G. Bell

PDP-X Technical Memorandum # 32

Title: EOP Class Instructions

Author: H. Burkhardt

Index Keys: Double Precision
Floating-Point
Instruction Set
Software Specifications

Distribution Keys: A,B,C,D

Obsolete: None

Revision: None

Date: December 6, 1967

0. Introduction

PDP-X architecture includes a class of instructions, the Extended OPeration codes, that may either cause a program trap or may be executed directly by the processor hardware. This class of instructions may be divided into three sub-classes depending upon their function:

1. User communication with the IO monitor or supervisor ($D1 < 40_8$)
2. User communication with himself ($40_8 \leq D1 < 100_8$)
3. Arithmetic or logical operations, special multiprocessor functions, etc. ($100_8 \leq D1 < 400_8$)

EOP's in the first sub-class will be defined by the software system. User programs will use and define the functions in the second sub-class (for example, the Fortran Operating System). The third sub-class will be defined by specific hardware implementation or by universally accepted conventions. PDP-X/II has 15 implemented EOP's as described in PDP-X Technical Memorandum #29. This document will define a further class of EOP instructions, i.e., the floating-point and double-precision operations. At some later date, it may be possible to implement these operations in hardware, but for the present, these operations will be performed by resident software. From the user's point of view, there is essentially no difference except that:

1. The floating-point instructions will take longer to execute when performed by software.
2. The software to execute the floating-point operations requires that some portion of the available memory space be allocated for their residence.

Commonly used functions such as floating-point add will be defined as EOP class instructions since:

1. They may be assembled with the normal assembler.
2. The calling sequences are concise (i.e., 2 words for the EOP instruction).
3. The addition of hardware to execute these instructions will not necessitate the alteration of any existing software.

1. Double-Precision Operations

Double-precision instructions operate upon signed (2's complement) double-precision integers (programming conventions allow these quantities to be interpreted as signed binary fractions) as described in PDP-X Technical Memorandum #29, section 2.2.1.

Double-precision quantities are addressed by the address of the high-order word (which must be even). If the address of a double-precision quantity is not even, an address exception error will occur and bit 5 of the Program Status double-Word will be set. All double-precision instructions are long form EOP class instructions. The D1 field indicates the operation to be performed.

Most double-precision operations are between one operand in the accumulators (AC-field of the instruction) and one operand located by the effective address (in the accumulators or in main storage). The AC-field of such instructions must be even or an address exception error will occur and bit 5 of the Program Status double-Word will be set.

When referring to double-precision operations, the term "effective double-word" will imply the double-word located by the effective address.

D1	mnem	Definition
120	DLDA	Double-precision LoaD Accumulator. The double-word accumulator specified by the AC-field of the instruction (must be even) is loaded with the effective double-word. The condition code bits remain unchanged.
121	DSTA	Double-precision STore Accumulator. The double-word accumulator specified by the AC-field of the instruction (must be even) replaces the effective double-word. The condition code bits remain unchanged.
122	DADD	Double-precision ADD. The sum of the double-word accumulator specified by the AC-field of the instruction (must be even) and the effective double-word replaces the double-word accumulator. The addition is carried out following the rules of two's complement arithmetic. condition code bit 0 is set if carry out of bit 0, cleared otherwise

D1	mnem	Definition
122 (cont.)		1 is set if negative result, cleared otherwise 2 is set if non-zero result, cleared otherwise The arithmetic error bit (2) of the PSW is set if the sign of the result does not agree with the signs of the operands (add overflow).
123	DSUB	Double-precision SUBtract. The effective double-word is subtracted from the double-word accumulator. The result replaces the double-word accumulator specified by the AC-field of the instruction (must be even). The subtraction is carried out following the rules of two's complement arithmetic. condition code bit 0 is set if carry out of bit 0, cleared otherwise 1 is set if negative result, cleared otherwise 2 is set if non-zero result, cleared otherwise The arithmetic error bit (2) of the PSW may be set by this instruction.

