes that ADVAST and FINST are not locked, and further, that the MSU is neither held nor locked if a memory operation or PE indexing is required by the executed instruction (TCC8, 9, 11, and 13 are zeros). If bit 8 is true, a single clock option is invoked. All clocks are shut off after ADVAST receives the X2 control to start the instruction and the TMU will exit, leaving those clocks off. If the ADVAST instruction is LIT, it will not complete. If the ADVAST instruction is BIN or LOAD with an ADB address, it will complete only upon the execution of the next ADVAST instruction, whether called for by EFA or otherwise.

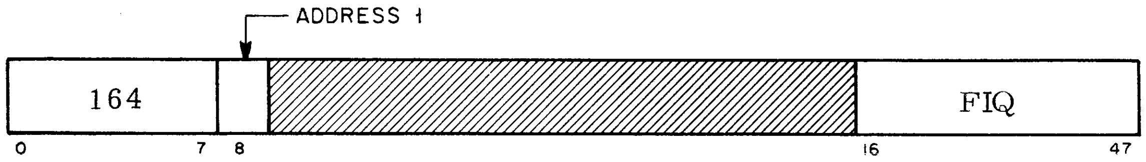
FLOW CHART: See next page.


EFA
ENABLE CLOCKS TO ADVAST AND FINST (0 → TCC10, 12)
TCR 16:32 → AIR 0:32
TCR = 8 ?
NO
START ADVAST
YES
EXECUTE THE INSTRUCTION IN AIR COMPLETELY
HOLD CLOCKS FROM ADVAST AND FINST (1 → TCC10, 12)
IS REPEAT SET? (TCC3 = 1)
YES
NO
O.C.

MNEMONIC CODE: EFF

OPERATION: Execute from FINST

TCR:



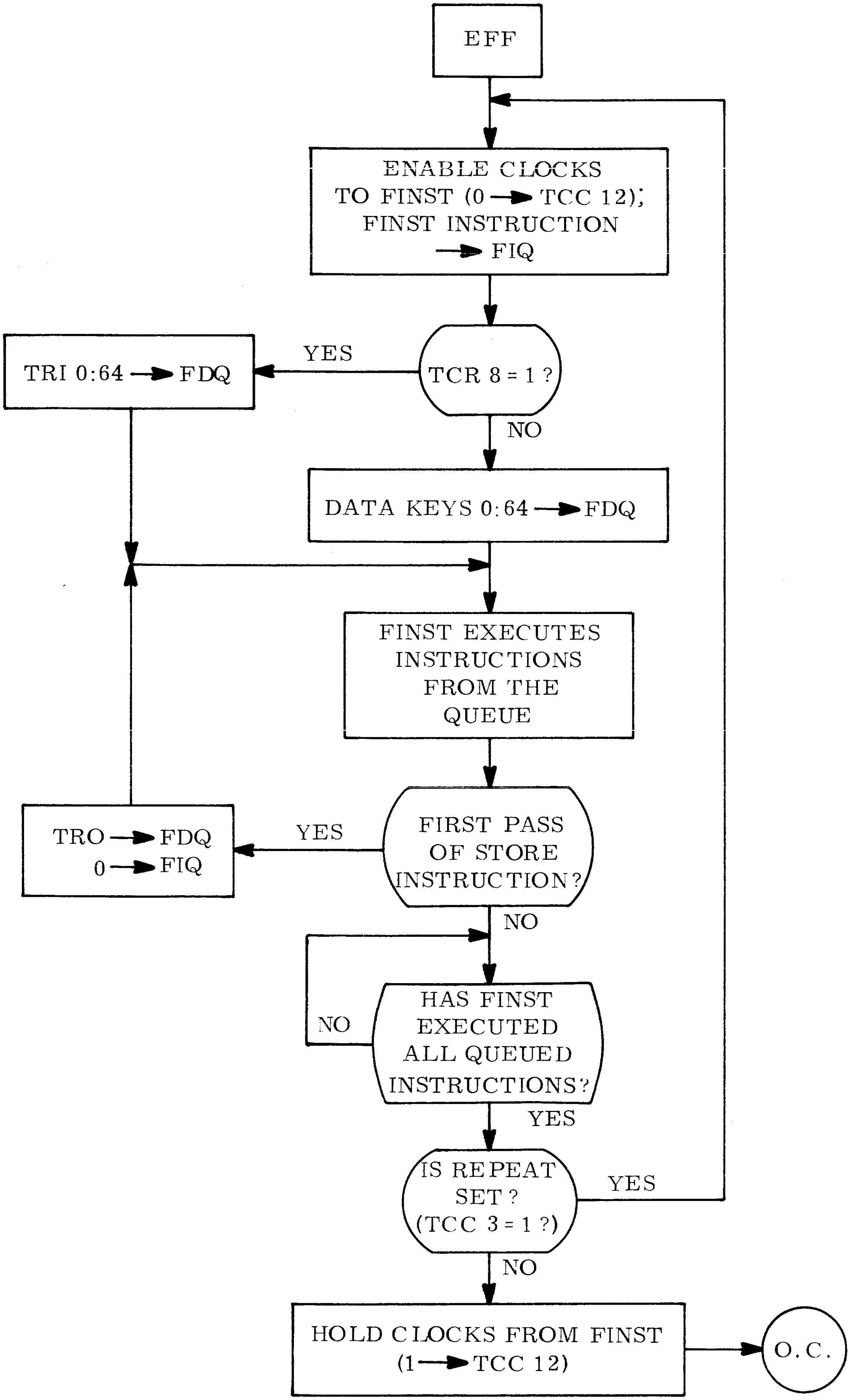
DESCRIPTION: As soon as the queue is not full, the FINST instruction is transferred from the TCR to the FINST instruction queue. The FINST data queue is loaded with a 64-bit value which is taken from the Test/Maintenance Panel data keys or the input register (TRI). Address 1 being "one" specifies the TRI and being "zero" specifies the data keys. This instruction assumes that FINST is not locked and that the MSU is neither held nor locked if a memory operation is required by the FINST instruction (TCC8, 9, and 13 are zeros).

If TCR 8 is set, the contents of TRI will be sent to the data queue (FDQ) and TCR 16 to 47 will be sent to the instruction queue (FIQ). If TCR 8 is not set. the Data Switches will be sent to the data queue (FDQ) and the TCR switches 16 to 47 will be sent to the instruction queue (FIQ).

If the instruction is a STORE the first (relative) queue position will be filled as above. The next instruction queue will be filled with zeros. The accompanying data queue will be filled with the contents of TRO.

Data destined for the instruction queue should be formatted the same as if it were to be preprocessed by advast. The instruction will be packed into twelve bits at AGF. Note that any advast preprocessing, such as indexing, will not be done.

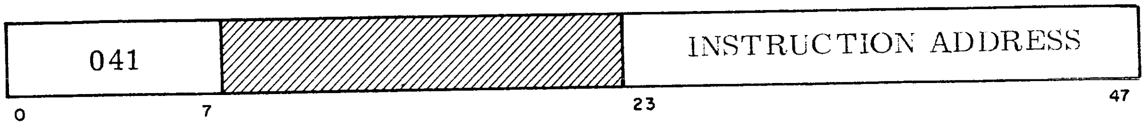
FLOW CHART: See next page.


EFF
ENABLE CLOCKS TO FINST (0 → TCC 12); FINST INSTRUCTION → FIQ
TCR 8 = 1 ?
YES
TRI 0:64 → FDQ
NO
DATA KEYS 0:64 → FDQ
FINST EXECUTES INSTRUCTIONS FROM THE QUEUE
FIRST PASS OF STORE INSTRUCTION?
YES
TRO → FDQ
0 → FIQ
NO
HAS FINST EXECUTED ALL QUEUED INSTRUCTIONS?
NO
YES
IS REPEAT SET ? (TCC 3 = 1 ?)
YES
NO
HOLD CLOCKS FROM FINST (1 → TCC 12)
O.C.

MNEMONIC CODE: LICR

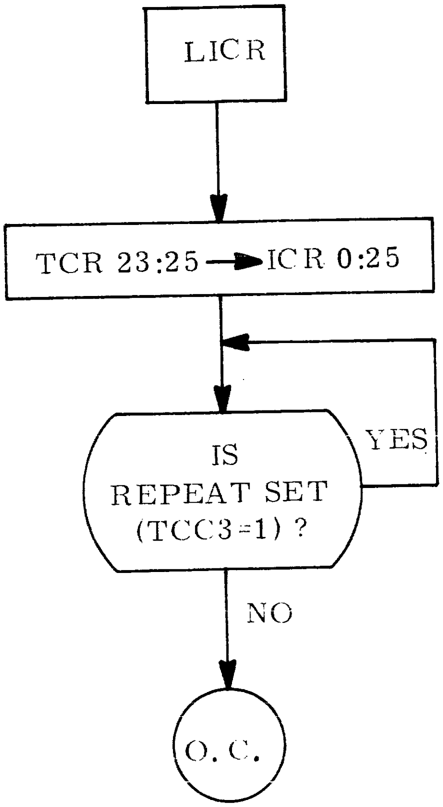
OPERATION: Load Instruction Counter (ICR)

ICR:



DESCRIPTION: The low order 25 bits of the TCR are transferred into the instruction counter (ICR). Subsequent instructions will be fetched and executed beginning with this address value.

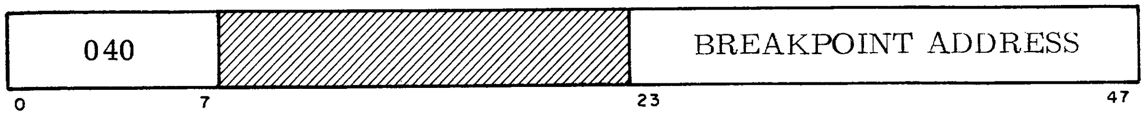
FLOW CHART:



MNEMONIC CODE: LISR

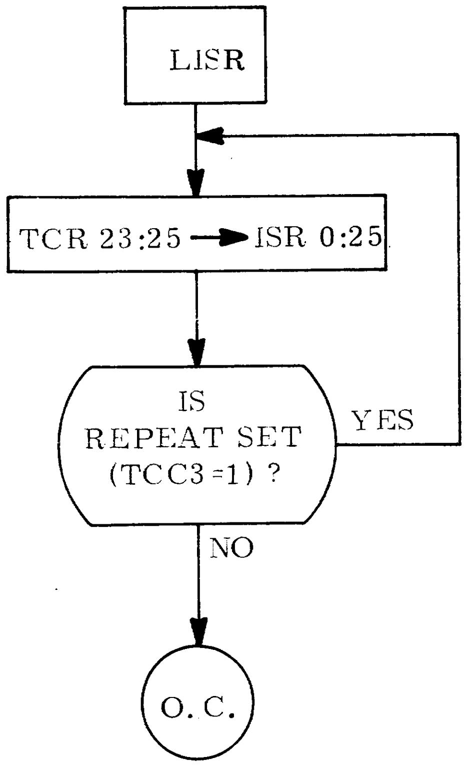
OPERATION: Load Breakpoint Register (ISR)

TCR:



DESCRIPTION: The low order 25 bits of the TCR are transferred into the breakpoint register (ISR). Should the contents of the instruction counter register (ICR) become equal to the contents of the ISR while the CU is running, the CU will come to an orderly halt as though an ADVAST halt instruction had been executed. The instruction at the breakpoint address will not be executed and TCI 5 will be set which will cause an interrupt to be sent to the B6700.

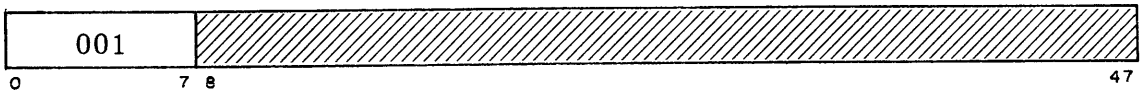
FLOW CHART:



MNEMONIC CODE: RPT

OPERATION: Repeat

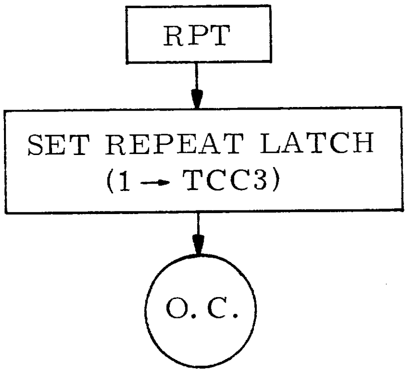
TCR:



DESCRIPTION: The TMU will repeat the next instruction indefinitely. The iteration will end whenever a quadrant disable is received or TCC3 is reset, or when the desired comparison is made on an SOC instruction.

After each iteration of a repeated instruction, a check is made to determine if the I/O has generated a request to the TMU. If so, the TMU will answer the request.

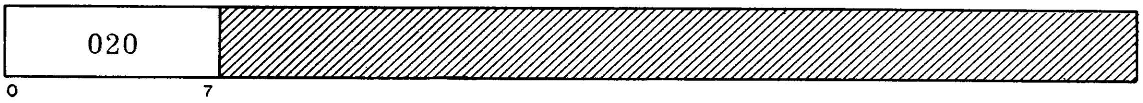
FLOW CHART:



MNEMONIC CODE: RUN

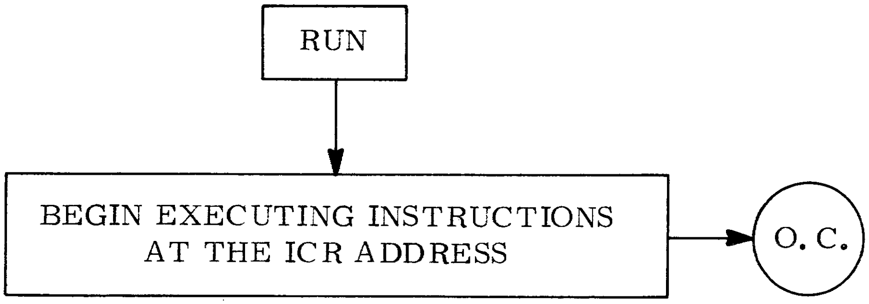
OPERATION: Run

TCR:



DESCRIPTION: The CU begins processing instructions starting at the location specified by the instruction counter (ICR). The CU will continue to execute instructions until it recognizes any of the conditions that will cause it to stop or halt (including breakpoint).

