TECHNICAL MEMO NO. 15
REVISION NO. 2
DECEMBER 1959

LARC Computing-Unit Instructions

REMINGTON RAND UNIVAC
DIVISION OF SPERRY RAND CORP.
PHILADELPHIA, PENNSYLVANIA

Publications Engineering Department

Appendix B

LARC COMPUTING-UNIT INSTRUCTIONS

Appendix B

LARC COMPUTING-UNIT INSTRUCTIONS

B-1 INTRODUCTION

This appendix is designed to acquaint the programmer with the LARC computing-unit instruction repertoire. For the programmer's convenience, the instructions, as presented in section B-5, are classified according to function.

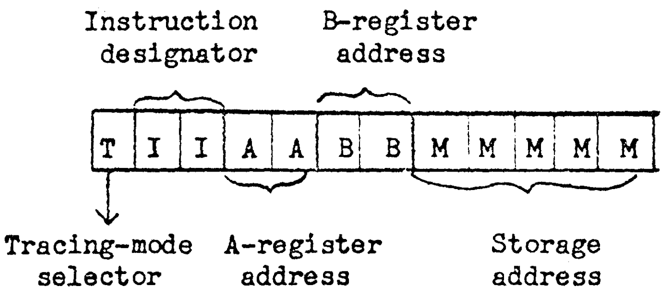
B-2 WORD FORMAT

Each computing-unit instruction word consists of 12 decimal digits; all these instructions are written in accordance with a standard format. Operands are written in a 12-digit format or a 24-digit format for single-precision or double-precision operations, respectively. The contents of an index register are written in a special format which is described in section 5.8.

In the following discussion, digit position references, by number, apply in ascending order, from right to left.

B-2.1 Instruction Words

The standard format for a computing-unit instruction word is as follows:



The tenth through twelfth digit positions contain the instruction-designator digits (TII). The I-digits specify the number of any legitimate computing-unit instruction. The T-digit contains one of the tracing-mode selectors (1, 2, ... 9) or, when an instruction is not to be traced, a period (.). An ignore sign (⋀) may also be specified in the T-digit; this causes the computer to enter the indirect-addressing mode. Any other character in the T-digit of an instruction word causes a transfer of control to the error routine.

The A-digits of a computing-unit instruction word contain the address of a fast register which is used to store an operand and/or the computational result of the operation specified by the I-digits; in certain instructions, the A-digits are used to specify the number of a flip-flop. The B-digits also specify the address of a fast register; in this case, however, the contents of the specified fast register are used to modify the M-digits of the current instruction before that instruction is executed.

The M-digits are used to specify any one of the following items:

(1) The memory address of an operand. (In this case M may refer to a standard memory location or to a fast register; see the note at the end of this section.)

(2) The memory address of the next instruction.

(3) The number of digit positions a word is to be shifted. (This number is specified by the two least significant M-digits.)

(4) The position of the decimal point in a conversion operation. (This is indicated by a scale factor in the two least-significant M-digits. The scale factor consists of a base-ten exponent expressed in excess-fifty notation.)

NOTE: A computing unit may contain up to 99 fast registers (addresses 01 through 99) all of which may be addressed and used interchangeably as accumulator registers, as index registers, or in the same manner as standard memory locations (using M-addresses 99901 through 99999). Although there is no corresponding fast register, the address digits 00 may be used in any of the digit positions specified, as follows:-

A = 00: May be used to supply an operand, consisting of a period and eleven decimal zeros (.00 000 000 000), in instructions which do not store in A. (The significance of a period in the sign position of an operand is discussed in section 2.2.1.)

B = 00: Used when no modification of the M-digits is required.

M=99900: May be used to supply an operand consisting of a period and eleven decimal zeros.

B-2.2 Operands

In single-precision, fixed-point operations, operands are written in this format:

S∧X X X X X X X X X X X

where S = the sign digit, and X = a decimal digit. The computer assumes the decimal point (∧) to be between the sign and the most significant decimal digit.

In single-precision, floating-point notation, the two digit positions immediately following the S-digit contain an excess-fifty, base-ten exponent. Thus, the format is:

S E E∧X X X X X X X X X X

where S = the sign digit, E = an exponent digit, and X = a decimal digit. The decimal point occurs between the E and X digits, and the operand is normalized (i.e., the most significant X-digit is not equal to zero).

The two-word format for double-precision, fixed-point operands allows for 22 decimal digits and an algebraic sign in the twelfth digit position of each word:

```
S^X X X X X X X X X X X|S X X X X X X X X X X X
```

Here, the decimal point is assumed to be between the S-digit and the most significant X-digit of the left-hand word (i.e. most significant half). The S-digit should be the same in both words.

The double-precision, floating-point operand consists of 20 decimal digits, an algebraic sign, and an excess-50, base-ten exponent, which are arranged as follows in two 12-digit words:

```
S E E^X X X X X X X X X|S X X X X X X X X X X X
```

In this notation, the decimal point occurs in the left-hand word between the E and X digits, and, as was the case in single-precision, floating-point notation, the operand is normalized. It is important to note that the sign is repeated in the twelfth digit of the right-hand word (as in double-precision, fixed-point notation), but the exponent is not repeated.

B-2.2.1 Sign-Digit Specification

Words written in alphanumeric code must contain a numeric character, 1 through 9, in the S-digit position. (In this case, the S-digit contains the first digit of a pair representing one of the alphanumeric characters.)

The character in the S-digit position of an algebraic number written in numeric code should be one of the following:

(1) A zero, indicating that the number is positive.

(2) A minus sign, indicating that the number is negative.

(3) A period.

In the sign position of an operand, a period has the general effect of causing an operation to be performed in an absolute sense. In floating point notation a period followed by all decimal zeros is used to indicate absolute zero (see section 2.2.2).

The computational effect of the character in the S-digit position (especially a period) varies according to the type of operation, as follows:

(1) In all arithmetic operations and in negative data transfers, if there is any anomaly in the sign, which causes a transfer of control to the contingency routine, a zero is deposited in the sign of the result.

