ILLIAC IV MACHINE REFERENCE MANUAL FOR THE PROGRAMMER
IAC DOCUMENT NUMBER: PG-11700-0000-B
MAY 1, 1975

ILLIAC IV MACHINE REFERENCE MANUAL FOR THE PROGRAMMER

Revised July 1977

INSTITUTE FOR ADVANCED COMPUTATION

1095 EAST DUANE AVENUE
SUNNYVALE, CALIFORNIA 94086
TELEPHONE (408) 735-0635

ADVAST Instruction Index

Mnemonic	Field A OP Code	Field B OP Code	Page
ALIT	16	- -	4-9
BIN	06	10	4-10
BINX	06	11	4-10
CACRB	00	01	4-11
CADD	04	02	4-12
CAND	04	10	4-13
CCB	11	01	4-14
CEXOR	04	07	4-15
CLC	00	05	4-16
COMPC	00	06	4-17
COR	04	11	4-18
CRB	02	07	4-19
CROTL	00	15	4-20
CROTR	00	17	4-21
CSB	00	13	4-22
CSHL	00	14	4-23
CSHR	00	16	4-24
CSUB	04	03	4-25
CTSBF	11	02	4-44
CTSBT	11	00	4-44
DUPI	04	01	4-26
DUPO	04	00	4-27
EQLXF	14	17	4-45
EQLXT	14	15	4-45
EXCHL	04	06	4-28
EXEC	00	04	4-29
FINQ	00	10	4-30
GRTRF	15	03	4-46
GRTRT	15	01	4-46
HALT	00	00	4-31
INCRXC	00	02	4-32
JUMP	17	- -	4-33
LDC	00	11	4-34
LDL	04	05	4-35
LEADO	02	01	4-36
LEADZ	02	00	4-36
LESSF	15	07	4-47
LESST	15	05	4-47
LIT	00	03	4-37
LOAD	06	00	4-38
LOADX	06	01	4-38
ONESF	10	07	4-48
ONEST	10	05	4-48
ONEXF	10	17	4-49
ONEXT	10	15	4-49
SETC	00	12	4-39
SKIP	11	03	4-50
SKIPF	11	07	4-51
SKIPT	11	05	4-51
SLIT	16	- -	4-40
STL	04	04	4-41
STORE	06	02	4-42
STOREX	06	03	4-42
TXEF	14	13	4-52
TXEFM	12	17	4-53
TXET	14	11	4-52
TXETM	12	15	4-53
TXGF	14	03	4-54
TXGFM	13	03	4-55
TXGT	14	01	4-54
TXGTM	13	01	4-55
TXLF	14	07	4-56
TXLFM	13	07	4-57
TXLT	14	05	4-56
TXLTM	13	05	4-57
ZERF	10	03	4-58
ZERT	10	01	4-58
ZERXF	10	13	4-59
ZERXT	10	11	4-59

ILLIAC IV MACHINE REFERENCE MANUAL FOR THE PROGRAMMER

INSTITUTE FOR
ADVANCED COMPUTATION

1095 EAST DUANE AVENUE
SUNNYVALE, CA 94086
TELEPHONE (408) 735-0635

Preface

The primary purpose of this manual is to describe the instruction set used by ILLIAC IV programmers. A few instructions that are of interest only for test and maintenance purposes are omitted, as are all instructions and instruction variants that apply only to a four-quadrant machine. Hence the subset of instructions described in this manual is intended primarily for the user/programmer of the existing ILLIAC IV. As such, it supersedes all previous documentation of the instruction set used by ILLIAC IV programmers.

CONTENTS

LIST OF ILLUSTRATIONS AND TABLES

Section 1
ILLIAC IV Addressing

CONTENTS

Section 1

ILLIAC IV Addressing

The addressable entities in the ILLIAC IV may be divided into three groups, each with its own type of addressing: *processor memory (PM) locations, PE registers,* and *CU registers.* The addresses are generally contained in the ADR field of an ADVAST or FINST/PE instruction and may, in general, be indexed in the CU by adding the contents of a specified ACAR.

Each of the following subsections describes one type of addressing.

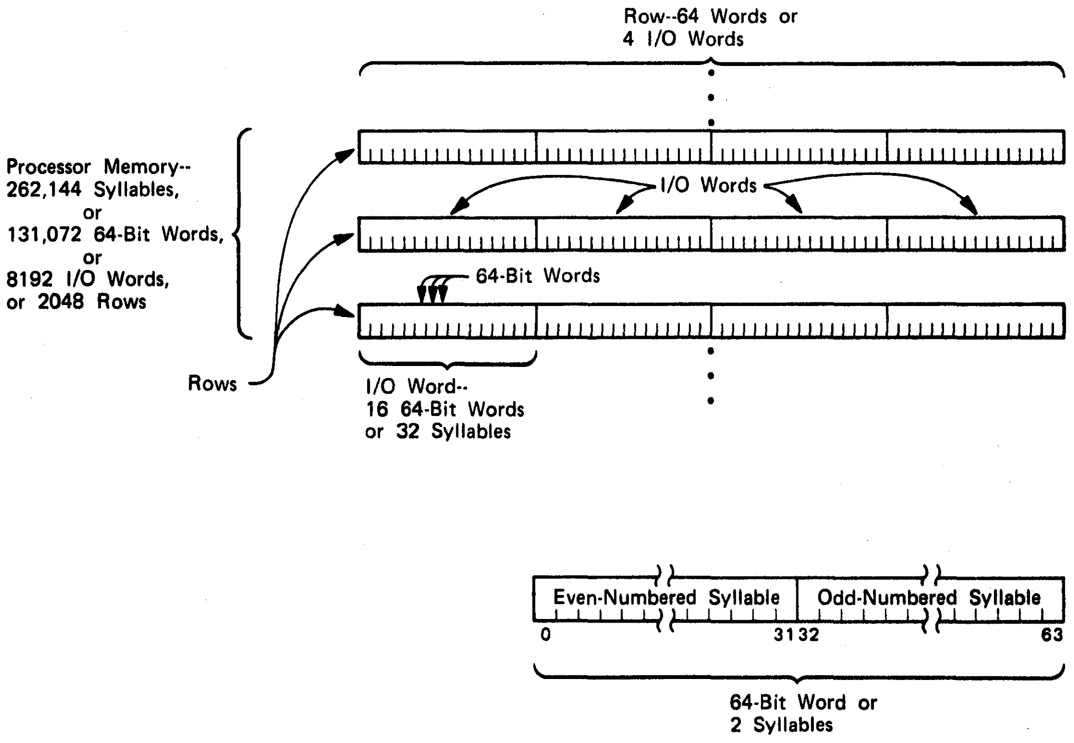


Figure 1-1 Basic Organization of Processor Memory

1.1 PROCESSOR MEMORY ADDRESSING

Processor memory can be addressed by:

(a) syllables,
(b) words,
(c) I/O words,
(d) rows.

Figure 1.1 illustrates the basic organization of processor memory.

1.1.1 SYLLABLE ADDRESSES

The smallest addressable entity in processor memory is the 32-*bit syllable* - not to be confused with the inner or outer portion of a 64-bit word in 32-bit mode.

Processor memory contains $262{,}144_{10}$ or $1{,}000{,}000_8$ contiguous syllables, numbered consecutively from 0 to $777{,}777_8$.

Note: A syllable address consists of 18 bits forming a value from 0 to $777{,}777_8$.

Since instructions are the sole entities stored in syllable format, syllable addresses are used *only* in the instruction counter register (ICR). The ILLIAC IV instruction set does not include any instructions that contain a direct syllable address. For example, SKIP instructions modify the contents of ICR by incrementing or decrementing the syllable address already contained in ICR. The JUMP instruction contains a *word address* in its address field; this word address is loaded into ICR, and in the process it is converted into a syllable address (by right-appending a 0 bit - see the following discussion of word addresses and also p. 1-5). The ILLIAC IV programmer therefore encounters syllable addresses only when dealing with assembler pseudo-instructions and when examining dump files.

1.1.2 WORD ADDRESS

The next type of addressable entity in processor memory hierarchy is the 64-bit word.

Note: Thirty-two bit "inner" and "outer" portions of a 64-bit word are not separately addressable entities.

A 64-bit word occupies the same space as two syllables. Processor memory contains $131{,}072_{10}$ or $400{,}000_8$ contiguous words, numbered consecutively from 0 to $377{,}777_8$.

Note: A word address consists of 17 bits forming a value from 0 to $377{,}777_8$.

Word addresses are used in ADVAST instructions to control CU accessing of processor memory on a word basis (LOAD, LOADX, STORE, and STOREX instructions), to access a block of eight contiguous PM words (BIN and BINX instructions), and to load ICR (JUMP instruction). The word address in a JUMP instruction is modified by appending a 0 bit to the right-hand end (thus converting it to a syllable address) and then loading it into ICR.

If the rightmost bit is dropped from any syllable address, the result is the word address of the processor memory word that contains the syllable.

1.1.3 I/O Word Address

An "I/O word" is a block of 16 contiguous 64-bit words. Only data-transfer processes use I/O word addressing. Hence the ILLIAC IV programmer encounters I/O word addresses only when dealing with assembler pseudo-instructions (specifically, the initialization of processor memory), data-transfers SYSCALLS or ASK data-transfer macros, and the examination of dump files.

Processor memory contains $8{,}192_{10}$ or $20{,}000_8$ I/O words, numbered consecutively from 0 to 17777_8.

Note: An I/O word address consists of 13 bits forming a value from 0 to 17777_8.

If the rightmost four bits are dropped from any word address, the resulting entity is the I/O word address of the I/O word that contains the word.

1.1.4 Row Addresses

A row consists of 64 contiguous 64-bit words.[1] Processor memory contains 2048_{10} rows, numbered consecutively from 0 to 3777_8.

Note: A row address consists of 11 bits forming a value from 0 to 3777_8.

Row addresses are used in FINST/PE instructions, and each row address is broadcast to the 64 PEs along with the microprogram sequence for performing an instruction. To understand how the row address is used by the PEs, it is first necessary to consider how processor memory is accessed by each PE.

Every PE has a position number that corresponds to its physical position in the array of 64 PEs; that is, the position number is a number from 0 to 77_8 (0 to 63_{10}). Each PE has access to one word in each row of processor memory - the word whose position in the row corresponds to the PE's position. This set of 2048 words (one for each row) is sometimes referred to as the processor memory "column" that is associated with the particular PE.

Note: Remember that each PE has access only to the words in its "column" of processor memory.

[1]For data-transfer processes, it may be regarded as four contiguous I/O words.

When the ILLIAC IV processes a FINST/PE instruction that uses a row address, this address is received simultaneously by all the PEs. In the simplest case, each PE will access one word in the row, and the overall effect is to process the complete row of 64 words.

If a PE has its E and E1 bits reset (=0), however, it will not alter processor memory contents. Hence, if one or more PEs have E and E1 bits reset, only some of the words in the row will be processed, leaving other words untouched.

In the most general case, *PE-indexing* may be invoked by the instruction. This means that after receiving the broadcast row address, each PE then modifies it by adding the contents of one of its own registers. Since the indexing register in each PE may have different contents, the set of words processed may not be a row at all. It may, for example, be a diagonal or some other configuration. In the special case where PE-indexing is invoked and all the PEs index by the same quantity, the effect will be to process a row (or a row with gaps) but *not* the same row specified by the broadcast row address.

In summary, therefore, a row address (11 bits) is associated with a FINST/PE instruction and is broadcast by the CU to all 64 PEs. Each PE then uses the row address as a base address to access a single word in its particular column of processor memory. The overall effect, in the simple case, is to process one row of processor memory. In other cases, the set of words processed may not be a row, but the term "row address" is used nevertheless.

1.1.5 Processing of Word Addresses

We will now return to word addresses to learn how the 17 bits of a word address are actually used. Note that if the rightmost six bits are dropped from any word address, we are left with the row address of the row containing the word. Also, the rightmost six bits of the word address are actually the PE position number of the PE whose "column" in processor memory contains the word. In other words, the leftmost eleven bits of a word address give the row containing the word and the rightmost six bits give the position of the word in the row (beginning with 0 for the first word in the row).

The JUMP instruction *converts* a word address to a syllable address by right-appending a 0 bit. The only other instructions that use a word address are LOAD, LOADX, STORE, STOREX, BIN, and BINX. These instructions treat the word address as a row address and a position number.

LOAD, LOADX, STORE AND STOREX INSTRUCTIONS

In these instructions, the CU uses a single PE to access a single word of processor memory. The last six bits of the word address select the PE; the first eleven bits (a row address in effect) are sent to this PE. The PE indexes the

the row address (if the instruction is LOADX or STOREX) and then accesses the corresponding word in its "column" in processor memory.

BIN AND BINX INSTRUCTIONS

In these instructions, the CU uses a set of eight contiguous PEs to access eight words in processor memory and then fetch them to the CU, where they are stored in ADB registers. The first of the eight words must be at an address that is congruent to 0 mod 8; that is, the block must begin at the 0th, 8th, 16th, 24th, 32nd, 40th, 48th, or 56th word[2] in some row.

Again, the last six bits of the word address are used to select one PE, but the last three bits are disregarded and treated as 0s. This adjusts the address to an 8 word boundary. The resulting 6-bit quantity is used to select eight contiguous PEs, starting at the 8-word boundary; the first eleven bits of the word address (a row address in effect) are sent to these PEs. Each PE independently indexes the row address (if the instruction is BINX) and then uses the result to access one word in its "column" in processor memory. Finally, each of the eight PEs sends its word to the CU, and the eight words are stored.

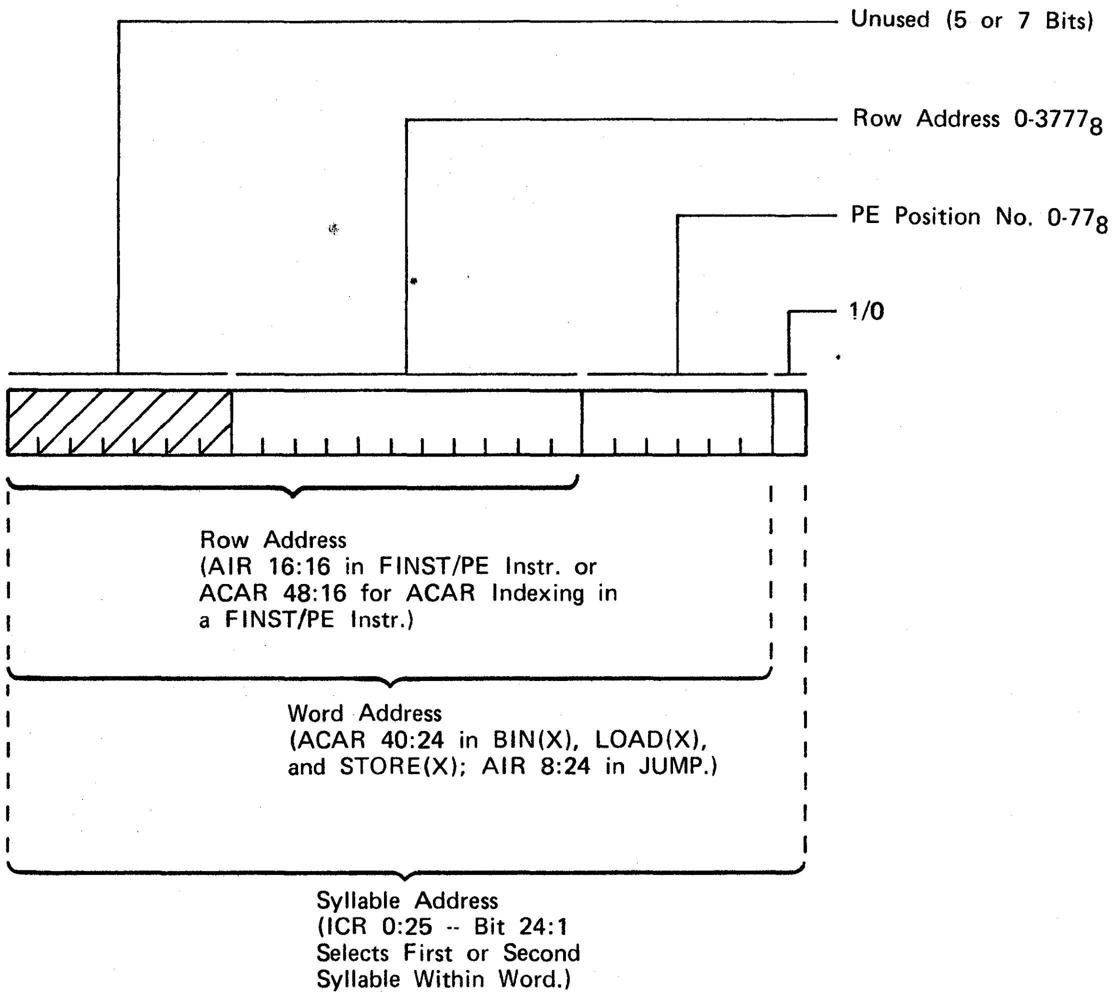
1.1.6 Address Field Formats

Although row addresses consist of 11 bits, the address field (ADR) of a FINST/PE instruction is defined as 16 bits. [Figure 1-2 shows how addresses are formatted within address fields.] Address arithmetic (indexing) operates on all 16 bits, but only the rightmost 11 bits in the field are used for determining a physical address. The five leftmost bits are unused in the present ILLIAC IV configuration.

Similarly, a word address consists of 17 bits, but the address field of an ACAR or the JUMP instruction is 24 bits. Again, all manipulations of the address field operate on the entire 24 bits, but only the rightmost 17 bits are used as an address.

Finally, although ICR is 25 bits long, the syllable address it contains is only 18 bits. When ICR is loaded from an ACAR or processor memory location (or modified by a SKIP or JUMP instruction), all 25 bits may be affected, but only the rightmost 18 bits are used as a syllable address.

[2]These positions are called *8-word boundaries*.



Unused (5 or 7 Bits)
Row Address 0-3777$_8$
PE Position No. 0-77$_8$
1/0
Row Address (AIR 16:16 in FINST/PE Instr. or ACAR 48:16 for ACAR Indexing in a FINST/PE Instr.)
Word Address (ACAR 40:24 in BIN(X), LOAD(X), and STORE(X); AIR 8:24 in JUMP.)
Syllable Address (ICR 0:25 -- Bit 24:1 Selects First or Second Syllable Within Word.)

**Note:* ICR 24:1 corresponds to ACAR/Memory 0:1 in any ADVAST instruction that loads or stores ICR.

Figure 1-2 Address Format and Address Fields

1.1.7 Jumps and Skips

A JUMP instruction, as described previously, loads a word address into ICR, where it is converted to a syllable address by right-appending a 0 bit. The result is always an *even* syllable address! It is impossible to use the JUMP instruction to branch to an instruction having an odd-numbered syllable address. The ASK assembler permits the programmer to force any instruction to be located at a word address (i.e., an even-numbered syllable address).

Certain instructions contain a "skip" field, which is used to modify ICR. This field is added to ICR with the least significant bits aligned. In other words, a syllable address is modified directly to another syllable address, so the program can "skip" to any syllable within the range provided by the 8-bit skip field.

1.2 PE REGISTER ADDRESSING

PE register addresses are specified by setting one bit in a 16-bit field (the ADR field) of a PE instruction word. If more than one of these bits is set, the results are unspecified. The bit assignments are listed in Table 3-1, p. 3-3.

One ADVAST instruction, LDC, uses a PE register address that is specified by setting one bit in the instruction word. [See Section 2, p. 2-36.]

1.3 CU REGISTER ADDRESSING

CU registers are addressed by 8-bit codes in the ADR field of ADVAST instructions. Table 2-1, p. 2-8, lists the codes and provides a brief description of all addressable CU registers mentioned in this manual. There are also other CU registers (not mentioned in this manual) that may be legally addressed by ADVAST instructions. However, the results of addressing these registers are unspecified, and their use is not recommended.

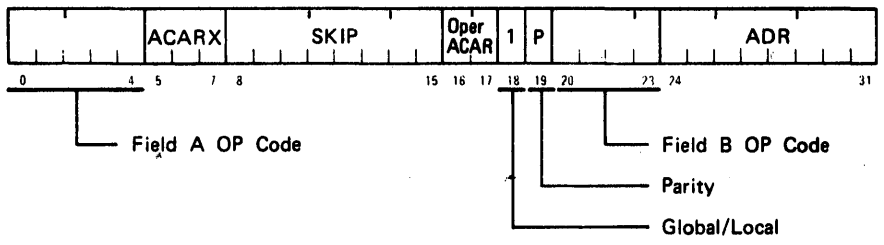
Section 2
ADVAST Instructions

CONTENTS

Section 2
ADVAST Instructions

2.1 INSTRUCTION FORMAT AND FIELD USAGE

This section begins with an illustration of the general format for ADVAST instruction words, followed by a listing of each field and its usage.



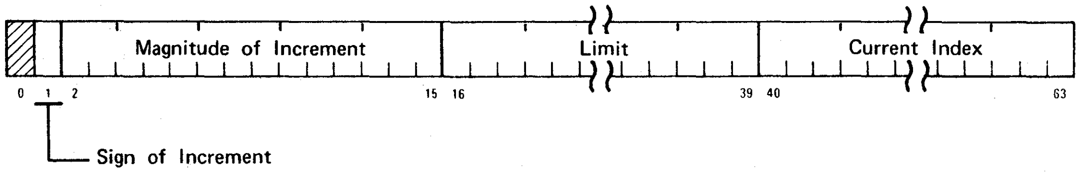
Field	Description
Field A OP Code	Bits 0:2 and 2:3. First and second digits of octal OP code. Bit 0 is "zero" for ADVAST instructions.
ACARX	Bits 5:3. When bit 5 is "one," the contents of the ACAR specified by bits 6 and 7 are used to index the quantity found in the ADR field. When bit 5 is "zero," the ADR field is used without indexing, and the values in bits 6:2 are irrelevant. The SLIT and ALIT instructions utilize this and the remainder of the fields differently. See instruction details for exact field definitions.

SKIP	Bits 8:8. This field is used in Test-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 from ICR, and "zero" means add to ICR. Bits 9:7 specify the magnitude. The JUMP instruction utilizes this and the remainder of the fields differently. See instruction details for exact field definitions.
Operand ACAR	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	Bit 18:1. A "one" indicates "local." Local operation is the only possibility in the present ILLIAC IV configuration.
Parity	Bits 19:1. This is an odd parity bit. ALIT, SLIT, and JUMP instructions do not use the parity bit.
Field B OP Code	Bits 20:1 and 21:3. Second and third digits of the octal OP code.
ADR	Bits 24:8. The particular usage of this field is described separately for each instruction. Generally, it is indexable (see ACARX), and it specifies the CU register 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.

2.2 ADVAST ARITHMETIC

Three different formats are used by various ADVAST instructions in handling the 64 bits in an ACAR or ADB location. The format used depends on the instruction and, in some cases, the specified operand.

Here, we are concerned with cases in which the *index word format* is used. The following diagram illustrates this format.



Bits 1:15, the sign and magnitude of the increment, are called the *Increment Field;* bits 16:24 are called the *Limit Field;* and bits 40:24 are called the *Current Index Field.*

All ADVAST arithmetic is 24-bit, twos-complement integer arithmetic (addition and subtraction). It is both performed in and restricted to the Current Index Field of an ACAR; that is, bits outside the Current Index Field are never affected by ADVAST arithmetic operations. The first operand is always found in the Current Index Field of the operand ACAR, and the result is always left in the same field of that same ACAR. The second operand can be any one of the following entities.

(a) Current Index Field of another ACAR or ADB location.
(b) Most significant 24 bits of ICR (a 25-bit register).
(c) Least significant 24 bits of some other register.
(d) Increment Field of the *same* ACAR that contains the first operand.
(e) A 24-bit literal contained in the instruction word (i.e., the 24 least significant bits of the instruction register, AIR).

ADVAST arithmetic uses unsigned twos-complement operands, except when the second operand is the Increment Field (a sign-magnitude quantity). In this case, twos-complement arithmetic is still used, and a negative result produced by subtraction is left in twos-complement form.

Note: Overflow is disregarded; thus all results are mod 2^{24}.

The Limit Field is used only in certain Test-Skip instructions, where it is compared with the Current Index Field of either the same ACAR or another ACAR. The Limit Field is *never* modified or used as an operand in ADVAST arithmetic instructions.

2.2.1 ACAR-INDEXING

Both ADVAST instructions and FINST/PE instructions may be ACAR-indexed. When an ADVAST instruction is indexed, the eight least significant bits of the instruction word (the ADR field) are modified by adding the eight least insignificant bits of a specified ACAR (ACARX). Overflow is disregarded.

This description of ACAR-indexing applies to *all* ADVAST instructions where an ACARX field is shown in the instruction word layout. Hence the instruction descriptions, presented in a later subsection [see Section 2.5], do not mention ACAR-indexing except where special cautions apply.

2.3 CATEGORIZATION OF ADVAST INSTRUCTIONS

The following subsections are a generalized categorization of ADVAST instructions. Each instruction mnemonic is followed by a page reference in brackets []. These page numbers refer to the detailed descriptions of these instructions, which begin on p.2-11. For a complete alphabetical listing of these instructions, see the inside front cover of this manual.

OPERATIONS ON ACARS

Current Index Field Manipulation

ALIT [2-11]	Add literal to ACAR (40:24)
CADD [2-14]	Add operand to ACAR (40:24)
CSUB [2-27]	Subtract CU register from ACAR (40:24)
SLIT [2-42]	Replace ACAR (40:24) with literal
INCRXC [2-34]	Modify ACAR (40:24) by ACAR (1:15)

Whole Register Manipulation

CAND [2-15]	64-bit "and" of ACAR and CU register
COR [2-20]	64-bit "or" of ACAR and CU register
CEXOR [2-17]	64-bit "exclusive" or ACAR and CU register
CLC [2-18]	Clear ACAR
COMPC [2-19]	Complement ACAR
CROT(L/R) [2-22, 2-23]	Rotate ACAR left/right, end-around
CSH(L/R) [2-25, 2-26]	Shift ACAR left/right, end-off, zero fill
LEAD(O/Z) [2-38]	Find leading one/zero in ACAR
EXCHL [2-30]	Exchange contents of ACAR and CU register
LDL [2-37]	Load ACAR from CU register
LIT [2-39]	Load ACAR with literal
STL [2-43]	Store ACAR contents in CU register

Bit Manipulation

CCB [2-16]	Complement bit in ACAR
CRB [2-21]	Reset (=0) bit in ACAR
CSB [2-24]	Set (=1) bit in ACAR

Half Word Manipulation

DUP(I/O) [2-28, 2-29]	Duplicate inner/outer 32 bits of ADB word in ACAR

REFERENCE TO PROCESSOR MEMORY

BIN(X) [2-12]	Fetch 8 words from processor memory to ADB
LOAD(X) [2-40]	Fetch 1 word from processor memory to CU register
STORE(X) [2-44]	Store CU register in 1 word of processor memory

REFERENCE TO PE INFORMATION

LDC [2-36]	Load ACAR from PE register
SETC [2-41]	Load each ACAR bit with mode bit from a PE

CONTROL

CACRB [2-13]	Set/reset one bit in ADVAST Control Register, ACR
EXEC [2-31]	Execute instruction in ACAR (32:32)
FINQ [2-32]	Stop ADVAST until FINST is idle
HALT [2-33]	Programmed halt; CU comes to orderly idle state
JUMP [2-35]	Jump to specified word address

In the following Test-Skip instructions, the notes apply to the "T" case in conditional skips; that is, skip if test is TRUE. "F" means skip if test is FALSE.