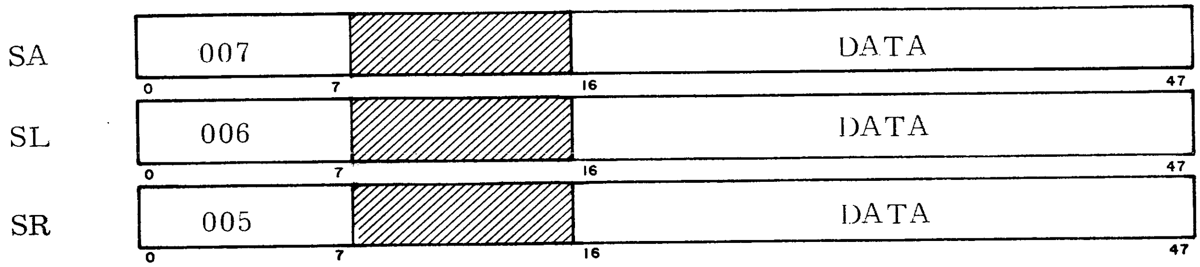
FLOW CHART:



MNEMONIC CODE: SA, SL, SR

OPERATION: Set All, Set Left, Set Right (Set TRI)

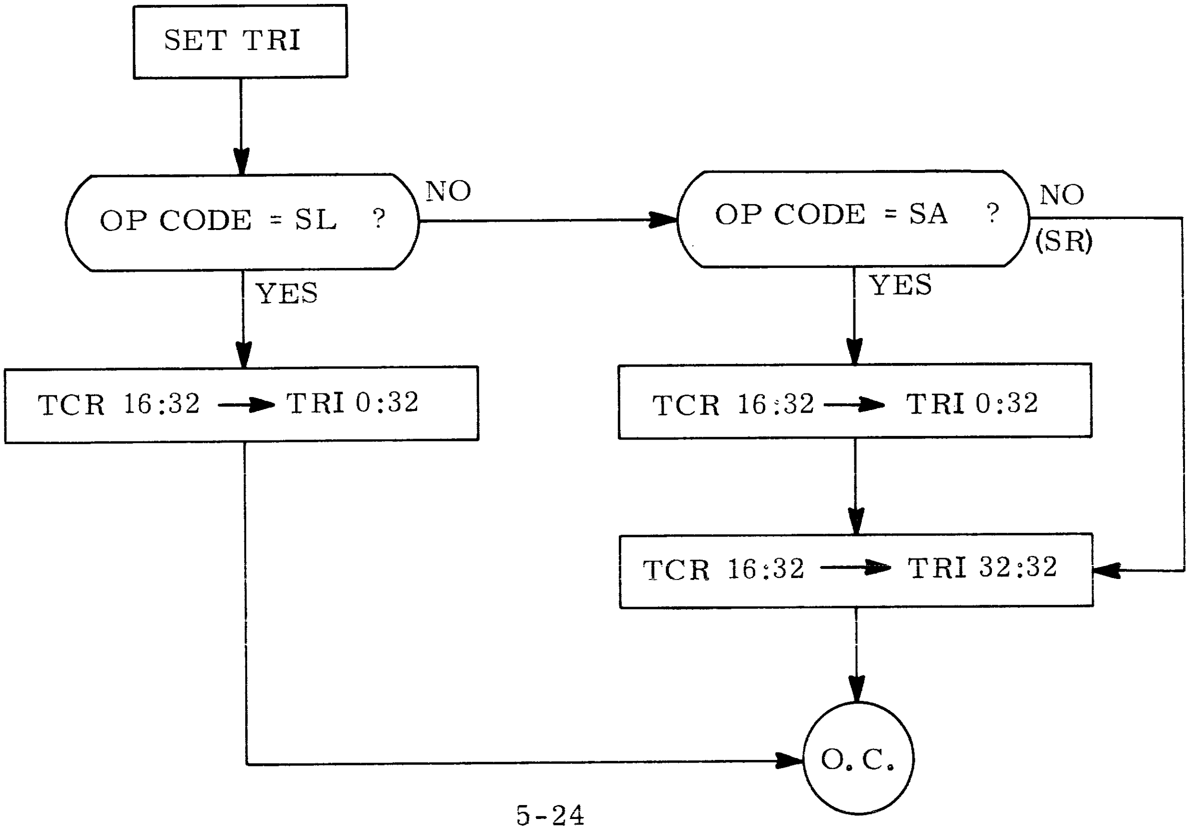
TCR:



DESCRIPTION: The contents of the data field are duplicated into both halves of the TMU input register (TRI), or replace the high-order or low-order 32 bits of this register, depending upon the operation code of SA, SL, or SR respectively.

NOTE: If the TRI is loaded and not read, this instruction will be treated as an illegal instruction.

FLOW CHART:



MNEMONIC CODE: SAT, SLT, SRT

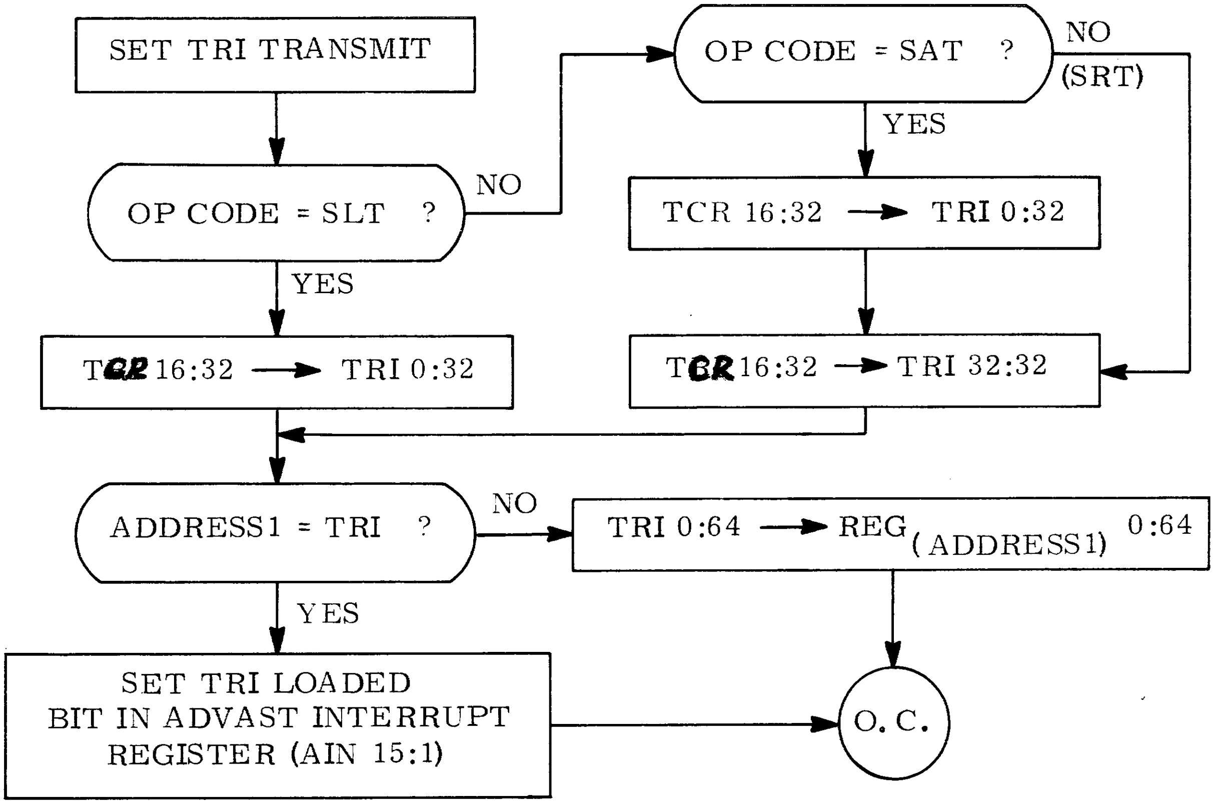
OPERATION: Set All Transmit, Set Left Transmit, Set Right Transmit (Set TRI Transmit)

TCR:

	0 – 7	8 – 15	16 – 47
SAT	047	ADDRESS 1	DATA
SLT	046	ADDRESS 1	DATA
SRT	045	ADDRESS 1	DATA

DESCRIPTION: The contents of the data field are duplicated into both halves of the TMU input register (TRI), or replace the high-order or low-order 32 bits of this register, depending upon the operation code of SAT, SLT, or SRT respectively. The contents of TRI are then transferred to the register addressed by the address 1 field. If this field specified TRI, then no data is moved and the "TRI loaded" bit in the ADVAST interrupt register (AIN 15:1) is set.

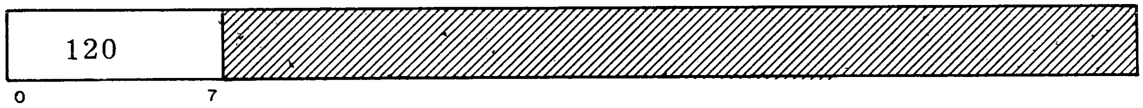
FLOW CHART:



MNEMONIC CODE: SIS

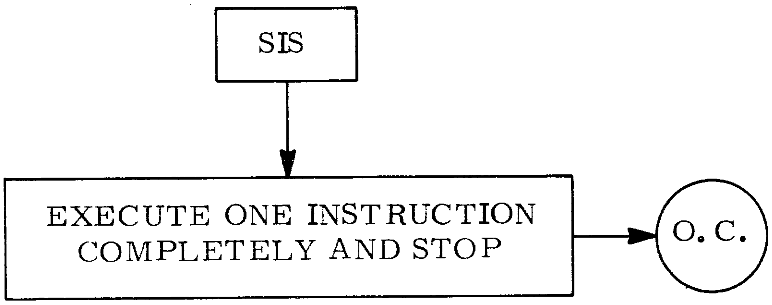
OPERATION: Single Step

TCR:



DESCRIPTION: The non-overlap mode is set. The CU executes the instruction addressed by the instruction counter (ICR) and stops when the instruction has been processed at all applicable CU subunits.

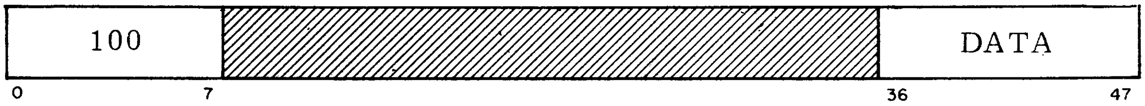
FLOW CHART:



MNEMONIC CODE: SIV

OPERATION: Set to Initialized Value

TCR:



DESCRIPTION: The registers or controls indicated in the data field are caused to be set to their machine idle state. In general, counting and indicator registers are reset (to zero) by this operation. Any combination of accessible registers and controls may be initialized by this instruction. A "one" in the corresponding bit positions of the data field will cause the following registers to be initialized:

TCR Bit	Register	Initialized Value
47	TCI	Zero (except indicators TCI 0:8)
46	TCC	Zero
* 45	MC0	Own quadrant bit set, all others reset
	MC1, MC2	High order bit set, all others reset
* 44	MSU Controls	Idle state
43	IAM Presence Bits	Vacant
42	ILA Controls	Idle state
41	FLP	Bit 0 set, all others reset
	FRP	Bit 0 set, all others reset
	FIR	Zero
	FINST Controls	Idle state

TCR Bit	Register	Initialized Value
40	FCC	Zero
	FTC	Zero
	FOC	Zero
39	AIT	Zero
	ADVAST CONTROLS	Initial State
	ADC	Zero
	ALR	Zero
	ACR (and other controls)	(See table below)
38	AIN, AMR	Zero
37	AC0, 1, 2, 3	Interrupt Index value → AC0 0 → AC1, 2, 3
36	TRO, TRI	Zero - TRI One - TRO

*Performed only if quadrant is disabled or if in "Local" operation.

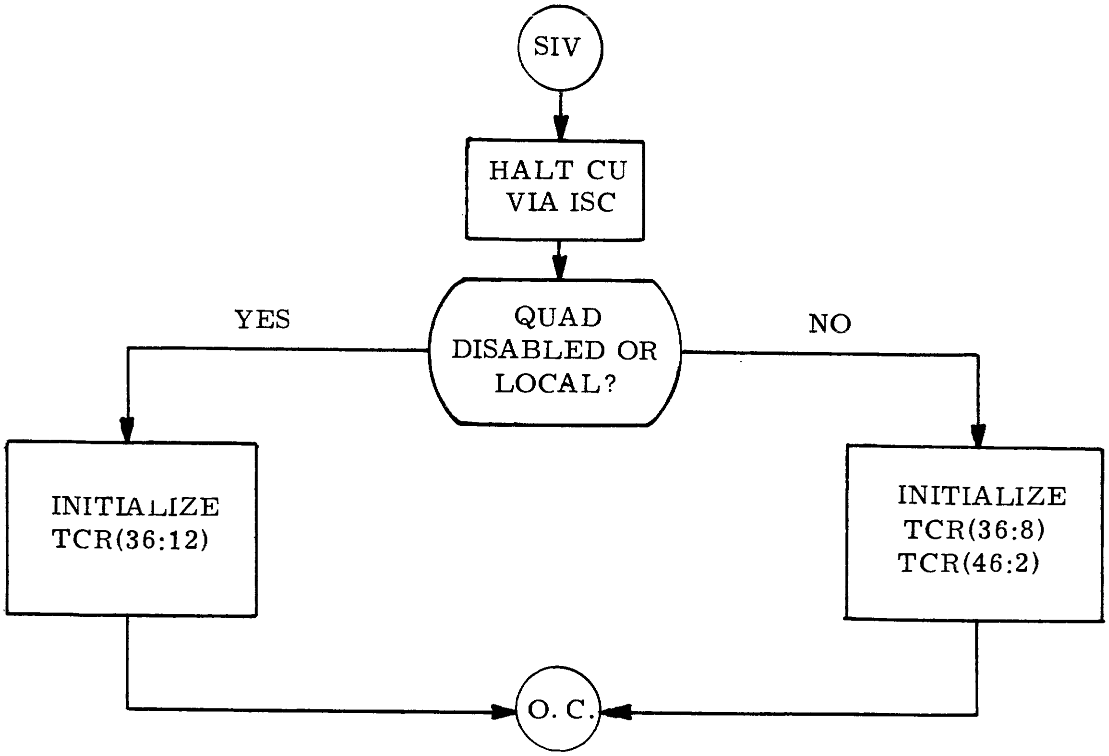
The SIV instruction is executed under program control via the TMU. The symbols in the table below have the following meaning:

NC - bit remains unchanged
S - set bit
R - reset bit

ACR Bit Number	SIV
0	NC
1	R
2	S
3	R
4	NC
5	R
6	NC
7	NC

ACR Bit Number	SIV	
8	R	
9	NC	
10	NC	
11	NC	
12	R	
13	S	(NC for the interrupt SIV instruction)
14	R	
15	R	

FLOW CHART:



MNEMONIC CODE: SOC

OPERATION: Scan Out Compare

TCR:

011	ADDRESS 1	CS	▨
0 7	8 15	16	

DESCRIPTION: Nothing happens until the ILA has relinquished control to the TMU. Then, if the TRO is already loaded for transmission, the instruction is bypassed. If the TRO is not already loaded, the contents of the register specified by the ADDRESS1 field are sent to the comparator (VALUE1). At the same time if the CS bit equals "one", the contents of TRI are sent to the comparator (VALUE2); if the CS bit equals "zero", the Test Maintenance Panel data switch configuration is sent to the comparator (VALUE2). If VALUE1 equals VALUE2, the TCL EQUAL BIT is set in the test condition indicator register (TCI2). Further, if VALUE1 equals VALUE2, and the "interrupt DC on equal bit"(TCC5) is set, or, if VALUE1 does not equal VALUE2, and the "interrupt DC on unequal bit"(TCC4) is set, then the following steps are taken:

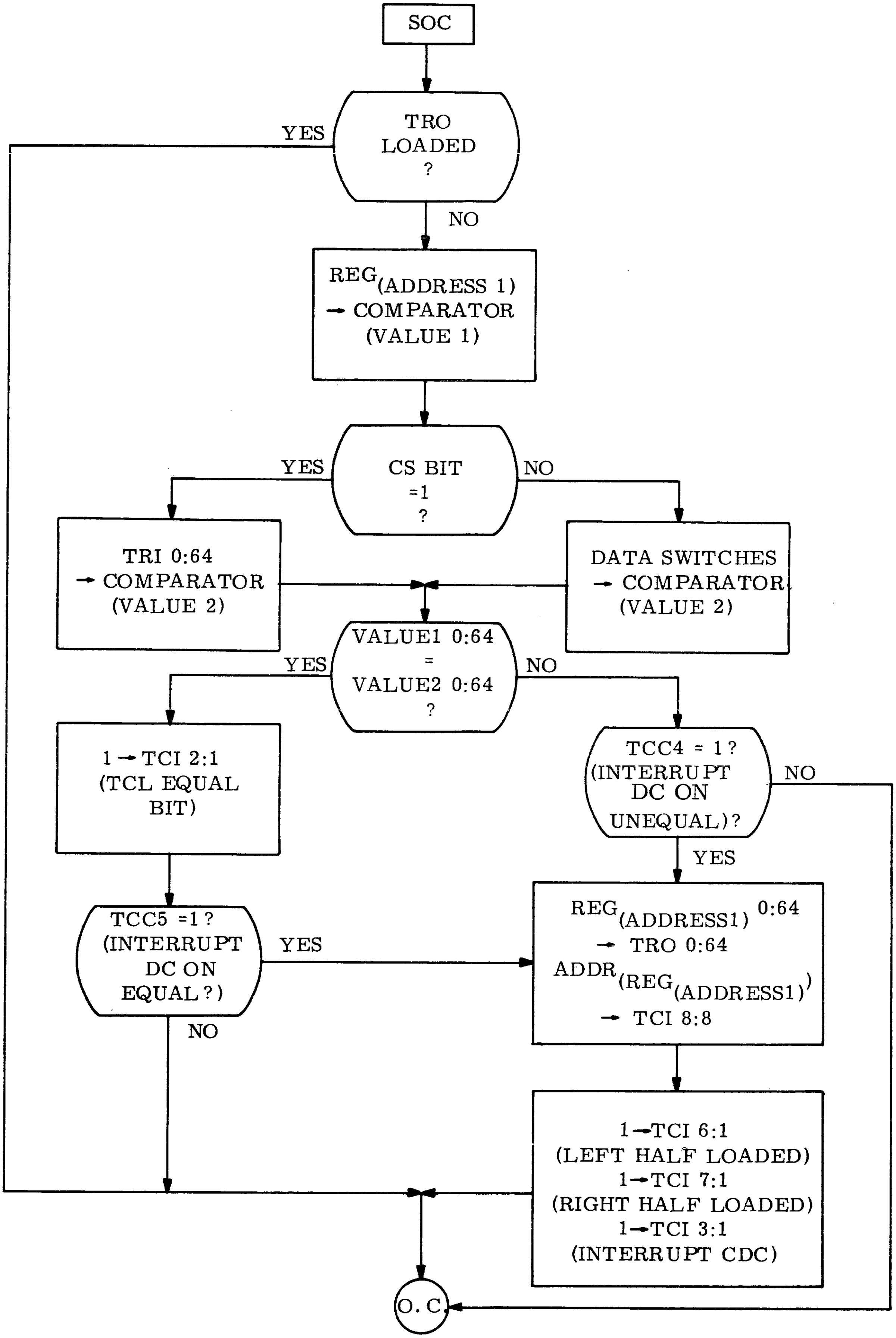
(a) The TRO is loaded with the contents of the register specified by ADDRESS1;

(b) Bits 8 through 15 of the TCI register are loaded with the address of the register specified by ADDRESS1;

(c) The "left half loaded"(TCI6) and "right half loaded"(TCI7) bits are set in TCI; and

(d) TCI3 is set, causing the DC to recognize an interrupt from the TMU.

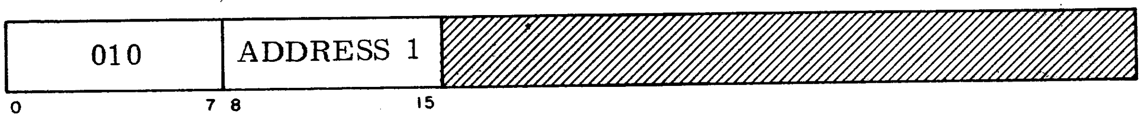
FLOW CHART: See next page.


SOC
TRO LOADED ?
YES
NO
REG(ADDRESS 1) → COMPARATOR (VALUE 1)
CS BIT =1 ?
YES
NO
TRI 0:64 → COMPARATOR (VALUE 2)
DATA SWITCHES → COMPARATOR (VALUE 2)
VALUE1 0:64 = VALUE2 0:64 ?
YES
NO
1 → TCI 2:1 (TCL EQUAL BIT)
TCC4 = 1 ? (INTERRUPT DC ON UNEQUAL)?
NO
YES
TCC5 =1 ? (INTERRUPT DC ON EQUAL ?)
YES
NO
REG(ADDRESS1) 0:64 → TRO 0:64
ADDR(REG(ADDRESS1)) → TCI 8:8
1→TCI 6:1 (LEFT HALF LOADED)
1→TCI 7:1 (RIGHT HALF LOADED)
1→TCI 3:1 (INTERRUPT CDC)
O. C

MNEMONIC CODE: SOD

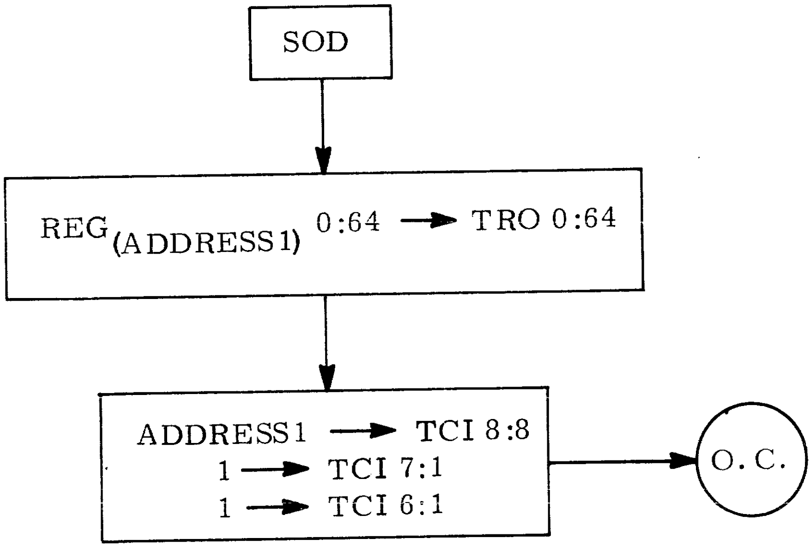
OPERATION: Scan Out Data

TCR:



DESCRIPTION: Nothing happens until the ILA has relinquished control to the TMU. Then if the test output register (TRO) is already loaded for transmission the instruction is bypassed. If the TRO is not loaded, its contents are replaced by the contents of the register specified by the address 1 field. Bits 8 through 15 of the test condition indicator (TCI) are loaded with the address of the register specified in address 1 and the "left half loaded" and "right half loaded" bits are set. All other TCI bits are left unaltered.

FLOW CHART:



MNEMONIC CODE: TIC

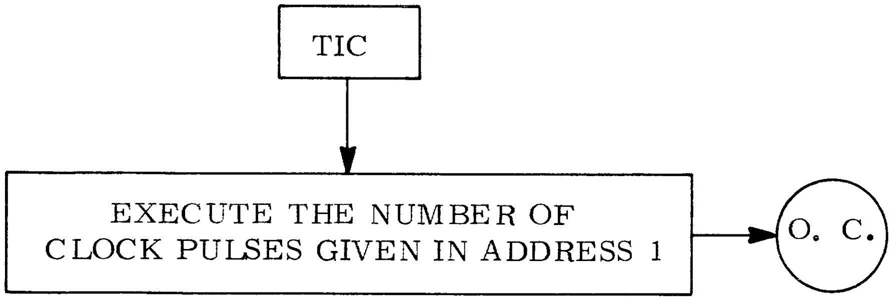
OPERATION: Trigger I Clocks

TCR:

121	ADDRESS 1	▨
0 7	8 15	

DESCRIPTION: All enabled stations are advanced through the number of clock pulses given in the address 1 field.

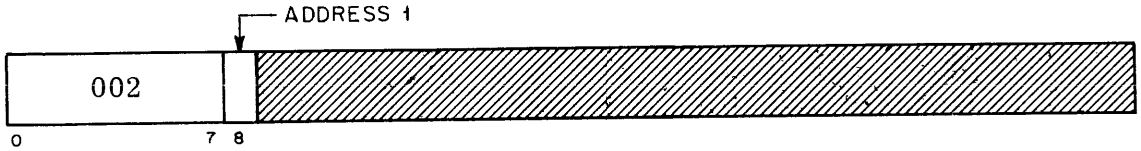
FLOW CHART:



MNEMONIC CODE: TOC

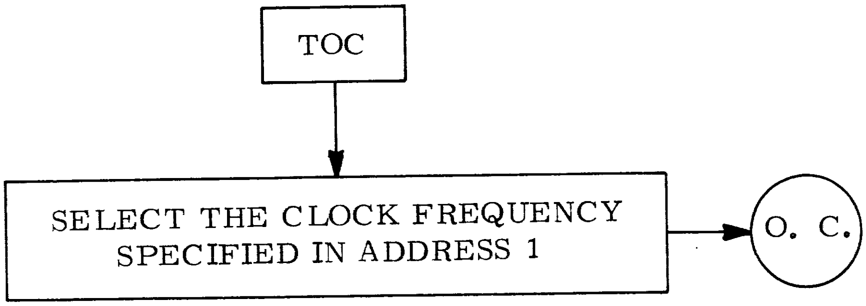
OPERATION: Timing Oscillator Connect

TCR:



DESCRIPTION: This instruction selects one of two frequencies that control the speed at which the ILLIAC IV quadrant will operate. The frequency is specified in the high order bit of address 1. If this bit is "one", the quadrant will operate at one-sixteenth the normal frequency. All instructions or operations within the CU subsequent to the issuance of this instruction will be performed at the selected frequency.

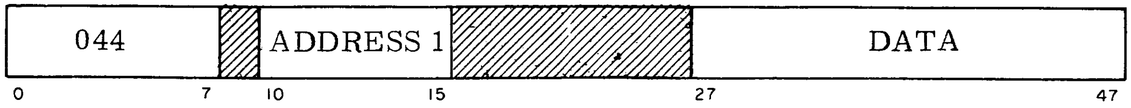
FLOW CHART:



MNEMONIC CODE: WIS

OPERATION: Write Instruction Storage

TCR:



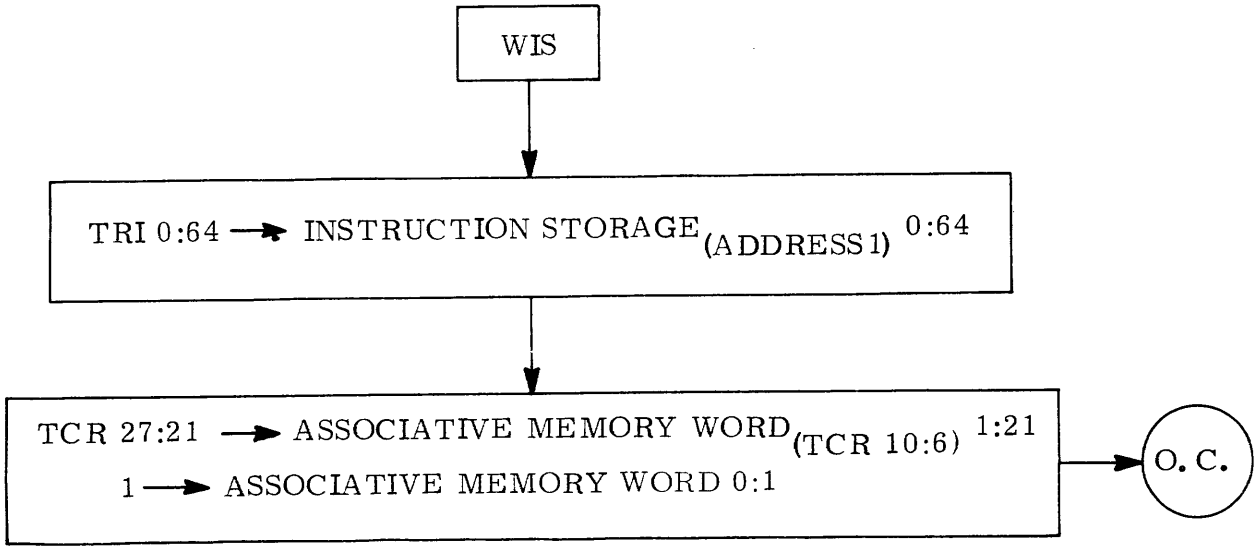
DESCRIPTION: The contents of the TRI are transferred to the instruction storage word specified in the address 1 field. The 21-bit data field is transferred to the associative memory in the instruction lookahead unit (ILA) into the word specified by the address 1 field (three high-order bits) and the "present" bit set in the word. The address 1 field (bits 10-15) are defined as follows:

Bits 10-12 = Block

Bits 13-15 = Word in block

Bits 10-12 define one of eight blocks in IWS, and bits 13-15 define one of eight words within the block. The 21-bit data field corresponds to the array address (less the three PUC bits) from which the ILA unit will assume that the instruction was fetched.