(2) Addition and Subtraction

(a) Fixed Point

(i) A non-numeric character, other than a minus sign or a period, in the S-digit of either operand causes an automatic transfer of control to the contingency routine.

(ii) In double precision operations a numeric character, other than zero, in the S-digit of either operand causes an automatic transfer of control to the contingency routine [see (2) (d)].

(iii) In single precision operations a numeric character, other than zero, in the S-digit of one operand (either but not both) appears unchanged in the result. Numerics other than zero in the S-digits of both operands cause an automatic transfer of control to the contingency routine.

(iv) A period in the S-digit of either operand causes arithmetic addition without complementing.

(v) If a number with a period in the S-digit is added to or subtracted from another number, the result has the sign of the other number. If a number with a zero or a minus sign in the S-digit is subtracted from a number with a period in the S-digit the sign of the subtracted number is inverted in the result.

(b) Floating Point (except $|(M)| \oplus (A) \dashrightarrow A$)

(i) Any character other than a zero, a minus sign, or a period in the S-digit of either operand causes an automatic transfer of control to the contingency routine.

(ii) A period in the S-digit of either operand behaves as in fixed point operation [see (a) - (iv), (v)], with the added restriction that the exponent overflow and underflow contingency flip-flops are inhibited.

(c) $|(M)| \oplus (A) \dashrightarrow A$

(i) The character in the S-digit of the "A"-operand behaves exactly as in other floating point operations - see (b).

(ii) Any character is permissible in the S-digit of the M-operand. This character behaves as a zero, in all respects.

(d) Double Precision

(i) The S-digit in the most significant half only, of each operand, is examined and used in the computation. The character in the S-digit of the least significant half of each operand has no effect.

(ii) The characters in the S-digits of both halves of the result are identical.

(3) Multiplication and Division

(a) Any character other than a zero, a minus sign or a period in the

S-digit of either operand causes an automatic transfer of control to the contingency routine.

(b) A period in the S-digit of either operand causes a period to be deposited in the sign of the result.

(c) Floating Point

If there is a period in the S-digit of either operand the exponent overflow and underflow contingency flip-flops are inhibited and the result exponent is replaced by 00.

(d) Double Precision

(i) Division: see (2)(d)

(ii) Multiplication: the S-digit of the least significant half only is examined.

(4) Shift

(a) There is no restriction on the character in the S-digit.

(b) In all shift operations, except left circular shift, the character in the S-digit is neither shifted nor changed.

(c) In a left-circular shift the S-digit is shifted but not changed.

(5) Conversion

(a) There is no restriction on the character in the S-digit.

(b) The character in the S-digit is carried forward unaltered.

(6) Fetch and Store

(a) Except in a negative store, there is no restriction on the character in the S-digit.

(b) Negative Store

(i) Any character other than a zero, a minus sign or a period, in the S-digit, causes an automatic transfer of control to the contingency routine.

(ii) A period in the S-digit is transferred unaltered.

(c) Store Absolute Value

The character in the S-digit is always replaced by a zero.

(d) Double Precision

The S-digits of both words are handled independently.

(7) Comparisons

(a) All Comparisons

(i) A non-numeric, other than a minus sign or a period, in the S-digit blocks any transfer of control due to the comparison and causes an automatic transfer of control to the contingency routine. (In double precision operation all S-digits are examined independently)

(ii) A period in the S-digit behaves as a zero.

(b) (A) = (A+1)? : (A)>(A+1)?

(i) A numeric character, other than zero, in the S-digit of one operand causes that operand to be the greater.

(ii) A numeric character in the S-digit of each operand causes a twelve decimal digit comparison.

(c) (A)>0?

A numeric character other than zero in the S-digit causes the number to be greater than zero.

(d) (A) negative?

The S-digit only is examined.

(e) (A)=0?

(i) Compares eleven decimal digits, disregarding the sign.

(ii) Any character other than a zero, a period or a minus sign in the S-digit blocks any transfer of control due to the comparison and causes an automatic transfer of control to the contingency routine.

(f) (A') = ([A+2]')?

A digit by digit comparison is made, for all twenty-four digit positions.

(g) $(A') > ([A+2]')$?

(i) The most significant halves of both operands are compared following the same rules as for a single precision comparison [see (b)]

(ii) The least significant halves of both operands are compared only if the most significant halves are equal (in sign and magnitude). In this case the result of the comparison is based solely on the relative values of the least significant halves, following the same rules as in a single precision comparison [see (b)]

B-2.2.2 Specification of Floating Point Zero

In floating point notation, an absolute zero is represented by a period in the S-digit followed by eleven decimal zeros.

A floating point relative zero should not normally be represented by an exponent and all decimal zeros, since this can cause various anomalies in floating point arithmetic operations. The relative zero may be represented by an absolute zero or by an assumed very small non-zero, in the form, SEE 500 000 000, depending on how the number is to be used.

A floating point zero, consisting of an exponent and all decimal zeros may be obtained either as the result of an algebraic add or subtract operation, or by converting a fixed point zero to floating point form.

In either case this result is detected automatically, in the execution of the instruction, and sets contingency flip-flop 40 ("zero floating point adder result").

NOTE: In single precision addition and subtraction a zero result sets the contingency flip-flop, only if the exponents are equal, i.e. it is assumed that both operands are normalized.

The appropriate representation of floating point zero can be determined in the contingency routine.

For a fixed to floating point conversion it might be assumed that the unknown part of the fixed point number can be represented by a five in the twelfth significant digit. In the corresponding floating point representation this number is normalized and given an exponent equal to the scale factor minus eleven.

It may be noted that the conversion instruction, which is completed before entering the contingency routine, shifts out eleven zeros trying to normalize and subtracts this number from the scale factor to give the correct exponent. If the scale factor is less than eleven, the exponent underflow contingency (flip-flop 43) also occurs.

e.g. the fixed decimal number 000 000 000 000, with a scale factor of 50 would be converted to the floating point zero, 039 000 000 000. The required representation is 039 500 000 000.