UNCONDITIONAL SKIP

SKIP [2-52]	Skip specified number of syllables forward or backward

SKIP ON CONDITION OF CU TRUE/FALSE FLIP FLOP (TFFF)

SKIP(T/F) [2-53]	Skip on preexisting TFFF value

SKIP ON VALUE OF BIT IN ACAR

CTSB(T/F) [2-46]	Skip if specified ACAR bit is set

ZEROS AND ONES

ONES(T/F) [2-50]	Skip if ACAR (0:64) = all "ones"
ONEX(T/F) [2-51]	Skip if ACAR (40:24) = all "ones"
ZER(T/F) [2-60]	Skip if ACAR (0:64) = all "zeros"
ZERX(T/F) [2-61]	Skip if ACAR (40:24) = all "zeros"

COMPARE ACAR CURRENT INDEX FIELD TO OPERAND CURRENT INDEX FIELD

EQLX(T/F) [2-47]	Skip if ACAR (40:24) = CU register (40:24)
GRTR(T/F) [2-48]	Skip if ACAR (40:24) > CU register (40:24)
LESS(T/F) [2-49]	Skip if ACAR (40:24) < CU register (40:24)

COMPARE ACAR CURRENT INDEX FIELD TO OPERAND LIMIT FIELD

TXE(T/F) [2-54]	Skip if ACAR (40:24) = CU register (16:24)
TXG(T/F) [2-56]	Skip if ACAR (40:24) > CU register (16:24)
TXL(T/F) [2-58]	Skip if ACAR (40:24) < CU register (16:24)

COMPARE ACAR CURRENT INDEX FIELD TO LIMIT FIELD OF SAME ACAR AND MODIFY CURRENT INDEX FIELD BY INCREMENT FIELD OF SAME ACAR

Note: The skip is conditional, but the Current Index Field modification is unconditional.

TXE(T/F)M [2-55]	Skip if ACAR (40:24) = ACAR (16:24); modify ACAR (40:24) by ACAR (1:15)
TXG(T/F)M [2-57]	Skip if ACAR (40:24) > ACAR (16:24); modify ACAR (40:24) by ACAR (1:15)
TXL(T/F)M [2-59]	Skip if ACAR (40:24) < ACAR (16:24); modify ACAR (40:24) by ACAR (1:15)

2.4 REGISTER ALIGNMENT IN ADVAST INSTRUCTIONS

As indicated in Table 2.1, the addressable CU registers range from 8 to 64 bits in length. The following rules describe the way in which two registers are aligned with each other in transfers and in arithmetic operations.

2.4.1 Rules For All Registers Except ICR

- The least significant bits of the two registers are aligned.
- If the destination register in a transfer operation is longer than the source register, all destination register bits not replaced by bits from the source register are *cleared*.
- If the source register in a transfer operation is longer than the destination register, the excess source register bits are disregarded.

2.4.2 Rules For ICR

ICR is a 25-bit register that is used to hold a syllable address. The most significant 24 bits of ICR contain a word address; the least significant bit is used to select the first or second syllable of the addressed word (i.e., "0" for the first syllable or "1" for the second).

An ADVAST instruction may cause ICR to be aligned with an ACAR or with a processor memory location - no other combinations are possible. The following rules describe the alignment.

- The 24 *most significant* bits of ICR are aligned with the 24 *least significant* bits of the ACAR or processor memory location. In other words, the word address portion of the ICR aligns with the Current Index Field of an ACAR.

Table 2-1 Summary of Addressable CU Registers*

Register	Description
ACR (ADVAST Control Register)	A 16-bit register containing 1-bit indicators and control bits for various states of the CU. ACR can be read by LDL and STORE(X). It cannot be written, but certain bits can be set/reset by CACRB. *Address Code* = 140_8.
ADB (ADVAST Data Buffer)	An array of sixty-four, 64-bit storage registers with general read/write access by ADVAST instructions. ADB access requires only two clocks, while processor memory access requires seven clocks. *Address Codes* = $000 - 077_8$.
ACARs (Accumulator Registers)	An array of four 64-bit, general-purpose accumulator registers. *Address Codes* = $100_8 - 103_8$.
ICR (Instruction Counter Register)	A 25-bit register containing the syllable address of the instruction currently being processed by ADVAST. ICR can be addressed by CADD, CSUB, EXCHL, LDL, STL, LOAD(X), and STORE(X). See Section 2.4.2 for a description of bit alignment between ICR and other operand registers. *Address Code* = 104_8.
AIN (ADVAST Interrupt Register)	A 16-bit register containing 1-bit flags set by the hardware to indicate the occurrence of error conditions. AIN is masked by AMR; if the hardware sets a bit in AIN and the corresponding bit of AMR is set, a HALT occurs. AIN can be addressed by EXCHL, LDL, STL, LOAD(X), and STORE(X). AIN is NOT automatically cleared each time it is read. *Address Code* = 142_8.
AMR (ADVAST Mask Register)	A 16-bit register used to mask AIN, as described above. AMR can be addressed by EXCHL, LDL, STL, LOAD(X), and STORE(X). *Address Code* = 145_8.
ALR (ADVAST Local Memory Address Register)	An 8-bit register used to store the address code of a CU register addressed in an ADVAST instruction. ALR can be addressed by EXCHL, LDL, STL, LOAD(X), and STORE(X). *Address Code* = 144_8.
TRI (TMU Input Register)	A 64-bit register used to hold information sent to the ILLIAC IV by the operating system. TRI can be addressed (read) only by LDL and STORE(X). *Address Code* = 155_8.
TRO (TMU Output Register)	A 64-bit register used to send information from the ILLIAC IV to the operating system. TRO can be addressed by EXCHL, LDL, STL, LOAD(X), and STORE(X). When TRO is written by LOAD(X), ACR bit 15 is automatically set. *Address Code* = 156_8.

*This table contains *only* those registers that are referenced in this manual.

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

Bit No.	Initial Value	Can Be Set/Reset via CACRB	Description
0	OFF	YES/YES	*CU TRUE/FALSE Flip-Flop:* Used in Test-Skip instructions. Set for TRUE and reset for FALSE.
1	OFF	NO/NO	*Interrupt Processing Mode:* When set, CU is in interrupt processing mode. Set whenever ACR bit 11 is reset, some bit in AIN is set, and the corresponding bit in AMR is also set. Reset by INR.
2	OFF	NO/YES	*Hardware Mask in Use:* When set, all AIN bits except 0-3 are ignored; AMR is also ignored. Set when ACR bit 1 is set. Reset by INR.
3	OFF	NO/NO	*ALR Busy:* Set by a BIN or LOAD instruction that references an ADB location. ACR bit 7 indicates whether BIN or LOAD caused ALR BUSY condition. Reset automatically when the BIN or LOAD completes.
4	OFF	YES/YES	*Alternate Interrupt Base in Use.*
5	OFF	YES/YES	*Quadrants Awaiting Sync.:* Meaningless in one-quadrant ILLIAC IV.
6	OFF	NO/NO	*FINST Idle:* Set whenever FINST is idle.
7	OFF	NO/NO	*BIN/LOAD Indicator:* When set, it indicates that last operation to set ACR bit 3 was a BIN. When reset, the operation was a LOAD.
8	ON	YES/YES	*Nonoverlap Mode:* When set, ILLIAC IV operates in nonoverlap mode. When reset, overlap mode is used.
9	ON	YES/YES	*Exponent Underflow Inhibit:* When set, exponent underflow condition in a PE will not cause F or F1 bit to be set.
10	OFF	YES/YES	*32-Bit Mode:* When reset, machine is in 64-bit mode.
11	ON	YES/YES	*Halt on Interrupt Condition:* When set, a halt occurs instead of an interrupt.
12	OFF	YES/YES	*Partial Overlap:*
13	OFF	YES/YES	*Spare*
14	OFF	YES/YES	*Branch Trace Enable:* When set, alteration of ICR sets AIN bit 14 and loads old contents of ICR into TRO(40:24).
15	OFF	YES/YES	*TRO Loaded:* Set automatically by LOAD or LOADX instruction that loads TRO. Reset automatically when TRO has been read by the operating system.

Table 2-3 Meanings of Bits in Interrupt (AIN) and Mask (AMR) Registers

Bit No.	Initial Value in AMR	Type *	Conditions
0	ON	2	*Spare:* Available for indicating power failure.
1	ON	2	*Parity Error in Instruction:* Sum of bits loaded from IWS to AIR is even. Does not apply to SLIT, ALIT, JUMP, or any instruction loaded by an EXEC instruction.
2	ON	2	*Undefined OP Code:* OP code not a member of total ILLIAC IV instruction set.
3	ON	2	*CU Stalled:* CU has waited 15 ms for another instruction, or HALT was executed, or breakpoint was reached, or second-level interrupt has halted all operations.
4	ON	2	*Improper Setting of MC0, MC1, MC2 Registers:* Configuration control registers, invisible in one-quadrant ILLIAC IV.
5	ON	2	*Improper CU Register Address:* Nonexistent or inaccessible CU register address in ADVAST instruction. Not applicable to BIN or BINX.
6	ON	2	*ADB Wraparound:* Effective ADB address greater than 77_8 in BIN or BINX.
7	ON	2	*Execute Loop:* AIR contents replaced by ACAR value having an identical OP code and ACAR address. This can occur with an EXEC instruction.
8	ON	2	*Skip Loop:* Skip instruction encountered with skip distance equal to -1.
9	ON	1	*User Program Requested Interrupt:* INR instruction executed, while ACR bit 1 is reset. (CU not in interrupt processing mode.)
10	ON	2	*Spare*
11	ON	2	*Fault Bit Set:* F bit is set in any PE; or ACR bit 10 is set (machine is in 32-bit mode) and F1 bit is set in any PE.
12	ON		*Spare*
13	ON		*Spare*
14	ON	1	*Branch Trace:* ACR bit 14 is set (branch trace enable), and ICR has been altered by EXCHL, STL, LOAD, LOADX, JUMP, or any SKIP instruction.
15	OFF	1	*TRI Loaded:* TRI has been loaded by the operating system.

*"Type 1" conditions cause a halt or interrupt after completion of the current instruction. "Type 2" conditions cause a halt or interrupt on the next clock after the condition is detected.

- The *least significant* bit of ICR is not used in arithmetic operations. In transfers, this bit is aligned with the *most significant* bit of the ACAR or processor memory location.
- Bits 1:39 of the ACAR or processor memory location are not aligned with any ICR bits. If the ACAR or processor memory location is the destination of a transfer, these bits are *cleared*. If the ACAR or processor memory location is the source of the transfer, these bits are disregarded.

Figure 2-1 illustrates these rules.

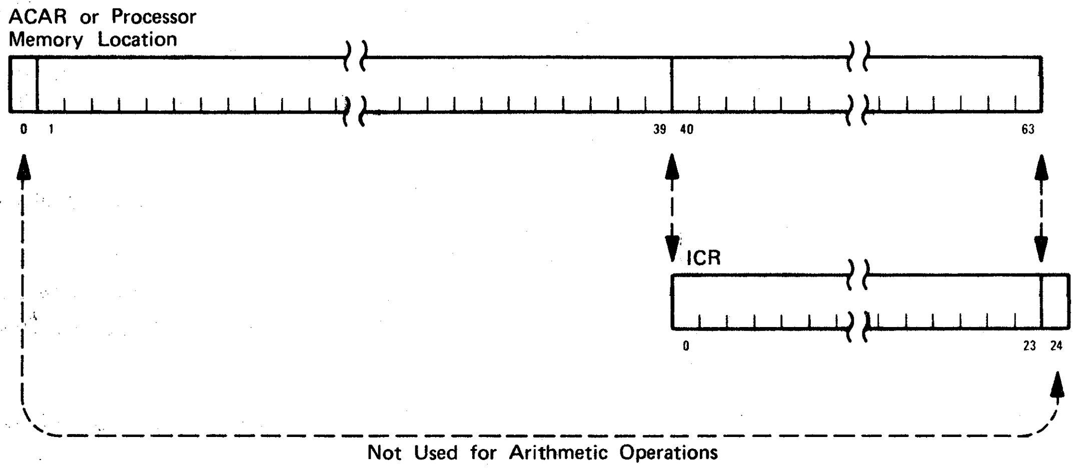

ACAR or Processor Memory Location
0
1
39
40
63
ICR
0
23
24
Not Used for Arithmetic Operations

Figure 2-1 Bit Alignment in Transfers To or From ICR

2.5 ADVAST INSTRUCTION DESCRIPTIONS

Throughout this section the ADVAST instructions are described in detail. Each description includes the mnemonic code, specification of operands (where applicable), a description of the operation of the instruction, and the bit layout of the instruction word. Any fields for which a particular instruction has specific use are described. Shading is used to indicate those fields that are ignored for a particular instruction.

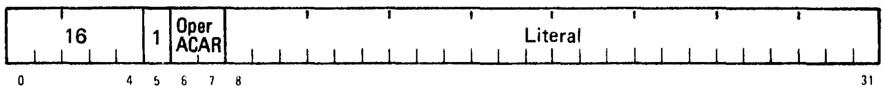
Mnemonic Code: ALIT

First Operand: Current Index Field, bits 40:24 of operand ACAR.

Second Operand: 24-bit literal found in bits 8:24 of the instruction.

Operation: 24-bit, unsigned, twos-complement add; overflow disregarded; result replaces first operand in operand ACAR (bits 40:24).

Instruction Word:



Any overflow from the addition is disregarded, so bits 0:40 of the operand ACAR are unaltered.

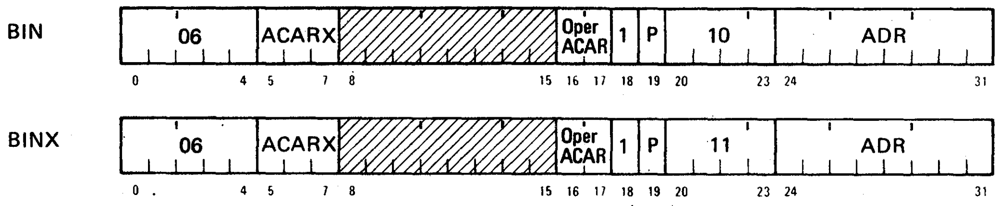
Also bit 5 of the instruction must be a "1", while bits 6 and 7 designate the operand ACAR.[1]

[1]This is an exception to the normal instruction format.

Mnemonic Codes: BIN, BINX

Operation: Block fetch from processor memory to ADB.

Instruction Words:



The operand ACAR is assumed to contain a word address. To select a block of eight contiguous PE positions,[2] this address is treated as if its three least significant bits were zero. Each PE in the selected block receives the row address portion of the word address (modifying it by RGX if the instruction is BINX) and gets a word from its column of processor memory. The resulting set of eight words from processor memory is sent to the CU.

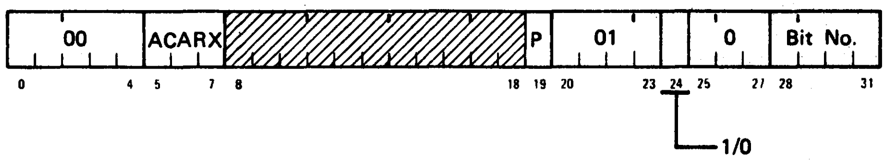
The ADR field of the instruction (which may be ACAR-indexed) specifies an address in ADB. To select a block of 8 contiguous ADB locations (starting at ADB0, 10, 20, 30, 40, 50, 60, or 70 octal), this ADB address is treated as if its three least significant bits were zero. The eight words received from processor memory are stored in these eight ADB locations.

[2]The first of these eight positions will be octal PE position 0, 10, 20, 30, 40, 50, 60, or 70.

Mnemonic Code: CACRB

Operation: Set/reset specified bit in ADVAST Control Register (ACR).

Instruction Word:



This instruction changes a bit in the ADVAST Control Register (ACR). The bit number is specified in bits 28:4 of the instruction and may be ACAR-indexed. Bit 24 of the instruction contains a "1" if the ACR bit is to be set or a "0" if the ACR bit is to be reset.

ACR bits 1, 3, 6, and 7 cannot be changed by CACRB. If the instruction specifies one of these bits, the instruction is a no-op. ACR bit 2 can be reset (=0) but not set (=1) by CACRB. If the instruction specifies setting ACR bit 2, it is a no-op.

Caution: Bits 24:8 of the instruction may be modified by ACAR-indexing, which could result in an alteration of bit 24, thus changing the meaning of the instruction.

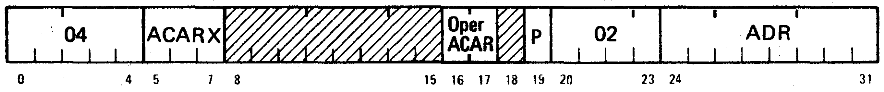
Mnemonic Code: CADD

First Operand: Current Index Field, bits 40:24 of operand ACAR.

Second Operand: 24-bit field from a CU register specified in ADR field; limited to ADB, ACARs, IIA and ICR. [See below.]

Operation: 24-bit, unsigned, twos-complement add; overflow disregarded; result replaces first operand in operand ACAR (bits 40:24).

Instruction Word:



If the ADR field designates an ADB location or an ACAR, the second operand is the Current Index Field (bits 40:24) of the ADB location or ACAR; that is, the 24-bit add is performed with the least significant bits aligned.

If the ADR field designates ICR or IIA, the second operand is bits 0:24 of ICR or IIA; that is, the 24-bit add is performed with the second least significant bit of ICR or IIA aligned with the least significant bit of the operand ACAR. The least significant bit of ICR or IIA is not used.

Any carry from the addition is disregarded, and bits 0:40 of the operand ACAR are unaltered.

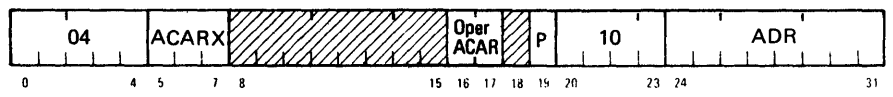
Mnemonic Code: CAND

First Operand: Operand ACAR(0:64).

Second Operand: ADB location or ACAR(0:64) specified in ADR field.

Operation: 64-bit logical AND; result replaces first operand in operand ACAR (bits 0:64).

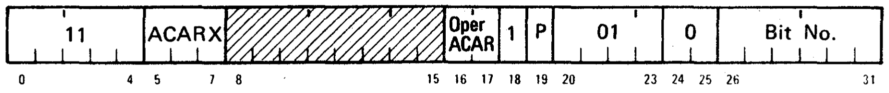
Instruction Word:



Mnemonic Code: CCB

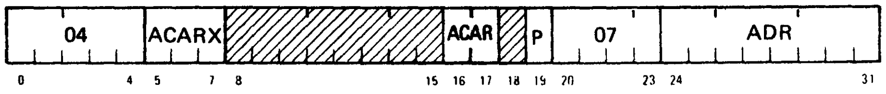
Operation: Complement specified bit in operand ACAR.

Instruction Word:



This instruction complements one bit in the operand ACAR. The bit number is specified in bits 26:6 of the instruction and may be ACAR-indexed.

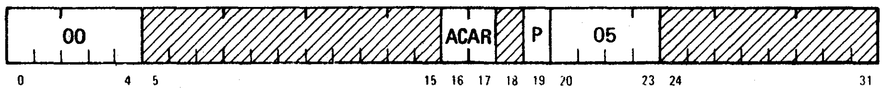
Mnemonic Code:	CEXOR
First Operand:	Operand ACAR(0:64).
Second Operand:	ADB location or ACAR(0:64) specified in ADR field.
Operation:	64-bit logical EXCLUSIVE-OR; result replaces first operand in operand ACAR (bits 0:64).
Instruction Word:	



Mnemonic Code: CLC

Operation: Clear operand ACAR.

Instruction Word:

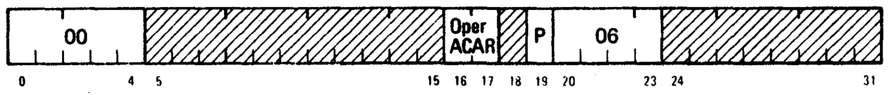


This instruction causes CU to reset the operand ACAR to all zeros.

Mnemonic Code: COMPC

Operation: Complement operand ACAR.

Instruction Word:



This instruction causes each bit of the operand ACAR to be replaced by its complement.

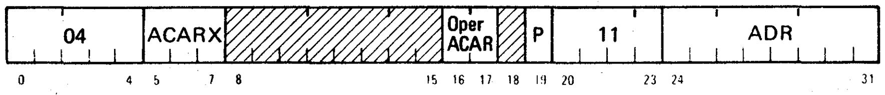
Mnemonic Code: COR

First Operand: Operand ACAR(0:64).

Second Operand: ADB location or ACAR specified in ADR field.

Operation: 64-bit logical OR; result replaces first operand in operand ACAR (bits 0:64).

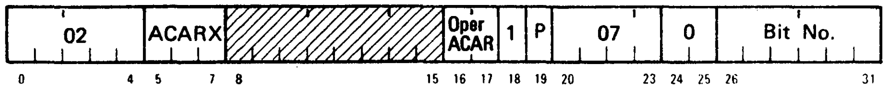
Instruction Word:



Mnemonic Code: CRB

Operation: Reset specified bit in operand ACAR.

Instruction Word:

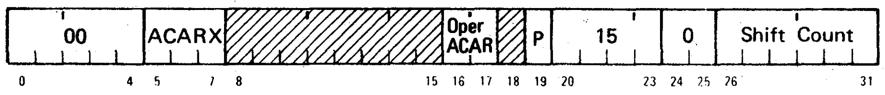


This instruction resets one bit in the operand ACAR. The bit number is specified in bits 26:6 of the instruction and may be ACAR-indexed.

Mnemonic Code: CROTL

Operation: Rotate operand ACAR left (end-around).

Instruction Word:

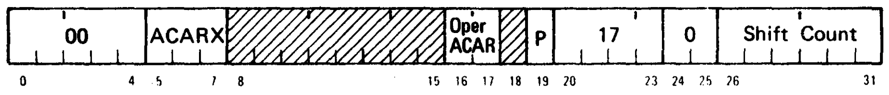


Bits 26:6 of the instruction (which may be ACAR-indexed) specify a shift distance. The operand ACAR is shifted left (end-around) by the specified number of bit positions.

Mnemonic Code: CROTR

Operation: Rotate operand ACAR right (end-around).

Instruction Word:

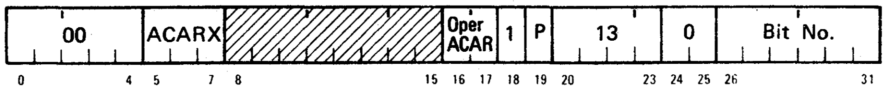


Bits 26:6 of the instruction (which may be ACAR-indexed) specify a shift distance. The operand ACAR is shifted right (end-around) by the specified number of bit positions.

Mnemonic Code: CSB

Operation: Set specified bit in operand ACAR.

Instruction Word:

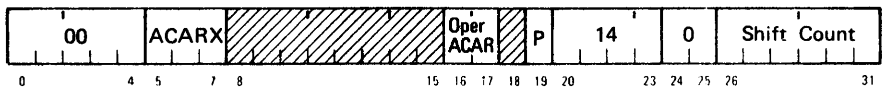


This instruction sets one bit in the operand ACAR. The bit number is specified in bits 26:6 of the instruction and may be ACAR-indexed.

Mnemonic Code: CSHL

Operation: Shift operand ACAR left (end-off).

Instruction Word:

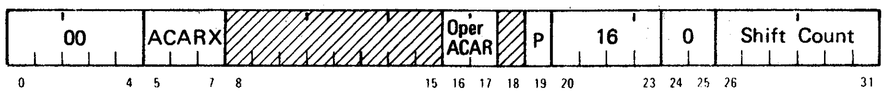


Bits 26:6 of the instruction (which may be ACAR-indexed) specify a shift distance. The operand ACAR is shifted left (end-off) by the specified number of bit positions. The vacated bit positions at the right end of the ACAR are filled with zeros.

Mnemonic Code: CSHR

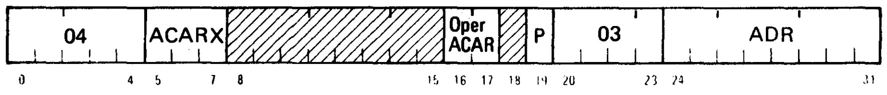
Operation: Shift operand ACAR right (end-off).

Instruction Word:



Bits 26:6 of the instruction (which may be ACAR-indexed) specify a shift distance. The operand ACAR is shifted right (end-off) by the specified number of bit positions. The vacated bit positions at the left end of the ACAR are filled with zeros.

Mnemonic Code:	CSUB
First Operand:	Current Index Field, bits 40:24 of operand ACAR.
Second Operand:	24-bit field from a CU register specified in ADR field; limited to ADB, ACARs, IIA, and ICR. [See below.]
Operation:	24-bit, unsigned, twos-complement subtraction of second operand from first operand. Result replaces first operand in operand ACAR (bits 40:24).
Instruction Word:	



If the ADR field designates an ADB location or an ACAR, the second operand is the Current Index Field (bits 40:24) of the ADB location of ACAR; that is, the 24-bit subtraction is performed with least significant bits aligned.

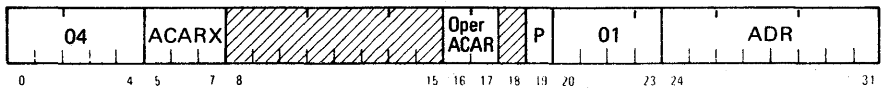
If the ADR field designates ICR or IIA, the second operand is bits 0:24 of ICR or IIA; that is, the 24-bit subtraction is performed with the second least significant bit of ICR or IIA aligned with the least significant bit of the ACAR specified in bits 16:2 of the instruction. The least significant bit of ICR or IIA is not used.

Bits 0:40 of the operand ACAR are unaltered.

Mnemonic Code: DUPI

Operation: Duplicate inner 32 bits of ADB word into operand ACAR.

Instruction Word:



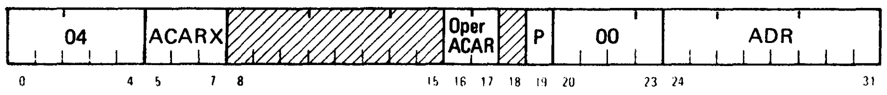
The ADR field of the instruction (which may be ACAR-indexed) contains a CU register address that is restricted to ADB. The instruction causes the CU to duplicate the inner 32 bits of the word found in ADB into both inner and outer portions of the operand ACAR, with the following bit alignment.

ADB Location	Operand ACAR
8:8	0:8 and 8:8
16:24	16:24 and 40:24

Mnemonic Code: DUPO

Operation: Duplicate outer 32 bits of ADB word into operand ACAR.

Instruction Word:



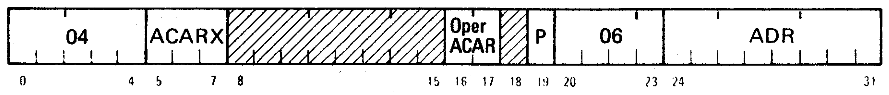
The ADR field of the instruction (which may be ACAR-indexed) contains a CU register address that is restricted to ADB. The instruction causes the CU to duplicate the outer 32 bits of the word found in ADB into both inner and outer portions of the operand ACAR, with the following bit alignment.

ADB Location	Operand ACAR
0:8	0:8 and 8:8
40:24	16:24 and 40:24

Mnemonic Code: EXCHL

Operation: Exchange the specified CU register and operand ACAR.

Instruction Word:



The ADR field of the instruction (which may be ACAR-indexed) specifies a CU register address that should be one of the following: an ADB location, an ACR, AIN, ALR, ACARs, AMR, ICR, MCO-2, IIA and TRO.

The contents of the specified register are interchanged with the contents of the operand ACAR.

If the specified register is ICR, the incrementing of ICR after completion of the instruction is inhibited, thus effecting a jump. The bit alignment is: (a) the second least significant bit of ICR or IIA is aligned with the least significant bit of the operand ACAR; and (b) the least significant bit of ICR or IIA is interchanged with the most significant bit of the operand ACAR.

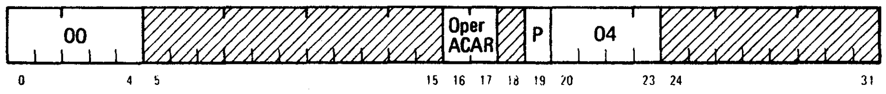
In all cases, any operand ACAR bits that are not replaced by corresponding bits in the specified register are cleared.

Loading MCO or MC1 clears the IWS presence bits. Loading MCO or MC2 causes the FINST Queue to empty before the interchange is performed.

Mnemonic Code: EXEC

Operation: Execute instruction found in operand ACAR (bits 32:32).

Instruction Word:



Bits 32:32 of the operand ACAR are assumed to be an instruction. These bits are transferred to AIR and are executed as the next instruction following the EXEC instruction.

The normal incrementing of ICR and the fetching of the next instruction from IWS are inhibited during execution of EXEC - but not inhibited for the instruction placed in AIR by EXEC. Thus normal operation resumes automatically after this instruction is completed.