FLOW CHART:


WIS
TRI 0:64 → INSTRUCTION STORAGE(ADDRESS 1) 0:64
TCR 27:21 → ASSOCIATIVE MEMORY WORD(TCR 10:6) 1:21
1 → ASSOCIATIVE MEMORY WORD 0:1
O. C.

SECTION VI

INSTRUCTION TIMING

In addition to the obvious parallelism of the array computer, there is parallelism (overlap) in ILLIAC IV between consecutive instructions in the instruction stream. Four significant mechanisms for achieving overlap are:

a. Parallelism between ADVAST and FINST, which are operating on different instructions in nearby portions of the instruction stream;

b. Parallelism between automatic hardware actions which are called upon to complete certain instructions, and the execution of subsequent instructions; and

c. Parallelism between noninterfering portions of successive FINST/PE instructions.

d. Parallelism between MSU operations and non-interfering FINST/PE instructions which use common data paths.

The first and primary source of overlap is the execution of instructions at two relatively independent stations, ADVAST and FINST. An instruction queue, FINQ, exists between these two stations to smooth the flow of instructions through both stations.

The second source of overlap is the completion, in ADVAST, of certain instructions after the start of subsequent instructions. An example is the ADVAST instruction, BIN. After ADVAST has initiated the memory access of BIN, the instruction is sent on to FINST (and the MSU) to be completed while ADVAST processes subsequent instructions. Only if ADVAST needs to use the data which have not yet been fetched by BIN, will ADVAST become idle while waiting for the data. Thus, the access time from ADVAST to memory, which

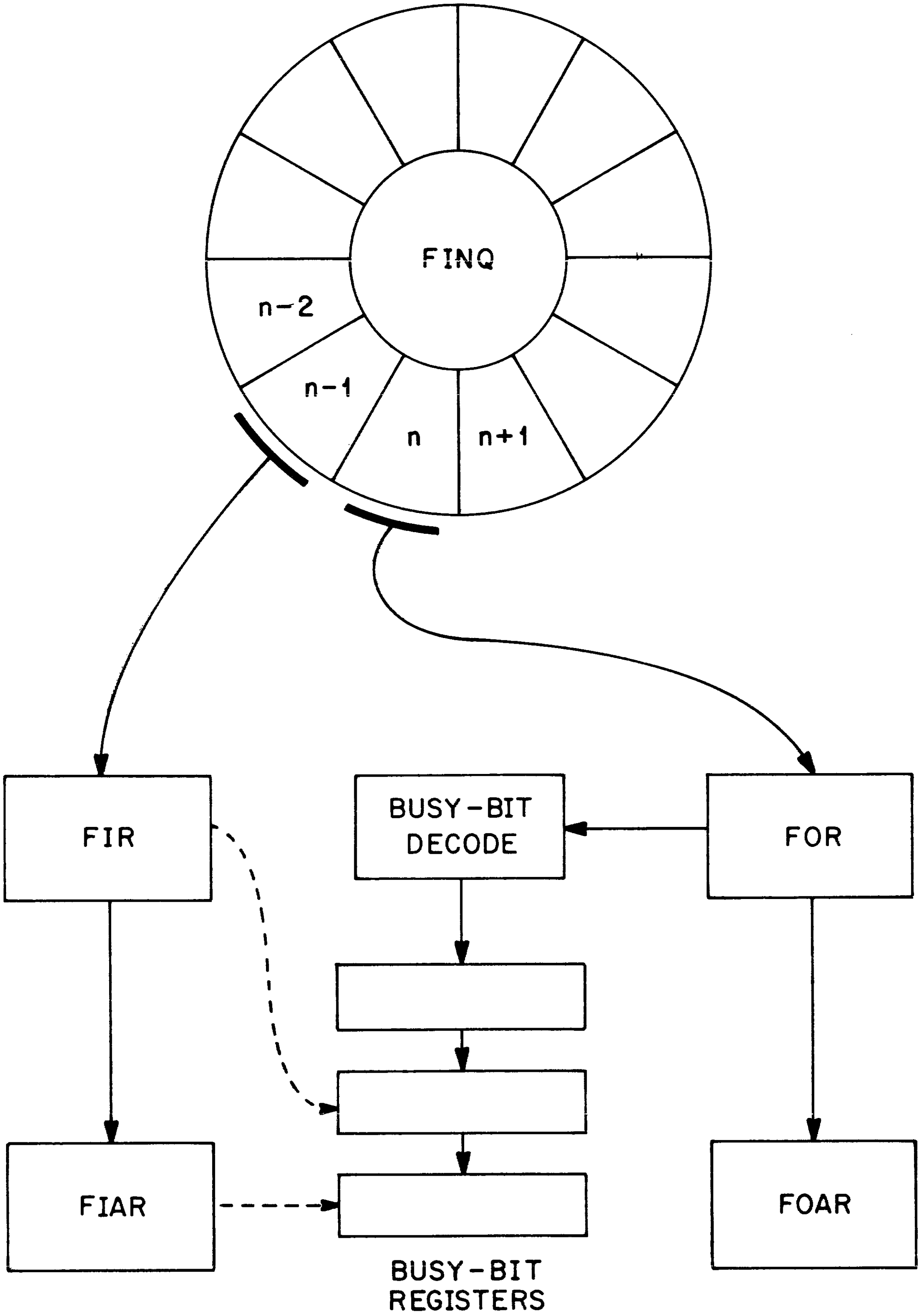


Figure 6-1. FINST Overlap Structure

is long compared to the ADVAST clock cycle time, is overlapped with other ADVAST operations. Another example is arithmetic overflow in the PE. When an arithmetic overflow occurs, it takes time for the fault bit to reach the CU and cause an interrupt. However, the PE does not wait for the interrupt to occur before starting to execute the next instruction. Therefore, the FINST/PE ADD instruction is not quite finished while the next instruction in the PE is being executed.

The third source of overlap is provided by two execution stations within FINST. These stations are called the early, or overlap ("O") station, and the late, or instruction execution ("I") station. Each station contains a register, FOR or FIR respectively for the two stations, which receives the next instruction from the instruction queue, FINQ, and examines the instruction to determine when it is time to start the instruction execution. The actual execution of the instruction is carried out from a second register, FOAR or FIAR respectively. The "A" in their designation indicates that these registers serve as the address register for the microprogram storage at FINST. The two stations receive their inputs from neighboring slots of the instruction queue, FINQ. The stations also share a group of "busy bit registers" which record the parts of the PE that are needed by unexecuted instructions in the instruction station. This mechanism is shown in Figure 6-1.

The fourth source of overlap is the interlacing of operations over common data paths by the MSU and FINST. The common data paths consist of the signal buses between the CU and the MLU, and between registers of the MLU. For instance, the MSU could be executing a number of memory operations while FINST was executing instructions which do not require memory accesses.

In any calculations of execution time of an instruction stream, the individual mechanisms for achieving overlap, which appear reasonable and straightforward by themselves, cause great difficulty when taken collectively. Therefore, a listing of instruction execution times must of necessity be qualified with statements of conditions to the extent that the time savings accruing from the compound overlap of ILLIAC IV will be incorporated. An algorithm for the approximate timing of ILLIAC programs is to divide the program into sections, each section having consistency in the ratio of ADVAST to FINST/PE instructions. Then, determine for each section whether the ADVAST or FINST/PE time is longer, and sum the longer times of each section.

Some exceptions to this simplified algorithm should be noted. Some instructions require a variable amount of time depending upon some variant such as the address field. For example, STL(MC0) will require more time for execution than STL(ADB) because the effect of MC0 on FINST/PE instructions must be waited for before the actual change is made. Similarly, shifts of zero length require less time than shifts of any other length. Also, additional time is taken by the fetching of instructions. Each block of instructions fetched (see Section I, Fetching the Program) requires a time equal to the time of a BIN instruction, assuming these fetches are not delayed by higher priority memory requests (see Section III, ILLIAC IV Addressing). With regard to FINST/PE instructions, the total run time at FINST is somewhat shorter than the sum of the FINST/PE instruction times. Operations which occur at the beginning of instructions, and which are overlapped with the preceding FINST/PE instruction, are generally memory fetches, other uses of the common data bus, and register-to-register transfers. In routing the transfer from source register to RGR is executed from the overlap station, and the transfer between PE's is executed from the normal execution station.

Instructions with their corresponding execution times expressed in clock times are given in the following tables: ADVAST Instruction Timing, Table 6-1, and FINST/PE Instruction Timing, Table 6-2*.

*TMU instruction timing has been omitted since the use of TMU instructions in manual mode precludes the need.

In Table 6-1, instruction times are given in terms of the number of clock cycles required to execute the instruction at ADVAST in the CU, and where applicable, at FINST in the CU, and at the PE. The execution time of a sequence of ADVAST instructions, at ADVAST, is the sum of the times listed in the CU-ADVAST column. The times given in the CU-ADVAST column assume single quadrant operation and therefore do not include the time required for synchronization at the various CU stations in a multiquadrant configuration. In addition, the times given in the CU-ADVAST column do not include waits caused by FINQ being full or memory access time, i.e., LOAD or BIN operations, if required. Eight ADVAST instructions (BIN, BINX, LOAD, LOADX, LDC, SETC, STORE, STOREX) require execution time at FINST in the CU and at the PE in addition to the execution time given in the CU-ADVAST column. These times are given in the CU-FINST and PE columns and are the same for both 32-bit and 64-bit modes of operation. The BIN or BINX operation requires 36 clock times to complete, however, the instruction requires only 2 clock times at ADVAST before the next instruction, if allowed, can start executing. After FINST, PE, and memory access times have been expended on BIN or BINX, an additional 16 clock times at ADVAST are required while the information is loaded into ADB. The execution time added by a BIN or BINX in the program can therefore be either 2 clock times (the delay until the start of the next instruction), or 18 clock times (the total ADVAST time), or 36 clock times (the elapsed time from beginning to end), or some intermediate number depending upon the details of the instruction stream at ADVAST, as noted previously. In the case of LOAD and LOADX, a similar situation applies, except that the ADVAST time required by the returning data is only 2 clock times. In the case of STORE and STOREX, ADVAST is finished with the instruction as soon as it passes it on to FINST via FINQ. In the case of LDC and SETC, the FINST and PE operations are required to be simultaneous with the ADVAST operations, so that the instruction cannot start until FINST is finished with the previous instruction(s). Therefore the minimum (assuming that execution time is ADVAST limited) and maximum (depending on the amount of overlap that can be achieved with other ADVAST instructions) execution times for these eight instructions are:

	Minimum	Maximum
BIN	19	36
BINX	19	36
LDC	17	17
LOAD	4	20
LOADX	4	20
SETC	17	17
STORE	4	4
STOREX	4	4

The execution time of the instructions at the PE is not longer than the sum of the times listed in the CU-FINST and PE columns, and can be less because of the capability of the PE sequencer to sometimes overlap noninterfering portions of successive instructions, resulting in the FINST time being masked.

In Table 6-2, instruction times are given in terms of the number of clock times required to execute the instruction at the PE. All FINST/PE instructions require one clock time for execution in ADVAST or two clock times if ACAR indexing is required. Also, any time FINST becomes idle, two clock times must be added to the next FINST/PE instruction. Except for waits caused by an idle FINST, or memory access if required, the execution time of a sequence of FINST/PE instructions is no longer than the times listed in the appropriate PE mode columns. The times given are those appropriate for the operand being found in the RGA and in the RGB or RGR as is appropriate to the instruction. When a second operand is required and is not found in the RGB or RGR, a memory access time of seven clock times or a literal or register transfer of one clock time must be added. Examples of minimum (overlap) and maximum (no overlap) execution times, assuming no memory access or register transfer is required, 64-bit mode of operation, and subject to other conditions given in Table 6-2, for several FINST/PE instructions are:

	Minimum	Maximum
ADD	1	2
SUB	1	2
ML	8	9
DVM	52	53

Table 6-1. ADVAST Instruction Timing

Mnemonic Code	Operation	Clock Times			Sync Req'd	Notes
		CU		PE*		
		ADVAST	FINST			
ALIT	Add literal to address field of ACAR	2				
BIN	Block fetch from PE memory to ADB	18	2	1	Yes	a, b, c
BINX	Block fetch (RGX-indexed) from PE memory to ADB	18	2	1	Yes	a, b, c
CACRB	Set/Reset nth bit in ADVAST control register	2				d
CADD	Add local memory to ACAR	3				
CAND	Logical AND of local memory and ACAR	3				
CCB	Complement nth bit of ACAR	6				e
CEXOR	Logical exclusive OR of local memory and ACAR	3				
CLC	Clear ACAR	2				
COMPC	Complement ACAR	2				
COPY	Copy ACAR	4			Yes	
COR	Logical OR of local memory and ACAR	3				
CRB	Reset nth bit in ACAR	6				e
CROTL	Rotate ACAR left (end around)	3				e
CROTR	Rotate ACAR right (end around)	3				e
CSB	Set nth bit in ACAR	6				e
CSHL	Shift ACAR left (end off)	3				e
CSHR	Shift ACAR right (end off)	3				e
CSUB	Subtract local memory from ACAR	3				
CTSBF	Skip if nth bit in ACAR is not "one"	4			Yes	g, h
CTSBT	Skip if nth bit in ACAR is "one"	4			Yes	g, h
DUPI	Duplicate inner-half of ADB memory word	3				
DUPO	Duplicate outer-half of ADB memory word	3				
EXCHL	Exchange local operand and ACAR	3				i, o
EXEC	Execute	4				j
FINQ	Stop ADVAST until FINST is idle	2				k
HALT	CU comes to orderly idle state	2				k, l
INCRXC	Modify index field of ACAR by increment field of same ACAR	3				
INR	Return to normal processing after interrupt	20				b, k, l, m, n
JUMP	Jump to address in ADR field	2				g
LDC	Transfer specified PE register to ACAR	17	2	1		k, m, f
LDL	Load from local address	3				
LEADO	Find leading "one" in ACAR	5			Yes	
LEADZ	Find leading "zero" in ACAR	5			Yes	
LIT	Store next 64 bits in ACAR	4				l
LOAD	Word fetch from PE memory to CU local memory	4	2	1	Yes	a, b, c, i, o, t
LOADX	Word fetch (RGX-indexed) from PE memory to CU local memory	4	2	1	Yes	a, b, c, i, o, t

*PE clock times are the same for 32-bit or 64-bit mode of operation.