A floating point zero, resulting from an algebraic addition or subtraction may be similarly represented:

In a floating point arithmetic subtraction, if the result contains significant zeros, the number is automatically normalized and the exponent adjusted accordingly. In the case of a zero result the operation will shift out nine zeros trying to normalize, and subtract nine from the exponent.

For use in further floating point computation, this result may be represented by assuming that the tenth significant digit, before normalizing, contains a five.

e.g. In the instruction $-(M)\oplus(A) \dashrightarrow A$, where (M)=(A)= 050 123 456 789

The initial result = 050 000 000 000

The final result = 041 000 000 000

The assumed value of the initial result = 050 000 000 000 5

The required representation = 041 500 000 000

A floating point absolute zero, represented by .00 000 000 000, is not changed by a floating-to-fixed-point conversion. This number may be used in fixed-point arithmetic and will behave as a normal fixed point zero, 000 000 000 000 (see section B-2.2.1).

B-3 CONVENTIONS

The following conventions are used in the description of the computing unit-instructions, in section B-5.

M — The M-digits of the instruction being described. Except in shift instructions (section B-5.5) and Conversion instructions (B-7.7), M is a storage address: M may refer either to a core-storage memory location or to a fast register; the possible memory addresses range from 00000 through 97499, and the fast-register addresses range from 99901 through 99999.

In the description of Shift and Conversion instructions, M signifies the two least significant M-digits, used to specify either the number of places a word is to be shifted, or the scale factor.

A , B — Address of a fast register (01 through 99): A denotes a fast register that is used as an accumulator register, (the next succeeding fast register is denoted by A+1, and the preceding fast register is denoted by A-1). In certain instructions, A is the number of a flip-flop (the addressable flip-flops are described in section B-6). B denotes a fast register used as an index register.

A_A Capital-letter subscripts denote a particular part of a word in accordance with the instruction-word format:

A_A denotes the A-register-address digits of the word in fast register A

A_B denotes the B-register-address digits of the word in fast register A

A_I denotes the tracing-mode selector and instruction-designator digits (TII) of the word in fast register A

A_M denotes the memory-address digits of the word in fast register A

$A_{A,B}$ denotes both the A-register-address and B-register-address digits of the word in fast register A. (more than one part of a word may be designated by means of successive capital-letter subscripts.)

M_A etc. the same notation is used to denote a portion of a word in memory location M

C A control counter which can be assumed to contain the storage address of the instruction currently being executed

Two consecutive storage locations: A' denotes the two fast registers A and A+1. Normally, the location of a double-precision word

( ) The contents of (a fast register, memory location, or control counter)

$\lvert\ \rvert$	The absolute value of (whatever is represented by the symbol between the vertical lines)
$\bigcirc$	A circled arithmetic symbol denotes a floating-point operation: (M) $\oplus$ (A) denotes floating-point addition of (M) and (A).
Rdd	Rounded result (All other results are unrounded.)
M ---> C	Control is transferred to a new sequence of instructions starting with the instruction whose storage address is specified in the M-digits of the instruction being described.
(C)+1 ---> C	The present sequence of executing instructions is continued. (That is, the control counter is stepped by 1 to give the address of the next instruction in sequence.)

B-4 INSTRUCTION-EXECUTION TIME

The execution time in microseconds is specified for each instruction in section B-5. The times given are all-inclusive; that is, they include the time required for obtaining operands and instructions from storage, the time required for modifying operand addresses, the time required for calculating floating-point exponents, the time required for error, contingency, and tracing-mode checking, etc. All input-output operations may be assumed to be performed in parallel with the instructions.

B-5 COMPUTING-UNIT INSTRUCTIONS

NOTE: The four items in the heading of each instruction are (from left to right) the numeric code, the mnemonic code, the symbolic notation, and the execution time in microseconds.

B-5.1 Arithmetic Instructions

The following descriptions of the arithmetic instructions have an algebraic connotation. In all cases the contents of M remain unchanged.

---

01 AX $(M) + (A) \dashrightarrow A$ 4 μsecs.

Add the contents of memory location M (addend) to the contents of fast register A (augend).

Store the sum, with the correct sign, in fast register A.

This is a fixed-point, single-precision operation.

---

02 A $(M) \oplus (A) \dashrightarrow A$ 4 μsecs.

This instruction is the same as instruction 01 except that it performs a floating-point operation.

---

03 AM $|(M)| \oplus (A) \dashrightarrow A$ 4 μsecs.

Add the absolute value of the contents of memory location M to the contents of fast register A.

Store the sum, with the correct sign, in fast register A.

This is a floating-point, single-precision operation.

04	AU	$(M) \oplus (A) \dashrightarrow A+1$	4 µsecs.

This instruction is the same as instruction 02 except that the sum is stored in fast register A+1 and the augend is retained in fast register A.

05	AAX	$(M') + (A') \dashrightarrow A'$	12 µsecs.

Add the contents of memory locations M and M+1 (addend) to the contents of fast registers A and A+1 (augend).

Store the sum, with the correct sign, in fast registers A and A+1.

This is a fixed-point, double-precision operation.

06	AA	$(M') \oplus (A') \dashrightarrow A'$	16 µsecs.

This instruction is the same as instruction 05 except that it performs a floating-point operation.

11	NX	$-(M) + (A) \dashrightarrow A$	4 µsecs.

Change the sign of the contents of memory location M and add to the contents of fast register A.

Store the sum, with the correct sign, in fast register A.

This is a fixed-point, single-precision operation.

12	N	$-(M) \oplus (A) \dashrightarrow A$	4 µsecs.

This instruction is the same as instruction 11 except that it performs a floating-point operation.

14	NU	$-(M) \oplus (A) \dashrightarrow A+1$	4 μsecs.

This instruction is the same as instruction 12 except that the sum is stored in fast register A+1 and the contents of A remain unchanged.