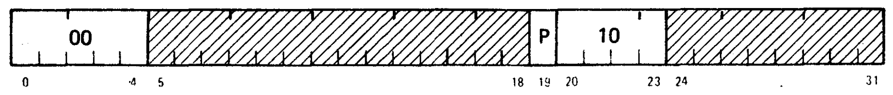
No parity check is performed on the instruction loaded into AIR by EXEC.

If the instruction found in the operand ACAR by EXEC is another EXEC instruction specifying the same operand ACAR, the original EXEC instruction is not executed and AIN bit 7 is set. This avoids the infinite loop that would result if the EXEC were executed.

Mnemonic Code: FINQ

Operation: Stop ADVAST until FINST is idle.

Instruction Word:

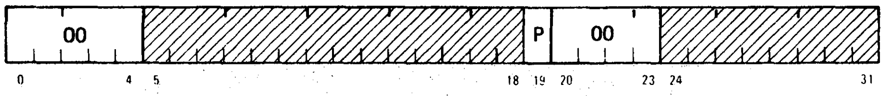


This instruction **causes** ADVAST to stall until FINQ empties and FINST becomes idle. As soon as FINST is idle, ADVAST resumes normal operation.

Mnemonic Code: HALT

Operation: CU comes to an orderly idle state.

Instruction Word:



This instruction causes ADVAST to cease fetching instructions into AIR. FINST continues to operate normally until FINQ is empty; then it becomes idle.

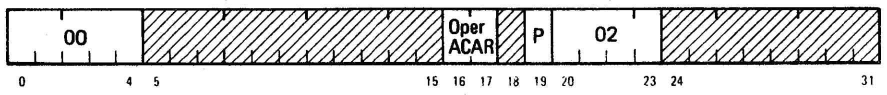
All pending processor memory fetches and data transfer operations being carried out by the operating system will be completed. Communication between the CU and the operating system remains open.

A signal is sent to the operating system to indicate that a HALT has been executed. (AIN bit 3 is set.)

Mnemonic Code: INCRXC

Operation: Modify Current Index Field of operand ACAR by Increment Field of *same* ACAR.

Instruction Word:

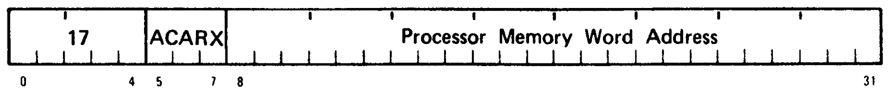


Bits 40:24 of the operand ACAR are modified by the Increment Field (bits 1:15). Bit 1 is the sign of the increment ("0" for positive or "1" for negative). Bits 2:14 are added to or subtracted from bits 40:24 with least significant bits aligned. Any overflow is disregarded, and a negative result will be in twos-complement form. Bits 0:40 of the operand ACAR are unaltered.

Mnemonic Code: JUMP

Operation: Jump to specified word address.

Instruction Word:



Bits 8:24 of the instruction are assumed to contain a word address (right-justified). Bits 24:8 - the eight least significant bits of the address - may be modified (mod 256) by ACAR-indexing.

The word address is loaded into bits 0:24 of ICR, and ICR bit 24 is reset (=0). This results in an even 25-bit syllable address; that is, the first syllable in the processor memory word referenced by the word address. Normal incrementing of ICR is inhibited. The effect is a jump to the instruction in this syllable location.

It is not possible to jump to an odd-numbered syllable.[3]

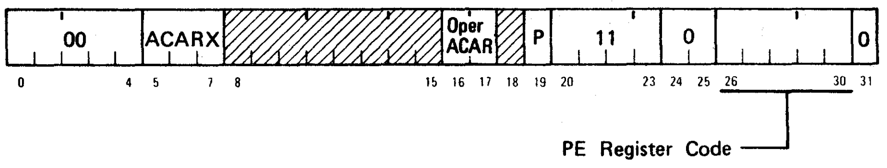
The ASK assembler provides means for forcing instructions to be located at word addresses (even-numbered syllable addresses).

[3]Instructions in the Test-Skip group may be used to transfer to any odd or even syllable location within 128 syllables of the current syllable address in ICR.

Mnemonic Code: LDC

Operation: Transfer specified PE register to operand ACAR.

Instruction Word:



Bits 26:5 of the instruction (which may be ACAR-indexed) specify a PE register code, which is restricted to the following:

PE Register Code in Bits 26:5	PE Register
10000	RGA
01000	RGB
00100	RGX
00010	RGS
00001	RGR

Each "enabled" PE (see below) sends the contents of the specified register to the CU. All of the values are ORed together, and the result is placed in the operand ACAR.

The RGX register in each PE is 16 bits long. If RGX is specified in bits 26:5 of the instruction, bits 0:48 of the operand ACAR will be cleared, and operand ACAR bits 48:16 will contain the OR of the RGX registers.

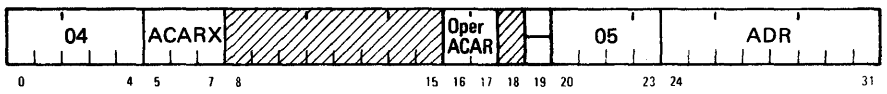
The LDC instruction is affected in a special way by RGD bits E and E1 in each PE [see Section 3.3.3]. In 64-bit mode, only those PEs that have both E and E1 set (=1) will transmit information to the CU.

This is the only ADVAST instruction affected by the E and E1 bits. These bits protect the transfer path itself - unlike FINST/PE instructions, where the E and E1 bits protect only certain registers and processor memory. Thus in the LDC instruction, the E and E1 bits affect the execution regardless of which PE register is addressed.

Mnemonic Code: LDL

Operation: Load operand ACAR from specified CU register.

Instruction Word:



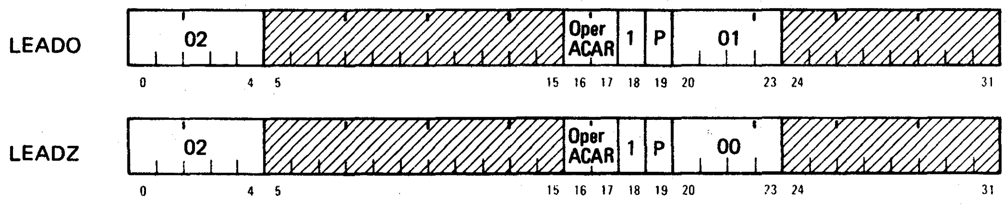
The ADR field of the instruction (which may be ACAR-indexed) specifies a CU register address that should be one of the following: an ADB location, an ACAR, ACR, AIN, AMR, ALR, ICR, MCO-2, IIA, TRO, TRI, ACU and PEM(ARE).

The contents of the specified register are loaded into the operand ACAR. Least significant bits are aligned, except when the specified register is ICR or IIA. The second least significant bit of ICR or IIA is aligned with the least significant bit of the operand ACAR, and the least significant bit of ICR or IIA is transferred to the most significant bit of the operand ACAR. Any operand ACAR bits not replaced by bits from the specified register are cleared.

Mnemonic Codes: LEADO, LEADZ

Operation: Find leading one or zero in operand ACAR.

Instruction Words:



This instruction causes the leftmost "one" (LEADO) or "zero" (LEADZ) in the operand ACAR to be found. The information is returned in the same ACAR in the following manner.

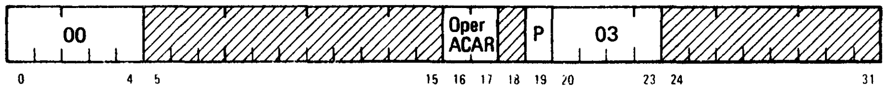
(a) If no leading "one" or "zero" is found, the operand ACAR will be cleared.

(b) If a leading "one" or "zero" is found, operand ACAR bit 55 will be set (=1) and the bit position number of the leading "one" or "zero" is placed in operand ACAR bits 58:6. All other bits in the operand ACAR are cleared.

Mnemonic Code: LIT

Operation: Store next 64 bits in operand ACAR.

Instruction Word:



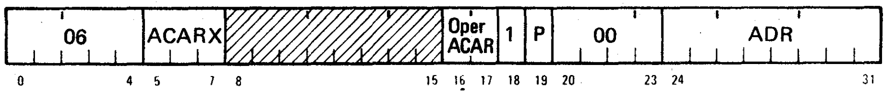
This instruction stores a 64-bit literal into the operand ACAR. The literal is found in IWS in the two syllables immediately following the LIT instruction.

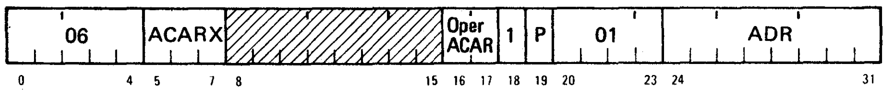
ICR is incremented by 3 instead of 1, to cause the next instruction to be taken from the location following the literal.

Mnemonic Codes: LOAD, LOADX

Operation: Fetch word from processor memory to specified CU register.

Instruction Words:

LOAD


LOADX


The ADR field of the instruction (which may be ACAR-indexed) specifies a CU register address that should be one of the following: an ADB location, an ACAR, AIN, AMR, ALR, ICR, MCO-2, IIA and TRO.

Bits 40:24 of the operand ACAR are assumed to contain a word address. The row address portion of this address is sent to the appropriate PE [see Section 1.1.2], where it is modified by RGX if the instruction if LOADX. The word is then fetched to the CU and stored in the specified CU register.

Least significant bits are aligned, except when the selected CU register is ICR or IIA. The second least significant bit of ICR or IIA is aligned with the least significant bit of the processor memory word. The least significant bit of ICR or IIA is loaded from the most significant bit of the processor memory word.

When the specified CU register is TRO, ACR bit 15 is automatically set (=1) after the transfer is completed.

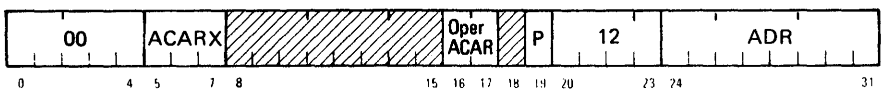
If ICR is modified by this instruction, normal incrementing of ICR is inhibited; hence the next instruction fetched will be the one pointed to by the modified contents of ICR.

When ADR references MCO or MC1, the presence indicators are cleared upon completion of the load.

Mnemonic Code: SETC

Operation: Copy specified MODE (RGD) bit from each PE to operand ACAR.

Instruction Word:



This instruction causes each bit in the operand ACAR to be loaded with a particular MODE (RGD) bit from the corresponding PE; that is, operand ACAR bit 0 gets a MODE (RGD) bit from PE 0, and so forth.

Bits 24:8 of the instruction (which may be ACAR-indexed) contain the following code for selecting a MODE (RGD) bit.

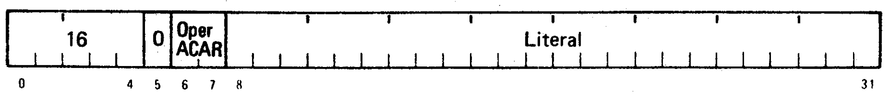
Instruction Bits 24:8 (After Indexing)	RGD Bit in Each PE
10000000	H
01000000	G
00100000	J
00010000	I
00001000	E1
00000100	E
00000010	F1
00000001	F
00000000	F "OR" F1

If more than one bit in bits 24:8 is set (=1), the results are unspecified.

Mnemonic Code: SLIT

Operation: Replace Current Index Field (bits 40:24) of operand ACAR with a literal.

Instruction Word:



Bits 8:24 of this instruction contain a literal quantity , which is stored in bits 40:24 of the specified ACAR. Bits 0:40 of the operand ACAR are not changed.

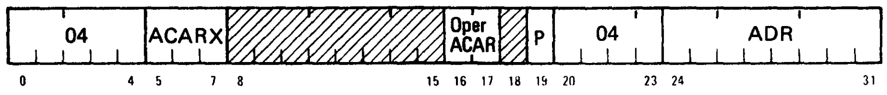
Bit 5 of this instruction must be 0, while bits 6:2 specify the operand ACAR.[4]

[4]This is an exception to the normal instruction format.

Mnemonic Code: STL

Operation: Store operand ACAR in specified CU register.

Instruction Word:



The ADR field of the instruction (which may be ACAR-indexed) specifies a CU register address that should be one of the following: an ADB location, an ACAR, AIN, AMR, ALR, ICR, MCO-2, IIA and TRO. The contents of the operand ACAR are stored in the specified register.

Least significant bits are aligned, except when the specified register is ICR or IIA. The second least significant bit of ICR or IIA is aligned with the least significant bit of the operand ACAR. The most significant bit of the operand ACAR is transferred to the least significant bit of ICR or IIA.

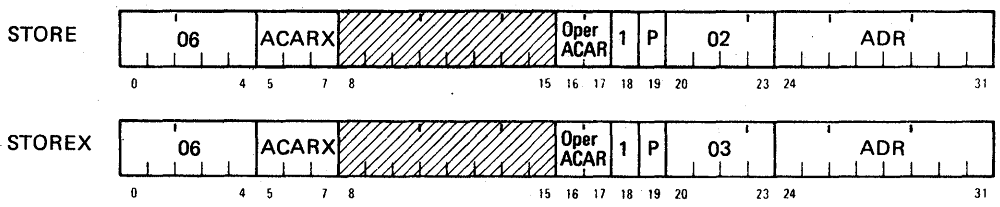
If this instruction is used to change the contents of ICR, the normal incrementing of ICR is inhibited; so the next instruction executed will be the one addressed by the value transferred into ICR by the STL instruction.

Storing into MCO or MC1 clears the IWS presence indicators. Storing into MC2 causes the FINST Queue to empty before the storing takes place.

Mnemonic Codes: STORE, STOREX

Operation: Store from specified CU register into specified processor memory location.

Instruction Words:



The ADR field of the instruction (which may be ACAR-indexed) specifies a CU register address that should be one of the following: an ADB location, an ACAR, ACR, AIN, AMR, ALR, ICR, IIA, MCO-2, TRI, PEM(ARE) and TRO.

Bits 40:24 of the operand ACAR are assumed to contain a word address. The row address portion of this word address is sent to the appropriate PE [see Section 3.1.2], where it is modified by RGX if the instruction is STOREX. The contents of the selected CU register are stored in the selected processor memory location.

Least significant bits are aligned except when the selected CU register is ICR or IIA. The second least significant bit of ICR or IIA is aligned with the least significant bit of the processor memory word. All bits in the processor memory word that are not replaced by bits from the CU register are cleared.

If the specified CU register is AIN, it is not automatically cleared after it is read.

2.6 GENERAL DESCRIPTION OF TEST-SKIP INSTRUCTIONS

The instructions in the remainder of this section cause specific tests to be performed on the contents of the operand ACAR and (in some cases) on a specified CU register that must be another ACAR or an ADB location. The CU TRUE/FALSE Flip-Flop (TFFF)[5] is set (=1) or reset (=0) according to the result of the test.

A skip is then conditionally performed, according to the test result in the TFFF. For each type of test, two instructions are provided: skip if the test result is TRUE, and skip if the test result is FALSE. The skip is performed by modifying the contents of ICR with the contents of the SKIP field in the instruction. The SKIP field consists of instruction bits 8:8. Bit 8 is the sign of the skip - "0" for a positive (forward) skip or "1" for a negative (backward) skip. Bits 9:7 are the skip distance; that is, the number of instructions (syllables) to be skipped. Depending on the value of bit 8, the skip distance is added to or subtracted from the contents of ICR (with least significant bits aligned) to yield a new syllable address. As usual, ICR is also incremented by 1. The following examples illustrate the effects.

SKIP Value	Effect
-1	Infinite loop (see *Special Actions* below)
0	No effect
+1	Skip next instruction

In certain Test-Skip instructions, the Current Index Field of the operand ACAR is modified by the Increment Field after the test is performed. This modificaiton of the operand ACAR is *unconditional*. The test is performed; the result is saved in the TFFF; the operand ACAR is modified; and a skip is conditionally performed.

In two instructions, SKIPT and SKIPF, no test is performed. The preexisting value of the TFFF is used as the sole condition for the skip.

The unconditional skip instruction, SKIP, is also discussed in this section.

Special Actions: *A skip of -1 would result in an infinite loop, executing the same skip repeatedly. If a skip of -1 is specified, AIN bit 8 is set.*

NOTE: *Although CU arithmetic operations are done in two's complement arithmetic, quantities used in the comparison testing of the following test instructions are to be considered positive. Therefore - 1 is considered greater than 0 and it follows -1 > -2.*

[5]The TFFF is ACR bit 0.

Mnemonic Codes: CTSBT, CTSBF

Test Performed: Is specified bit in operand ACAR a "1"? (Result to TFFF.)

Operations: CTSBT skips if test result is TRUE; CTSBF skips if test result is FALSE.

Instruction Words:

	0 4	5 7	8 15	16 17	18	19	20 23	24 25	26 31
CTSBF	11	ACARX	SKIP	Oper ACAR	1	P	02	0	Bit No.

	0 4	5 7	8 15	16 17	18	19	20 23	24 25	26 31
CTSBT	11	ACARX	SKIP	Oper ACAR	1	P	00	0	Bit No.

Bits 26:6 of the instruction, which may be ACAR-indexed, specify a bit number in the operand ACAR. This bit is tested, and the TFFF is loaded with the value of the bit. A conditional skip is then performed, depending on the value in the TFFF. [Please see the general description of the Test-Skip instructions at the beginning of Section 2.6 for a more detailed explanation of the skip action.]

Mnemonic Codes: EQLXT, EQLXF

First Operand: Current Index Field (bits 40:24) of operand ACAR.

Second Operand: Bits 40:24 of ADB location of ACAR specified in ADR field.

Test Performed: Is first operand equal to second operand? (Result to TFFF.)

Operations: EQLXT skips if test result is TRUE; EQLXF skips if test result is FALSE.

Instruction Words:

	0–4	5–7	8–15	16–17	18	19	20–23	24–31
EQLXT	14	ACARX	SKIP	Oper ACAR	1	P	15	ADR
EQLXF	14	ACARX	SKIP	Oper ACAR	1	P	17	ADR

[Please see the general description of the Test-Skip instructions at the beginning of Section 2.6 for a more detailed explanation of the skip action.]

Mnemonic Codes: GRTRT, GRTRF

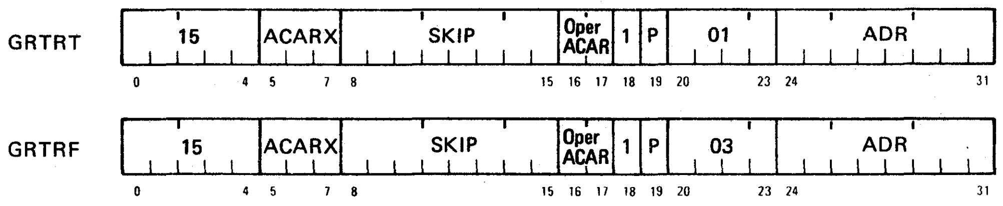
First Operand: Current Index Field (bits 40:24) of operand ACAR.

Second Operand: Bits 40:24 of ADB location or ACAR specified in ADR field.

Test Performed: Is first operand greater than second operand?
(Result to TFFF.)

Operations: GRTRT skips if test result is TRUE; GRTRF skips if test result is FALSE.

Instruction Words:



[Please see the general description of the Test-Skip instructions at the beginninq of Section 2.6 for a more detailed explanation of the skip action.]

Mnemonic Codes: LESST, LESSF [redacted]

First Operand: Current Index Field (bits 40:24) of operand ACAR.

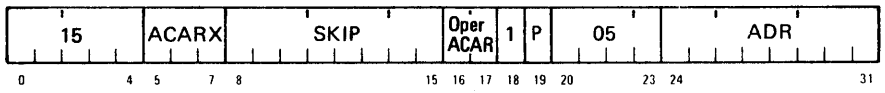
Second Operand: Current Index Field (bits 40:24) of ADB location or ACAR specified in ADR field.

Test Performed: Is first operand less than second operand?
(Result to TFFF.)

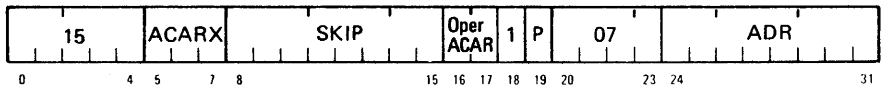
Operations: LESST skips if test result is TRUE; LESSF skips if test result is FALSE.

Instruction Words:

LESST



LESSF



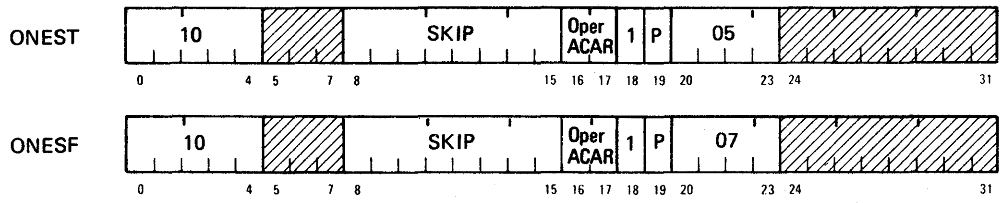
[Please see the general description of the Test-Skip instructions at the beginning of Section 2.6 for a more detailed explanation of the skip action.]

Mnemonic Codes: ONEST, ONESF

Test Performed: Does operand ACAR contain all "ones"? (Result to TFFF.)

Operations: ONEST skips if test result is TRUE; ONESF skips if test result is FALSE.

Instruction Words:



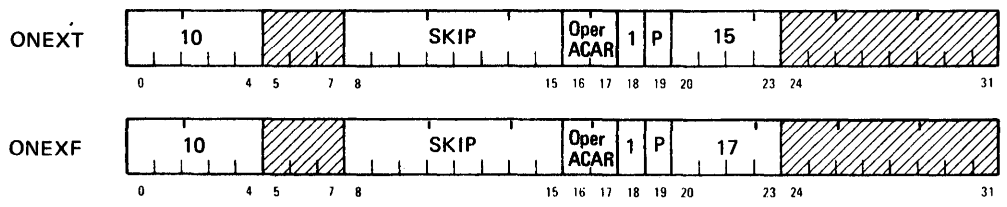
These instructions test all 64 bits of the operand ACAR. [Please see the general description of the Test-Skip instructions at the beginning of Section 2.6 for a more detailed explanation of the skip action.]

Mnemonic Codes: ONEXT, ONEXF

Test Performed: Does Current Index Field of operand ACAR contain all "ones"? (Result to TFFF.)

Operations: ONEXT skips if test result is TRUE; ONEXF skips if test result is FALSE.

Instruction Words:

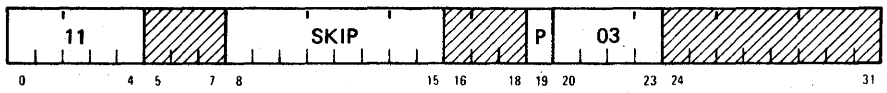


These instructions test bits 40:24 of the operand ACAR. [Please see the general description of the Test-Skip instructions at the beginning of Section 2.6 for a more detailed explanation of the skip action.]

Mnemonic Code: SKIP

Operation: Unconditional skip.

Instruction Word:



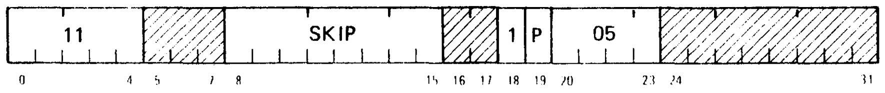
In this instruction, there is no test and the TFFF is not referenced. [Please see the general description of the Test-Skip instructions at the beginning of Section 2.6 for a more detailed explanation of the skip action.]

Mnemonic Codes: SKIPT, SKIPF

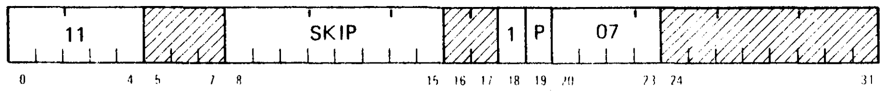
Operations: No test is performed. SKIPT skips if value found in TFFF is TRUE; SKIPF skips if value is FALSE.

Instruction Words:

SKIPT



SKIPF



[Please see the general description of the Test-Skip instructions at the beginning of Section 2.6 for a more detailed explanation of the skip action.]

Mnemonic Codes: TXET, TXEF

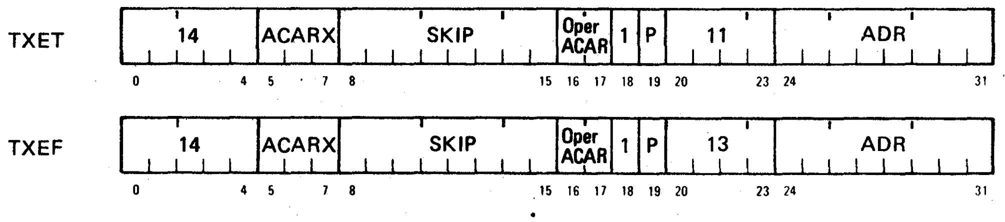
First Operand: Current Index Field (bits 40:24) of operand ACAR.

Second Operand: Limit Field (bits 6:24) of ADB location or ACAR specified in ADR field.

Test Performed: Is first operand equal to second operand? (Result to TFFF.)

Operations: TXET skips if test result is TRUE; TXEF skips if test result is FALSE.

Instruction Words:



[Please see the general description of the Test-Skip instructions at the beginning of Section 2.6 for a more detailed explanation of the skip action.]

Mnemonic Codes: TXETM, TXEFM

Test Performed: Is Current Index Field of operand ACAR equal to Limit Field of same ACAR? **(Result to TFFF.)**

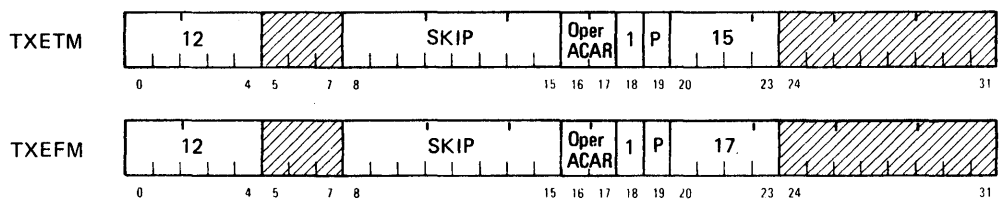
Operations:

(a) Test is performed and result saved in TFFF.

(b) Current Index Field is modified (unconditionally) by Increment Field of *same* ACAR.

(c) TXETM skips if test result is TRUE; TXEFM skips if test result is FALSE.

Instruction Words:



The unconditional modification of the Current Index Field (bits 40:24) by the Increment Field (bits 1:15) is performed as follows:

(a) Bit 1 is the sign of the increment - "0" for positive or "1" for negative.

(b) Using twos-complement arithmetic, bits 2:14 are added to or subtracted from bits 40:24 with the least significant bits aligned; any overflow is disregarded.

(c) Bits 0:40 of the operand ACAR are unaltered.

[Please see the general description of the Test-Skip instructions at the beginning of Section 2.6 for a more detailed explanation of the skip action.]

Mnemonic Codes: TXGT, TXGF

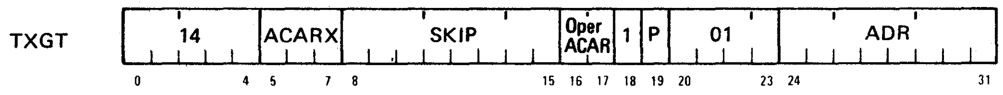
First Operand: Current Index Field (bits 40:24) of operand ACAR.

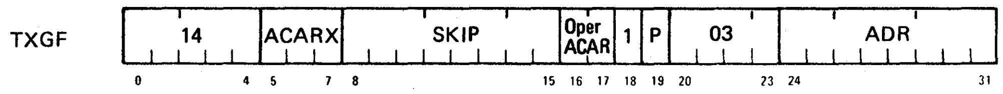
Second Operand: Limit Field (bits 16:24) of ADB location or ACAR specified in ADR field.

Test Performed: Is first operand greater than second operand?
(Result to TFFF.)

Operations: TXGT skips if test result is TRUE; TXGF skips if test result is FALSE.