Table 6-1. ADVAST Instruction Timing (Cont.)

Mnemonic Code	Operation	Clock Times			Sync Req'd	Notes
		CU		PE*		
		ADVAST	FINST			
ORAC	Inclusive-OR of operand in ACAR of all CUs executing the instruction	2			Yes	
SETC	Specified mode bit from PEs to ACAR	17	14	14		k, f
SKIP	Skip forward/backward	4				g, h
SLIT	Replace address field of ACAR	2				
STL	Store ACAR in local address	3				o
STORE	Store from local address into specified PE location	4	3	3		c, p
STOREX	Store from local address into specified PE location (RGX-indexed)	4	3	3		c, p
TCCW	Transmit ACAR counterclockwise (to next lower numbered CU)	2			Yes	
TCW	Transmit ACAR clockwise (to next higher numbered CU)	2			Yes	
Test-Skip	Test and skip conditionally:**	▨	▨	▨	▨	▨
EQLX- -TA,-T,-FA,-F	Skip if ACAR 40:24 equal operand 40:24	6			Yes	q
		9			Yes	g, r
GRTR- -TA,-T,-FA,-F	Skip if ACAR 40:24 are greater than operand 40:24	6			Yes	q
		9			Yes	g, r
LESS- -TA,-T,-FA,-F	Skip if ACAR 40:24 are less than operand 40:24	6			Yes	q
		9			Yes	g, r
ONES- -TA,-T,-FA,-F	Skip if ACAR 0:64 are all "ones"	4			Yes	q
		7			Yes	g, r
ONEX- -TA,-T,-FA,-F	Skip if ACAR 40:24 are all "ones"	4			Yes	q
		7			Yes	g, r
SKIP- -TA,-T,-FA,-F	Skip dependent upon CU true/false flip-flop	2			Yes	q
		5			Yes	g, r
TXE- -TA,-T,-FA,-F	Skip if ACAR 40:24 equal bits 16:24 in local memory	5			Yes	q
		8			Yes	g, r
TXE- -TAM,-TM,-FAM,-FM	Skip if ACAR 40:24 equal bits 16:24 (also, 40:24 are modified by 1:15) of same ACAR	6			Yes	q
		8			Yes	g, r
TXG- -TA,-T,-FA,-F	Skip if ACAR 40:24 are greater than bits 16:24 in local memory	5			Yes	q
		8			Yes	g, r
TXG- -TAM,-TM,-FAM,-FM	Skip if ACAR 40:24 are greater than bits 16:24 (also, 40:24 are modified by 1:15) of same ACAR	6			Yes	q
		8			Yes	g, r
TXL- -TA,-T,-FA,-F	Skip if ACAR 40:24 are less than bits 16:24 in the local memory	5			Yes	q
		8			Yes	g, r
TXL- -TAM,-TM,-FAM,-FM	Skip if ACAR 40:24 are less than bits 16:24 (also, 40:24 are modified by 1:15) of same ACAR	6			Yes	q
		8			Yes	g, r
ZER- -TA,-T,-FA,-F	Skip if ACAR 0:64 are all "zeros"	4			Yes	q
		7			Yes	g, r
ZERX- -TA,-T, -FA,-F	Skip if ACAR 40:24 are all "zeros"	4			Yes	q
		7			Yes	g, r
WAIT	Synchronize all CUs in array or join all CUs specified by ADR 4:4	2			Yes	
		5			Yes	s
TIO	Send descriptor to I/O	4			Yes	u

*PE clock times are the same for 32-bit or 64-bit mode of operation.

**Variants of the Test-Skip instruction sample the condition of the true-false (TF) flip-flops, as follows:

Variant	Meaning	Variant	Meaning
F	Any false	T	Any true
FA	All false	TA	All true
FM	Any false, magnitude of modifier only	TM	Any true, magnitude of modifier only
FAM	All false, magnitude of modifier only	TAM	All true, magnitude of modifier only

Table 6-1. ADVAST Instruction Timing (Cont.)

NOTES

a. The words of ADB which are to be loaded by this instruction cannot be referenced for the next 36 clock times (22 for the case of LOAD). Any ADVAST instruction which references them before this time will cause ADVAST to hang up until the fetched copy of the word has arrived in ADB.

b. CU-FINST time includes four clock times for sync.

c. CU-FINST clock times given for this instruction may be masked by overlap operation.

d. If CACRB13 is resetting ACR13, nine clock times are required.

e. Clock times given are for the normal case. For the NOOP case, two clock times are required.

f. The clock times required for FINST and PE execution are concurrent with ADVAST time.

g. Clock times given are for the case of no jump. Four additional clock times are required if there is a jump and the block is present. If the block is not present, an instruction fetch time of approximately 36 clock times is required.

h. Plus one clock time if ILA adder is busy.

i. For MC's, one additional clock time is required.

j. Plus time for execution of instruction after it is loaded into AIR.

k. Plus time required for FINST to become idle.

l. Six clock times (minimum) required for ILA to complete Look-Ahead, if necessary.

m. ADVAST time given assumes no wait for memory access.

n. Clock times given are for the case of ACR01 set. If ACR01 is not set, two clock times are required.

o. If MC0, MC2, note "k" applies. If ICR, note "g" applies. If MC0, MC1, note "l" applies.

p. Clock times given include time for a memory access.

q. Test failed.

r. Test successful.

s. The special option WAIT requires additional time for JOIN in multi-quadrant mode of operation.

Table 6-2. FINST/PE Instruction Timing

Program Mnemonic Code	Operation	PE Clock Times		Notes
		32-Bit Mode	64-Bit Mode	
AD	Add (ADR) to RGA. Variants are:	6	4	
ADA	Suffix — Meaning	6	4	
ADM	A — Unsigned	5	3	
ADMA	M — Fixed point	5	3	
ADN	N — Normalized floating	6	5	
ADNA	R — Rounded	6	5	
ADR		10	6	
ADRA		10	6	
ADRN		10	7	
ADRNA		10	7	
ADB	Add (ADR) to RGA in 8-bit bytes	1	1	
ADD	Add 64-bit unsigned fixed-point numbers (ADR) to RGA	1	1	
ADEX	Add (ADR) exponent field(s) to RGA exponents	1	1	
ASB	Place the sign(s) of RGA into the sign(s) of RGB	1	1	
Boolean Operations:	Place the result of the specified logical function of RGA with (ADR) into RGA:	▨	▨	▨
AND	Logical AND of RGA with (ADR)	1	1	
ANDN	Logical AND of RGA with complement of (ADR)	1	1	
EOR	Logical EXCLUSIVE-OR of RGA with (ADR)	1	1	
EQV	Logical EQUIVALENCE of RGA with (ADR)	1	1	
NAND	Logical AND of complement of RGA with (ADR)	1	1	
NANDN	Logical AND of complement of RGA with complement of (ADR)	1	1	
NOR	Logical OR of complement of RGA with (ADR)	2	2	
NORN	Logical OR of complement of RGA with complement of (ADR)	2	2	
OR	Logical OR of RGA with (ADR)	2	2	
ORN	Logical OR of RGA with complement of (ADR)	2	2	
Change RGA Bit.	Perform the indicated operation on the specified RGA bit:	▨	▨	▨
CAB	Complement bit(s) in RGA	3	2	a
CHSA	Change sign bit(s) in RGA	3	2	a
RAB	Reset bit(s) in RGA	3	2	a
SAB	Set bit(s) in RGA	3	2	a
SAN	Set sign bit(s) in RGA	3	2	a
SAP	Reset sign bit(s) in RGA	3	2	a
CLRA	Clear RGA	1	1	
COMPA	Complement RGA	1	1	

Table 6-2. FINST/PE Instruction Timing (Cont.)

Program Mnemonic Code	Operation	PE Clock Times		Notes
		32-Bit Mode	64-Bit Mode	
DV	Divide RGA and RGB, double-length mantissa, by (ADR). Variants are: Suffix — Meaning A — Unsigned M — Fixed point N — Normalized R — Rounded	65	53	
DVA		65	53	
DVM		63	52	
DVMA		63	52	
DVN		68	55	
DVNA		68	55	
DVR		66	54	
DVRA		66	54	
DVRM		64	53	
DVRMA		64	53	
DVRN		69	56	
DVRNA		69	56	
EAD	Recover extended precision after floating-point add	13	13	
ESB	Recover extended precision after floating-point subtract	13	13	
GB	Test for RGA greater than (ADR) in 8-bit bytes.	2	2	
(I|J) A (G|L)	RGA arithmetic test to mode bit (for 32-bit mode, result also goes to G or to H):	▨	▨	▨
IAG	Place result of test for RGA arithmetically greater than (ADR) into I (and G)	6	3	
IAL	Place result of test for RGA arithmetically less than (ADR) into I (and G)	6	3	
JAG	Place result of test for RGA arithmetically greater than (ADR) into J (and H)	6	3	
JAL	Place result of test for RGA arithmetically less than (ADR) into J (and H)	6	3	
(I|J) B	Move RGA bit to mode bit:	▨	▨	▨
IB	Transfer RGA bit(s) to I (and G)	5	3	a
ISN	Transfer RGA sign(s) to I (and G)	5	3	a
JB	Transfer RGA bit(s) to J (and H)	5	3	a
JSN	Transfer RGA sign(s) to J (and H)	5	3	a
(I|J)(L|M)(E|G|L)	RGA logical test to mode bit (for 32-bit mode, results go to I and G or to J and H):	▨	▨	▨
ILE	Place result of test for RGA logically equal to (ADR) into I	2	1	
ILG	Place result of test for RGA logically greater than (ADR) into I	2	1	
ILL	Place result of test for RGA logically less than (ADR) into I	2	1	
IME	Place result of test for RGA mantissa logically equal to (ADR) mantissa into I	2	1	

Table 6-2. FINST/PE Instruction Timing (Cont.)

Program Mnemonic Code	Operation	PE Clock Times		Notes
		32-Bit Mode	64-Bit Mode	
IMG	Place result of test for RGA mantissa logically greater than (ADR) mantissa into I	2	1	
IML	Place result of test for RGA mantissa logically less than (ADR) mantissa into I	2	1	
JLE	Place result of test for RGA logically equal to (ADR) into J	2	1	
JLG	Place result of test for RGA logically greater than (ADR) into J	2	1	
JLL	Place result of test for RGA logically less than (ADR) into J	2	1	
JME	Place result of test for RGA mantissa logically equal to (ADR) mantissa into J	2	1	
JMG	Place result of test for RGA mantissa logically greater than (ADR) mantissa into J	2	1	
JML	Place result of test for RGA mantissa logically less than (ADR) mantissa into J	2	1	
(I|J)(L|M)(O|Z)	RGA zeros or ones test to mode bit (for 32-bit mode, results also go into G or H):	▨	▨	▨
ILO	Place result of test for RGA logically equal to all "ones" into I	2	1	
ILZ	Place result of test for RGA logically equal to zero into I	2	1	
IMO	Place result of test for RGA mantissa logically equal to all "ones" into I	2	1	
IMZ	Place result of test for RGA mantissa logically equal to zero into I	2	1	
JLO	Place result of test for RGA logically equal to all "ones" into J	2	1	
JLZ	Place result of test for RGA logically equal to zero into J	2	1	
JMO	Place result of test for RGA mantissa logically equal to all "ones" into J	2	1	
JMZ	Place result of test for RGA mantissa logically equal to zero into J	2	1	
(I|J)(S|X)(E|G|L)	Index test to mode bit:	▨	▨	▨
ISE	Place result of test for (RGS) arithmetically equal to (ADR) into I	1	1	
ISG	Place result of test for (RGS) arithmetically greater than (ADR) into I	1	1	
ISL	Place result of test for (RGS) arithmetically less than (ADR) into I	1	1	
IXE	Place result of test for (RGX) arithmetically equal to (ADR) into I	1	1	
IXG	Place result of test for (RGX) arithmetically greater than (ADR) into I	1	1	
IXL	Place result of test for (RGX) arithmetically less than (ADR) into I	1	1	

Table 6-2. FINST/PE Instruction Timing (Cont.)