15	NNX	$-(M') + (A') \dashrightarrow A'$	12 μsecs.

Change the sign of the contents of memory locations M and M+1 and add to the contents of fast registers A and A+1.

Store the sum, with the correct sign, in fast registers A and A+1.

This is a fixed-point, double-precision operation.

16	NN	$-(M') \oplus (A') \dashrightarrow A'$	16 μsecs.

This instruction is the same as instruction 15 except that it performs a floating-point operation.

20	MXR	$[(M) \times (A)]$ Rdd $\dashrightarrow A$	8 μsecs.

Multiply the contents of fast register A (multiplicand) by the contents of memory location M (multiplier).

Store the rounded product, with the correct sign, in fast register A.

This is a fixed-point, single-precision operation.

21	MXE	$(M) \times (A) \dashrightarrow A'$	12 μsecs.

This instruction is the same as instruction 20 except that a double-precision unrounded product is stored in fast registers A and A+1.

---

22 MR $[(M) \otimes (A)]$ Rdd ---> A 12 μsecs.

This instruction is the same as instruction 20 except that it performs a floating-point operation.

---

23 M $(M) \otimes (A)$ ---> A 8 μsecs.

This instruction is the same as instruction 22 except that the product is not rounded.

---

24 MU $(M) \otimes (A)$ ---> A+1 8 μsecs.

This instruction is the same as instruction 23 except that the product is stored in fast register A+1 and the multiplicand is retained in fast register A.

---

25 ME $(M) \otimes (A)$ ---> A' 12 μsecs.

This instruction is the same as instruction 21 except that it performs a floating-point operation.

---

26 MMX (M') x (A') ---> A' 36 μsecs.

Multiply the contents of fast registers A and A+1 (multiplicand) by the contents of memory locations M and M+1 (multiplier).

Store the product, with the correct sign, in fast registers A and A+1.

This is a fixed-point, double-precision operation.

---

27 MM $(M') \otimes (A')$ ---> A' 36 μsecs.

This instruction is the same as instruction 26 except that it performs a floating-point operation.

---

30 DX (A) ÷ (M) ---> A 32 µsecs.

Divide the contents of fast register A (dividend) by the contents of memory location M (divisor).

Store the quotient, with the correct sign, in fast register A; the remainder is not retained.

This is a fixed-point, single-precision operation.

---

31 DXE (A) ÷ (M) ---> A' 36 µsecs.

This instruction is the same as instruction 30 except that the remainder, which retains the sign of the dividend, is stored in fast register A+1.

---

32 DR [(A) ÷ (M)] Rdd ---> A 28 µsecs.

This instruction is the same as instruction 30 except that it performs a floating-point operation and produces a rounded quotient.

---

34 DUR [(A) ÷ (M)] Rdd ---> A+1 28 µsecs.

This instruction is the same as instruction 32 except that the rounded quotient is stored in fast register A+1 and the dividend is retained in fast register A.

---

35 DDX (A') ÷ (M') ---> A' 184 µsecs.

Divide the contents of fast registers A and A+1 by the contents of memory locations M and M+1.

Store the quotient, with the correct sign, in fast registers A and A+1; the remainder is not retained.

This is a fixed-point, double-precision operation.

36	DD	(A') ⊖ (M') ---> A'	168 μsecs.

This instruction is the same as instruction 35 except that it performs a floating-point operation.

37	DSE	(A') ⊖ (M) ---> A'	56 μsecs.

Divide the contents of fast registers A and A+1 by the contents of memory location M. Store the quotient, with the correct sign, in fast registers A and A+1; the remainder is not retained.

This is a floating-point operation. A double precision dividend is divided by a single precision divisor giving a double precision quotient.

B-5.2 Data-Transfer Instructions

40	S	(A) ---> M	4 μsecs.

Transfer the contents of fast register A to memory location M.

The contents of A remain unchanged.

41	SN	-(A) ---> M	4 μsecs.

This instruction is the same as instruction 40 except that the negative value of the quantity in fast register A is transferred.

42	SM	|(A)| ---> M	4 μsecs

This instruction is the same as instruction 40 except that the absolute value of the quantity in fast register A is transferred.

43	F	(M) ---> A	4 μsecs.

Transfer the contents of memory location M to fast register A.

The contents of M remain unchanged.

45	SS	(A') ---> M'	8 μsecs.

This instruction is the same as instruction 40 except that it performs a double-precision operation. (That is, the contents of fast registers A and A+1 are transferred to memory locations M and M+1, respectively, and both A and A+1 remain unchanged.)

46	SSN	-(A') ---> M'	8 μsecs.

This instruction is the same as instruction 41 except that it performs a double-precision operation.

47	SSM	|(A')| ---> M'	8 μsecs.

This instruction is the same as instruction 42 except that it performs a double-precision operation.

48	FF	(M') ---> A'	8 μsecs.

This instruction is the same as instruction 43 except that it performs a double-precision operation.

EOP | EA EB | EM
TIIAABBMMMMM

Extr. op.

60 EOP $(M)_I$ ---> A_I 4 μsecs.

Transfer the tracing-mode selector digit and the instruction-designator digits of the word in memory location M to the corresponding digit positions of the word in fast register A; all other digit positions in A remain unchanged.

---

61 EA $(M)_A$ ---> A_A 4 μsecs.

This instruction is the same as instruction 60 except that the two A-digits are transferred.

---

62 EB $(M)_B$ ---> A_B 4 μsecs.

This instruction is the same as instruction 60 except that the two B-digits are transferred.

---

63 EAB $(M)_{AB}$---> A_{AB} 4 μsecs.

This instruction is the same as instruction 60 except that both the A-digits and B-digits are transferred.

---

64 EM $(M)_M$ ---> A_M 4 μsecs.

This instruction is the same as instruction 60 except that the five M-digits are transferred.