Instruction Words:





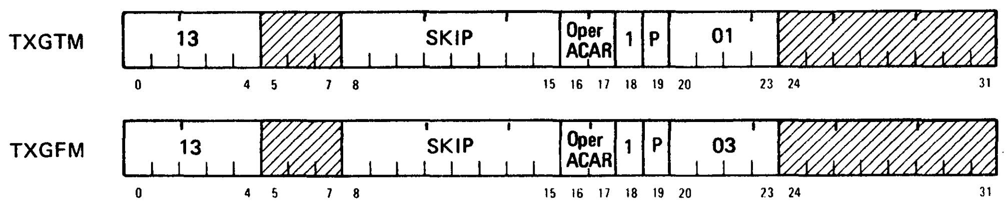
[Please see the general description of the Test-Skip instructions at the beginning of Section 2.6 for a more detailed explanation of the skip action.]

Mnemonic Codes: TXGTM, TXGFM

Test Performed: Is Current Index Field of operand ACAR greater than Limit Field of *same* ACAR? **(Result to TFFF.)**

Operations:
(a) Test is performed and result saved in TFFF.
(b) Current Index Field is modified (unconditionally) by Increment Field of *same* ACAR.
(c) TXGTM skips if test result is TRUE; TXGFM skips if test result is FALSE.

Instruction Words:



The unconditional modification of the Current Index Field (bits 40:24) by the Increment Field (bits 1:15) is performed as follows:

(a) Bit 1 is the sign of the increment - "0" for positive or "1" for negative.
(b) Using twos-complement arithmetic, bits 2:14 are added to or subtracted from bits 40:24 with least significant bits aligned; any overflow is disregarded.
(c) Bits 0:40 of the operand ACAR are unaltered.

[Please see the general description of the Test-Skip instructions at the beginning of Section **2.6** for a more detailed explanation of the skip action.]

Mnemonic Codes: TXLT, TXLF

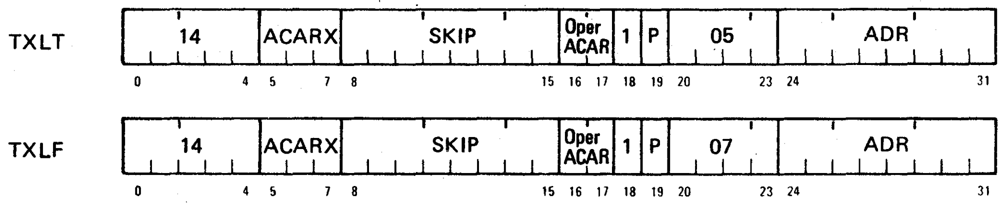
First Operand: Current Index Field (bits 40:24) of operand ACAR.

Second Operand: Limit Field (bits 16:24) of ADB location or ACAR specified in ADR field.

Test Performed: Is first operand less than second operand?
(Result to TFFF.)

Operations: TXLT skips if test result is TRUE; TXLF skips if test result is FALSE.

Instruction Words:



[Please see the general description of the Test-Skip instructions at the beginning of Section 2.6 for a more detailed explanation of the skip action.]

Mnemonic Codes: TXLTM, TXLFM

Test Performed: Is Current Index Field of operand ACAR less than Limit Field of the same ACAR? **(Result to TFFF.)**

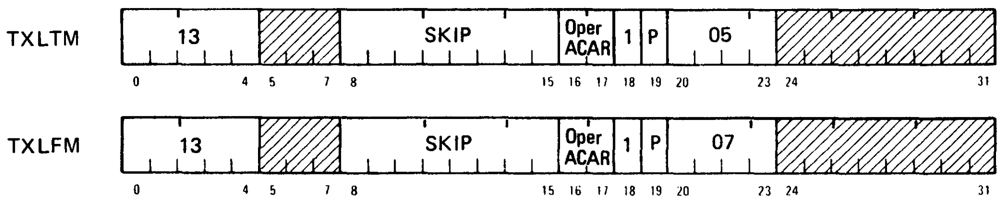
Operations:

(a) Test is performed and result saved in TFFF.

(b) Current Index Field is modified (unconditionally) by Increment Field of *same* ACAR.

(c) TXLTM skips if test result is TRUE; TXLFM skips if test result is FALSE.

Instruction Words:



The unconditional modification of the Current Index Field (bits 40:24) by the Increment Field (bits 1:15) is performed as follows:

(a) Bit 1 is the sign of the increment - "0" for positive or "1" for negative.

(b) Using twos-complement arithmetic, bits 2:14 are added to or subtracted from bits 40:24 with the least significant bits aligned; any overflow is disregarded.

(c) Bits 0:40 of the operand ACAR are unaltered.

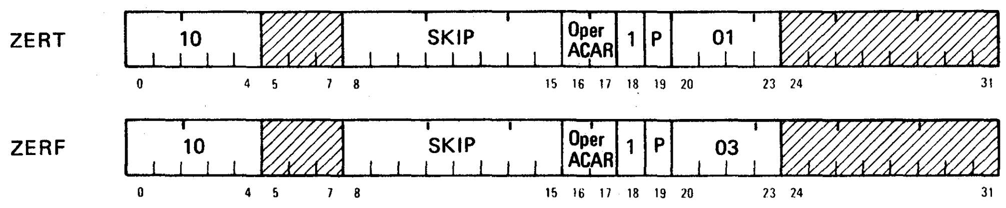
[Please see the general description of the Test-Skip instructions at the beginning of Section 2.6 for a more detailed explanation of the skip action.]

Mnemonic Codes: ZERT, ZERF

Test Performed: Does operand ACAR contain all "zeros"? (Result to TFFF.)

Operations: ZERT skips if test result is TRUE; ZERF skips if test result is FALSE.

Instruction Words:



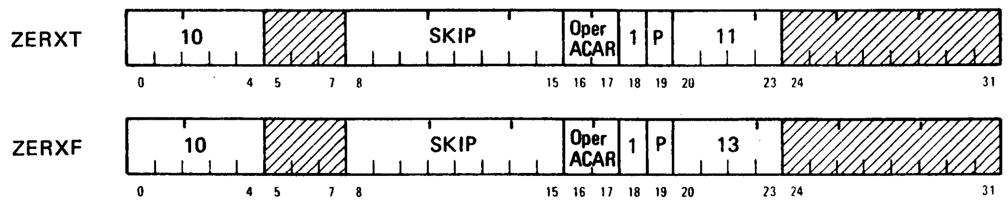
These instructions test all 64 bits of the operand ACAR. [Please see the general description of the Test-Skip instructions at the beginning of Section 2.6 for a more detailed explanation of the skip action.]

Mnemonic Codes: ZERXT, ZERXF

Test Performed: Does Current Index Field of operand ACAR contain all "zeros"? (Result to TFFF.)

Operations: ZERXT skips if test result is TRUE; ZERXF skips if test result is FALSE.

Instruction Words:



These instructions test bits 40:24 of the operand ACAR. [Please see the general description of the Test-Skip instructions at the beginning of Section 2.6 for a more detailed explanation of the skip action.]

Section 3

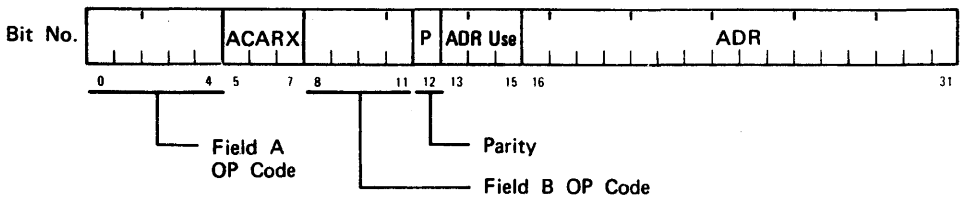
FINST/PE Instructions

CONTENTS

Section 3
FINST/PE Instructions

3.1 INSTRUCTION FORMAT AND FIELD USAGE

Like Section 2, this section begins with an illustration of the general format for FINST/PE instruction words, followed by a listing of each field and its usage.



Field	Description
ADR	Bits 16:16. This field may be indexed in the CU by an ACAR before being sent to the PEs [see ACARX in this list and also Section 3.3.1]. Depending on the instruction and the contents of the ADR USE field, ADR may be interpreted as one of the following.

(a) Processor memory address, bit number, or shift count. Each may be indexed in the PE by RGX or RGS, depending on ADR USE. The address is a right-justified, 11-bit row address.[1]

(b) PE register code. The code is formed by setting (=1) one bit within bits 17:6 of the instruction [see Table 3-1, p. 3-3].

(c) Literal operand. Via the Common Data Bus (CDB), each PE receives a 64-bit value that is formed in the following manner.

No ACAR-indexing: CDB (bits 0:48) = 0; CDB (bits 48:16) = ADR.
Using ACAR-indexing: CDB (bits 0:48) = ACAR (bits 0:48); CDB (bits 48:16 = ACAR (bits 48:16) + ADR (high-order carry from the addition is lost). If ADR = 0, however, the literal operand in CDB is the 64-bit value found in the ACAR.

[1]Bits 16:5 are unused.

ADR USE

Bits 13:3. These bits govern the use of the ADR data (possibly ACAR-indexed) received from the CU. The following are the possible cases.

(a) Bit 15 set. ADR data is a processor memory address, a bit position, or shift count. PE-indexing is permitted. Bits 13 and 14 govern PE-indexing in the following ways.

Bits 13 and 14 reset: No PE-indexing.

Bit 13 set, bit 14 reset: Index by adding RGS to ADR data.

Bit 14 set, bit 13 reset: Index by adding RGX to ADR data.

Bits 13 and 14 set: Unspecified results.

(b) Bit 15 reset, bit 13 set. ADR data is a PE register code [see Table 3-1]. No PE-indexing is possible. Bit 14 is disregarded.

(c) Bit 15 reset, bit 13 reset. CU transmitting a 64-bit literal of which the ADR data is a component [see Section 3.3.1].

(d) Tables 3-2 and 3-3 [see p. 3-4] summarize the various configurations of ADR USE for two classes of instructions: those that use *operand addressing and indexing logic*, and those that use *bit/shift counting and indexing logic*.

ACARX

Bits 5:3. These bits specify ACAR-indexing. If bit 5 is set (=1), ACAR-indexing is performed; bits 6 and 7 contain the ACAR number (0 - 3). If bit 5 is reset (=0), no ACAR-indexing is performed; bits 6 and 7 are disregarded. [See Section 3.3.1 for a description of ACAR-indexing.]

Field A OP Code

Bits 0:2 and 2:3. This field contains the first and second digits of the octal OP code. Bit 0 is *always* "1" for FINST/PE instructions.

Field B OP Code

Bits 8:1 and 9:3. This field contains the third and fourth digits of the octal OP code.

Parity

Bit 12:1. Odd parity bit.

Table 3-1 Summary of Addressable PE Registers*

Register	Description
RGA (Accumulator Register)	A 64-bit, general-purpose accumulator used by the PE for arithmetic and logical operations. RGA may be protected by the E and E1 bits against modification by operations performed in the PE. *Address Code:* Bit 17 set (=1) in the instruction word [see No. 1 and 2].
RGB ("B" Register)	A 64-bit register used to hold second operands and intermediate results. It is also used as an extension of RGA in double-precision operations. RGB cannot be protected against modification by the E and E1 bits. *Address Code:* Bit 18 set (=1) in the instruction word [see No. 1 and 2].
RGX (Index Register)	A 16-bit register used as an index register for independent modification of addresses received by the PE ("X-indexing"). RGX may be protected by the E bit against modification by operations performed in the PE. *Address Code:* Bit 19 set (=1) in the instruction word [see No. 1 and 2].
RGS (Storage Register)	A 64-bit, general-purpose storage register that may also be used for indexing ("S-indexing") in the same manner as the RGX. RGS may be protected by the E and E1 bits against modification by operations performed in the PE. *Address Code:* Bit 20 set (=1) in the instruction word [see No. 1 and 2].
RGR (Routing Register)	A 64-bit register that receives data routed from another PE and that is also used in some instructions to hold intermediate results. RGR cannot be protected against modification by the E and E1 bits. *Address Code:* Bit 21 set (=1) in the instruction word [see No. 1 and 2].
RGD (Mode Register)	An 8-bit register containing control and indicator bits related to the status of the PE [see Section 3.3.3 for a more detailed discussion]. *Address Code:* Bit 22 set (=1) in the instruction word [see No. 1, 2, and 3].

*** This table contains *only* those registers that have been referenced in this manual.**

In the preceding table [Table 3-1], three important codicils were referred to and should be kept in mind whenever those registers are implemented.

(1) To form a PE register address code, exactly one bit in the ADR field of the instruction should be set (=1), as indicated in Table 3-1. If more than the ADR bit is set (=1), the results are unspecified. In the ADR USE field, bit 15 must be zero and bit 13 must be one to indicate that the ADR field contains a register code.

(2) All register codes shown in Table 3-1 may be used in any PE instruction that permits a register code, except instructions RTL and LD (A/D/R/S/X). See descriptions [p. 3-45 and 3-59] of these instructions for details.

(3) Within the PE, a transfer to or from RGD is always between the eight bits of RGD and the eight most significant bits of RGB. This is the only physical path in the PE hardware. Thus, if eight bits are transferred between RGD and some other PE register, RGB will always be modified as a side-effect of the transfer, since the transfer will go via RGB.

Table 3-2 Operand Addressing and Indexing Logic

ADR USE (Bits 13:3)	Interpretation of ADR Contents
000	CU is transmitting a 64-bit literal*
001	Processor memory address — no PE-indexing
010	CU is transmitting a 64-bit literal*
011	Processor memory address -- index by RGX
100	PE register code -- no PE-indexing allowed
101	Processor memory address — index by RGS
110	PE register code -- no PE-indexing allowed
111	Unspecified results

*Contents of ADR will be a component of the literal, depending on whether ACAR-indexing is used [see Section 3.3.1 for a more detailed description].

Table 3-3 Bit/Shift Counting and Indexing Logic

ADR USE (Bits 13:3)	Interpretation of ADR Contents*
000	No PE-indexing
001	No PE-indexing
010	Index by RGX
011	Index by RGX
100	Index by RGS
101	Index by RGS
110	Unspecified results
111	Unspecified results

*In all instructions where this table applies, bit 15 is assumed to be on, and the ADR contents are interpreted as a bit number or shift count (depending on the instruction). Hence the table indicates the PE-indexing used on the bit number or shift count.

3.2 CATEGORIZATION OF FINST/PE INSTRUCTIONS

The following subsections are a generalized categorization of FINST/PE instructions. Each instruction mnemonic is followed by a page reference in brackets []. These page numbers refer to the detailed descriptions of these instructions, which begin on p 3-14. For a complete alphabetical listing of these instructions, see the inside back cover of this manual.

<u>LOAD REGISTER</u>

LD(A/B/D/R/S/X) [3-45]	Load specified register
LEX [3-49]	Load RGA exponent

<u>STORE REGISTER TO PROCESS MEMORY</u>

ST(A/B/R/S/X) [3-72]	Store specified register

<u>ROUTE</u>

RTL [3-59]	Route from specified register to RGR of another PE

<u>CHANGE RGA CONTENTS</u>

CLRA [3-23]	Clear RGA
COMPA [3-24]	Complement RGA
CAB [3-21]	Complement specified RGA bit
RAB [3-57]	Reset specified RGA bit
SAB [3-57]	Set specified RGA bit
CHSA [3-22]	Complement RGA sign
SAP [3-60]	Make RGA positive
SAN [3-60]	Make RGA negative

<u>BASIC ARITHMETIC</u>

AD(A,M,N,R variants) [3-14]	Add
ADD [3-17]	Add 64-bit logical words
ADEX [3-18]	Add exponent fields
EAD [3-28]	Add (extended precision results)
SB(A,M,N,R variants) [3-61]	Subtract
SUB [3-73]	Subtract 64-bit logical words
SBEX [3-64]	Subtract exponent fields
ESB [3-31]	Subtract (extended precision result)

ML(A,M,N,R variants) [3-50]	Multiply
MULT [3-52]	Multiply (32-bit only)
DV(A,M,N,R variants) [3-25]	Divide
NORM [3-54]	Normalize
ASB [3-20]	Transfer RGA sign to RGB sign

ADDRESS ARITHMETIC

XI [3-78]	Add to RGX
(I/J)XGI [3-42]	Same, with overflow to RGD bit I or J
XD [3-77]	Subtract from RGX
(I/J)XLD [3-43]	Same, with complemented overflow to RGD bit I or J

BOOLEAN

(N)AND(N) [3-19]	Logical AND (operands may be complemented)
(N)OR(N) [3-56]	Logical OR (operands may be complemented)
EOR [3-29]	Logical EXCLUSIVE-OR
EQV [3-30]	Logical EQUIVALENCE

SHIFT/ROTATE

RTA(L/R) [3-58]	Rotate RGA left/right
SHA(L/R) [3-70]	Shift RGA left/right
SHAM(L/R) [3-71]	Shift RGA matissa left/right
SHAB(L/R) [3-68]	Shift RGA + RGB left/right
SHABM(L/R) [3-69]	Shift RGA mantissa + RGB mantissa left/right

LOAD/SET RGD BIT

LD(E/El/EEl/G/H/I/J) [3-47]	Load RGD bit from one bit of a literal
SET(E/El/F/Fl/G/H/I/J) [3-65]	Set RGD bit to a function of two RGD bits

LOAD RGD BIT I OR J FROM RGA BIT

(I/J)B [3-34]	Load I or J from specified RGA bit
(I/J)SN [3-40]	Load I or J from RGA sign bit

LOAD RGD BIT I OR J WITH RESULT OF TEST

(I/J)A(G/L) [3-33]	Signed floating comparison of RGA and operand
(I/J)L(G/E/L) [3-35]	Logical-word comparison of RGA and operand
(I/J)M(G/E/L) [3-37]	Mantissa-only comparison of RGA and operand

(I/J)L(O/Z) [3-36]	Test RGA logical word
(I/J)M(O/Z) [3-38]	Test RGA mantissa only
(I/J)S(G/E/L) [3-39]	Address-field comparison of RGS and operand
(I/J)X(G/E/L) [3-41]	Address-field comparison of RGX and operand

Note: Tests are indicated by G for "greater than," E for "equal to," L for "less than," O for "all ones," and Z for "all zeros."

BYTE-ORIENTED INSTRUCTIONS

ADB [3-16]	Add bytes
SBB [3-63]	Subtract bytes
GB [3-32]	Test for RGA bytes "greater than" operand bytes
LB [3-44]	Test for RGA bytes "less than" operand bytes
NEB [3-53]	Test for RGA bytes "not equal" to operand bytes
OFB [3-55]	Recover byte carries or test results from RGC to RGB

SWAP INSTRUCTIONS

SWAP [3-74]	Swap RGA and RGB
SWAPA [3-75]	Swap RGA inner and outer words
SWAPX [3-76]	Swap RGA outer word and RGB inner word

3.3 GENERAL INFORMATION ON FINST/PE INSTRUCTIONS

The following subsections contain general information that applies in broad terms to the FINST/PE instructions, especially those instructions that are arithmetic. The reader should become familiar with this general information before proceeding to the more detailed descriptions of the various FINST/PE instructions.[2]

3.3.1 Transmission of ADR Data to the PE - Also ACAR-Indexing and Literals

The 16-bit ADR field of an instruction is sent to each PE on the Common Data Bus (CDB), which is 64 bits wide. If ACAR-indexing is not used (instruction bit 5 reset), the ADR data is in CDB bits 48:16; the remaining CDB bits (0:48) are all zeros.

If ACAR-indexing is used (bit 5 set), bits 6 and 7 specify an ACAR. Indexing is performed in the CU in the following manner.

(a) The ADR data (16 bits) are added to ACAR bits 48:16, the 16 least significant bits of the ACAR, and the 16-bit result is placed in CDB bits 48:16. Any carry resulting from the addition is lost.

[2]These are arranged alphabetically according to instruction mnemonic, except where instructions are grouped together under a single description.

(b) ACAR bits 0:48 are placed in CDB bits 0:48.

The handling of the 64 bits in the CDB depends on both the OP code and the ADR USE field [see Section 3.1]. In most cases, only the 16 (or fewer) least significant bits of the CDB are used. Hence the data that is used will effectively consist of the contents of the ADR field plus the contents of the ACAR; that is, the first 48 bits of the CDB do not matter.

When a literal is transmitted, however, all 64 bits of the CDB are used. The programmer must then bear in mind that this quantity is not, strictly speaking, the sum of the ADR and ACAR contents, since the carry from the 16-bit add is lost.

Note: If ADR = 0, then CDB = ACAR. And if ACAR = 0 or if ACAR-indexing is not used, then CDB = ADR, with 48 leading zeros.

This description of ACAR-indexing applies to *all* instructions where the instruction word layout shows an ACARX field. Accordingly, the detailed instruction descriptions [see Section 3.4] do not mention ACAR-indexing or the ACARX field - except where special cautions apply.

3.3.2 64-Bit and 32-Bit Modes

At any moment, the ILLIAC IV is in either 64-bit *or* 32-bit mode, as determined by ACR bit 10. This bit can be reset (=0) programmatically by the CACRB instruction for 64-bit mode and set (=1) for 32-bit mode. The CU sends the current value of this bit to FINST, along with each FINST/PE instruction.

The mode has no effect on the execution of ADVAST instructions. Certain FINST/PE instructions are also mode-independent, while others operate differently in each mode. The latter instructions are designed to produce meaningful results in 64-bit mode (if the operands are formatted as 64-bit words) and in 32-bit mode (if the operands are formatted as 32-bit inner and outer words).

3.3.3 The RGD Register

RGD is an 8-bit register in each PE. Each of the eight bits is used individually. Therefore each bit is individually named: E, E1, F, F1, I, G, J, and H for RGD bit positions 0 through 7, respectively.

When set (=1), the E and E1 bits "enable" operations on the outer and inner portions, respectively, of RGA, RGS, and all processor memory locations that are addressable by the PE. When reset (=0), they "disable" operations by protecting the outer and inner portions of these registers and locations against writes.

The F, I, and J bits are related to operations in 64-bit mode and to outer-word operations in 32-bit mode. The F1, G, and H bits are related to inner-word operations in 32-bit mode. The following is a more detailed description.

E BITS

RGD bits E and E1 in each PE are used to enable changes to the outer and inner portions, respectively, of RGA, RGS, and all processor memory locations that are addressable by the PE. In addition, the E bit enables changes to RGX, and the E and E1 bits enable changes to the F and F1 bits, respectively. When the E and/or E1 bits are reset (=0), the corresponding items are *protected;* that is, they cannot be changed by any FINST/PE instruction (except as noted below).

The action of the E and E1 bits is *independent* of the ILLIAC IV mode (64-bit or 32-bit). The condition of the E and E1 bits does not affect the logic of instruction execution, except to prevent an alteration of the protected items when E or E1 is reset (=0).

As previously stated, RGA, RGS, RGX, processor memory, and the F and F1 bits may be protected by resetting (=0) the E and E1 bits.

Note: All other registers in the PE remain enabled (unprotected) at all times.

Also resetting of the E and E1 bits does not protect processor memory against writes performed by the operating system (data transfer operations) or against writes performed by ADVAST instructions, STORE or STOREX, nor does it protect the F and F1 bits against the instructions SETF and SETF1.

In 32-bit mode, the E and E1 bits may be used independently to enable or disable operations on 32-bit outer and inner operands. In 64-bit mode, the E and E1 bits may be set/reset concurrently (E = E1 *always*) to enable or disable operations on 64-bit operands. If E ≠ E1 in 64-bit mode, the results in many cases will be unspecified, so the use of this configuration is not recommended. There is no hardware provision to ensure that E = E1 in 64-bit mode.

The following chart summarizes the effects of all possible configurations of the E and E1 bits.

		RGA, RGS, and Processor Memory				
E	E1	Outer	Inner	RGX	F Bit	F1 Bit
0	0	Protected	Protected	Protected	Protected	Protected
0	1	Protected	Enabled	Protected	Protected	Enabled
1	0	Enabled	Protected	Enabled	Enabled	Protected
1	1	Enabled	Enabled	Enabled	Enabled	Enabled

E and E1 bits are set/reset by the LDE, LDE1, LDEE1, SETE, and SETE1 instructions. The E and E1 bits have a special effect on the ADVAST instruction LDC [see Section 2.5]. Also, they do *not* protect RGA in the DV instructions [see Section 3.4].

F AND F1 BITS

These bits are automatically set (=1) to indicate the occurrence of various fault conditions in PE arithmetic operations. In 64-bit mode, the F bit indicates a fault; in 32-bit mode, it indicates a fault in the outer word. A fault in the inner word in 32-bit mode is indicated by the F1 bit. Whenever any F bit is set in 64-bit mode or any F or F1 bit is set in 32-bit mode due to a PE error condition, the CU sets AIN bit 11 [causing a halt under normal conditions].

Four conditions may cause the F and F1 bits to be set.

(a) Exponent overflow
(b) Mantissa overflow in mantissa-sized, fixed-point arithmetic
(c) Zero or unnormalized divisor
(d) Exponent underflow, if ACR bit 9 is reset and if, in normalized operation, the resultant mantissa is nonzero

Remember that if the E bit is reset (=0), the F bit will not be set (=1) automatically and if the E1 bit is set, the F1 bit will not be set automatically. However, the F and F1 bits can be set/reset programmatically by the SETF and SETF1 instructions, regardless of the condition of E and E1.

Note: In other words, once an F or F1 bit is set, it will not be reset automatically under any circumstances; SETF or SETF1 must be used to reset it programmatically.

I AND J BITS

These two bits are used programmatically to store single bits of information, such as isolated bits from other registers or the results of tests performed in the PE. Instructions that can load these bits have mnemonics beginning with I or J - these letters indicate which of the two bits is loaded by each instruction. Information stored in the I and J bits may be used in SET instructions to cause other RGD bits to be set or reset; they may also be read by the CU by using the SETC instruction. These bits may also be set or reset via the SETI or SETJ instructions.

G AND H BITS

These two bits are used in 32-bit mode in parallel with the I and J bits; that is, in 32-bit mode the I or J bit is used for the outer 32-bit quantity, while the G or H bit is used for the inner 32-bit quantity. In 32-bit mode, instructions whose mnemonics begin with I load the I and G bits, while instructions whose mnemonics begin with J load the J and H bits. In 64-bit mode these bits may be used by the programmer to hold logical information. (via the SETG and SETH instructions). These bits may also be read by the CU using the SETC instruction.

3.3.4 Conditions Arising in PE Arithmetic Operations

The following conditions, which are briefly discussed, may arise in PE arithmetic operations on 64-bit or 32-bit operands.

ARITHMETIC ZERO

When a signed, floating-point arithmetic operation produces a zero result (e.g., a result of exponent underflow or, in normalized operation, when the resultant mantissa is zero), the entire 64- or 32-bit word is zeroed. A word of all "zeros" is interpreted as a mantissa of +0 and an exponent of $-(2^{14})$ for a 64-bit word and $-(2^6)$ for a 32-bit word. Thus a zero result from signed, floating-point arithmetic is always +0; it is never -0.

In mantissa-sized, fixed-point arithmetic and in unsigned, floating-point (e.g., "M" and "A" variants), the word is *not* automatically zeroed, so a result of -0 is possible.

Note: If +0 and -0 are compared arithmetically, they will be considered unequal; that is, $-0 < +0$.

MANTISSA OVERFLOW

In mantissa-sized, fixed-point operations, mantissa overflow is a fault condition, thus causing the F or F1 bit to be set as previously described. In all other cases, mantissa overflow is automatically adjusted by shifting the mantissa one place right, then placing a "one" in the leading bit position, and finally adding a 1 to the exponent. Rounding (if specified) is not affected by this operation.

EXPONENT OVERFLOW

When exponent overflow occurs in a floating-point operation, the resultant exponent is the true exponent mod 2^{14} (in 64-bit mode) or 2^6 (in 32-bit mode). The resultant mantissa will be correct, unless the exponent overflow occurred in response to normalizing a mantissa overflow.

EXPONENT UNDERFLOW

In signed, floating-point arithmetic operations, exponent underflow causes the entire 64-bit and 32-bit word to be cleared [described previously in "Arithmetic Zero"]. In unsigned operations, this is not done.

Exponent underflow will cause the F or F1 bit to be set (=1) if ACR bit 9 is set and if, in normalized operation, the resultant mantissa is nonzero.

3.3.5 Variants of PE Arithmetic Instructions

For the arithmetic instructions AD, SB, ML, and DV,[4] various suffixes may be appended to the mnemonic to produce *variant* OP codes. These suffixes are composed of various combinations of the letters A, M, N, and R, each denoting a specific optional type of arithmetic operation.