Program Mnemonic Code	Operation	PE Clock Times		Notes
		32-Bit Mode	64-Bit Mode	
JSE	Place result of test for (RGS) arithmetically equal to (ADR) into J	1	1	
JSG	Place result of test for (RGS) arithmetically greater than (ADR) into J	1	1	
JSL	Place result of test for (RGS) arithmetically less than (ADR) into J	1	1	
JXE	Place result of test for (RGX) arithmetically equal to (ADR) into J	1	1	
JXG	Place result of test for (RGX) arithmetically greater than (ADR) into J	1	1	
JXL	Place result of test for (RGX) arithmetically less than (ADR) into J	1	1	
(I|J) XGI	Index add overflow to mode bit:	▨	▨	▨
IXGI	Add (ADR) 48:16 to RGX; store overflow in mode register bit I	1	1	
JXGI	Add (ADR) 48:16 to RGX; store overflow in mode register bit J	1	1	
(I|J) XLD	Index subtract underflow to mode bit:	▨	▨	▨
IXLD	Subtract (ADR) 48:16 from RGX; store complement of overflow in I	1	1	
JXLD	Subtract (ADR) 48:16 from RGX; store complement of overflow in J	1	1	
LB	Test for RGA less than (ADR) in 8-bit bytes	2	2	
LEX	Load RGA exponent(s) from (ADR) exponent(s)	1	1	
ML	Multiply RGA by (ADR). Variants are: Suffix — Meaning A — Unsigned M — Fixed point N — Normalized R — Rounded	10	8	
MLA		10	8	
MLM		10	8	
MLMA		10	8	
MLN		10	9	
MLNA		10	9	
MLR		10	8	
MLRA		10	8	
MLRM		10	8	
MLRMA		10	8	
MLRN		10	9	
MLRNA		10	9	
Mode Register Instructions:	Load mode register bit from ACAR	▨	▨	▨
LDE	Set mode register bit with result of logical function specified in instruction address field.	1	1	
LDE1		1	1	
LDEE1		1	1	
LDG		1	1	
LDH		1	1	
LDI		1	1	
LDJ		1	1	

Table 6-2. FINST/PE Instruction Timing (Cont.)

Program Mnemonic Code	Operation	PE Clock Times		Notes
		32-Bit Mode	64-Bit Mode	
SETE	Set mode register bit with result of logical function specified in instruction address field	1	1	
SETE1		1	1	
SETF		1	1	
SETF1		1	1	
SETG		1	1	
SETH		1	1	
SETI		1	1	
SETJ		1	1	
MULT	For 32-bit mode, both halves enabled, multiply RGA by ADR contents; leave inner double-length product mantissa in RGA, outer in RGB	12	12	
NEB	Test for RGA not equal to (ADR) in 8-bit bytes	3	3	
NORM	Normalize	3	2	
OFB	Overflow bits of previous 8-bit byte instruction are transmitted from RGC to RGB	1	1	
RT (G|L)	Route:	▨	▨	▨
RTG	Transmit register (Y) of every PE to RGR of PE number (N+D) modulo a, where Y = a specified PE register, N = initial PE no., D = routing distance, and a = number of PEs in array (64/128/256)	5	5	b
RTL	Same as above, except single quadrant (a = 64)	3	3	b
SB	Subtract (ADR) from RGA. Variants are: Suffix — Meaning A — Unsigned M — Fixed point N — Normalized R — Rounded	6	4	
SBA		6	4	
SBM		5	3	
SBMA		5	3	
SBN		6	5	
SBNA		6	5	
SBR		10	6	
SBRA		10	6	
SBRN		10	7	
SBRNA		10	7	
SBB	Subtract (ADR) from RGA in 8-bit bytes	1	1	
SBEX	Subtract exponent(s) of (ADR) from RGA exponent(s)	1	1	
SCM	Execute one cycle of a multiplication	2	2	

Table 6-2. FINST/PE Instruction Timing (Cont.)

Program Mnemonic Code	Operation	PE Clock Times		Notes
		32-Bit Mode	64-Bit Mode	
Shift Instructions:	Shift: Variant — Meaning SH | RT — Shift | rotate A | AB — RGA | RGA + RGB (single | double) _ | M — Full register | mantissa L | R — Left | right	▨	▨	▨
RTAL		2	1	a
RTAR		2	1	a
SHABL		3	4	c
SHABR		3	4	c
SHABML		3	4	c
SHABMR		3	4	c
SHAL		3	1	a
SHAR		3	1	a
SHAML		2	1	a
SHAMR		2	1	a
STA	Store from RGA to memory	1	1	
STB	Store from RGB to memory	1	1	
STR	Store from RGR to memory	1	1	
STS	Store from RGS to memory	1	1	
STX	Store from RGX to memory	1	1	
SUB	Subtract 64-bit unsigned fixed point number (ADR) from RGA	1	1	
SWAP	Interchange (RGA) and (RGB)	1	1	a
SWAPA	Interchange the inner and outer operands in RGA	2	2	a
SWAPX	Interchange the outer operand of RGA and the inner operand of RGB	2	2	a
TCY	Transfer data from CDB to MAR	1	1	
TCYS	Add RGS to CDB and store in MAR	1	1	
TCYX	Add RGX to CDB and store in MAR	1	1	
T3A	Transfer contents of C register (RGC) to RGA	3	3	
Transmit Instructions:	Transmit source data to register indicated in op code. (Source is specified in bits 13:3 and 16:16.)	▨	▨	▨
LDA	Transmit to RGA	1	1	
LDB	Transmit to RGB	1	1	
LDD	Transmit to RGD	9	9	
LDR	Transmit to RGR	1	1	
LDS	Transmit to RGS	1	1	
LDX	Transmit to RGX	1	1	
XD	Subtract (ADR) 48:16 from RGX	1	1	
XI	Add (ADR) 48:16 to RGX	1	1	

NOTES

a. If overlap does not occur, one (1) additional clock time is required in both the 32-bit and 64-bit modes.

b. Routing times given are for D equal to plus or minus 1 or plus or minus 8. For other D's, the times are either 1 + 4N (for RTG) or 1 + 2N (for RTL) where N is the number of elementary shifts of distance eight and one required to make up the specified distance D.

c. DOUBLE LENGTH SHIFTS ARE EXECUTED OUT OF THE FINST OVERLAP STATION, AND CAN THEREFORE POSSIBLY OVERLAP A PRECEDING INSTRUCTION, BUT CANNOT BE OVERLAPPED WITH A FOLLOWING INSTRUCTION.

CONTENTS

SECTION VII
PERIPHERAL TIMING

I/O SUBSYSTEM TIMING

The I/O subsystem receives and passes on control actions from one portion of the ILLIAC IV system to another. It also passes data from one portion to another in response to controls.

Each action in the I/O subsystem is initiated by transmitting a command from the B6700 to the I/O subsystem. This command causes the fetching of a descriptor from memory. One of a number of things may happen in response to a descriptor, as follows:

1. A disk descriptor is written into the disk queuer, from which it is executed. An option is that a "list" of descriptors can be loaded into the queuer, or several lists, interlaced in the same block of memory space, in response to a single "initiate I/O" command.

2. A descriptor to read or write to the CU will cause action to take place between the I/O subsystem and the TMU portion of the CU.

3. A transfer from BIOM to array memory may take place.

Timing information therefore consists of:

(a) The time to get a descriptor from the B6700 into the I/O subsystem;

(b) The time for the CU to respond to the descriptor, if a CU descriptor;

(c) The time to load descriptors into the queuer, if a disk descriptor;

(d) The time for the disk system to respond to a descriptor which is contained in the queuer;

(e) Data transfer rates for disk to array;

(f) Data transfer rates for BIOM to array; and

(g) Data transfer rates for B6700 loading BIOM.

From the above, the time for I/O subsystem response can be determined since the total time for any one such response is the sum of the appropriate times listed above. A discussion of the individual time components follows.

OBTAINING A DESCRIPTOR

An "Initiate I/O" command is held on the scan bus until the I/O subsystem responds to the B6700. Therefore, the scan bus is tied up, after the beginning of the operation, for twice the cable delay (which is constrained to be less than one clock time in each direction, or 200 nanoseconds maximum) plus the time for logic in the I/O subsystem. Following the scan bus operation, two (sometimes one) memory cycles are required to fetch the descriptor. At 1.2 microseconds per memory cycle, 0.2-microsecond response on the scan bus, and 0.1-microsecond for logic, the total time required for fetching the descriptor to the I/O subsystem is 2.7 microseconds.

CU RESPONSE TO DESCRIPTOR

Descriptors controlling the interaction with the CU are of four types:

> Read CU: This descriptor causes a word, obtained from the CU, to be inserted into B6700 memory;
>
> Write CU: This descriptor causes a word obtained from B6700 memory to be inserted into the designated register in the CU;
>
> Scan CU: This descriptor causes a word obtained from B6700 memory to be inserted into the control register in the CU, and a word obtained from the CU in response to those controls to be inserted back into the B6700 memory at the same location (a "double length" variation on the scan retrieves two words from the CU for each word of control information sent to the CU);
>
> Stop CU: This is the "quadrant disable" descriptor, which issues a stop command to the CU, and retrieves nothing.

The read, write, and scan CU descriptors all operate in a "handshaking" mode, so that cable delays, regardless of the length of cable, are waited out. Further, no synchronism between CU and I/O is required, each being treated as though it were asynchronous with the other; each signal passed back and forth across this interface is resynchronized.

"Read" requires, as a result of the above, two cycles (200 nanoseconds) of I/O clock and two cycles (100 nanoseconds) of CU clock, plus twice the cable delay from I/O to CU (300 nanoseconds if 100 feet), plus the time for TMU operations, plus the time for the memory cycle (1.2 microseconds) in the B6700. The TMU time is seven CU clocks if no action internal to the CU is taking place, therefore, the total "read" time takes slightly more than 2 microseconds.

"Write" requires the same steps as "read". However, "write" will result in an instruction being inserted into the TMU's control register (TCR). This instruction must await an opportunity to be executed (in general, just before ILA passes the next instruction to ADVAST if the clocks are running, although there is no wait if the quadrant is in single-step mode for debugging), and the instruction will then take time of its own for execution in TMU. A typical TMU instruction (LICR, LISR, SA, SL, SR, SOC, SOD, TIC, and TOC) is estimated to be five clocks long in the execution phase, although this varies. Therefore "write" takes about 2.5 microseconds plus the time that the TMU is waiting to get control of the CU.

"Scan" is divided in the I/O into separate write and read commands, taking 4.6 or 6.6 microseconds depending on whether the read is single or double.

"Stop", or quadrant disable, requires only a one-way cable delay, plus the TMU time needed to recognize the command, or about 0.400 microsecond.

LOADING DESCRIPTORS INTO QUEUER

The queuer action is divided into scan cycles; a scan cycle being the time it takes to scan all possible queuer contents for transactions which refer to a single storage unit. If there are "S" storage units in the system, there will be no more than "S" scan cycles before the first storage unit is scanned once again. A storage unit for which there is no descriptor in the queuer does not get a scan cycle.

A scan cycle, for the i^{th} storage unit, takes $0.2 + 0.4\ D_i$ microseconds when there are D_i descriptors stored in the queuer for the i^{th} storage unit. At the end of any one scan cycle, one word, if descriptors are waiting to be loaded, is fetched from memory and inserted into the queuer. The memory access time of the next word allows the next scan cycle to start. The worst-case for loading descriptors into the queuer occurs when there is only one storage unit, say j, having valid descriptors, and D_j is large. Then each scan cycle takes $0.2 + 0.4 D_j$ microseconds, and each two-word descriptor takes two scan cycles to be loaded into the queuer. There are 24 slots in the queuer. One descriptor can be loaded when D_j is as high as 23, or one descriptor can take as long as 18.8 microseconds. A list of 24 descriptors, all destined for the same storage unit, can take as long as 249.6 microseconds to be loaded.

The times given above are worst-case maxima. Typical times are much shorter, but depend on the number of descriptors in the queuer and their distribution among storage units.

RESPONSE TIME OF DISK TO DESCRIPTOR FOUND IN QUEUER

The address is offset from its associated information segment by an amount dependent on circuit settling time. This offset is estimated at two segment's worth, or 65.4 microseconds. There will be, therefore, a maximum of 65.4 microseconds between the finding of an address in the queuer and the start of the associated data segment(s) transfer to or from the disk.

A related question is whether there is any possibility that a disk transaction when due, can somehow be missed, having to wait an unnecessary revolution. This cannot happen, as shown by the following analysis.

The time that it takes the queuer to go through a complete series of scan cycles for each storage unit in the system may be expressed as:

$$T = 0.1\,N + \sum_{i=1}^{S} (0.2 + 0.4\,D_i)$$

Where S = number of storage units present in the system,

N = number of storage units for which no descriptor is found,

D_i = number of descriptors in the queuer for the i^{th} storage unit.

The upper limit on D_i is given by:

$$\sum_{i=1}^{S} D_i \leq 24$$

since there are only 24 slots in the queuer. For any value of N or S up to the maximum S of 32, the time T is always less than 19.1 microseconds. The addressable segment on each storage unit therefore is compared at least once with the contents of the queuer, so there is no possibility of waiting a revolution unnecessarily. Only during an actual transfer which makes the DFC (disk file controller) busy will disk transactions be passed over.

DATA TRANSFER RATES FROM DISK TO ARRAY

The time per data segment on the disk is stated to be 32.7 microseconds per 16,384-bit segment. This is one 1024-bit I/O word per 2.04 microseconds in either or both of the DFCs, and results in interference to memory of a maximum of one memory cycle per I/O word.

DATA TRANSFER RATES FROM BIOM TO ARRAY

Data from BIOM to array is transferred through the same channels designed for use by the disk system. Since these channels are designed to keep ahead of the disk rate of 2.04 microseconds per 1024-bit I/O word, and since BIOM is an on-demand device potentially much faster than the disk, the disk transfer rate also serves as a lower bound on the transfer rate between the BIOM and array. More precise estimates of this rate are not fruitful, since the rate depends on unspecified cable lengths, and on the phase and frequency relationships between I/O clock and quadrant clock.

DATA TRANSFER RATES BETWEEN B6700 and BIOM

To the B6700 the BIOM appears as one of its normal memory modules, as far as the hardware is concerned. A response time of 1.2 microseconds is expected when the B6700 is operating one of its own memory modules from a single processor or I/O. This time is also an estimate of the time per memory cycle required to load a series of words into the BIOM.

This limit will not usually be the controlling factor in loading the BIOM. One visualizes loading the BIOM from a peripheral device, or creating within it a file by executing a program. This file would be written onto the disk or into the array memory. In one case, the data rate of the peripheral device will be the controlling factor; in the other case it depends on the execution time of the program.