---

93 SLJ [9T(C2)] ---> M 4 μsecs.

Transfer the contents of C2 (as the M-address digits of a 90 instruction) to memory location M.

93 (continued)

In the notation, [9T(C2)]: 9 = the tracing-mode selector (no other digit may be used in this particular case)

T = the 90 instruction

C2 = a five digit register containing the address of the instruction which would have followed the last conditional or unconditional transfer of control instruction if this had operated in the opposite sense. Specifically: whenever an instruction which could cause a transfer of control is executed, the M-digits of that instruction are stored in C2. If no transfer of control occurs this address is retained in C2; if a transfer of control does take place the contents of C2 are replaced by (C)+1 (the address of the next

the instr. following the transfer?

instruction in sequence).

At the completion of this 93 instruction, M contains 990 00 00 mmmmm, where mmmmm = (C2).

The 93 instruction may also be used to perform this transfer:

$$(C2) \dashrightarrow A_M \qquad 4\ \mu\text{secs.}$$

In this case, the M-digits of the 93 instruction contain an A-register address which is specified by 999AA.

93 $(C2) \dashrightarrow A_M$ (cont'd.)

At the completion of this transfer, fast register A contains 000 00 00 mmmmm, where mmmmm = (C2).

<u>NOTE</u>: A 93 instruction may be employed most usefully at the beginning of a sub-routine which is entered via a test instruction. The 93 instruction ensures that the point of origin, several of which may be scattered throughout the program, is available for use as a return point or for selecting some branch in the subroutine.

B-5.3 <u>Conditional-Transfer-of-Control Instructions</u>

---

70 TE (A) = (A+1) ?

Test to see if the contents of fast register A are equal to the contents of fast register A+1.

If (A) = (A+1), M ---> C.	12 μsecs.
If (A) $\neq$ (A+1), (C)+1 ---> C.	4 μsecs.

---

71 TG (A) = (A+1) ?

Test to see if the contents of fast register A are greater than the contents of fast register A+1.

If (A) > (A+1), M ---> C.	12 μsecs.
If (A) $\leq$ (A+1), (C)+1 ---> C.	4 μsecs.

---

72 TZ $(A) = 0$?

Test to see if the contents of fast register A are numerically equal to zero.

If $(A) = 0$, M ---> C.	12 μsecs.
If $(A) \neq 0$, (C)+1 ---> C.	4 μsecs.

---

73 TGZ $(A) > 0$?

Test to see if the contents of fast register A are greater than zero.

If $(A) > 0$, M ---> C.	12 μsecs.
If $(A) \leq 0$, (C)+1 ---> C.	4 μsecs.

---

74 TLZ (A) negative ? so is -0 ?

Test to see if the contents of fast register A are negative.

If (A) negative, M ---> C.	12 μsecs.
If (A) not negative, (C)+1 ---> C.	4 μsecs.

---

75 TTE $(A') = ([A+2]')$?

Test to see if the contents of fast registers A and A+1 are equal to the contents of fast registers A+2 and A+3.

If $(A') = ([A+2]')$, M ---> C.	16 μsecs
If $(A') \neq ([A+2]')$, (C)+1 ---> C.	8 μsecs.

---

76 TTG $(A') > ([A+2]')$?

Test to see if the contents of fast registers A and A+1 are greater than the contents of fast registers A+2 and A+3.

If (A') > ([A+2]'), M ---> C. 16 µsecs.

If (A') ≤ ([A+2]'), (C)+1 ---> C. 8 µsecs.

---

95 TF Test FFA

Test to see if flip-flop A is set.

If FFA is set, M ---> C. 12 µsecs.

If FFA is reset, (C)+1 ---> C. 4 µsecs.

NOTE: The number of the flip-flop is specified in the A-digits of the instruction word. Refer to section B-6 for a description of the addressable flip-flops.

B-5.4 Unconditional-Transfer-of-Control Instructions

---

90 T M ---> C 8 µsecs.

Transfer control to the instruction in memory location M.

---

91 TR [9T(C)+1] ---> M and M+1 ---> C 12 µsecs.

Store in memory location M a 90 instruction which specifies the address of the next instruction in sequence (that is, the instruction immediately following the 91 instruction).

Transfer control to the instruction in memory location M+1.

Memory location M+1 contains the first instruction in a subroutine. At the completion of that subroutine, control is transferred to memory location M which contains the exit instruction of that subroutine; this exit instruction returns control to the originating program.

91 (continued)

NOTE: In the notation, [9T(C)+1]: 9 = the tracing-mode selector (no other digit may be used in this particular case)

T = the 90 instruction

(C)+1 = the address of the next instruction in the originating program.

92	TB	(C) ---> A_M and M ---> C	8 µsecs.

Store the contents of the control counter (that is, the current address of the 92 instruction) in the M-digits of fast register A. The contents of the remaining seven digit positions in A are not changed.

Transfer control to the instruction in memory location M.

Memory location M contains the first instruction in a subroutine. The contents of fast register A are used to modify the exit instruction of that subroutine so that, at the completion of the subroutine, control is returned to the originating program. More specifically, the exit instruction of the subroutine is in the form T90 00 BB 00001 (where the B-digits of the 90 instruction and the A-digits of the 92 instruction specify the same fast register); the M-digits of this instruction, when modified by the M-digits of the specified B-register, specify the address C+1.

B-5.5. Shift Instructions

52	PR	$(A)10^{-M}$ ---> A	4 µsecs.

Shift the contents of fast register A to the right M places.

52 (continued)

Fill the digit positions which are emptied by the shift with decimal zeros.

Store the result in fast register A.

The sign digit is neither shifted nor changed in this operation.

53	PL	$(A)10^M \dashrightarrow A$	4 μsecs.

This instruction is the same as instruction 52 except that the digits are shifted to the left.