THE "A" VARIANT - UNSIGNED OPERANDS

The signs of both operands are ignored during the operation. Hence the sign bit in RGA is unchanged; that is, the result has the same sign as the operand found in RGA.

THE "M" VARIANT - MANTISSA-SIZED, FIXED-POINT OPERATION

The exponent fields of both operands are ignored during the operation. Thus the exponent field in RGA is unchanged; that is, the result has the same exponent as the operand found in RGA.

THE "N" VARIANT - NORMALIZATION

The resultant mantissa is shifted so that its most significant bit is placed in the leading bit position of the mantissa field (i.e., left-justified). The exponent is adjusted accordingly.

Note: The NORM instruction may be used to normalize an operand found in RGA without causing any other operation to be performed on it.

When the "N" variant is used with ML or DV, *both* operands are assumed to be normalized. In addition, the *divisor* is assumed to be normalized in *all* variations of the DV instruction, and *both* operands of the MULT instruction are assumed to be normalized.

THE "R" VARIANT - ROUNDING

Rounding is always performed by addding 1 to the least significant bit of RGA, assuming the next bit of the result would be a 1. In the ML and DV instructions, this makes the results in RGB meaningless.

Note: When both the "N" variant and the "R" variant are used in the same instruction, rounding is performed before normalization.

[4]This rule does not apply to the following arithmetic instructions: ADD, ADDEX, EAD, SUB, SUBEX, ESB, or MULT.

3.4 FINST/PE INSTRUCTION DESCRIPTIONS

Beginning on the next page and continuing through p. 3-78, the FINST/PE instructions are described in detail. Each description includes the mnemonic code, specification of operands (where applicable), a description of the operation of the instruction, a list of the registers affected, and the bit layout of the instruction word. Any fields for which a particular instruction has specific use are described. Shading is used to indicate those fields that are ignored for a particular instruction.

Mnemonic Codes: AD, ADA, ADM, ADMA, ADN, ADNA, ADR, ADRA, ADRN, ADRNA [redacted]
Add floating-point numbers or mantissa-sized, fixed-point numbers.

First Operand: Found in RGA.

Second Operand: Specified by ADR and ADR USE fields, using operand addressing and indexing logic, as shown in Table 3-2.

Operation: <u>64-Bit Mode.</u> The second operand is added to RGA, and the result is left in RGA. If both E and E1 bits are reset (=0), RGA will be unchanged. If E ≠ E1, the results are unspecified.

<u>32-Bit Mode.</u> Similar to a 64-bit operation, except the inner and outer pairs of words are added independently. If E bit is reset (=0), RGA outer word will be unchanged. If E1 bit is reset, RGA inner word will be unchanged.

Registers Affected: RGA - Contains result, as enabled by E and E1.
RGB - Contains second operand, which is either modified or unmodified by bits shifted off in aligning operand mantissas.
RGD - F or F1 bits may be set [see Section 3.3.3 for a detailed explanation].

Instruction Words:

AD

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

ADA

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

ADM

34	ACARX	14	P	ADR USE	ADR
0 4	5 7	8 11	12	13 15	16 31

ADMA

34	ACARX	15	P	ADR USE	ADR
0 4	5 7	8 11	12	13 15	16 31

ADN

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

ADNA

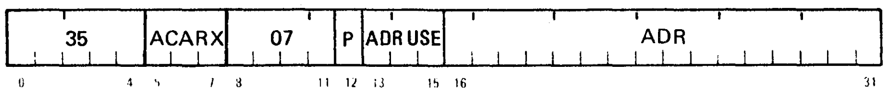
34	ACARX	05	P	ADR USE	ADR
0 4	5 7	8 11	12	13 15	16 31

ADR

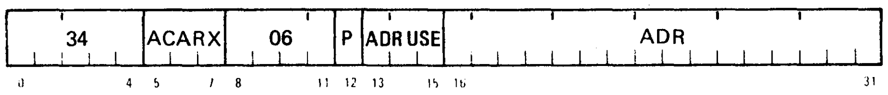
35	ACARX	06	P	ADR USE	ADR
0 4	5 7	8 11	12	13 15	16 31

Instruction Words (Cont.):

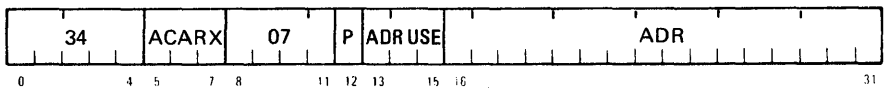
ADRA



ADRN



ADRNA



The following variants are permitted - in the combinations shown above.

(a) No suffix - Both operands are treated as signed, floating-point numbers (no rounding or normalization).

(b) A - Signs of both operands are ignored. The sign of the result is the same as the sign of the augend found in RGA. A result of -0 is possible.

(c) M - Exponents of both operands are ignored; that is, both are treated as mantissa-sized, fixed-point numbers. The exponent of the result is the same as the exponent of the augend found in RGA. A result of -0 is possible.

(d) N - Result is normalized (after rounding, where specified).

(e) R - Result is rounded in RGA.

See Sections 3.3.4 and 3.3.5 for further details.

Alignment for floating-point addition is performed in the following sequence.

(a) The exponent of the result is determined as the larger of the two operand exponents (which are subsequently adjusted if normalization is used).

(b) The mantissa of the operand with the smaller exponent is shifted right end-off, until it is properly aligned with the mantissa of the other operand. If the difference between the exponents is greater than 47 (or 23 in 32-bit mode), this process will zero the mantissa of the operand with the smaller exponent.

(c) The most significant bit shifted off in aligning the mantissa is saved and used for rounding (where specified).

Mnemonic Code: ADB
Add bytes.

First Operand: Found in RGA.

Second Operand: Specified by ADR and ADR USE fields, using operand addressing and indexing logic, as shown in Table 3-2.

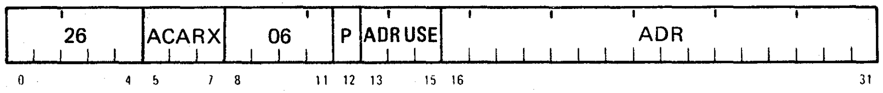
Operation: 64-Bit and 32-Bit Modes (Identical). Each operand is considered to be eight 8-bit bytes. Corresponding bytes are added, and the results are left in RGA. The high-order carry from each byte is stored in the most significant bit of the corresponding byte in RGC. RGC is not an addressable register. The OFB instruction transfers the contents of RGC to the least significant bits of corresponding bytes in RGB and clears the outer bits of RGB.

16 Bit, 24 Bit etc., extended precision from the 8 bit mode is facilitated by the proper combination of OF8 and ADS instructions.

If the E bit is reset (=0), RGA bytes 0, 5, 6, and 7 will be unchanged. If the E1 bit is reset, RGA bytes 1, 2, 3, and 4 will be unchanged. (The carries from these bytes will still be stored in RGC.)

Registers Affected: RGA - Contains results, as enabled by E and E1.
RGC - Contains high-order carries in the most significant bit of each byte; all other RGC bits are cleared.
RGB - Contains second operand.

Instruction Word:



Mnemonic Code: ADD

Add 64-bit logical words.

First Operand: Found in RGA.

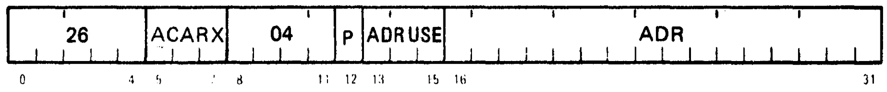
Second Operand: Specified by ADR and ADR USE fields, using operand addressing and indexing logic, as shown in Table 3-2.

Operation: 64-Bit and 32-Bit Modes (Identical). Both operands are treated as 64-bit, unsigned integers and are added together. The result is left in RGA. Overflow causes an end-around carry. If both E and E1 bits are reset (=0), RGA will be unchanged. If E ≠ E1, the results are unspecified.

Registers Affected: RGA - Contains result, as enabled by E and E1.

RGB - Contains second operand.

Instruction Word:



Mnemonic Code: ADEX

Add exponents.

First Operand: Found in RGA.

Second Operand: Specified by ADR and ADR USE fields, using operand addressing and indexing logic, as shown in Table 3-2.

Operation: <u>64-Bit Mode</u>. The exponent of the second operand is added to the exponent of RGA, and the result is left in RGA. Sign and mantissa fields are unchanged, unless exponent underflow occurs - then RGA is cleared. The exponents are treated as exponents in offset notation, not as binary numbers. If both E and E1 bits are reset (=0), RGA will be unchanged. If E ≠ E1, the results are unspecified.

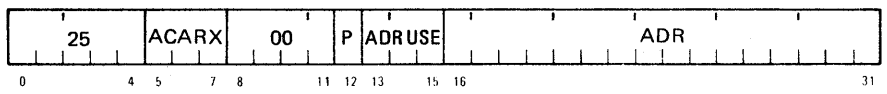
<u>32-Bit Mode</u>. Similar to 64-bit operation, except the inner and outer exponents are added independently. If the E bit is reset (=0), the outer exponent in RGA will be unchanged. If the E1 bit is reset, the inner exponent in RGA will be unchanged.

Registers Affected: RGA - Exponent field(s) modified, as enabled by E and E1. All 64 bits are cleared in case of exponent underflow (inner or outer 32 bits in 32-bit mode).

RGB - Contains second operand.

RGD - F or F1 bits may be set [see Section 3.3.3 for a detailed explanation].

Instruction Word:



Exponents are represented by offset code. This instruction makes the necessary adjustments for the offset, effectively adding the true exponents, and then puts the result (in offset code) into RGA.

Mnemonic Codes: AND, ANDN, NAND, NANDN

Logical AND of two operands or their complements.

First Operand: Found in RGA.

Second Operand: Specified by ADR and ADR USE fields, using operand addressing and indexing logic, as shown in Table 3-2.

Operation: 64-Bit and 32-Bit Modes. Bitwise logical AND of two operands or their complements, where OP1 is the first operand and OP2 is the second operand.

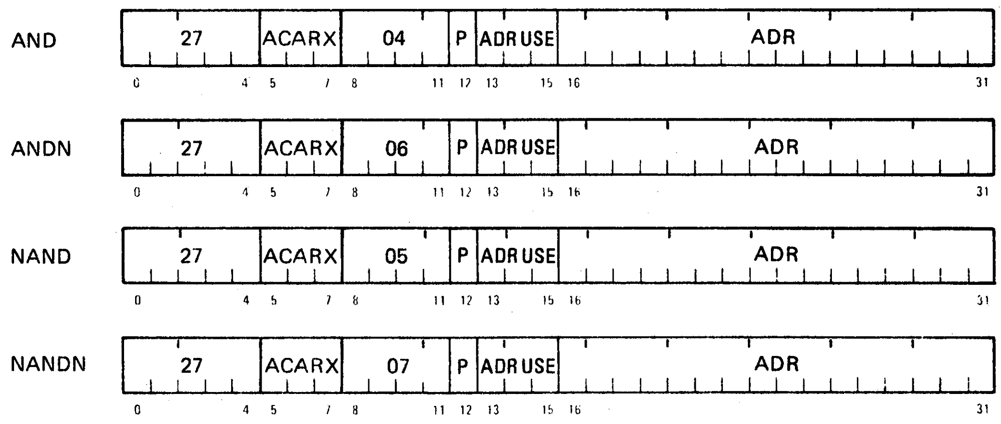
Mnemonic	Function
AND	OP1 and OP2
ANDN	OP1 and $\overline{\text{OP2}}$
NAND	$\overline{\text{OP1}}$ and OP2
NANDN	$\overline{\text{OP1}}$ and $\overline{\text{OP2}}$

The 64-bit result is left in RGA. If the E bit is reset (=0), RGA bits 0:8 and 40:24 will be unchagned. If the E1 bit is reset, RGA bits 8:32 will be unchanged.

Registers Affected: RGA - Contains result, as enabled by E and E1.

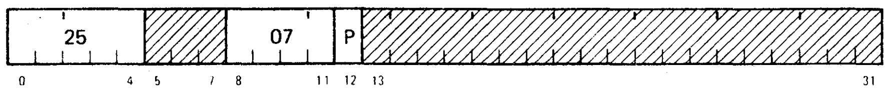
RGB - Contains second operand.

Instruction Words:



Observe that the mnemonic "NAND" does *not* result in a logical "NOT AND" operation.

Mnemonic Code:	ASB
	Transfer RGA sign to RGB sign.
Operation:	64-Bit Mode. Transfer RGA bit 0 (sign bit) to RGB bit 0.
	32-Bit Mode. Transfer RGA bits 0 and 8 (sign bits) to RGB bits 0 and 8.
Register Affected:	RGB - Sign bit(s) may be changed.
Instruction Word:	



Notice that the configuration of the E and E1 bits does not affect the operation of this instruction.

Mnemonic Code: CAB

Complement specified bit in RGA.

Bit Number: Specified by ADR and ADR USE fields, using bit/shift counting and indexing logic, as shown in Table 3-3.

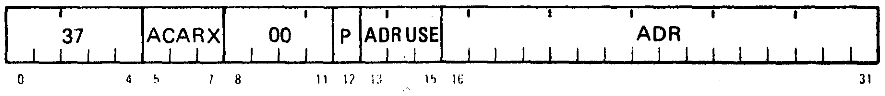
Operation: 64-Bit Mode. The bit number is taken mod 64, and the corresponding bit in RGA is complemented. Other RGA bits are unchanged. If the E bit is reset (=0), RGA bits 0:8 and 40:24 cannot be changed. If the E1 bit is reset, RGA bits 8:32 cannot be changed.

32-Bit Mode. The bit number is taken mod 32 and then used to select a bit position within each of the two 32-bit inner and outer words of RGA. For example, if the bit number is 7, it selects bit 7 of the outer word and bit 7 of the inner word (these are RGA bits 7 and 14). The selected bits are complemented, while other RGA bits are unchanged. If the E bit is reset (=0), the bit in the outer word cannot be changed. If the E1 bit is reset, the bit in the inner word cannot be changed.

Registers Affected: RGA - Specified bit(s) complemented, as enabled by E and E1.

RGB - Contains a mask in which bit(s) corresponding to selected bit(s) in RGA are set (=1), and all other RGB bits are reset (=0).

Instruction Word:



Mnemonic Code: CHSA

Change sign of RGA.

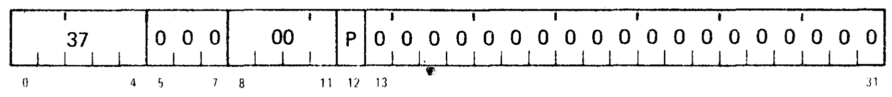
Operation: 64-Bit Mode. RGA bit 0 (sign bit of 64-bit word) is complemented if E bit is set (=1).

32-Bit Mode. RGA bits 0 and 8 (sign bits of 32-bit outer and inner words) are complemented if E and E1 bits, respectively, are set.

Registers Affected: RGA - Sign bit(s) complemented, as enabled by E and E1.

RGB - Contains a mask in which bit 0 (bit 8 in 32-bit mode) are set (=1) and all other RGB bits are reset (=0).

Instruction Word:



This is the CAB instruction where ADR and ADR USE equal zero in order to specify the sign bit(s).

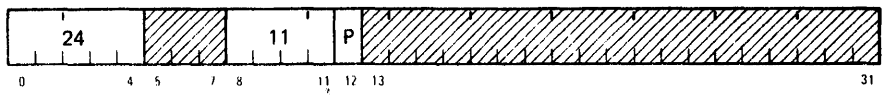
Mnemonic Code: CLRA

Clear RGA.

Operation: <u>64-Bit and 32-bit Modes</u>. All bits of RGA are cleared if both E and E1 are set (=1). If E is reset (=0), RGA bits 0:8 and 40:24 will be unchanged. If E1 is reset, RGA bits 8:32 will be unchanged.

Register Affected: RGA - Cleared, as enabled by E and E1.

Instruction Word:



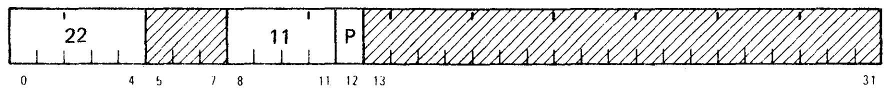
Mnemonic Code: COMPA

Complement RGA.

Operation: 64-Bit and 32-Bit Modes. All bits of RGA are complemented if both E and E1 are set (=1). If E is reset (=0), RGA bits 0:8 and 40:24 will be unchanged. If E1 is reset, RGA bits 8:32 will be unchanged.

Register Affected: RGA - Complemented, as enabled by E and E1.

Instruction Word:



Mnemonic Codes: DV, DVA, DVM, DVMA, DVN, DVNA, DVR, DVRA, DVRM, DVRMA, DVRN, DVRNA

Divide floating-point numbers or mantissa-sized, fixed-point numbers.

First Operand: Double-precision dividend found in RGA and RGB.

Second Operand: Single-precision, *normalized* divisor specified by ADR and ADR USE fields, using operand addressing and indexing logic, as shown in Table 3-2.

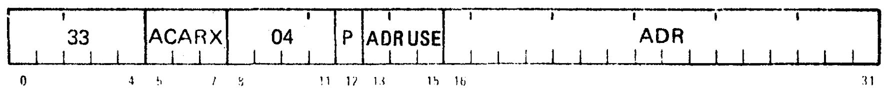
Operation: 64-Bit Mode. The RGB mantissa is considered to be the low-order extension of the RGA mantissa. RGB sign and exponent are ignored. The divisor is assumed to be normalized. The 64-bit quotient is left in RGA, and the 64-bit remainder is left in RGB. If both the E and E1 bits are reset (=0), RGA will be unchanged. If E ≠ E1, the results are unspecified.

32-Bit Mode. Similar to the 64-bit operation, except the inner and outer pairs of operands are divided. RGA and RGB outer portions contain outer double-precision dividend; RGA and RGB inner portions contain inner double-precision dividend. Two 32-bit quotients are left in RGA, and two 32-bit remainders are in RGB. If the E bit is reset (=0), the outer portion of RGA is *unspecified* rather than unchanged. Similarly, if the E1 bit is reset, the inner portion of RGA is *unspecified* rather than unchanged.

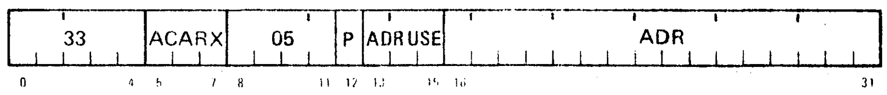
Registers Affected:
RGA - Contains quotient(s).
RGB - Contains remainder(s).
RGR - Contains copy of divisor(s); in 32-bit mode inner and outer mantissas are swapped.
RGD - F or F1 bits may be set [see Section 3.3.3 for a detailed explanation].

Instruction Words:

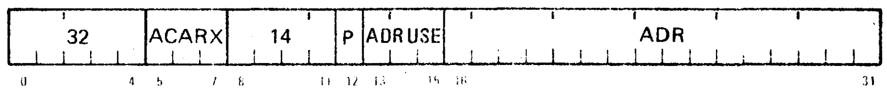
DV



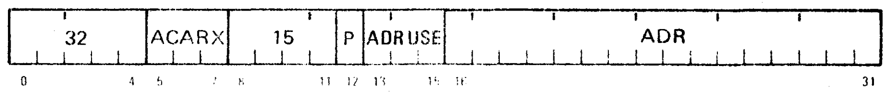
DVA



DVM



DVMA



Instruction Words (Cont.):

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

If the divisor is not normalized, the results are unspecified and the F or F1 bit will be set (=1).

The following are *variants* of the preceding combinations.[5] Note, in particular, the cautions that apply to the M option (mantissa-sized, fixed-point operation).

(a) No suffix - Both operands are treated as signed floating-point numbers. No rounding or normalization.

(b) A - Signs of both operands are ignored. The sign of the result is the same as the sign of the dividend (first operand). A result of -0 is possible.

(c) M - Exponents of both operands are ignored; that is, both are treated as mantissa-sized, fixed-point numbers. The exponent of the result is the same as the exponent of the dividend (first operand). A result of -0 is possible.

(d) N - Result is normalized (after rounding, when specified). *Both operands* are assumed to be normalized; the result will *not* be normalized unless the operands are.

(e) R - Result is rounded in RGA. The remainder left in RGB is meaningless if this variant is used.

[5]See Sections 3.3.4 and 3.3.5 for further details.

Cautions:

Side-Effects. RGR is altered.

Unnormalized Divisor. The divisor (second operand) must be normalized for *all* variants of DV, including mantissa-sized, fixed-point variants (M variants). In effect, this means that the most significant bit in the mantissa of the divisor must be "1". If this bit is not "1", the F or F1 bit will be set (=1) and the result will be unspecified. In most cases, this makes fixed-point division impractical.

E and E1 Bits. In 32-bit mode, the E and E1 bits do *not* protect the contents of RGA against modification. If one or both of these bits are set (=1), the corresponding portion of RGA will contain unspecified results.

Remainder. The remainder left in RGB will be *meaningless* if rounding (R variant) is used. Furthermore, even when rounding is not used, the remainder is meaningful only if the magnitude of the divisor is greater than the magnitude of the dividend. (Note also that the exponent of the remainder is not meaningful.) Therefore, when the quotient is normalized (N variant), the remainder mantissa may be correct but its positional significance is unspecified. To summarize: the remainder in RGB is meaningful only in unnormalized, unrounded operations in which the magnitude of the divisor is greater than the magnitude of the dividend.

Timing. The number of clocks required to execute DV instructions ranges from 52 to 56 (in 64-bit mode) or from 63 to 69 (in 32-bit mode). ML instructions require only 8 or 9 clocks (in 64-bit mode) or 10 clocks (in 32-bit mode).

Mnemonic Code: EAD

Floating-point add with extended-precision result.

First Operand: Found in RGA.

Second Operand: Specified by ADR and ADR USE fields, using operand addressing and indexing logic, as shown in Table 3-2.

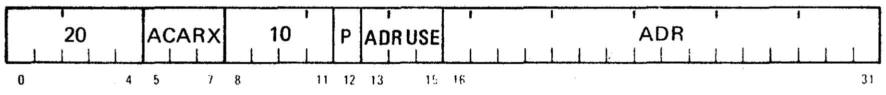
Operation: 64-Bit Mode. The operands are treated as signed, 64-bit, floating-point numbers and are added without rounding or normalization. The result left in RGB is the same as what would be produced in RGA by an AD instruction. RGA will contain a low-order result. The mantissa will contain any bits shifted off the mantissa of one of the operands for alignment, modified by any bit shifted off in adjusting for mantissa overflow during the addition. The exponent will be 48 less than the exponent in the high-order result in RGB. The sign will be the sign of the original operand with the smaller exponent. (The signs of the high-order and low-order results may differ.) If both the E and E1 bits are reset (=0), RGA will be unchanged but RGB will still contain the high-order result. If E ≠ E1, the results are unspecified.

32-Bit Mode. This instruction will produce unspecified results in 32-bit mode.

Registers Affected:

RGB - Contains high-order result.
RGA - Contains low-order result, as enabled by E and E1.
RGR - Contains copy of low-order result.
RGD - F bit may be set [see Section 3.3.3 for a detailed explanation].

Instruction Word:



See AD instruction and Section 3.3.4 for a more detailed description.

Mnemonic Code: EOR

Logical EXCLUSIVE-OR of two 64-bit operands.

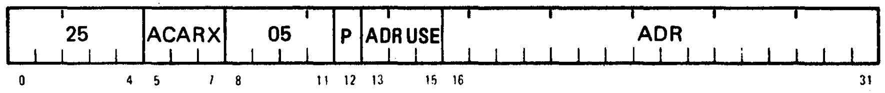
First Operand: Found in RGA.

Second Operand: Specified by ADR and ADR USE fields, using operand addressing and indexing logic, as shown in Table 3-2.

Operation: 64-Bit and 32-Bit Modes. Bitwise logical EXCLUSIVE-OR or two 64-bit operands. The result is left in RGA. If the E bit is reset (=0), RGA bits 0:8 and 40:24 will be unchanged. If the E1 bit is reset, RGA bits 8:32 will be unchanged.

Registers Affected: RGA - Contains result, as enabled by E and E1.
RGB - Contains second operand.

Instruction Word:



Mnemonic Code: EQV

Logical EQUIVALENCE of two 64-bit operands.

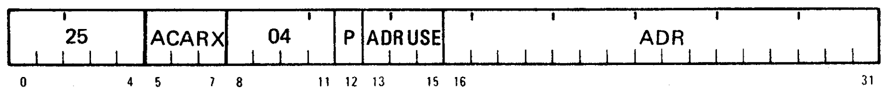
First Operand: Found in RGA.

Second Operand: Specified by ADR and ADR USE fields, using operand addressing and indexing logic, as shown in Table 3-2.

Operation: 64-Bit and 32-Bit Modes. Bitwise logical EQUIVALENCE of the two 64-bit operands. (Each bit of the result is True if the corresponding bits of the two operands have the same value and False if the corresponding bits of the two operands have different values.) The result is left in RGA. If the E bit is reset (=0), RGA bits 0:8 and 40:24 will be unchanged. If the E1 bit is reset, RGA bits 8:32 will be unchanged.

Registers Affected: RGA - Contains result, as enabled by E and E1.
RGB - Contains second operand.

Instruction Word:



Mnemonic Code: ESB

Floating-point subtraction with an extended-precision result.

First Operand: Found in RGA.

Second Operand: Specified by ADR and ADR USE fields, using operand addressing and indexing logic, as shown in Table 3-2.

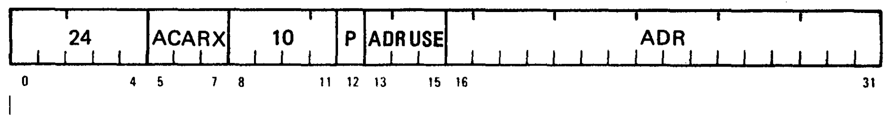
Operation: 64-Bit Mode. The operands are treated as signed, 64-bit, floating-point numbers. The second operand is subtracted from the first without rounding or normalization. The result left in RGB is the same as what would be produced in RGA by an SB instruction. RGA will contain a low-order result. The mantissa will contain any bits shifted off the mantissa of one of the operands for alignment, modified by any bit shifted off in adjusting for mantissa overflow during the subtraction. The exponent will be 48 less than the exponent of the high-order result in RGB. The sign will be either the sign of the first operand (if it had the larger exponent) or the complement of the sign of the second operand (if it had the larger exponent).[6] If both the E and E1 bits are reset (=0), RGA will be unchanged but RGB will still contain the high-order result. If E ≠ E1, the results are unspecified.

32-Bit Mode. This instruction will produce unspecified results in 32-bit mode.

Registers Affected:

RGB - Contains a high-order result.
RGA - Contains a low-order result, as enabled by E and E1.
RGR - Contains a copy of the low-order result.
RGD - F bit may be set [see Section 3.3.3 for a detailed explanation].

Instruction Word:



See SB instruction and also Section 3.3.4 for a detailed description.

[6]The signs of the high-order and low-order results may differ.

Mnemonic Code: GB [redacted]

Test bytes for RGA byte "greater than" second operand byte.

First Operand: Found in RGA.

Second Operand: Specified by ADR and ADR USE fields, using operand addressing and indexing logic, as shown in Table 3-2.

Operation: 64-Bit and 32-Bit Modes. Each 8-bit byte in RGA is compared with the corresponding byte in the second operand. If the RGA byte is greater than the second operand byte, the least significant bit of the RGA byte is set (=1) and the other bits in the byte are cleared. If the RGA byte is not greater than the second operand byte, all bits in the RGA byte are cleared. The OFB instruction transfers the contents of RGC to the least significant bits of the corresponding byte in RGB and clears the outer bits of RGB. 16-Bit, 24-Bit, 32-Bit (etc.) extended precision from the 8-Bit mode is facilitated by the proper combination of byte mode instructions. If the E bit is reset (=0), RGA bytes 0, 5, 6, and 7 will be unchanged. If the E1 bit is reset, RGA bytes 1, 2, 3 and 4 will be unchanged.