Loading the BIOM from the B6700 side is not permitted when the BIOM is busy with a disk transfer. There is not sufficient time to inject B6700 cycles between the BIOM cycles required to keep pace with the disk.

B6700 PERIPHERAL DEVICE TIMING

Table 7-1 presents a summary of the operating times for several typical B6700 peripheral devices. More detailed information can be found in the Burroughs B6700/B7500 Information Processing System Characteristics Manual.

Table 7-1. Peripheral Device Timing

Unit	Characteristics
B-9111 Card Reader	800 CPM, 2400 card capacity
B-9213 Card Punch	300 CPM, 1000 card capacity
B-9243-1 Line Printer	1040 lines per minute, 120 print positions (standard)
B-9220 Paper Tape Punch	100 characters per second, BCL or Baudot code
B-9391 Magnetic Tape Unit	7 channels, 800/556/200 BPI, 72/50/18 KB per second
B-9382-4 Magnetic Tape Cluster	9 channels, 1600 BPI, 72 KB per second
B-9375-10 Disk File	133×10^6 characters, 23 milliseconds access time

APPENDIX A

GLOSSARY

APPENDIX A

GLOSSARY

AC0-3
(ACARs)

ADVAST Accumulator Registers 0-3: Serve as accumulators for all CU operations; as the source of memory address for ADVAST memory operations (bits 40-63); as the source of the first operand for instructions requiring two operands; as a local memory location; as an index register (bits 1-15 are the increment bits, bits 16-39 are the limit bits, and bits 40-63 are the address bits). Bit 0 is the half-word designator for transfers between ICR or IIA.

ACR

ADVAST Control Register: A 16-bit register that contains indicators which may be used to determine the state of various conditions within the CU.

ACU

Absolute CU Number: A hard-wired 4-bit register which contains the CU number of this CU in logical form, i.e., CU0 = 1000; CU1 = 0100; CU2 = 0010; CU3 = 0001.

ADA

PE Address Adder: A 16-bit adder in the PEs used to generate PE memory addresses and index values.

ADB

ADVAST Data Buffer: A 64-word, 64-bit per word random access memory that is part of local memory.

ADC

ADVAST Data Control: An 8-bit register used to control the loading of local memory.

ADVAST

Advanced Station, CU: Processes all instructions either completely or transmits them to the final station (FINST) if they are to be executed by the PUs. Program control and interrupt handling are accomplished in this station.

AFR

ADVAST-to-FINST Instruction Register: A 12-bit register which contains the instruction which is being transferred to the FIQ.

AIN

ADVAST Interrupt Register: A 16-bit register used to store the various conditions which will cause the CU to transfer to the interrupt routine.

AIR — ADVAST Instruction Register: A 32-bit register which receives instructions from IWS during normal instruction execution.

AIT — ADVAST Instruction Timer: Generates the proper timing commands for instruction execution at ADVAST.

ALR — ADVAST Local Memory Address Register: An 8-bit register used to store the local memory address. It receives the address from the AIR for all instructions that require access to PE Memory.

AMR — ADVAST Mask Register: A 16-bit register that establishes which of the bits in AIR will be permitted to cause an interrupt.

APC — ADVAST Instruction Parity Checker: Used to check parity of the instruction contained in the AIR, before modification, for instructions loaded from IWS.

Array — An array is composed of the set of PUs that are being used in a coordinated manner. The set is specified in the MC registers.

AWR — ADVAST-to-FINST Data Register: A 64-bit register which contains data that is being transferred to the FDQ.

BIOM — Buffer Input-Output Memory: A 1024-word, 128-bit memory when used by the DC and a 2730-word 48-bit memory when used.by the B6700 system. It functions as a speed differential buffer between the two systems. Its cycle time is 250 nanoseconds for a read/restore cycle for 128-bit words.

Broadcast — To make information available to all CUs or PUs in an array simultaneously.

BSW — Barrel Switch: A logic network that accomplishes shifts of up to 64 bits in one clock time.

CDB — Common Data Bus: The set of 64 lines on which addresses, counts, and data words are broadcast from CUs to PUs.

DC — Descriptor Controller, DC: A section of the IOS/ DC. It contains the queuer and control registers, and performs descriptor interpretation and generation functions.

CPA — Carry Propagate Adder: A device in which one output is a sum representation of the quantities represented by its inputs. The unit is a 64-bit carry propagate adder partitioned into group segments of four bits each. The bit- and group-level carry signals are propagated in parallel segments to form the adder summed output.

CU — Control Unit: The CU is composed of the ADVAST, FINST, ILA, MSU, and TMU subunits. Its functions are to control a quadrant of 64 PUs directly and to synchronize multiquadrant operation.

Descriptor — A block of control information exchanged between either a processor or memory and its connected input-output controllers.

DFC — Disk File Control: Controls data flow between a disk storage unit (SU) and another memory. It can select among five attached electronics units (EUs) to which the SUs are connected.

DFDC — Disk File Descriptor Control: That portion of the DC associated with, and containing the controls for, a specific DFC.

Disk, II AP — The bulk storage unit for the ILLIAC IV system. It has an instantaneous transfer rate of approximately 0.5 billion bits per second.

Driver (DR) — A circuit capable of transmitting a signal over long distances through a cable.

EU — Electronics Unit, Disk: Controls the disk units (SU) attached to it and the packing and unpacking of data being exchanged with a disk unit.

FATR — FINST Address Transmission Register: These 24 bits of storage contain the memory addresses for MSU sampling. Its content is updated for every memory reference required by the PE instructions processed by FINST..

FCC — FINST Control Counter: This 8-bit counter is used to control the repetitions of commands within instructions. From the decremented contents of this register are derived the controls for the routing interconnections in the cabinet logic.

FCDA
FCDB
FCDC
FCDD — FINST Command Drivers: Each group of 260 drivers sends the PE subcommands from FINST to two PU cabinets, where subcommands are distributed to each PE. All of these drivers receive inputs from the FINST control register.

FCE — FINST Compare Equal: This logic compares the values of the load pointer and the read pointer to determine when the final queue becomes empty or full. Its outputs are fundamental to the coordination of FINST and ADVAST.

FCLDA
FCLDB
FCLDC
FCLDD — FINST Cabinet Logic Drivers: Each of these four groups of eight drivers sends controls to the cabinet logic for routing and other MSU uses of the quadrant data paths. One group of drivers provides signals to two cabinets where they are distributed to the separate cabinet and PE Logic.

FCLR

FINST Cabinet Logic Register: This 8-bit register stores for both retiming and buffering the six FINST-generated and six MSU-generated enables (4 are common) to be sent to the cabinet logic.

FCR

FINST Command Register: The first 260 bits of this register receive inputs from the ROM which are PE subcommands to be retimed before being driven from the CU cabinet. The next 100 bits of FCR retime the internally used subcommands generated by the ROM for FINST and other parts of the CU. The final eight bits of FCR retime subcommands generated in the MSU for transmission to the PE concurrently with related subcommands.

FDB

FINST Data Buffer: A 16-bit buffer which provides drive for the outputs of FDS.

FDDA
FDDB
FDDC
FDDD

FINST Data Drivers: Each group of 64 drivers sends the PE data, shift amount, or addresses over the common data bus to two PU cabinets, where the bits are distributed to each PE. All of these drivers receive inputs from the FDR.

FDE

FINST Mode Enables: A group of 64 drivers used to transmit the contents of a selected ACR to the proper mode bit of each PE in the quadrant. Each bit is sent to a unique PE.

FDF
FDH

FINST Data Queue Select Gates: These units select one of the five or three, respectively, words of the data queue for use in FINST.

FDG

FINST Data Gates: Select the 64-bit parallel word from the preceding gates and eight registers of the data queue for use in FINST or MSU.

FDQ0
through
FDQ7

FINST Data Queue: Registers 0 through 7. The queue stores data in these eight registers of 64 bits each. The data is interpreted by the nature of the instruction to be literal, address, shift amount, or variant. The queue is loaded from AWR of ADVAST.

FDR

FINST Data Register: This 64-bit register provides retiming and buffering of the common data bus information before it leaves the CU through the data drivers.

FDS

FINST Data Selection: Gates for the selection of address from either MSU or the queue as a 24-bit parallel word. An adder in FDS corrects shift amounts, and other gates force special shift values onto the common data bus.

FGS

A 5-input operand select gate 64 bits wide used to supply the TMU with the desired FCR or FDR outputs.

FGT — A set of multiple input operand select gates arranged to format into 64-bit words the outputs of important shorter registers for display by the TMU.

FIAD — FINST Instruction Address Decoder: These gates decode the instruction into the first (perhaps only) address for accessing of the read-only memory.

FIAR — FINST Instruction Addressing Register: A 12-bit register which stores all instructions as their first (perhaps only) address is decoded for accessing of the read-only memory.

FIB — FINST Instruction Buffer: A 12-bit buffer for driving the output of the instruction queue to the instruction register.

FICL — FINST Instruction Control Logic: These gates participate in the sequencing of instructions from FIR to the read-only memory.

FID — FINST Instruction Decoder: Logic for the decoding of the instruction register for use in developing the particular subcommands.

FIF
FIH — FINST Instruction Queue Select Gates: These units select one of four words of the instruction queue for use in FINST.

FIGR — FINST Instruction Gating Register: This register contains one bit for each of the PE instructions executed from the read-only memory which has more than one clock time. It is set to the proper value by the FIAD during the first step of the instruction and cleared at the end of the instruction.

FINQ — Final Queue: Eight 80-bit word storage composed of FIQ and FDQ. It stores instructions and data passed from ADVAST to FINST before FINST execution. FINST services FINQ on a first-in first-out basis.

FINST — Final Station CU: Accepts partially decoded instructions from the ADVAST, converts them into fully converted microsequences, and broadcasts the microsequences and other data to 64 connected PUs.

FIP — FINST Instruction Picker: A set of 12 operand select gates for selection of one of the TMU, FIF, or FIH inputs to be the instruction for execution.

FIQ0 through FIQ7 — FINST Instruction Queue: Registers 0 through 7. The queue stores instructions in these eight registers of 12 bits each. The queue is loaded from AFR to ADVAST.

FIR — FINST Instruction Register: A 12-bit register which stores the output of the instruction queue buffers as the instruction is being interpreted for execution.

FISC FINST Instruction Step Counter: A set of latches arranged as a shift register for sequencing the steps in a multiple-step PE instruction.

FITE FINST Instruction Gate Enables: This set of AND gates selects each word address for the steps of the multiple-step PE instructions in accordance with the outputs of FIAR and FISC.

FLP FINST Load Pointer: An 8-bit shift register used to select the next register of the FIQ and FDQ to be loaded to ADVAST.

FOAD FINST Overlap Address Decoder: These gates decode the overlap into the first (perhaps only) address for accessing of the read-only memory.

FOAR FINST Overlap Addressing Register: A 12-bit register which stores all overlaps as their first (perhaps only) address is decoded for accessing of the read-only memory.

FOB FINST Overlap Buffers: A set of 12 buffers for driving the output of the instruction queue to the overlap register.

FOCL FINST Overlap Control Logic: These gates participate in the sequencing of overlaps from FOR to the read-only memory.

FOD FINST Overlap Decoder: Logic for the decoding of the overlap register for use in developing the particular subcommands.

FOGR FINST Overlap Gating Register: This register contains one bit for each type of overlap executed from the read-only memory which has more than one clock time. It is set to the proper value by the FOAD during the first step of the overlap and cleared at the end of the overlap.

FOP FINST Overlap Picker: A set of 12 operand select gates for selection of one of the TMU, FIF, or FIH inputs to be the overlap for execution.

FOR FINST Overlap Register: A 12-bit register which stores the output of the overlap buffers as the overlap is being interpreted for execution.

FOSC FINST Overlap Step Counter: A set of latches arranged as a shift register for sequencing the steps in a multiple-step overlap.

FOTE FINST Overlap Gate Enables: 'This set of AND gates selects each word address for the steps of the multiple-step overlaps in accordance with the outputs of FOGR and FOSC.

FRCL FINST Route Control Logic: This logic generates the six cabinet logic control signals needed to arrange the quadrant data paths for the ROUTE instruction.

FROM FINST Read Only Memory: A transistor matrix memory of 720 words and 360 bits. Each word is a step of a PE instruction; each bit is a PE or FINST subcommand.

FRP FINST Read Pointer: An 8-bit shift register used to select the next register of the FIQ and FDQ to be read by FINST.

FRR FINST Route Register: An 8-bit register used to store the route distances derived from the address field of the data queue.

IAM ILA Associative Memory: A content-addressable memory which contains eight 21-bit block addresses of the words stored in the IWS. Associated with each of the eight addresses is a "present" bit (IAMP) which is set when the first word of a block of instructions is stored in IWS. Note that it is not necessary to clear the eight locations on an IWS clear operation; it is sufficient to clear the IAMPs and use their status to establish if the address stored in IAM is valid.

ICR ILA Instruction Counter: A 25-bit register which contains a binary number representing the address of the instruction presently under execution by ADVAST. The binary number is interpreted by the configuration control logic to determine the absolute memory address of the instruction.

IIA ILA Interrupt Address: A 25-bit register used to identify the location of the next program instruction in the interrupted program. It is also used as a base for storing pertinent information which is necessary to return to the noninterrupt program at the completion of the interrupt program.

ILA Instruction Look-ahead Unit, CU: A subunit of the Control Unit that maintains a queue of up to 128 instructions for the ADVAST unit. It contains an associative memory (IAM), the instruction counter (ICR), and a 64-word bit instruction store (IWS).

IOS Input Output Switch: A buffer between the DC unit and the ILLIAC IV memory. It blocks 256 words from the DC into 1024-bit words for ILLIAC IV memory and the reverse.

I/O Word The package of information transmitted to or from IOS from or to PEM during a single memory cycle. Initially, this is 1024 bits.

IWS ILA Instruction Word Storage: A 4096-bit memory of 64 × 64-bit words sectioned into eight-word blocks. The instructions are fetched from memory and stored in IWS in 512-bit blocks.

List In this manual, a sequence of I/O descriptors which can be initiated by a single command from the B6700 processor.

LOD Leading Ones Detector: A set of logic used to generate a shift count that indicates the bit distance between the high-order "one" bit and the high-order position of a word.