57	PPR	$(A')10^{-M} \dashrightarrow A'$	8 μsecs.

This instruction is the same as instruction 52 except that it performs a double-precision shift. (That is, the contents of fast registers A and A+1 are shifted simultaneously to the right so that digits shifted out of A occupy the digit positions vacated by the shift in A+1)

58	PPL	$(A')10^M \dashrightarrow A'$	8 μsecs.

This instruction is the same as instruction 53 except that it performs a double-precision shift. (That is, the contents of fast registers A and A+1 are shifted simultaneously to the left so that the digits shifted out of A+1 occupy the digit positions vacated by the shift in A)

59	PPC	$(A')10^M \dashrightarrow A'$ (circular)	12 μsecs.

Shift the contents of fast registers A and A+1 simultaneously to the left M places.

59 (continued)

The digits shifted out of the most significant end of fast register A re-enter fast register A+1 at the least significant end.

The sign digits are included in this circular-left shift operation.

B-5.6 Extract Instructions

65	EL	$(A-1) \xrightarrow[(M)]{} A$	8 μsecs.

In accordance with an extract pattern specified by the word in memory location M, replace certain digits of the word in fast register A with the corresponding digits of the word in fast register A-1.

Store the result in fast register A.

The contents of A-1 and M remain unchanged.

Extraction occurs in those digit positions occupied by a ONE in (M). In the sign position of (M) either a ONE or a minus sign causes extraction.

For example, if

(M) = -11 023 111 456

(A) = XXX XXX XXX XXX

(A-1) = YYY YYY YYY YYY

then, after the execution of a 65 instruction,

(A) = YYY XXX YYY XXX

66	EJ	(A+1) ---> A (M)	8 μsecs.

This instruction is the same as instruction 65 except that digits of the word in fast register A are replaced by digits from the word in fast register A+1.

B-5.7 Conversion Instructions

50	CX	FL ---> FX M = scale factor	4 μsecs.

Convert the single-precision, floating-point number in fast register A to a single-precision, fixed-point number.

Store the result in fast register A.

The conversion is made in accordance with a scale factor which is specified in the two least significant digits of the instruction word.

The floating-to-fixed point conversion process is illustrated by an example at the end of this section.

51	C	FX ---> FL M = scale factor	4 μsecs.

Convert the single-precision, fixed-point number in fast register A to a single-precision, floating-point number.

Store the result in fast register A.

The conversion is made in accordance with a scale factor which is specified in the two least significant digits of the instruction word.

The fixed-to-floating-point conversion process is illustrated by an example at the end of this section.

55	CCX	FL' ---> FX'	12 μsecs.
		M = scale factor	

This instruction is the same as instruction 50 except that it performs a double-precision, floating-point-to-fixed-point conversion.

56	CC	FX' ---> FL'	12 μsecs.
		M = scale factor	

This instruction is the same as instruction 51 except that it performs a double-precision, fixed-point-to-floating-point conversion.

Examples:

A fixed point number, as it appears in the machine, has associated with it a scale factor which indicates the true magnitude of the number. When this number is converted to floating-point form, the scale factor determines the value of the floating point exponent, subject to the restriction that the floating point number must be normalized. Conversely, when a floating point number is converted to fixed point form, based on some previously established scale factor, the apparent magnitude of the number as expressed in fixed point notation is determined by the relative values of the floating point exponent and the scale factor.

1. Fixed-to-Floating-Point Conversion

True magnitude of number	-.000198765432
Fixed point number as it appears in the computer	-01987654320

Scale Factor:-

Express the number as it appears in the computer in its true magnitude, using powers of 10 — $-01987654320 \times 10^{-2}$

Express the 10's exponent in excess 50 notation:

This number is the scale factor — 48

Conversion:-

Subtract from the scale factor, the number of zeros which must be shifted out to normalize the fixed-point number, as it appears in the computer.

The difference is the floating point exponent — 48-1 = 47

Normalized number in floating-point notation — -4719876543 2

2. Floating-to-Fixed-Point Conversion

Floating-point number to be converted — -54123456789

Fixed-point scale factor — 57

Conversion:-

Subtract the floating-point exponent from the scale factor — 57-54 = 3

Shift the normalized number right a number of places equal to the difference between the scale factor and the exponent. The

result is the number in fixed-point notation. 123456789 --->-00012345678

B-5.8 <u>Index-Register-Modification Instructions</u>

<u>NOTE 1:</u> In the six index-register-modification instructions, the B-register address (01, 02, ...99) is specified in the A-register-address digits.

<u>NOTE 2:</u> The format for words stored in a B-register is

N N N D D D D Δ Δ Δ Δ Δ

where NNN = cycle count: the number of times a program loop is to be repeated (Once in each iteration, NNN is reduced by 1; when NNN = 0, the iterative process is terminated.) Since, in the instructions (80 through 83) which modify the cycle counter, NNN is reduced by one before it is tested for zero, it is possible to count to one thousand by starting with NNN = zero.

DDDD = increment or decrement to ΔΔΔΔΔ: the amount which is added to or subtracted from the address modifier before or after each iteration

ΔΔΔΔΔ = address modifier: the amount which is added to the M-digits of an instruction that addresses the B-register before that instruction is executed

---

80 BIT

N-1 ---> N

Δ+D ---> Δ

N = 0 ?

Modify the specified B-register in this way:

(a) Reduce the cycle count by 1

(b) Increase the address modifier (Δ-digits) by the amount specified by the D-digits

Compare the reduced cycle count with zero:

If new N = 0, (C)+1 ---> C. 12 μsecs.

If new N ≠ 0, M --->C. 8 μsecs.

---

81 BDT

N-1 ---> N

Δ-D ---> Δ

N = 0 ?

This instruction is the same as instruction 80 except that the address modifier is decreased by the amount specified by the D-digits.

82	BIC	N-1 ---> N	
		Δ+D ---> Δ	
		N = 0 ?	

This instruction is the same as instruction 80 with one exception:

If new N = 0, M ---> C.	12 μsecs.
If new N $\neq$ 0, (C)+1 ---> C.	4 μsecs.