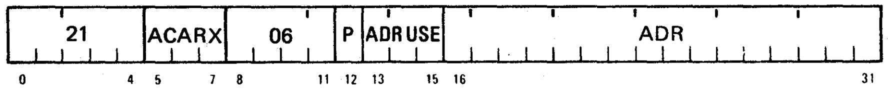
Registers Affected:

RGA - Contains test results in the least significant bit of each byte with other bits cleared, as enabled by E and E1.

RGC - Contains test results in the most significant bit of each byte with other bits cleared.

RGB - Contains second operand.

Instruction Word:



Mnemonic Codes: IAG, IAL, JAG, JAL [redacted]

Test for RGA arithmetically "greater than" or "less than" second operand.

First Operand: Found in RGA.

Second Operand: Specified by ADR and ADR USE fields, using operand addressing and indexing logic, as shown in Table 3-2.

Operation: 64-Bit Mode. The two operands are considered to be signed, floating-point numbers and are tested to find RGA either "greater than" or "less than" the second operand. Which test is indicated by the last letter of the mnemonic code. The test result is placed in either the I or J bit in RGD, depending on the first letter of the mnemonic code.

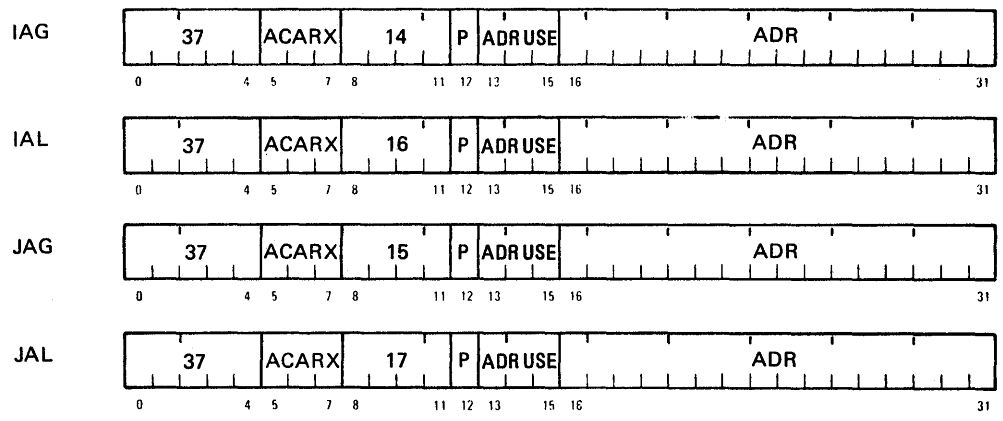
32-Bit Mode. Similar to the 64-bit operation - outer and inner words are tested independently. Results for outer and inner words are placed in the I and G bits (IAG and IAL), respectively, or in the J and H bits (JAG and JAL), respectively [see Section 3.3.3].

Note: Operation of these instructions is unaffected by the configuration of the E and E1 bits.

Registers Affected: RGD - I, J, G, and H bits may be changed.

RGB - Contains second operand.

Instruction Words:



No "IAE" or "JAE" instruction is provided for testing for arithmetic equality. Instead, this can be accomplished by testing for logical equality with the ILE or JLE instruction.

Mnemonic Codes: IB, JB

Transfer specified RGA bit to RGD bit I or J.

First Operand: Specified by ADR and ADR USE, using bit/shift counting and indexing logic, as shown in Table 3-3.

Operation: 64-Bit Mode. The bit number is taken mod 64, and the corresponding bit from RGA is transferred to either the I or J bit in RGD, depending on the first letter of the mnemonic code.

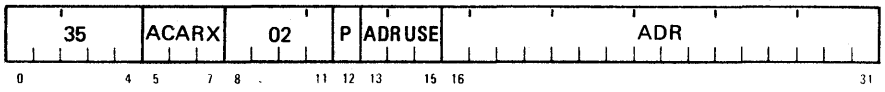
32-Bit Mode. The bit number is taken mod 32 and is used to select a bit position within each of the two 32-bit outer and inner words of RGA. For example, if the bit number is 7, it selects bit 7 of the outer word and bit 7 of the inner word (RGA bits 7 and 14, respectively). The selected bits are transferred from the outer and inner words to the I and G bits (IB instruction), respectively, or the J and H bits (JB instruction), respectively [see Section 3.3.3].

Note: Operation of these instructions is unaffected by the configuration of the E and E1 bits.

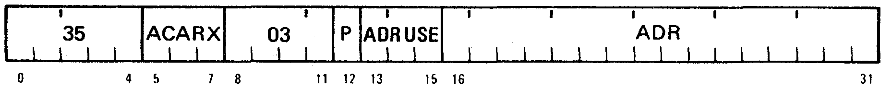
Registers Affected: RGD - I, J, G, and H bits may be changed.
RGB - Contains bit number.

Instruction Words:

IB



JB



Mnemonic Codes: ILG, ILE, ILL, JLG, JLE, JLL [redacted]

Test logical words for RGA "greater than", "equal to", or "less than" second operand.

First Operand: Found in RGA.

Second Operand: Specified by ADR and ADR USE fields, using operand addressing and indexing logic, as shown in Table 3-2.

Operation: 64-Bit Mode. Two 64-bit logical words (treated as 64-bit positive integers) are tested to find RGA "greater than", "equal to", or "less than" the second operand. Which test is indicated by the last letter of the mnemonic code. The test result is placed in the I or J bit in RGD, depending on the first letter of the mnemonic code.

32-Bit Mode. Similar to the 64-bit operation - outer and inner 32-bit logical words are tested independently. Results for outer and inner operands are placed in I and G bits (ILG, ILE, and ILL), respectively, or in J and H bits (JLG, JLE, and JLL), respectively [see Section 3.3.3].

Note: Operation of these instructions is unaffected by the configuration of the E and E1 bits.

Registers Affected: RGD - I, J, G, and H bits may be changed.

RGB - Contains second operand.

Instruction Words:

	0–4	5–7	8–11	12	13–15	16–31
ILE	35	ACARX	16	P	ADR USE	ADR
ILG	33	ACARX	14	P	ADR USE	ADR
ILL	33	ACARX	16	P	ADR USE	ADR
JLE	35	ACARX	17	P	ADR USE	ADR
JLG	33	ACARX	15	P	ADR USE	ADR
JLL	33	ACARX	17	P	ADR USE	ADR

Mnemonic Codes: ILO, ILZ, JLO, JLZ

Test RGA for "all ones" or "all zeros".

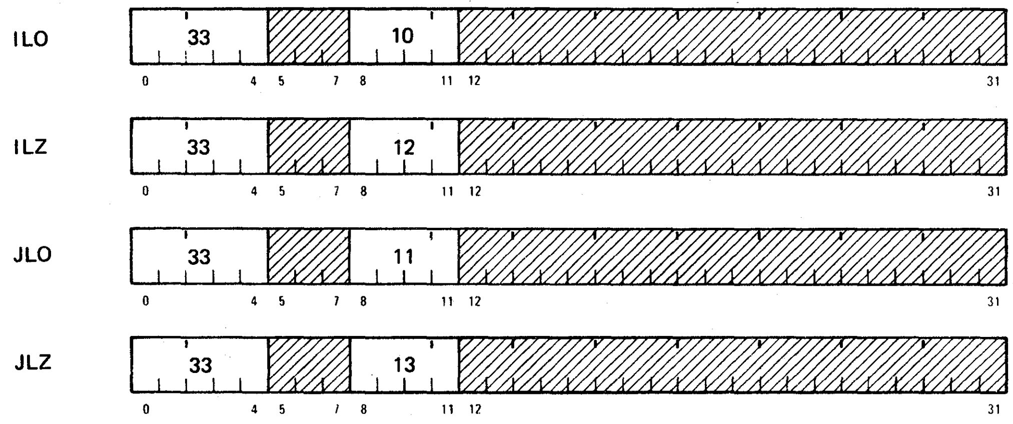
Operation: 64-Bit Mode. RGA is tested for either "all ones" or "all zeros". Which test is indicated by the last letter of the mnemonic code. The test result is placed in the I or J bit in RGD, depending on the first letter of the mnemonic code.

32-Bit Mode. Similar to the 64-bit operation - outer and inner portions of RGA are tested separately. Results for outer and inner words are placed in I and G bits (ILO and ILZ), respectively, or in the J and H bits (JLO and JLZ), respectively [see Section 3.3.3].

Note: Operation of these instructions is unaffected by the configuration of the E and E1 bits.

Register Affected: RGD - I, J, G, and H bits may be changed.

Instruction Words:



Mnemonic Codes: IMG, IME, IML, JMG, JME, JML [redacted]
Test for RGA mantissa "greater than", "equal to", or "less than" second operand mantissa.

First Operand: Found in RGA.

Second Operand: Specified by ADR and ADR USE fields, using operand addressing and indexing logic, as shown in Table 3-2.

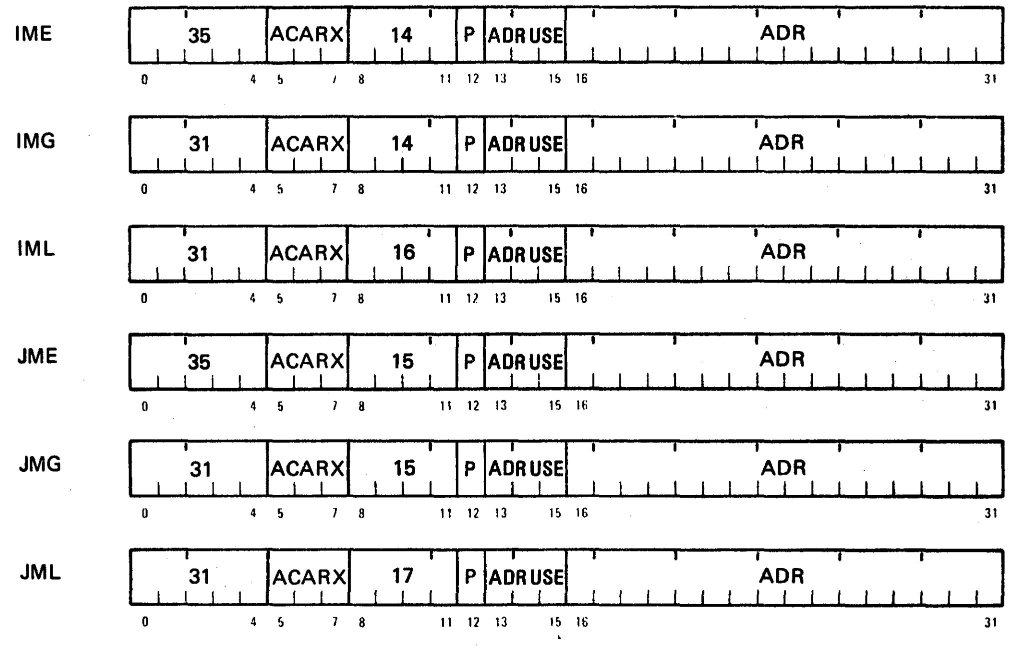
Operation: 64-Bit Mode. The operand mantissas are tested to find RGA mantissa "greater than", "equal to", or "less than" second operand mantissa. Which test is indicated by the last letter of the mnemonic code. The test result is placed in the I or J bit in RGD, depending on the first letter of the mnemonic code.

32-Bit Mode. Similar to the 64-bit operation - outer and inner mantissas are tested independently. Results for outer and inner operands are placed in I and G bits (IMG, IME, and IML), respectively, or in J and H bits (JMG, JME, and JML), respectively [see Section 3.3.3].

Note: Operation of these instructions is unaffected by the configuration of the E and E1 bits.

Registers Affected: RGD - I, J, G, and H bits may be changed.
RGB - Contains second operand.

Instruction Words:



These instructions do not use the sign bits in the operands.

Mnemonic Codes: IMO, IMZ, JMO, JMZ

Test RGA mantissa for "all ones" or "all zeros".

Operation: 64-Bit Mode. The mantissa field of RGA is tested for either "all ones" or "all zeros". Which test is indicated by the last letter of the mnemonic code. The test result is placed in the I or J bit in RGD, depending on the first letter of the mnemonic code.

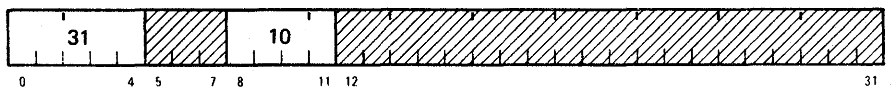
32-Bit Mode. Similar to the 64-bit operation - outer and inner mantissa fields of RGA are tested independently. Results for outer and inner mantissas are placed in I and G bits (IMO and IMZ), respectively, or in J and H bits (JMO and JMZ), respectively [see Section 3.3.3].

> *Note: Operation of these instructions is unaffected by the configuration of the E and E1 bits.*

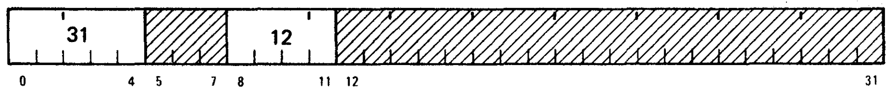
Register Affected: RGD - I, J, G, and H bits may be changed.

Instruction Words:

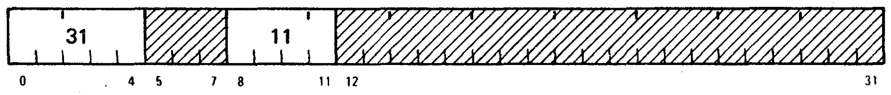
IMO



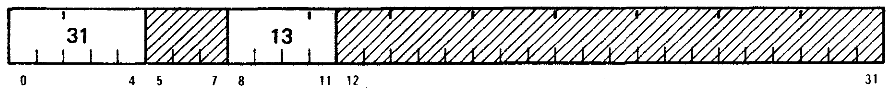
IMZ



JMO



JMZ



Mnemonic Codes: ISG, ISE, ISL, JSG, JSE, JSL [redacted]

Test for RGS bits 48:16 "greater than", "equal to", or "less than" second operand.

First Operand: Found in RGS. Only the 16 least significant bits are used.

Second Operand: Specified by ADR and ADR USE fields, using operand addressing and indexing logic, as shown in Table 3-2. Only the 16 least significant bits are used.

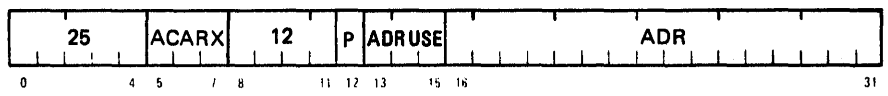
Operation: 64-Bit and 32-Bit Modes (Identical). The two 16-bit operands are tested to find RGS "greater than", "equal to", or "less than" the second operand. Which test is indicated by the last letter in the mnemonic code. The result of the test is placed in the I bit (ISG, ISE, and ISL) or the J bit (JSG, JSE, and JSL). The second operand (all 64 bits) is left in RGB.

Note: Operation of these instructions is unaffected by the configuration of the E and E1 bits.

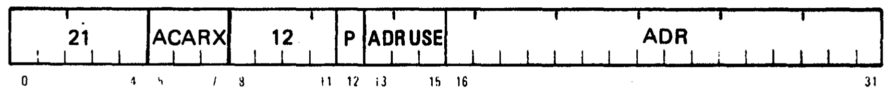
Registers Affected: RGD - I and J bits may be changed.
RGB - Contains 64-bit second operand.

Instruction Words:

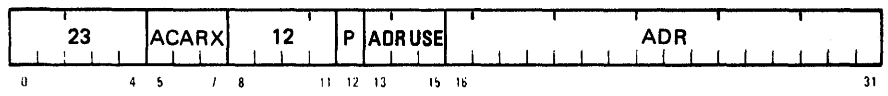
ISE



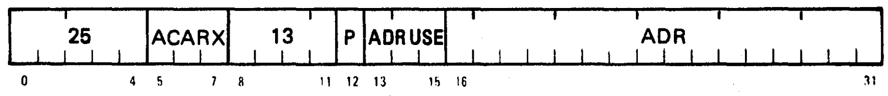
ISG



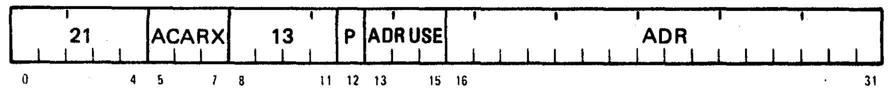
ISL



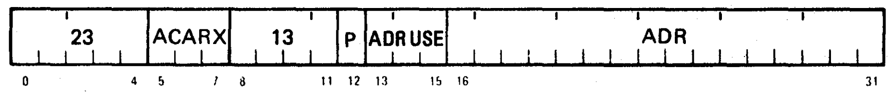
JSE



JSG



JSL



Mnemonic Codes: ISN, JSN [redacted]

Transfer RGA sign bit to RGD bit I or J.

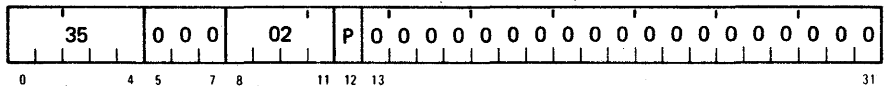
Operation: <u>64-Bit Mode</u>. RGA bit 0 (sign bit of 64-bit word) is transferred to RGD bit I or J, depending on the first letter of the mnemonic code.

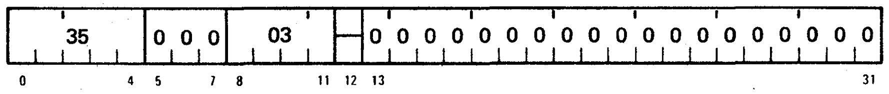
<u>32-Bit Mode</u>. RGA bits 0 and 8 (sign bits of outer and inner 32-bit words) are transferred to I and G bits (ISN instruction), respectively, or to J and H bits (JSN instruction), respectively [see Section 3.3.3].

Note: Operation of these instructions is unaffected by the configuration of the E and E1 bits.

Registers Affected: RGD - I, J, G, and H bits may be changed.
RGB - Contains all zeros.

Instruction Words:

ISN


JSN


The ISN and JSN instructions are the same as the IB and JB instructions. ADR and ADR USE fields are equal to 0 in order to specify the sign bit(s).

Mnemonic Codes: IXG, IXE, IXL, JXG, JXE, JXL [redacted]
Test for RGX "greater than", "equal to", or "less than" second operand.

First Operand: Found in RGX.

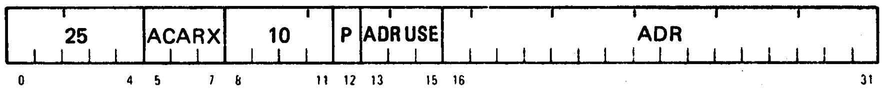
Second Operand: Specified by ADR and ADR USE fields, using operand addressing and indexing logic, as shown in Table 3-2. Only the 16 least significant bits are used.

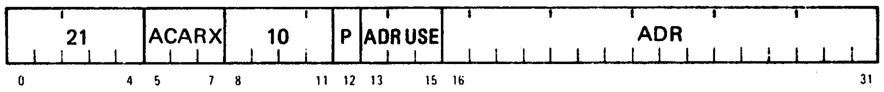
Operation: <u>64-Bit and 32-Bit Modes (Identical)</u>. The two 16-bit operands are tested to find RGX "greater than", "equal to", or "less than" the second operand. Which test is indicated by the last letter in the mnemonic code. The result of the test is placed in the I bit (IXG, IXE, and IXL) or in the J bit (JXG, JXE, and JXL). The second operand (all 64 bits) is left in RGB.

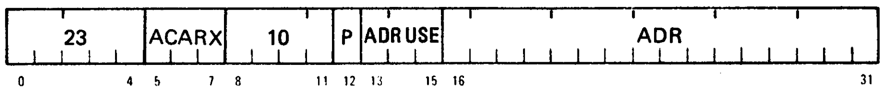
Note: Operation of these instructions is unaffected by the configuration of the E and E1 bits.

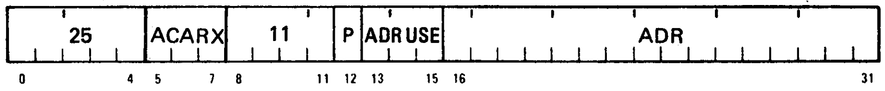
Registers Affected: RGD - I and J bits may be changed.
RGB - Contains 64-bit second operand.

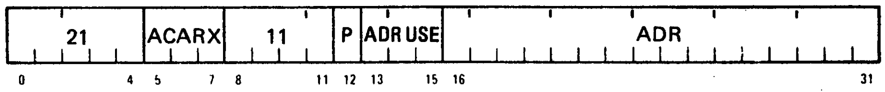
Instruction Words:

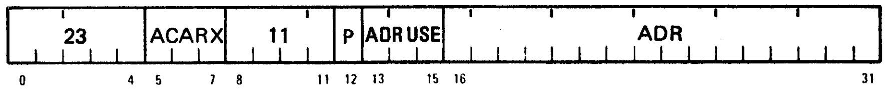
IXE


IXG


IXL


JXE


JXG


JXL


Mnemonic Codes: IXGI, JXGI [redacted]
Add to RGX and place carry in I or J bit.

First Operand: Found in RGX.

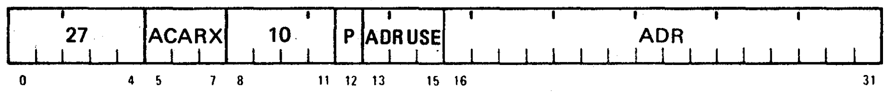
Second Operand: Specified by ADR and ADR USE fields, using operand addressing and indexing logic, as shown in Table 3-2. Only the 16 least significant bits are used.

Operation: 64-Bit and 32-Bit Modes (Identical). The two 16-bit operands are added, and the result is placed in RGX. The high-order carry is placed in the I bit (IXGI) or the J bit (JXGI). If the E bit is reset (=0), RGX will be unchanged, but the carry will still be stored in I or J. The second operand (all 64 bits) is left in RGB.

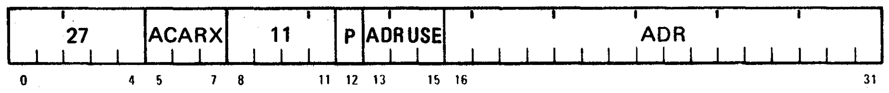
Registers Affected: RGX - Contains sum, as enabled by the E bit.
RGD - I or J bit contains high-order carry.
RGB - Contains 64-bit second operand.

Instruction Words:

IXGI



JXGI



The correct interpretation of the carry stored in the I or J bit depends on the operand values.

Mnemonic Codes: IXLD, JXLD [redacted]
Subtract from RGX and place complement of carry in I or J bit.

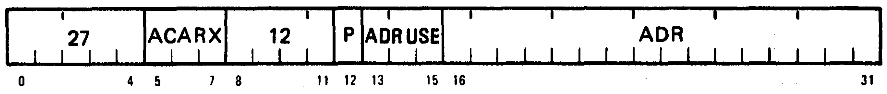
First Operand: Found in RGX.

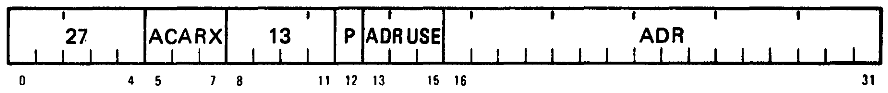
Second Operand: Specified by ADR and ADR USE fields, using operand addressing and indexing logic, as shown in Table 3-2. Only the 16 least significant bits are used.

Operation: <u>64-Bit and 32-Bit Modes (Identical)</u>. The second 16-bit operand is subtracted from RGX using twos-complement arithmetic, and the result is placed in RGX. The complement of the high-order carry is placed in the I bit (IXLD) or the J bit (JXLD). If the E bit is reset (=0), RGX will remain unchanged and the complement of the carry will still be stored in I or J. The second operand (eleven 64 bits) is left in RGB.

Registers Affected: RGX - Contains difference, as enabled by E bit.
RGD - I or J bit contains complement of high-order carry.
RGB - Contains 64-bit second operand.

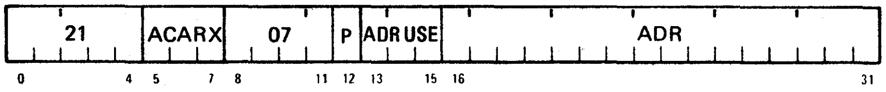
Instruction Words:

IXLD


JXLD


The correct interpretation of the carry stored in the I or J bit depends on the operand values.

Mnemonic Code:	LB [redacted] Test bytes for RGA byte "less than" second operand byte.
First Operand:	Found in RGA.
Second Operand:	Specified by ADR and ADR USE fields, using operand addressing and indexing logic, as shown in Table 3-2.
Operation:	64-Bit and 32-Bit Modes. Each 8-bit byte in RGA is compared with the corresponding byte in the second operand. If the RGA byte is less than the second operand byte, the least significant bit of the RGA byte is set (=1), and the other bits in the byte are cleared. If the RGA byte is not less than the second operand byte, all bits in the RGA byte are cleared. The OFB instruction transfers the contents of RGC to the least significant bits of the corresponding bytes in RGB and clears the outer bits of RGB. 16-bit, 24-bit, 32-bit (etc...) extended precision from 8 bit mode is facilitated by the proper combination of byte mode instructions. If the E bit is reset (=0), RGA bytes 0, 5, 6, and 7 will be unchanged. If the E1 bit is reset, RGA bytes 1, 2, 3, and 4 will be unchanged.
Registers Affected:	RGA - Contains test results in the least significant bit of each byte with other bits cleared, as enabled by E and E1.
	RGC - Contains test results in the most significant bit of each byte with other bits cleared. [Refer also to the comments following the ADB instruction.]
	RGB - Contains second operand.
Instruction Word:	



Mnemonic Codes: LDA, LDB, LDD, LDR, LDS, LDX [redacted]

Load specified PE register from specified source.

Source: Specified by ADR and ADR USE fields, using operand addressing and indexing logic, as shown in Table 3-2.

Operation: 64-Bit and 32-Bit Modes. Load the specified register from the specified source. The destination register is specified by the OP code, as indicated by the last letter of the mnemonic code.

RGD is 8 bits long and can be transferred *only* to or from RGB. Transfers to RGD (via the LDD instruction) are from the 8 most significant bits of RGB; transfers from RGD are to the 8 most significant bits of the RGB; the remaining bits of RGB are unspecified.

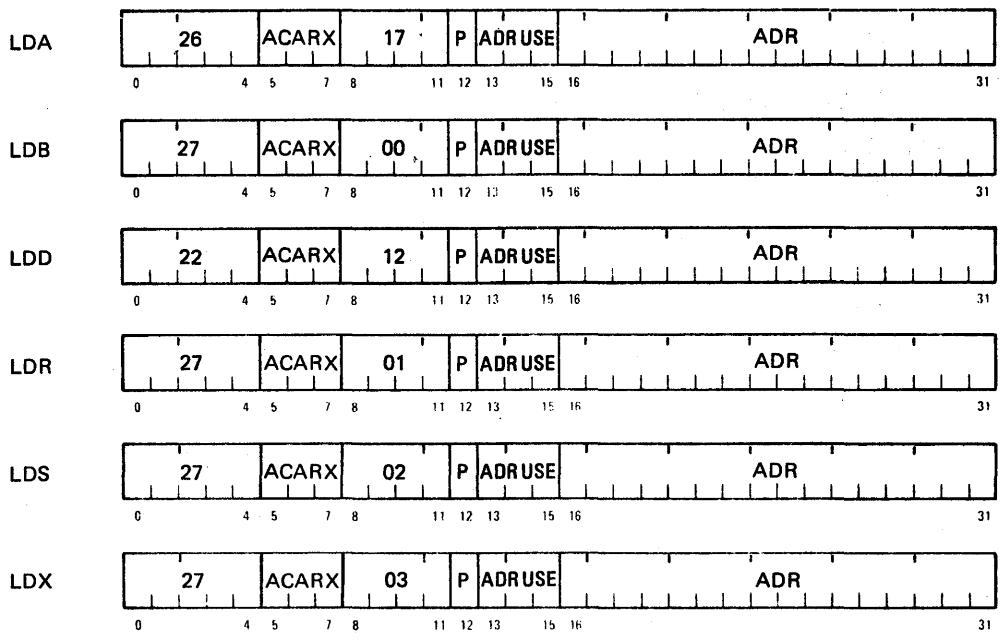
Note: Individual bits of RGD may be accessed by other instructions.