LOG Logic Unit: A set of logic that performs logical operations (such as NOT, AND, and OR) on operands

Mantissa As used in this manual, the fractional portion of a number in floating-point representation.

MAR Memory Address Register, PE: A 16-bit register that holds an effective address used to read or write a word in PE Memory.

MC0, MC1, MC2 MSU Configuration Control Registers: Three 4-bit registers used to determine which CUs are in the array for a particular operation. MC0 is the array size register; MC1 controls program fetches and establishes (relative to MC0) the location of the stored program; MC2 is used to control instruction execution.

MDG Multiplier Decoder Gates, PE: Generate the weight values used by the multiplicand select gates (MSG) for the multiply algorithm.

MLU Memory Logic Unit: A set of logic and registers that controls all memory accesses to its associated Processing Element Memory.

MPX Multiplexor, B6700: The unit that controls the operation of a B6700 input-output exchange.

MSG Multiplicand Select Gates, PE: Generate subproduct inputs that are used in mechanizing the multiply algorithm.

MSU Memory Service Unit, CU: A subunit of the Control Unit that resolves memory request conflicts and converts binary addresses into array addresses by using the configuration control logic (see MC0, MC1, MC2).

OSG Operand Select Gates: A set of logic that selects one of several inputs for transmission to another stage of logic or a register.

PAT Pseudo-adder Tree: Receives five 56-bit input word signals in parallel to three carry-save adders (CSA). The pseudo-adder tree inputs are reduced to sum and carry outputs, which are in turn added again until the input words are reduced to a single pseudo-adder tree sum and carry output. Implementation of this process requires three CSAs per bit position.

PE
Processing Element: A set of registers and combinational logic which is capable of executing a large complement of externally decoded instructions. Each PE can communicate directly with its orthogonal neighboring PE and has access to its own PE Memory.

PEM
Processing Element Memory: Contains 2048-word 64-bit per word memory which may be accessed from either its associated PE or the IOS. Its cycle time is approximately 250 nanoseconds.

PEM
The register of PE memory error latches (address 154 on p. 5-6).

PU
Processing Unit: A pluggable unit containing one PE, one MLU, and one PEM.

PUC
Processing Unit Cabinet: Holds eight PUs and contains their power supplies and other common circuitry.

Queuer
An associative storage unit in the DC that stores disk type descriptors from the B6700 and services them in such a way as to minimize disk rotation access latency.

Receiver (RCVR)
A circuit capable of detecting signals transmitted over long distances through cables.

Result Descriptor
A word of information prepared at the end of the execution of an I/O descriptor and passed back to the B6700 via the scan bus.

RGA
PE Register A: A 64-bit register in which the result of an arithmetic or logic operation is formed. The register is operative only when a PE status is enabled.

RGB
PE Register B: A 64-bit register in which second operands are stored for arithmetic or logic operations.

RGC
PE Register C: A 64-bit register in which carries are stored for later propagation. Its contents may be accessed by the OFB instruction.

RGD
PE Register D (Mode Register): Provides intermediate storage between a PE and its CU. The unit is an 8-bit register in which a result may be formed by signals obtained locally from executed instructions or remotely from the CU. Decoded register bit positions form outputs which specify a PE's status (operative or inoperative), arithmetic overflow, instruction test results, and the logic level of a specified bit.

RGR
PE Register R: A 64-bit register which provides intermediate storage during instruction execution. The register is accessible even though the PE's mode status is disabled.

RGS

PE Register S: A 64-bit register in which intermediate results obtained during PE processing are stored. The register is operative only when a PE status is enabled.

RGX

PE Register X: A 16-bit register whose content is added to an operand address or literal field prior to or during the execution of an instruction.

Scan Cycle

The time required by the queuer to search its own contents for all descriptors pertaining to a single storage unit.

Scan Bus

A bus linking all elements of the B6700 and the DC, used by the B6700 processor to initiate I/O operations, and used by the I/O controllers to pass result descriptors back to the processor.

SU

Storage Unit, Disk: A module of disk storage containing 20 million, 8-bit bytes of data.

TCC

TMU Condition Control Register: A 9-bit register that exercises control over operation of the CU subunits.

TCI

TMU Condition Indicator Register: A 16-bit register that describes the content of the TRO register.

TCR

TMU Command Register: A 48-bit register that receives its input from the TMP or the B6700 from which the data is interpreted as instructions.

TDC

TMU Data Comparator: A 64-bit comparator which may be used to compare test values with CU register contents.

TMP

Test Maintenance Panel, TMU: A set of indicators, switches, and controls which can be manually operated to control the operation of the CU.

TMU

Test Maintenance Unit, CU: A subunit of the Control Unit. It can be used to communicate with the B6700 via the DC and to exercise control over an ILLIAC IV quadrant.

TRI

TMU Input Register: A 64-bit register that may be used to hold data words sent from the B6700 until required by other sections of the CU.

TRO

TMU Output Register: A 64-bit register which may be used as a temporary storage register by the ADVAST instruction set or as a holding register for data words being sent to the B6700.

Word Bus

A port of the B6700 I/O multiplexer which allows access to B6500 memory via the multiplexer's path to memory. Used for DC access to the B6700's memory.

APPENDIX B

INSTRUCTION INDEX

ADVAST INSTRUCTION INDEX

Mnemonic Code	Octal Code	Ref. Page	Mnemonic Code	Octal Code	Ref. Page	Mnemonic Code	Octal Code	Ref. Page
ALIT	16XX	3-12	INCRXC	0002	3-41	TXEF	1413	3-70
BIN	0610	3-13	INR	0007	3-42	TXEFA	1412	3-70
BINX	0611	3-13	JUMP	17XX	3-43	TXEFAM	1216	3-71
CACRB	0001	3-15	LDC	0011	3-44	TXEFM	1217	3-71
CADD	0402	3-17	LDL	0405	3-45	TXET	1411	3-70
CAND	0410	3-18	LEADO	0201	3-46	TXETA	1410	3-70
CCB	1101	3-19	LEADZ	0200	3-46	TXETAM	1214	3-71
CEXOR	0407	3-20	LESSF	1507	3-66	TXETM	1215	3-71
CLC	0005	3-21	LESSFA	1506	3-66	TXGF	1403	3-72
COMPC	0006	3-22	LESST	1505	3-66	TXGFA	1402	3-72
COPY	0204	3-23	LESSTA	1504	3-66	TXGFAM	1302	3-73
COR	0411	3-24	LIT	0003	3-48	TXGFM	1303	3-73
CRB	0207	3-25	LOAD	0600	3-49	TXGT	1401	3-72
CROTL	0015	3-26	LOADX	0601	3-49	TXGTA	1400	3-72
CROTR	0017	3-27	ONESF	1007	3-67	TXGTAM	1300	3-73
CSB	0013	3-28	ONESFA	1006	3-67	TXGTM	1301	3-73
CSHL	0014	3-29	ONEST	1005	3-67	TXLF	1407	3-74
CSHR	0016	3-30	ONESTA	1004	3-67	TXLFA	1406	3-74
CSUB	0403	3-31	ONEXF	1017	3-68	TXLFAM	1306	3-75
CTSBF	1102	3-32	ONEXFA	1016	3-68	TXLFM	1307	3-75
CTSBT	1100	3-32	ONEXT	1015	3-68	TXLT	1405	3-74
DUPI	0401	3-34	ONEXTA	1014	3-68	TXLTA	1404	3-74
DUPO	0400	3-35	ORAC	0205	3-52	TXLTAM	1304	3-75
EQLXF	1417	3-64	SETC	0012	3-53	TXLTM	1305	3-75
EQLXFA	1416	3-64	SKIP	1103	3-54	WAIT	0206	3-78
EQLXT	1415	3-64	SKIPF	1107	3-69	ZERF	1003	3-76
EQLXTA	1414	3-64	SKIPFA	1106	3-69	ZERFA	1002	3-76
EXCHL	0406	3-36	SKIPT	1105	3-69	ZERT	1001	3-76
EXEC	0004	3-38	SKIPTA	1104	3-69	ZERTA	1000	3-76
FINQ	0010	3-39	SLIT	16XX	3-55	ZERXF	1013	3-77
GRTRF	1503	3-65	STL	0404	3-56	ZERXFA	1012	3-77
GRTRFA	1502	3-65	STORE	0602	3-58	ZERXT	1011	3-77
GRTRT	1501	3-65	STOREX	0603	3-58	ZERXTA	1010	3-77
GRTRTA	1500	3-65	TCCW	0203	3-60			
HALT	0000	3-40	TCW	0202	3-61			

TMU INSTRUCTION INDEX

Mnemonic Code	Octal Code	Ref. Page	Mnemonic Code	Octal Code	Ref. Page	Mnemonic Code	Octal Code	Ref. Page
EFA	160	5-16	SAT	047	5-25	SR	005	5-24
EFF	164	5-18	SIS	120	5-26	SRT	045	5-25
LICR	041	5-20	SIV	100	5-27	TIC	121	5-33
LISR	040	5-21	SL	006	5-24	TOC	002	5-34
RPT	001	5-22	SLT	046	5-25	WIS	044	5-35
RUN	020	5-23	SOC	011	5-30			
SA	007	5-24	SOD	010	5-32			

FINST/PE INSTRUCTION INDEX

Mnemonic Code	Octal Code	Ref. Page	Mnemonic Code	Octal Code	Ref. Page	Mnemonic Code	Octal Code	Ref. Page
AD	3504	4-17	IXL	2310	4-59	NORN	2307	4-31
ADA	3505	4-17	IXLD	2712	4-62	OFB	2506	4-76
ADB	2606	4-22	JAG	3715	4-52	OR	2304	4-31
ADD	2604	4-23	JAL	3717	4-52	ORN	2306	4-31
ADEX	2500	4-24	JB	3503	4-54	RAB	3701	4-36
ADM	3414	4-17	JLE	3517	4-55	RTAL	3513	4-87
ADMA	3415	4-17	JLG	3315	4-55	RTAR	3512	4-88
ADN	3404	4-17	JLL	3317	4-55	RTG	2413	4-77
ADNA	3405	4-17	JLO	3311	4-57	RTL	2412	4-77
ADR	3506	4-17	JLZ	3313	4-57	SAB	3702	4-36
ADRA	3507	4-17	JME	3515	4-55	SAN	3702	4-38
ADRN	3406	4-17	JMG	3115	4-55	SAP	3701	4-38
ADRNA	3407	4-17	JML	3117	4-55	SB	3704	4-79
AND	2704	4-27	JMO	3111	4-57	SBA	3705	4-79
ANDN	2706	4-27	JMZ	3113	4-57	SBB	2607	4-82
ASB	2507	4-26	JSE	2513	4-59	SBEX	2501	4-83
~~ASTRG~~	~~2417~~	~~4-26A~~	JSG	2113	4-59	SBM	3614	4-79
~~ASTRL~~	~~2416~~	~~4-26A~~	JSL	2313	4-59	SBMA	3615	4-79
CAB	3700	4-33	JSN	3503	4-54	SBN	3604	4-79
CHSA	3700	4-35	JXE	2511	4-59	SBNA	3605	4-79
CLRA	2411	4-39	JXG	2111	4-59	SBR	3706	4-79
COMPA	2211	4-40	JXGI	2711	4-61	SBRA	3707	4-79
DV	3304	4-41	JXL	2311	4-59	SBRN	3606	4-79
DVA	3305	4-41	JXLD	2713	4-62	SBRNA	3607	4-79
DVM	3214	4-41	LB	2107	4-63	SCM	2104	4-85
DVMA	3215	4-41	LDA	2617	4-104	SETE	2514	4-69
DVN	3204	4-41	LDB	2700	4-104	SETE1	2515	4-69
DVNA	3205	4-41	LDD	2212	4-104	SETF	2516	4-69
DVR	3306	4-41	LDE	2114	4-69	SETF1	2517	4-70
DVRA	3307	4-41	LDE1	2115	4-69	SETG	2714	4-70
DVRM	3216	4-41	LDEE1	2116	4-69	SETH	2715	4-70
DVRMA	3217	4-41	LDG	2314	4-69	SETI	2716	4-70
DVRN	3206	4-41	LDH	2315	4-69	SETJ	2717	4-70
DVRNA	3207	4-41	LDI	2316	4-69	SHABL	3711	4-89
EAD	2010	4-45	LDJ	2317	4-69	SHABML	3713	4-91
EOR	2505	4-29	LDR	2701	4-104	SHABMR	3712	4-92
EQV	2504	4-30	~~LDRAG~~	~~2415~~	~~4-106A~~	SHABR	3710	4-90
ESB	2410	4-48	~~LDRAL~~	~~2414~~	~~4-106A~~	SHAL	3501	4-93
GB	2106	4-50	LDS	2702	4-104	SHAML	3511	4-95
IAG	3714	4-52	LDX	2703	4-104	SHAMR	3510	4-96
IAL	3716	4-52	LEX	2117	4-64	SHAR	3500	4-94
IB	3502	4-54	ML	3104	4-65	STA	2612	4-97
ILE	3516	4-55	MLA	3105	4-65	STB	2613	4-97
ILG	3314	4-55	MLM	3014	4-65	STR	2614	4-97
ILL	3316	4-55	MLMA	3015	4-65	STS	2615	4-97
ILO	3310	4-57	MLN	3004	4-65	STX	2616	4-97
ILZ	3312	4-57	MLNA	3005	4-65	SUB	2605	4-99
IME	3514	4-55	MLR	3106	4-65	SWAP	3103	4-100
IMG	3114	4-55	MLRA	3107	4-65	SWAPA	3303	4-101
IML	3116	4-55	MLRM	3016	4-65	SWAPX	3703	4-102
IMO	3110	4-57	MLRMA	3017	4-65	T3A	2105	4-103
IMZ	3112	4-47	MLRN	3006	4-65	TCY	3100	---
ISE	2512	4-59	MLRNA	3007	4-65	TCYS	3101	---
ISG	2112	4-59	MULT	2213	4-72	TCYX	3102	---
ISL	2312	4-59	NAND	2705	4-27	XD	2503	4-107
ISN	3502	4-54	NANDN	2707	4-27	XI	2502	4-108
IXE	2510	4-59	NEB	2210	4-73			
IXG	2110	4-59	NOR	2305	4-31			
IXGI	2710	4-61	NORM	2013	4-74			