83	BDC	N-1 ---> N	
		Δ-D ---> Δ	
		N = 0 ?	

This instruction is the same as instruction 81 with one exception:

If new N = 0, M ---> C.	12 μsecs.
If new N $\neq$ 0, (C)+1 ---> C.	4 μsecs.

85	BI	Δ+D ---> Δ	4 μsecs.

Increase the address modifier (Δ-digits) by the amount specified by the D-digits.

86	BD	Δ-D ---> Δ	4 μsecs.

Decrease the address modifier (Δ-digits) by the amount specified by the D-digits.

B-5.9 Visual-Display-Register Instructions

09	FV	(5-digit register) ---> A_M	

If Interlock is set, transfer the contents of the 5-digit visual-display register to the M-digits of fast register A; the remaining digit positions of A are filled with zeros. Reset the Connect and Interlock flip-flops. 4 μsecs.

09 (continued)

If Interlock is reset, M ---> C. 12 µsecs.

---

19 FVK (12-digit register) ---> A

This instruction is the same as instruction 09 except that the entire contents of the 12-digit visual-display register are transferred.

---

29 SV $(A)_M$ ---> 5-digit register

If Interlock is reset, transfer the contents of the M-digits of fast register A to the 5-digit visual-display register. 4 µsecs.

If Interlock is set, M ---> C. 12 µsecs.

---

39 SVK (A) ---> 12-digit register

This instruction is the same as instruction 29 except that the contents of fast register A are transferred to the 12-digit visual-display register.

B-5.10 Miscellaneous Instructions

---

00 SK Skip 4 µsecs.

Go on to the next instruction in the sequence.

---

96 RF Reset FFA 4 µsecs.

Reset flip-flop A.

NOTE: The number of the flip-flop is specified in the A-digits of the instruction word. Refer to section B-6 for a description of the addressable flip-flops.

97	SF	Set FFA	4 μsecs.

Set flip-flop A.

<u>NOTE</u>: The number of the flip-flop is specified in the A-digits of the instruction word. Refer to section B-6 for a description of the addressable flip-flops.

99	H	Stop	———

Stop computation.

Numerical List of Instructions

Numeric Code	Mnemonic Code		Symbolic Notation	Time µs
00	SK	40	Skip	4
01	AX	19	(M) + (A) -----> A	4
02	A	19	(M) ⊕ (A) -----> A	4
03	AM	19	|(M)| ⊕ (A) ---> A	4
04	AU	20	(M) ⊕ (A) -----> A+1	4
05	AAX	20	(M') + (A') ---> A'	12
06	AA	20	(M') ⊕ (A') ---> A'	16
09	FV	39	(5 digit display register) -----> A_M and reset Connect and Interlock. If Interlock not set: M -----> C	4 12
11	NX	20	-(M) + (A) ----> A	4
12	N	20	-(M) ⊕ (A) ----> A	4
14	NU	21	-(M) ⊕ (A) ----> A+1	4
15	NNX	21	-(M') + (A') --> A'	12
16	NN	21	-(M') ⊕ (A') --> A'	16
19	FVK	40	(12 digit display register) -----> A and reset Connect and Interlock. If Interlock not set: M -----> C	4 12
20	MXR	21	(M) X (A)Rdd --> A	8
21	MXE	21	(M) X (A) -----> A'	12
22	MR	22	(M) ⊗ (A)Rdd --> A	12
23	M	22	(M) ⊗ (A) -----> A	8
24	MU	22	(M) ⊗ (A) -----> A+1	8
25	ME	22	(M) ⊗ (A) -----> A'	12
26	MMX	22	(M') X (A') ---> A'	36
27	MM	22	(M') ⊗ (A') ---> A'	36
29	SV	40	(A) ---> 5 digit display register. If Interlock set: M -----> C	4 12
30	DX	23	(A) ÷ (M) -----> A	32
31	DXE	23	(A) ÷ (M) -----> A'	36
32	DR	23	(A) ⨸ (M)Rdd --> A	28
34	DUR	23	(A) ⨸ (M)Rdd --> A+1	28
35	DDX	23	(A') ÷ (M') ---> A'	184
36	DD	24	(A') ⨸ (M') ---> A'	168
37	DSE	24	(A') ⨸ (M) ----> A'	56
39	SVK	40	(A) ---> 12 digit display register. If Interlock set: M -----> C	4 12
40	S	24	(A) --------> M	4
41	SN	24	-(A) -------> M	4
42	SM	24	|(A)| -------> M	4
43	F	25	(M)---------> A	4
45	SS	25	(A') -------> M'	8
46	SSN	25	-(A') ------> M'	8
47	SSM	25	|(A')| ----->,M'	8
48	FF	25	(M') -------> A'	8
50	CX	34	FL ---------> FX M = scale factor	4

Numeric Code	Mnemonic Code pg	Symbolic Notation	Time µs
51	C 34	FX ----------> FL M = scale factor	4
52	PR 31	$(A)10^{-M}$ ----> A	4
53	PL 32	$(A)10^{M}$ ------> A	4
55	CCX 35	FL' ---------> FX' M = scale factor	12
56	CC 35	FX' ---------> FL' M = scale factor	12
57	PPR 32	$(A')10^{-M}$ ---> A' (right shift M places)	8
58	PPL 32	$(A')10^{M}$ ----> A' (left shift M places)	8
59	PPC 32	Left circular shift A', M places	12
60	EOP 26	$(M)_I$ --------> A_I	4
61	EA 26	$(M)_A$ --------> A_A	4
62	EB 26	$(M)_B$ --------> A_B	4
63	EAB 26	$(M)_{AB}$ -------> A_{AB}	4
64	EM 26	$(M)_M$ --------> A_M	4
65	EL 33	(A-1) -------> A (M)	8
66	EU 34	(A+1) -------> A (M)	8
70	TE 28	(A) = (A+1) ? No: (C)+1 ----> C Yes: M ---------> C	 4 12