RGX is 16 bits long. Transfers to RGX (via the LDX instruction) are from the 16 least significant bits of the source; transfers from RGX are to the least significant 16 bits of the destination register; the remaining bits of the destination register are cleared.

In the LDA and LDS instructions, if the E bit is reset (=0), the outer portion of RGA or RGS will be unchanged; if the E1 bit is reset, the inner portion of RGA or RGS will be unchanged. In the LDX instruction, if the E bit is reset, RGX will be unchanged.

Register Affected: Destination register specified by OP code. Alteration is specified by the instruction and enabled by E and E1 in cases involving the LDA, LDS, and LDX instructions.

Instruction Words:



LDA
26 | ACARX | 17 | P | ADR USE | ADR
0 4 5 7 8 11 12 13 15 16 31
LDB
27 | ACARX | 00 | P | ADR USE | ADR
0 4 5 7 8 11 12 13 15 16 31
LDD
22 | ACARX | 12 | P | ADR USE | ADR
0 4 5 7 8 11 12 13 15 16 31
LDR
27 | ACARX | 01 | P | ADR USE | ADR
0 4 5 7 8 11 12 13 15 16 31
LDS
27 | ACARX | 02 | P | ADR USE | ADR
0 4 5 7 8 11 12 13 15 16 31
LDX
27 | ACARX | 03 | P | ADR USE | ADR
0 4 5 7 8 11 12 13 15 16 31

The following chart shows the permissible variations of and restrictions on these instructions in terms of source/destination combinations. A mnemonic code means that the combination can be used normally; an "U" means that the combination is illegal or will produce unspecified results.

	Destination Register					
Source	RGA	RGB	RGD	RGR	RGS	RGX
RGA	U	LDB	U	LDR	LDS	U
RGB	LDA	U	LDD	LDR	LDS	LDX
RGD	U	LDB	U	U	U	U
RGR	LDA	LDB	U	U	LDS	LDX
RGS	LDA	LDB	U	LDR	U	LDX
RGX	LDA	LDB	U	LDR	LDS	U
PM	LDA	LDB	U	LDR	LDS	LDX
Literal	LDA	LDB	U	LDR	LDS	LDX

Mnemonic Codes: LDE, LDE1, LDEE1, LDG, LDH, LDI, LDJ [redacted]
Load RGD bit from literal.

Operation: 64-Bit and 32-Bit Modes. Using a 64-bit literal specified by the ACARX and ADR fields, load a specified bit in RGD with the value of one bit in the literal. The bit in RGD is specified by the OP code, as indicated by the last letter of the mnemonic. LDEE1 loads both E and E1 with the same value. The bit in the literal used to load the RGD bit is the bit corresponding to the PE number of the PE executing the instruction. Hence PE 0 will load an RGD bit from bit 0 of the literal, and so forth.

Note: The operation of these instructions is unaffected by the configuration of the E and E1 bits.

ADR: This field is used as a component of a literal. Hence if ACAR-indexing is not used, bits 0:48 of the literal are zeros, and bits 48:16 of the literal are equal to ADR. But if ACAR-indexing is used, bits 0:48 of the literal are equal to bits 0:48 of the ACAR, while bits 48:16 of the literal are the sum of ADR and bits 48:16 of the ACAR (any high-order carry is lost).

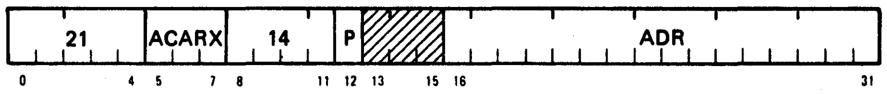
Note: If ACAR-indexing is used and ADR equals 0, then the literal is equal to the 64-bit value found in the ACAR.

ADR USE: This field is ignored and assumed to be zero, thereby indicating the transmission of a literal.

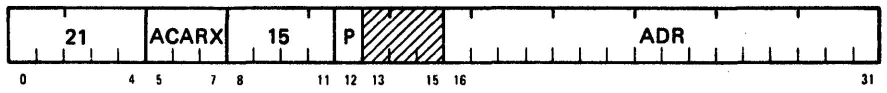
Register Affected: RGD - Altered as specified by the instruction.

Instruction Words:

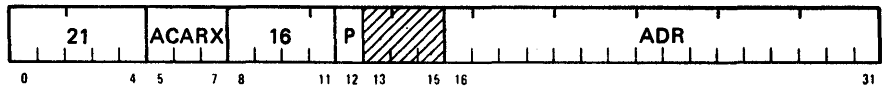
LDE



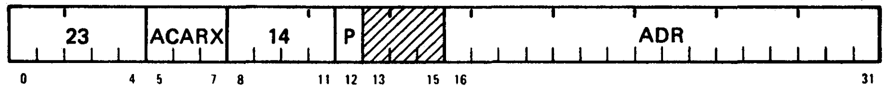
LDE1



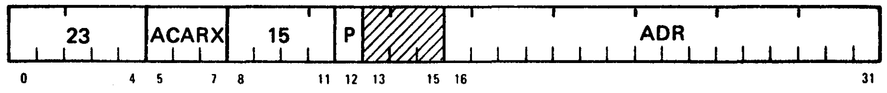
LDEE1



LDG

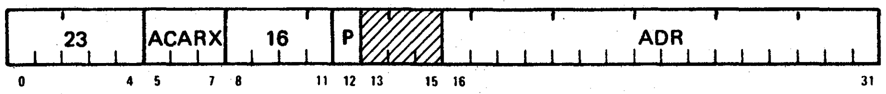


LDH

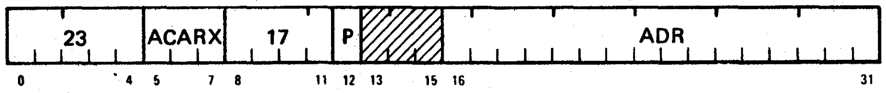


Instruction Words (Cont.):

LDI



LDJ



Mnemonic Code: LEX [redacted]

Load RGA exponent from second operand exponent.

First Operand: Found in RGA.

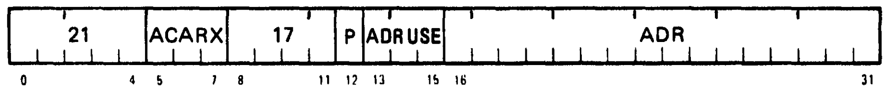
Second Operand: Specified by ADR and ADR USE fields, using operand addressing and indexing logic, as shown in Table 3-2.

Operation: 64-Bit Mode. The exponent field of the second operand is transferred to the exponent field of RGA. Other fields in RGA are unchanged. If E and E1 bits are both reset (=0), RGA will be unchanged. If E ≠ E1, the results are unspecified.

32-Bit Mode. The outer and inner exponent fields of the second operand are transferred to the outer and inner exponent fields of RGA. Other fields in RGA are unchanged. If the E bit is reset, the outer exponent in RGA will be unchanged. If the E1 bit is reset, the inner exponent field in RGA will be unchanged.

Registers Affected: RGA - Exponent field(s) altered, as enabled by E and E1 bits.
RGB - Contains second operand.

Instruction Word:



Mnemonic Codes: ML, MLA, MLM, MLMA, MLN, MLNA, MLR, MLRA, MLRM, MLRMA, MLRN, MLRNA

Multiply floating-point numbers or mantissa-sized fixed-point numbers.

First Operand: Found in RGA.

Second Operand: Specified by ADR and ADR USE fields, using operand addressing and indexing logic, as shown in Table 3-2.

Operation: 64-Bit Mode. The two operands are multiplied. The sign, exponent, and 48 high-order mantissa bits are left in RGA, while 48 low-order mantissa bits are left in the mantissa field of RGB. If both E and E1 bits are reset (=0), RGA will be unchanged but other registers will be affected. If E ≠ E1, the results are unspecified.

32-Bit Mode. Outer and inner words are multiplied independently. Single-precision outer and inner results are left in RGA. The 48-bit mantissa for the outer word is left in RGB (with an exponent of -1). If either E or E1 is reset, the corresponding portion of RGA will be unchanged but other registers will be affected.

Registers Affected:
RGA - Contains high-order results, as enabled by E and E1.
RGB - Contains 48-bit mantissa as described above (with 0 sign and -1 exponent).
RGD - F and F1 bits may be set [see Section 3.3.3 for a more detailed explanation].
RGR - Contains intermediate results.

Instruction Words:

	0–4	5–7	8–11	12	13–15	16–31
ML	31	ACARX	04	P	ADR USE	ADR
MLA	31	ACARX	05	P	ADR USE	ADR
MLM	30	ACARX	14	P	ADR USE	ADR
MLMA	30	ACARX	15	P	ADR USE	ADR

Instruction Words (Cont.):

	0–4	5–7	8–11	12	13–15	16–31
MLN	30	ACARX	04	P	ADRUSE	ADR
MLNA	30	ACARX	05	P	ADRUSE	ADR
MLR	31	ACARX	06	P	ADRUSE	ADR
MLRA	31	ACARX	07	P	ADRUSE	ADR
MLRM	30	ACARX	16	P	ADRUSE	ADR
MLRMA	30	ACARX	17	P	ADRUSE	ADR
MLRN	30	ACARX	06	P	ADRUSE	ADR
MLRNA	30	ACARX	07	P	ADRUSE	ADR

Caution: The contents of RGR are altered by these instructions.

The following variations are permissable on the preceding combinations.

(a) No suffix - Both operands are treated as signed floating-point numbers. No rounding or normalization.

(b) A - Signs of both operands are ignored. The sign of the result is the same as the sign of the first operand. A result of -0 is possible.

(c) M - Exponents of both operands are ignored; that is, both are treated as mantissa-sized, fixed-point numbers. The exponent of the result is the same as the exponent of the first operand. A result of -0 is possible.

(d) N - Result is normalized (after rounding, if specified). If both operands are not normalized, the result will not be normalized.

(e) R - Result is rounded in RGA. RGB mantissa field(s) will be cleared.

[See Sections 3.3.4 and 3.3.5 for further details].

Mnemonic Code: MULT

Multiply 32-bit, signed, floating-point numbers.

First Operand: Found in RGA. Assumed to be normalized.

Second Operand: Specified by ADR and ADR USE, using operand addressing and indexing logic, as shown in Table 3-2. Assumed to be normalized.

Operation: 64-Bit Mode. Operation in 64-bit mode is the same as in 32-bit mode. If the operands are not formatted as 32-bit inner/outer pairs, the results will be meaningless.

32-Bit Mode. The two pairs of operands (inner and outer) are treated as signed, floating-point numbers and are multipled (without rounding but with normalization).

Note: Both operands are assumed to be normalized.

The result for the outer operands is left in RGB. The sign is in bit 0, the exponent is in bits 1:7, and the double-length mantissa is in bits 16:48. In other words, the sign and exponent are in the normal fields for a 32-bit outer word, and the double-length mantissa is in the normal position for a 48-bit mantissa. Bits 8:8 are cleared.

The result for the inner operands is left in RGA. The sign is in bit 8, the exponent is in bits 9:7, and the double-length mantissa is in bits 16:48. Thus the sign and exponent are in the normal fields for a 32-bit inner word, and the mantissa is in the normal position for a 48-bit mantissa. Bits 0:8 are cleared.

If either the E or the E1 bit is reset (=0), the result in RGA is unspecified. If both E and E1 are reset, RGA will be unchanged but other registers will be changed.

Registers Affected:

RGA - Contains inner word result, as described above.

RGB - Contains outer word result, as described above.

RGD - F or F1 bits may be set [see Section 3.3.4 for a more detailed explanation].

Instruction Word:

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

[See Section 3.3.4 for further details.]

Mnemonic Code: NEB

Test bytes for RGA byte "not equal to" second operand byte.

First Operand: Found in RGA.

Second Operand: Specified by ADR and ADR USE fields, using operand addressing and indexing logic, as shown in Table 3-2.

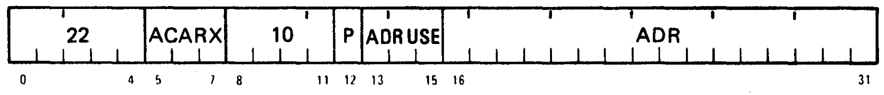
Operation: 64-Bit and 32-Bit Modes. Each 8-bit byte in RGA is compared with the corresponding byte in the second operand. If the RGA byte is not equal to the second operand byte, the least significant bit of the RGA byte is set (=1) and the other bits in the byte are cleared. If the RGA byte is equal to the second operand byte, all bits in the RGA byte are cleared. If the E bit is reset, RGA bytes 0, 5, 6, and 7 will be unchanged. If the E1 bit is reset, RGA bytes 1, 2, 3, and 4 will be unchanged.

Registers Affected: RGA - Contains test results in the least significant bit of each byte with other bits cleared, as enabled by E and E1.

RGC - Contains test results in the most significant bit of each byte with other bits cleared. [Refer also to the comments following the ADB instruction.]

RGB - Contains second operand.

Instruction Word:



The OFB instruction transfers the contents of RGC to the least significant bit of the corresponding byte in RGB and clears the outer bits of RGB. 16 bit, 24 bit, 32 bit, etc. ... extended precision from the 8 bit mode is facilitated by combinations of byte mode instructions.

Mnemonic Code: NORM

Normalize.

Operation: <u>64-Bit Mode</u>. If the RGA mantissa field does not contain a "1" in the leading (most significant) bit, the mantissa is shifted left (with zeros entering at the right) until a leading "1" is detected. The exponent is then adjusted by the number of bits shifted. The exponent adjustment is left in RGB. If both E and E1 are reset (=0), RGA will be unchanged but RGB will be changed. If E ≠ E1, the results are unspecified. F bits may be set if exponent adjustment caused underflow and ACR09 is set.

<u>32-Bit Mode</u>. Similar to 64-bit mode, except inner and outer words in RGA are normalized independently. The outer and inner exponent adjustments are left in the outer and inner exponent fields of RGB. If either E or E1 is reset, the corresponding portion of RGA will be unchanged.

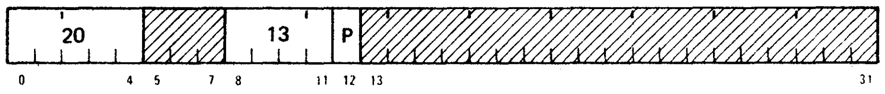
Registers Affected:

RGA - Mantissa and exponent fields altered, as enabled by E and E1.

RGB - Contains exponent adjustment(s).

RGD - F or F1 bits may be set [see Section 3.3.3 for a more detailed explanation].

Instruction Word:



The operation performed by this instruction is the same as the normalization performed on the results from AD, SB, DV, and ML instructions, when the "N" variant has been used. Any number with a zero mantissa (regardless of value of sign and exponent fields) is set to all zeros by NORM.

Mnemonic Code: OFB

Recover carries or test results from previous byte instructions (from RGC to RGB).

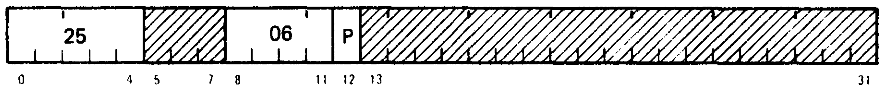
Operation: <u>64-Bit and 32-Bit Modes</u>. The ADB and SBB instructions leave high-order carries in the *most* significant bit of each byte in RGC. The GB, LB, and NEB instructions leave test results in these same bits. The OFB instruction transfers these bits to the *least* significant bits of corresponding bytes in RGB and clears the other bits of RGB. RGC is unchanged.

Because certain instructions may alter the contents of RGC, an OFB instruction should immediately follow the byte instruction whose results or carries are to be recovered.

Note: Operation of this instruction is unaffected by the configuration of the E and E1 bits.

Register Affected: RGB - Least significant bit of each byte contains a copy of most significant bit of corresponding byte in RGC. Other bits are cleared.

Instruction Word:



This instruction is not needed after GB, LB, or NEB, since these instructions automatically place the test results in RGA.

Mnemonic Codes: OR, ORN, NOR, NORN

Logical OR of two 64-bit operands or their complements.

First Operand: Found in RGA.

Second Operand: Specified by ADR and ADR USE fields, using operand addressing and indexing logic, as shown in Table 3-2.

Operation: 64-Bit and 32-Bit Modes. Bitwise logical OR of two operands or their complements, as shown in the following listing, where OP1 is the first operand and OP2 is the second operand.

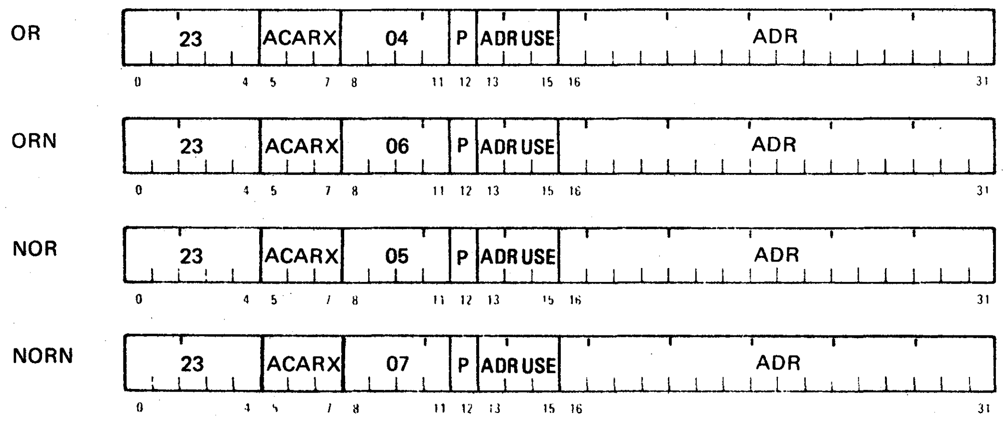
Mnemonic	Function
OR	OP1 OR OP2
ORN	OP1 OR $\overline{\text{OP2}}$
NOR	$\overline{\text{OP1}}$ OR OP2
NORN	$\overline{\text{OP1}}$ OR $\overline{\text{OP2}}$

The 64-bit result is left in RGA. If the E bit is reset (=0), RGA bits 0:8 and 40:24 will be unchanged. If the E1 bit is reset, RGA bits 8:32 will be unchanged.

Registers Affected: RGA - Contains result, as enabled by E and E1.

RGB - Contains second operand.

Instruction Words:



Mnemonic Codes: RAB, SAB

Reset or set specified bit in RGA.

Bit Number: Specified by ADR and ADR USE fields, using bit/shift counting and indexing logic, as shown in Table 3-3.

Operation: <u>64-Bit Mode</u>. The bit number is taken mod 64, and the corresponding bit in RGA is reset (RAB) or set (SAB). Other RGA bits are unchanged. If the E bit is reset (=0), RGA bits 0:8 and 40:24 cannot be changed. If the E1 bit is reset, RGA bits 8:32 cannot be changed.

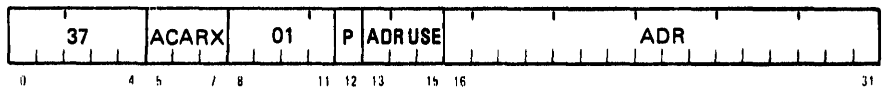
<u>32-Bit Mode</u>. The bit number is taken mod 32 and used to select a bit position within each of the two 32-bit inner and outer words of RGA. For example, if the bit number is 7, it selects bit 7 of the outer word and bit 7 of the inner word (these are RGA bits 7 and 14). The selected bits are set or reset; other RGA bits are unchanged. If the E bit is reset, the bit in the outer word cannot be changed. If the E1 bit is reset, the bit in the inner word cannot be changed.

Registers Affected: RGA - Specified bit(s) set or reset, as enabled by E and E1.

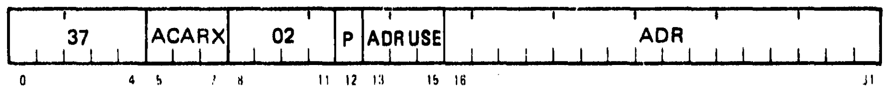
RGB - Contains a mask in which bit(s) corresponding to selected bit(s) in RGA are set and all other RGB bits are reset.

Instruction Words:

RAB



SAB



Mnemonic Codes: RTAL, RTAR

Shift left/right, end-around, logical, single-length.

Shift Count: Specified by ADR and ADR USE fields, using bit/shift counting and indexing logic, as shown in Table 3-3.

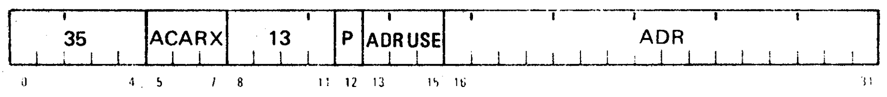
Operation: 64-Bit Mode. RGA is considered as a single 64-bit (i.e., logical) entity and is shifted end-around by the shift count, which is taken mod 64. The direction of the shift is specified by the OP code, as indicated by the last letter of the mnemonic code. If both E and E1 bits are reset (=0), RGA will be unchanged. If E ≠ E1, the result is unspecified.

32-Bit Mode. Similar to the 64-bit operation, except RGA is considered as two 32-bit logical entities (outer and inner) that are shifted independently. The shift count is taken mod 32. If the E bit is reset, the outer portion of RGA will be unchanged. If the E1 bit is reset, the inner portion of RGA will be unchanged.

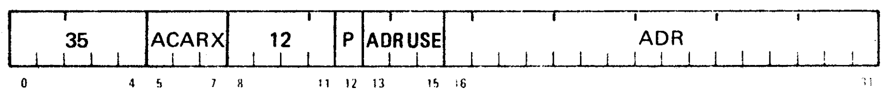
Register Affected: RGA - Altered as specified by the instruction and as enabled by E and E1.

Instruction Words:

RTAL



RTAR



Mnemonic Code: RTL

Send data from specified register to RGR of another PE.

Operation: 64-Bit and 32-Bit Modes. Transmit contents of specified register [see SOURCE Field below] to the RGR register of the PE in position number N + D, where N is the position number of the PE executing the instruction and D is a specified routing distance. N + D is taken mod 64.

> *Note: Operation of this instruction is unaffected by the configuration of the E and E1 bits. Every PE will execute the instruction, and after execution, the RGR register of every PE will contain a value sent from another PE.*

SOURCE Field (Bits 17:5): One bit in this field is set (=1) to specify the source register, as shown in the following listing.

Source Register	Bit Number in Instruction
RGA	17
RGB	18
RGX	19
RGS	20
RGR	21

RGD may not be used as the source register. If more than one bit is set in the SOURCE field - or if none is set - the results are unspecified.

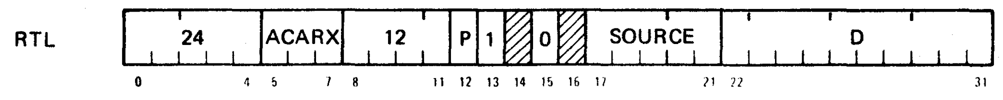
D Field (Bits 22:10): This field contains the routing distance (right-justified).

The SOURCE and D fields in this instruction are really an ADR field that may be altered in the CU by ACAR-indexing. If ACAR-indexing is used, the effective SOURCE and D fields will be the sum of bits 17:15 of the instruction and bits 49:15 of the ACAR.

ADR USE: Bit 13 should be set and bit 15 reset to indicate the presence of a register code. Bit 14 is ignored. If bit 13 is reset or bit 15 is set, the results are unspecified.

Register Affected: RGR - Contains data sent from the source register of PE number N - D mod 64, where N is the number of this PE and D is the routing distance.

Instruction Word:



Mnemonic Codes: SAN, SAP

Set RGA sign negative or positive.

Operation: 64-Bit Mode. RGA bit 0 (sign bit of 64-bit word) is set to "1" (SAN) or reset to "0" (SAP). If the E bit is reset, RGA is unaffected.

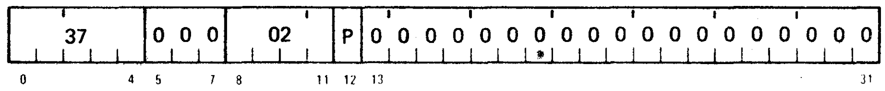
32-Bit Mode. RGA bits 0 and 8 (sign bits of 32-bit outer and inner words) are set to "1" (SAN) or reset to "0" (SAP). If the E bit is reset, RGA bit 0 is unchanged. If the E1 bit is reset, RGA bit 8 is unchanged.

Registers Affected: RGA - Sign bit(s) set or reset, as enabled by E and E1.

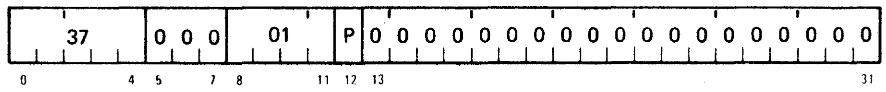
RGB - Contains a mask in which bit 0 and bit 8 (in 32-bit mode) are set and all other RGB bits are reset.

Instruction Words:

SAN



SAP



SAN and SAP are the SAB and RAB instructions, respectively, with ADR and ADR USE equal to 0 to specify the sign bit(s).

Mnemonic Codes: SB, SBA, SBM, SBMA, SBN, SBNA, SBR, SBRA, SBRN, SBRNA [redacted]
Subtract floating-point numbers or mantissa-sized, fixed-point numbers.

First Operand: (Minuend) Found in RGA.

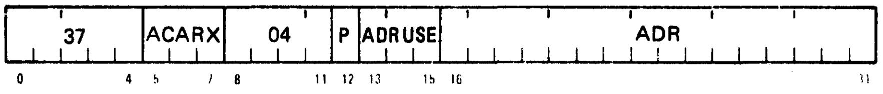
Second Operand: (Subtrahend) Specified by ADR and ADR USE, using operand addressing and indexing logic, as shown in Table 3-2.

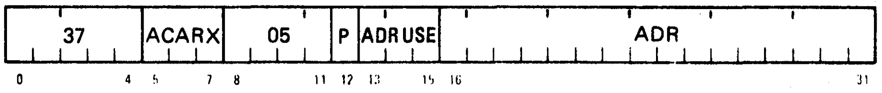
Operation: 64-Bit Mode. Second operand is subtracted from RGA and result is left in RGA. RGB will contain the second operand, which is either unmodified or modified by mantissa portion shifted off to align operands. If both E and E1 bits are reset (=0), RGA will be unchanged but RGB will be changed. If E ≠ E1, the results are unspecified.

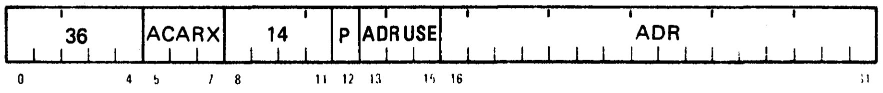
32-Bit Mode. Similar to 64-bit mode, except outer and inner operand pairs are subtracted independently. If E bit is reset, the outer word in RGA will be unchanged. If E1 bit is reset, the inner word in RGA will be unchanged. RGB will be changed in all cases.

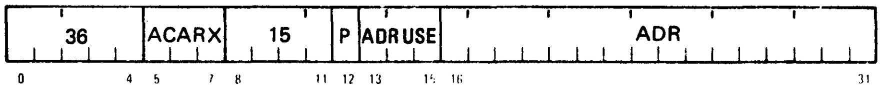
Registers Affected: RGA - Contains result(s), as enabled by E and E1.
RGB - Contains second operand, either unmodified or modified by mantissa portions shifted off to align operands.
RGD - F or F1 bits may be set [see Section 3.3.3 for a more detailed explanation.

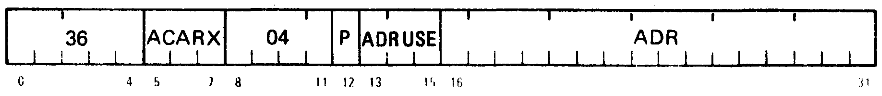
Instruction Words:

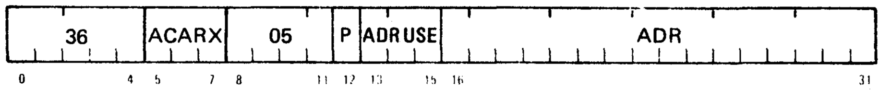
SB


SBA


SBM


SBMA


SBN


SBNA


Instruction Words (Cont.):

SBR

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

SBRA

37	ACARX	07	P	ADR USE	ADR
0 4	5 7	8 11	12	13 15	16 31

SBRN

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

SBRNA

36	ACARX	07	P	ADR USE	ADR
0 4	5 7	8 11	12	13 15	16 31

The SB instructions operate the same as the AD instructions, except that the sign of the second operand is changed before addition is performed.