Numeric Code	Mnemonic Code pg	Symbolic Notation	Time µs
71	TG	(A) > (A+1) ? No: (C)+1 ----> C Yes: M ---------> C	 4 12
72	TZ 29	(A) = 0 ? No: (C)+1 ----> C Yes: M ---------> C	 4 12
73	TGZ 29	(A) > 0 ? No: (C)+1 ----> C Yes: M ---------> C	 4 12
74	TLZ 29	(A) negative? No: (C)+1 ----> C Yes: M ---------> C	 4 12
75	TTE 29	(A') = (A+2)' ? No: (C)+1 ----> C Yes: M ---------> C	 8 16
76	TTG 29	(A') > (A+2)'? No: (C)+1 ----> C Yes: M ---------> C	 8 16
80	BIT 38	N-1 --> N and Δ+D-->Δ New N ≠ 0; M ------> C New N = 0; (C)+1--> C	 8 12
81	BDT 38	N-1--> N and Δ-D--> Δ New N ≠ 0; M ------> C New N = 0; (C)+1--> C	 8 12
82	BIC 39	N-1--> N and Δ+D--> Δ New N ≠ 0; (C)+1--> C New N = 0; M -----> C	 4 12

Numerical List of Instructions

Numeric Code	Mnemonic Code	Symbolic Notation	Time µs
83	BDC 39	N-1--> N and Δ-D-->Δ	
		New N ≠ 0;(C)+1 -->C	4
		New N = 0; M ------>C	12
85	BI 39	Δ + D ------------->Δ	4
86	BD 39	Δ - D ------------->Δ	4
90	T 30	M ----------------->C	8
91	TR 30	[9T(C) + 1] ------>M	12
		M + 1 ------------>C	

Numeric Code	Mnemonic Code	Symbolic Notation	Time µs
92	TB 31	(C) --------------> A_M	8
		M ----------------> C^M	
93	SLJ 26-28	[9T(C2)] ---------> M	4
95	TF 30	Test FF A	
		If reset: (C)+1--> C	4
		If set: M --------> C	12
96	RF 40	Reset FF A	4
97	SF 40	Set FF A	4
99	H 40	STOP	-

B-6 ADDRESSABLE FLIP-FLOPS IN THE COMPUTING UNIT

FF Number	Description	CU Program Can: Test (Inst. 95)	Reset (Inst 96)	Set (Inst. 97)
00, 01, ... 09	Sense FFs	X	X	X
10	Disclosure FF[1]	X		X
11	Processor-intervention contingency FF[2]	X	X	-
15	Manual intervention inhibit FF	X	X	X
20	Enter-tracing-mode FF	X	X	
21, 22, ... 29	Selected-tracing-mode FFs	X	X	X
30, 31, ... 34	Console-manual-intervention contingency FFs	X	X	
38	Improper-tape error FF	X	X	
39	Improper operand in arithmetic subtraction contingency FF	X	X	
40	Zero floating-point adder result contingency FF	X	X	

[1]This FF can also be tested and reset by the processor.

[2]This FF can also be tested and set by the processor.

FF Number	Description	CU Program Can: Test (Inst. 95)	Reset (Inst. 96)	Set (Inst. 97)
41	Non-normalized divisor contingency FF	X	X	
42	Exponent-overflow contingency FF	X	X	
43	Exponent-underflow contingency FF	X	X	
44	Fixed-point overflow contingency FF	X	X	
45	Sign-anomaly contingency FF	X	X	
46	Stall-error FF	X	X	
47	Control-error FF	X	X	
48	Fast-register control-error FF (on result time)	X	X	
49	Decoding-error FF (in tracing mode selector digit)	X	X	
50	B-adder Odd-even error FF (on instruction or operand call)	X	X	
51	Instruction odd-even error FF	X	X	
52	Operand odd-even error FF	X	X	
53	Fast-register odd-even error FF (in M-address modification)	X	X	

FF Number	Description	CU Program Can:		
		Test (Inst. 95)	Reset (Inst. 96)	Set (Inst. 97)
54	Fast-register odd-even error FF (on time-slot M: Time-slot M is the time at which the contents of a fast register are read out when addressed by the M-digits of an instruction. In certain instructions a fast register addressed by the A-digits is read out on time-slot M)	X	X	
55	A-register odd-even error FF (on result time)	X	X	
56	B-adder odd-even error FF (on output to control counter 1, or to the high-speed bus, or to the arithmetic unit)	X	X	
57	B-adder odd-even error FF (on output to the fast-register selector or to selector storage, or to the M-digits of instruction register 2)	X	X	
58	B-adder odd-even error FF (on output to control counter 2. Refer to section B-5.2, instruction 93, for a description of C2.)	X	X	

FF Number	Description	CU Program Can: Test (Inst. 95)	Reset (Inst. 96)	Set (Inst. 97)
59	Adder-output odd-even or non-numeric error FF	X	X	
60	Shifter-output odd-even error FF	X	X	
61	Comparator-error FF (single precision division)	X	X	
62	Multiplier, quotient, and extractor error FF	X	X	
63	Shift-control error FF	X	X	
64	Adder-overflow error FF	X	X	
65	Error FF for arithmetic-unit program counter and decoder	X	X	
66	Ending-Pulse error FF	X	X	
67	AH-register odd-even error FF	X	X	
68	AD-register odd-even error FF	X	X	
69	Sign-digit odd-even error FF	X	X	
70	A-register odd-even error FF (on time-slot A: Time-slot A is the time at which the contents of a fast register are normally read out when addressed by the A-digits of an instruction. See note on FF 54.)	X	X	

FF Number	Description	CU Program Can: Test (Inst. 95)	Reset (Inst. 96)	Set (Inst. 97)
71, 72, ... 82	Odd-even error, digit position FF's	X	*	
84	Cycling Unit Error FF	X	X	
90	Start-tape FF	X	X	X
98	Master error FF	X		
99	Master contingency FF	X		

*Flip-flops 71 through 82 are automatically reset when all of the following FF's are reset: 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 67, 68, 70.