The following variations are permitted in the preceding combinations.

(a) No suffix - Both operands are treated as signed, floating-point numbers. No rounding or normalization.

(b) A - Signs of both operands are ignored. The sign of the result is the same as the sign of the minuend found in RGA. A result of -0 is possible.

(c) M - Exponents of both operands are ignored; that is, both are treated as mantissa-sized, fixed-point numbers. The exponent of the result is the same as the exponent of the minuend found in RGA. A result of -0 is possible.

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

(e) R - Result is rounded in RGA.

[See Sections 3.3.4 and 3.3.5 for further details.]

Alignment for floating-point subtraction is performed in the following sequence.

1. The exponent of the result is determined as the larger of the two operand exponents (subsequently adjusted if normalization is used).
2. The mantissa of the operand with the smaller exponent is shifted right end-off, until it is properly aligned with the mantissa of the other operand. If the difference between the exponents is greater than 47 (or 23 in 32-bit mode), however, this process will zero the mantissa of the operand with the smaller exponent.
3. The most significant bit shifted off in aligning the mantissa is saved and used for rounding (if specified).

Mnemonic Code: SBB

Subtract bytes using ones-complement arithmetic.

First Operand: Found in RGA.

Second Operand: Specified by ADR and ADR USE fields, using operand addressing and indexing logic, as shown in Table 3-2.

Operation: 64-Bit and 32-Bit Modes. Each operand is considered as eight 8-bit bytes. Each byte of the second operand is complemented and added to the corresponding byte of RGA, and the results are left in RGA. The high order carry from each byte is stored in the most significant bit of the corresponding byte in RGC. The OFB instruction transfers the corresponding bytes in RGB and clears the outer bits of RGB. 16-bit, 23-bit, 32-bit (etc...) extended precision from the 8 bit mode is facilitated by combinations of byte mode instructions.

Bytes 0, 5, 6, and 7 will be unchanged. If the E1 bit is reset, RGA bytes 1, 2, 3, and 4 will be unchanged. Furthermore, the carries from these bytes will still be stored in RGC.

Registers Affected: RGA - Contains results, as enabled by E and E1.

RGC - Contains carries (for positive results) in the most significant bit of each byte, with other bits cleared.

RGB - Contains second operand.

Instruction Word:

26	ACARX	07	P	ADR USE	ADR
0 4	5 7	8 11	12	13 15	16 31

Mnemonic Code: SBEX

Subtract exponents.

First Operand: Found in RGA.

Second Operand: Specified by ADR and ADR USE fields, using operand addressing and indexing logic, as shown in Table 3-2.

Operation: 64-Bit Mode. Subtract exponent field of second operand from RGA exponent field, thus treating these fields as offset exponents rather than as binary numbers. Sign bit and mantissa in RGA are unchanged unless exponent underflow occurs (in which case RGA is cleared). If both E and E1 bits are reset, RGA is unchanged. If E $\neq$ E1, the results are unspecified. The second operand is left in RGB in all cases.

32-Bit Mode. Similar to 64-bit mode, except the inner and outer exponents are subtracted independently. If the E bit is reset, the outer exponent in RGA will be unchanged. If the E1 bit is reset, the inner exponent in RGA will be unchanged. In all cases, the second operand is left in RGB.

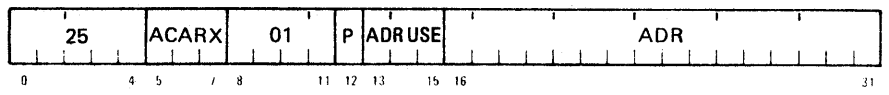
Registers Affected:

RGA - Exponent field(s) altered, as enabled by E and E1. Exponent underflow causes RGA or inner or outer word to be cleared, as enabled by E and E1.

RGB - Contains second operand.

RGD - F or F1 bits may be set [see Section 3.3.3 for a more detailed explanation].

Instruction Word:



Exponents are represented by offset code. This instruction makes the necessary adjustments for the offset, effectively subtracts the true exponents, and then puts the result (in offset code) into RGA.

Mnemonic Codes: SETE, SETE1, SETF, SETF1, SETG, SETH, SETI, SETJ [redacted]
Set RGD bit to value of logical function of two RGD bits.

First Operand: Value of a bit in RGD, specified by the setting of one bit in the B1 field of the instruction [see B1 listing below].

Second Operand: Value or complement of the E or E1 bit, as specified by the setting of one bit in the B2 field of the instruction [see B2 listing below].

Operation: 64-Bit and 32-Bit Modes. Load a bit in RGD with the value of a logical function of the two operands. The bit to be loaded is specified by the OP code, as indicated by the last letter of the mnemonic code. The logical function is specified by the LOG FUNC field of the instruction [see LOG FUNC, p. 3-66].

Note: The operation of these instructions is unaffected by the configuration of the E and E1 bits.

B1 Field (Bits 24:8): One of these bits is set to specify the first operand bit, as shown in the following listing.

First Operand Value	Bit Number in Instruction
H	24
G	25
J	26
I	27
E1	28
E	29
F1	30
F	31

If more than one bit is set in B1 or if no bits are set in B1, the results are unspecified.

B2 Field (Bits 20:4): One of these bits is set to specify the second operand bit, as shown in the following listing.

Second Operand Value	Bit Number in Instruction
$\overline{E1}$	20
E1	21
$\overline{E}$	22
E	23

If no bits are set in B2, the results are unspecified. If all bits are set in B2, the second operand value will be 0.

LOG FUNC Field (Bits 16:4): One of these bits is set to specify the logical function used. For example, the following listing shows that BIT1 is the value of the first operand (as specified by the B1 field) and BIT2 is the value of the second operand (as specified by the B2 field).

Logical Function of BIT1 and BIT2	Bit Number in Instruction
BIT1 OR BIT2	16
$\overline{\text{BIT1}}$ OR BIT2	17
BIT1 AND BIT2	18
$\overline{\text{BIT1}}$ AND BIT2	19

If more than one bit is set in the LOG FUNC field, the results are unspecified. If no bits are set in the LOG FUNC field, the logical function is BIT1 OR BIT2.

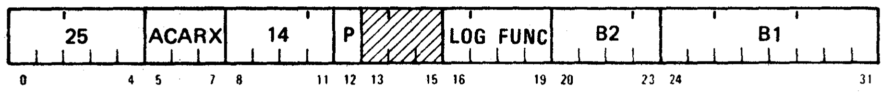
ACAR-Indexing: The B1, B2, and LOG FUNC fields are actually an ADR field treated as a component of a literal. PE-indexing is not allowed, but ACAR-indexing may normally be used. Only the last 16 bits (bits 48:16) of the 64-bit literal are used in executing these instructions. If ACAR-indexing is used, these bits will be the sum of instruction bits 16:16 and ACAR bits 48:16.

ADR USE: Bits 13:3 are ignored. They are also assumed to be all zeros, thus indicating transmission of a literal.

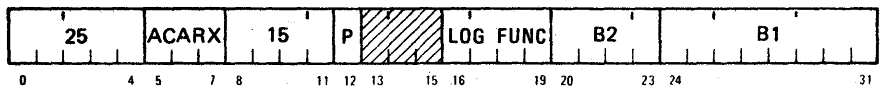
Register Affected: RGD - Altered, as specified by the instruction.

Instruction Words:

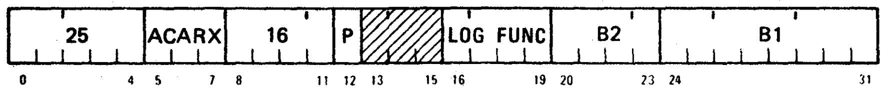
SETE



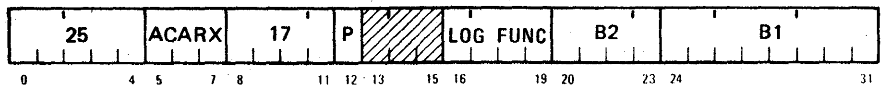
SETE1



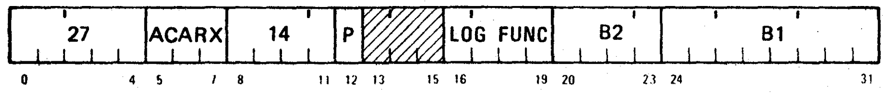
SETF



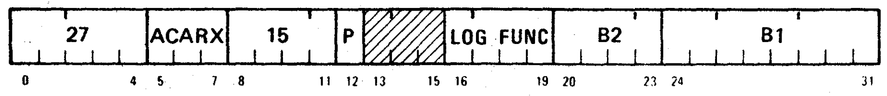
SETF1



SETG

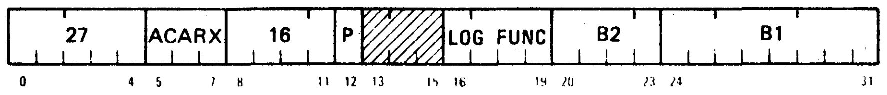


SETH

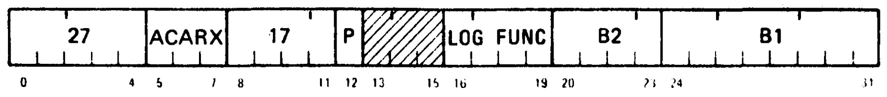


Instruction Words (Cont.):

SETI



SETJ



Mnemonic Codes: SHABL, SHABR [redacted]

Shift left/right, end-off, logical, double-length.

Shift Count: Specified by ADR and ADR USE fields, using the bit/shift counting and indexing logic shown in Table 3-3.

Operation: 64-Bit Mode. RGA and RGB are considered as the left and right halves, respectively, of a single 128-bit logical entity. This entity is shifted end-off (zero fill) by the shift count, which is taken mod 64. The direction of the shift is specified by the OP code, as indicated by the last letter of the mnemonic code. If both E and E1 bits are reset, RGA will be unchanged. RGB, however, will be altered irrespective of the E and E1 bits. If E $\neq$ E1, the result is unspecified.

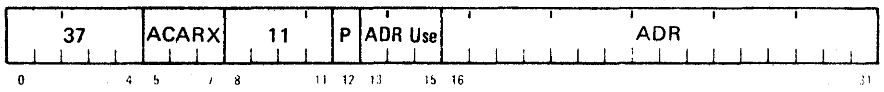
32-Bit Mode. In 32-bit mode, these instructions produce unspecified results.

Registers Affected: RGA - Altered, as specified by the instruction and as enabled by E and E1.

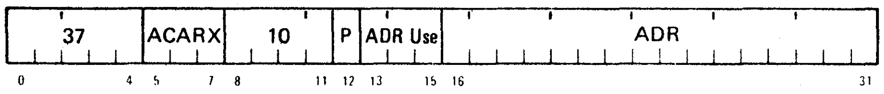
RGB - Altered, as specified by the instruction.

Instruction Words:

SHABL



SHABR



Mnemonic Codes: SHABML, SHABMR [redacted]
Shift left/right, end-off, mantissa only, double-length.

Shift Count: Specified by ADR and ADR USE fields, using the bit/shift counting and indexing logic shown in Table 3-3.

Operation: <u>64-Bit Mode</u>. RGA and RGB mantissa fields are considered as the left and right halves, respectively, of a single 96-bit logical entity. This entity is shifted end-off (zero fill) by the shift count, which is taken mod 64. The direction of the shift is specified by the OP code, as indicated by the last letter of the mnemonic code. If both E and E1 bits are reset, RGA will be unchanged. RGB, however, will be altered irrespective of the E and E1 bits. If E ≠ E1, the result is unspecified.

Note: If the shift count (mod 64) is greater than 47, the RGA and RGB mantissa fields will be cleared.

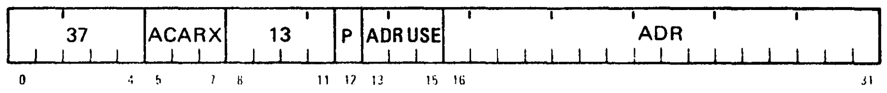
<u>32-Bit Mode</u>. In 32-bit mode, these instructions produce unspecified results.

Registers Affected: RGA - Mantissa field (bits 16:48) altered, as specified by the instruction and as enabled by E and E1.

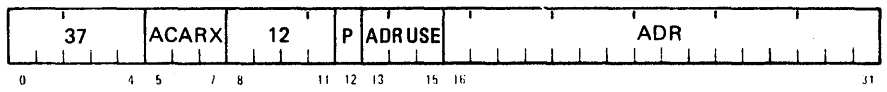
RGB - Mantissa field (bits 16:48) altered, as specified by the instruction.

Instruction Words:

SHABML



SHABMR



Mnemonic Codes: SHAL, SHAR [redacted]
Shift left/right, end-off, logical, single-length.

Shift Count: Specified by ADR and ADR USE fields, using the bit/shift counting and indexing logic shown in Table 3-3.

Operation: 64-Bit Mode. RGA is considered as a single 64-bit (i.e., logical) entity and is shifted end-off (zero fill) by the shift count, which is taken mod 64. The direction of the shift is specified by the OP code, as indicated by the last letter of the mnemonic code. If both E and E1 bits are reset, RGA will be unchanged. If E ≠ E1, the result is unspecified.

32-Bit Mode. Similar to the 64-bit operation, except RGA is considered as two 32-bit logical entites (outer and inner) that are shifted independently - the shift count is taken mod 32. If the E bit is reset, the outer portion of RGA will be unchanged. If the E1 bit is reset, the inner portion of RGA will be unchanged.

Registers Affected: RGA - Altered, as specified by the instruction and as enabled by E and E1.

Instruction Words:

SHAL

0–4	5–7	8–11	12	13–15	16–31
35	ACARX	01	P	ADR USE	ADR

SHAR

0–4	5–7	8–11	12	13–15	16–31
35	ACARX	00	P	ADR USE	ADR

Mnemonic Codes: SHAML, SHAMR [redacted]

Shift left/right, end-off, mantissa only, single-length.

Shift Count: Specified by ADR and ADR USE, using the bit/shift counting and indexing logic shown in Table 3-3.

Operation: 64-Bit Mode. The RGA mantissa field (48 bits) is shifted end-off (zero fill) by the shift count, which is taken mod 64. The direction of the shift is specified by the OP code, as indicated by the last letter of the mnemonic code. If both E and E1 bits are reset, RGA will be unchanged. If E ≠ E1, the result is unspecified.

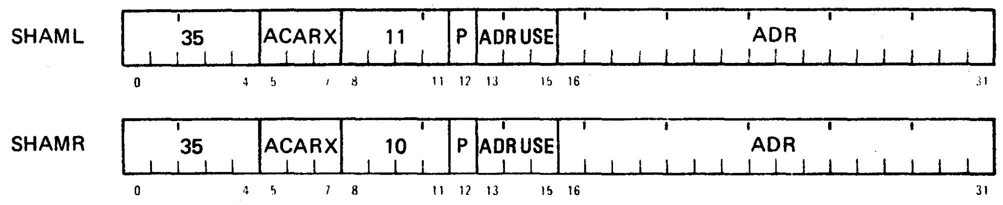
Note: If the shift count (mod 64) is greater than 47, the RGA mantissa field will be cleared.

32-Bit Mode. Similar to the 64-bit operation, except RGA outer and inner mantissas are shifted independently. The shift count is taken mod 32. If the E bit is reset, the outer mantissa in RGA will be unchanged. If the E1 bit is reset, the inner mantissa in RGA will be unchanged.

Note: If the shift count (mod 32) is greater than 23, the RGA mantissas will be cleared.

Register Affected: RGA - Mantissa field(s) altered, as specified by the instruction and as enabled by E and E1.

Instruction Words:



Mnemonic Codes: STA, STB, STR, STS, STX

Store from specified register to processor memory.

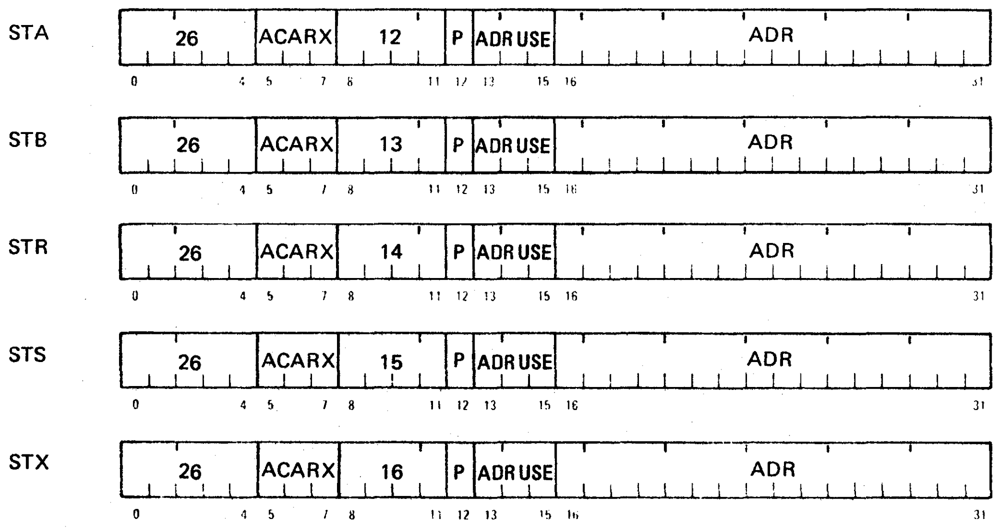
Destination: Specified by ADR and ADR USE fields, using the operand addressing and indexing logic shown in Table 3-2, with the restriction that a processor memory location must be specified (instruction bit 15 must be set).

Operation: <u>64-Bit and 32-Bit Modes</u>. Store the contents of the specified PE register into the processor memory location specified as its destination. The PE register is specified by the OP code, as indicated by the last letter of the mnemonic code. If the E bit is reset, the outer portion of the processor memory location will be unchanged. If the E1 bit is reset, the inner portion will be unchanged.

The RGX register is 16 bits long. The STX instruction stores these 16 bits in bits 48:16 of the processor memory location and clears bits 0:48.

Registers Affected: The processor memory location contains the contents of the source register, as enabled by E and E1.

Instruction Words:



Mnemonic Code: SUB

Subtract 64-bit unsigned integers using ones-complement arithmetic.

First Operand: Found in RGA.

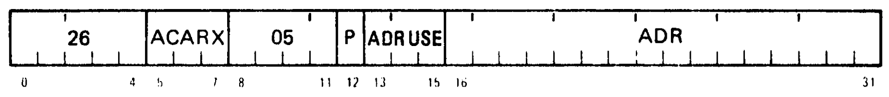
Second Operand: Specified by ADR and ADR USE fields, using the operand addressing and indexing logic shown in Table 3-2.

Operation: <u>64-Bit and 32-Bit Modes (Identical)</u>. The second operand is complimented and added to RGA, and the result is left in RGA. Overflow causes an end around carry. The second operand is left in RGB. If both E and E1 bits are reset, RGA will be unchanged but RGB will be changed. If E $\neq$ E1, the result is unspecified.

Registers Affected: RGA - Contains result, as enabled by E and E1.

RGB - Contains second operand.

Instruction Word:



Mnemonic Code: SWAP

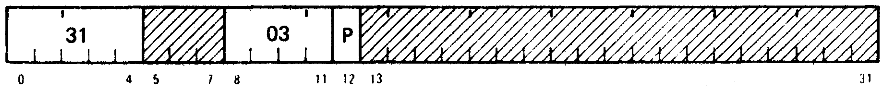
Interchange RGA and RGB contents.

Operation: <u>64-Bit and 32-Bit Modes (Identical)</u>. The contents of RGA and RGB are swapped. If the E is reset, the outer portion of RGA will be unchanged. If the E1 bit is reset, the inner portion of RGA will be unchanged. RGB is altered regardless of the E and E1 bits.

Registers Affected: RGA - Contains original contents of RGB, as enabled by E and E1.

RGB - Contains original contents of RGA.

Instruction Word:



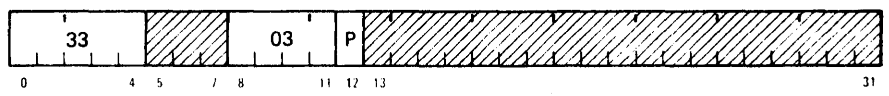
Mnemonic Code: SWAPA

Interchange inner and outer words of RGA.

Operation: 64-Bit and 32-Bit Modes. The inner and outer words of RGA are interchanged. If the E bit is reset, the outer word is unchanged. If the E1 bit is reset, the inner word is unchanged.

Register Affected: RGA - Inner and outer words interchanged, as enabled by E and E1.

Instruction Word:



Mnemonic Code: SWAPX

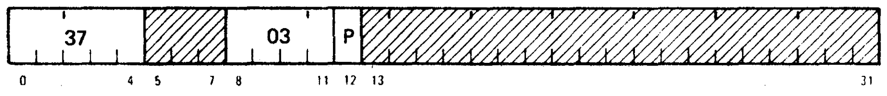
Interchange RGA outer word and RGB inner word.

Operation: <u>64-Bit and 32-Bit Modes</u>. The outer word of RGA and the inner word of RGB are interchanged. If the E bit is reset, RGA will be unchanged but RGB will be changed. The E1 bit has no effect.

Registers Affected: RGA - Outer word contains original contents of RGB inner word, if the E bit was set (=1).

RGB - Inner word contains original contents of RGA outer word.

Instruction Word:



Mnemonic Code: XD [redacted]
Subtract from RGX.

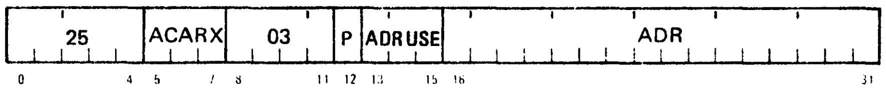
First Operand: Found in RGX.

Second Operand: Specified by ADR and ADR USE fields, using the operand addressing and indexing logic shown in Table 3-2.

Operation: <u>64-Bit and 32-Bit Modes (Identical)</u>. The 16 least significant bits of the second operand are subtracted from the contents of RGX, using twos-complement arithmetic, and the result is left in RGX. No end-around carry is generated; that is, any high-order carry resulting from the subtraction is lost. If the E bit is reset, RGX will be unchanged. The second operand (all 64 bits) is left in RGB.

Registers Affected: RGX - Contains difference, as enabled by E bit.
RGB - Contains second operand.

Instruction Word:



Mnemonic Code: XI

Add to RGX.

First Operand: Found in RGX.

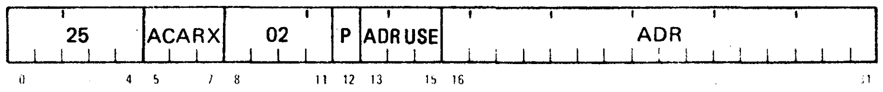
Second Operand: Specified by ADR and ADR USE fields, using the operand addressing and indexing logic shown in Table 3-2.

Operation: 64-Bit and 32-Bit Modes (Identical). The 16 least significant bits of the second operand are added to the contents of RGX, and the result is left in RGX. Any high-order carry is lost. If the E bit is reset, RGX will be unchanged. The second operand is left in RGB.

Registers Affected: RGX - Contains sum, as enabled by E bit.

RGB - Contains second operand.

Instruction Word:



FINST/PE Instruction Index

Mnemonic	Field A OP Code	Field B OP Code	Page
AD	35	04	5-14
ADA	35	05	5-14
ADB	26	06	5-16
ADD	26	04	5-17
ADEX	25	00	5-18
ADM	34	14	5-14
ADMA	34	15	5-14
ADN	34	04	5-14
ADNA	34	05	5-14
ADR	35	06	5-14
ADRA	35	07	5-14
ADRN	34	06	5-14
ADRNA	34	07	5-14
AND	27	04	5-19
ANDN	27	06	5-19
ASB	25	07	5-20
CAB	37	00	5-21
CHSA	37	00	5-22
CLRA	24	11	5-23
COMPA	22	11	5-24
DV	33	04	5-25
DVA	33	05	5-25
DVM	32	14	5-25
DVMA	32	15	5-25
DVN	32	04	5-25
DVNA	32	05	5-25
DVR	33	06	5-25
DVRA	33	07	5-25
DVRM	32	16	5-25
DVRMA	32	17	5-25
DVRN	32	06	5-25
DVRNA	32	07	5-25
EAD	20	10	5-28
EOR	25	05	5-29
EQV	25	04	5-30
ESB	24	10	5-31
GB	21	06	5-32
IAG	37	14	5-33
IAL	37	16	5-33
IB	35	02	5-34
ILE	35	16	5-35
ILG	33	14	5-35
ILL	33	16	5-35
ILO	33	10	5-36
ILZ	33	12	5-36
IME	35	14	5-37
IMG	31	14	5-37
IML	31	16	5-37
IMO	31	10	5-38
IMZ	31	12	5-38
ISE	25	12	5-39
ISG	21	12	5-39
ISL	23	12	5-39
ISN	35	02	5-40
IXE	25	10	5-41
IXG	21	10	5-41

Mnemonic	Field A OP Code	Field B OP Code	Page
IXGI	27	10	5-42
IXL	23	10	5-41
IXLD	27	12	5-43
JAG	37	15	5-33
JAL	37	17	5-33
JB	35	03	5-34
JLE	35	17	5-35
JLG	33	15	5-35
JLL	33	17	5-35
JLO	33	11	5-36
JLZ	33	13	5-36
JME	35	15	5-37
JMG	31	15	5-37
JML	31	17	5-37
JMO	31	11	5-38
JMZ	31	13	5-38
JSE	25	13	5-39
JSG	21	13	5-39
JSL	23	13	5-39
JSN	35	03	5-40
JXE	25	11	5-41
JXG	21	11	5-41
JXGI	27	11	5-42
JXL	23	11	5-41
JXLD	27	13	5-43
LB	21	07	5-44
LDA	26	17	5-45
LDB	27	00	5-45
LDD	22	12	5-45
LDE	21	14	5-47
LDE1	21	15	5-47
LDEE1	21	16	5-47
LDG	23	14	5-47
LDH	23	15	5-47
LDI	23	16	5-47
LDJ	23	17	5-47
LDR	27	01	5-45
LDS	27	02	5-45
LDX	27	03	5-45
LEX	21	17	5-49
ML	31	04	5-50
MLA	31	05	5-50
MLM	30	14	5-50
MLMA	30	15	5-50
MLN	30	04	5-50
MLNA	30	05	5-50
MLR	31	06	5-50
MLRA	31	07	5-50
MLRM	30	16	5-50
MLRMA	30	17	5-50
MLRN	30	07	5-50
MLRNA	30	07	5-50
MULT	22	13	5-52
NAND	27	05	5-19
NANDN	27	07	5-19
NEB	22	10	5-53

Mnemonic	Field A OP Code	Field B OP Code	Page
NOR	23	05	5-56
NORM	20	13	5-54
NORN	23	07	5-56
OFB	25	06	5-55
OR	23	04	5-56
ORN	23	06	5-56
RAB	37	01	5-57
RTAL	35	13	5-58
RTAR	35	12	5-58
RTL	24	12	5-59
SAB	37	02	5-57
SAN	37	02	5-60
SAP	37	01	5-60
SB	37	04	5-61
SBA	37	05	5-61
SBB	26	07	5-63
SBEX	25	01	5-64
SBM	36	14	5-61
SBMA	36	15	5-61
SBN	36	04	5-61
SBNA	36	05	5-61
SBR	37	06	5-61
SBRA	37	07	5-61
SBRN	36	06	5-61
SBRNA	36	07	5-61
SETE	25	14	5-65
SETE1	25	15	5-65
SETF	25	16	5-65
SETF1	25	17	5-65
SETG	27	14	5-65
SETH	27	15	5-65
SETI	27	16	5-65
SETJ	27	17	5-65
SHABL	37	11	5-68
SHABML	37	13	5-69
SHABMR	37	12	5-69
SHABR	37	10	5-68
SHAL	35	01	5-70
SHAML	35	11	5-71
SHAMR	35	10	5-71
SHAR	35	00	5-70
STA	26	12	5-72
STB	26	13	5-72
STR	26	14	5-72
STS	26	15	5-72
STX	26	16	5-72
SUB	26	05	5-73
SWAP	31	03	5-74
SWAPA	33	03	5-75
SWAPX	37	03	5-76
XD	25	03	5-77
XI	25	02	5-78