D1	mnem	Definition
124		General double-precision memory modification instruction; the AC bits specify a particular operation. Condition code bit 0 is changed only by the two rotate instructions (AC = 4,5). Condition code bits 1 and 2 are set as follows for all modify instructions: 1 set if negative result, cleared otherwise 2 set if non-zero result, cleared otherwise

	AC	Operation
DTST	0	Double-precision TeST, no operation but condition code bits 1 and 2 are set to reflect the state of the effective double-word.
DCOM	1	Double-precision logical COMplement, the effective double-word is complemented on a bit-by-bit basis.
DNC	2	Double-precision iNCrement, one is added to the effective double-word.
DNEG	3	Double-precision NEGate, the effective double-word is negated (complemented then incremented). May cause arithmetic error
DRR	4	Double-precision Right Rotate, the effective double-word and condition code bit 0 are rotated together as a 33-bit register one place to the right, loading condition code bit 0 from bit 31 and bit 0 of the memory word from condition code bit 0.
DLR	5	Double-precision Left Rotate, the effective double-word and condition code bit 0 are rotated left together as a 33-bit register, loading condition code bit 0 from bit 0 of the memory word and bit 31 of the memory word from condition code bit 0.
	6	No operation.
DCLR	7	Double-precision CLeaR, the effective double-word is cleared.

Dl	mnem	Definition
125	DCMP	Double-precision CoMPare, the double-word accumulator and the effective double-word are algebraically compared. Neither the accumulator nor the effective double-word are changed, but condition code bit 1 and 2 are set according to the result.

condition code bit 0 remains unchanged

1 set if;
accumulator < effective double-word

cleared if;
accumulator > effective double-word

2 set if;
accumulator = effective double-word

cleared if;
accumulator = effective double-word

126 DMUL — Double-precision MULtiply, the effective double-word is algebraically multiplied by the low-order double-word of the quadruple-word specified by AC. If AC is divisible by four, the quadruple-word product replaces the quadruple-word at AC and is properly signed. If the specified accumulator (R) is not divisible by four, the high-order double-word of the product is discarded. The multiplier and multiplicand are treated as signed quantities. The condition codes remain unchanged. May cause arithmetic error.

Example:

```
DMUL  4, 300
```

The double-word located at locations 6,7 is multiplied by the double-word located at locations 300, 301. The signed product is placed in locations 4, 5, 6, 7.

D1	mnem	Definition
126 (cont.)		DMUL 6, 300 As above, but the high-order product is discarded and the low-order is placed in location 6, 7. DMUL 5, 300 Illegal DMUL 4, 301 Illegal
127	DDIV	Double-precision DIVide, the signed arithmetic quadruple-word beginning at the specified accumulator (AC) is algebraically divided by the effective double-word. The signed double-word quotient replaces the low-order double-word and the remainder, signed the same as the dividend, replaces its high-order part. When the relative magnitude of dividend and divisor is such that the quotient cannot be expressed as a 32-bit signed number, a divide overflow trap occurs, no divisor takes place and the dividend may be lost. If AC is not divisible by four, the corresponding even double-word is filled to produce a two's complement quadruple-precision dividend. The divide then proceeds normally. The condition codes remain unchanged. Example: DDIV 4, 1136 Divide the contents of 4, 5, 6, 7 by the contents of 1136, 1137. Place the remainder in 4, 5 and the quotient in 6, 7. DDIV 6, 1136 Divide the sign-extended double-word in locations 6, 7 by the contents of

Dl	mnem	Definition
127 (cont.)		1136, 1137. Place the remainder in 4, 5 and the quotient in 6, 7.
		DDIV 5, 1136
		Illegal
		DDIV 4, 1137
		Illegal

The double-precision instructions

DADD

DSUB

DNEG

DDIV

DMUL

may cause arithmetic errors and result in bit 2 of the PSW being set. If the traps are enabled, a trap will occur with location 16_8 receiving the Program Counter pointing to the instruction which caused the error.

The conditions which cause arithmetic errors in these instructions are analogous to these single-precision counterparts

ADD

SUB

NEG

DN

MUL

and the reader is referred to section 2.9 of PDP-X Technical Memorandum #29 or its revisions for further details.

2. Floating-Point Operations

Floating-point instructions operate upon single or double precision floating-point quantities as described in PDP-X Technical Memorandum #29, section 2.2.3. Arithmetic operations are performed with one operand in a floating-point register (see below) and another either in a register or from storage. The result is developed in a register (with the exception of STORE and MODIFY operations).

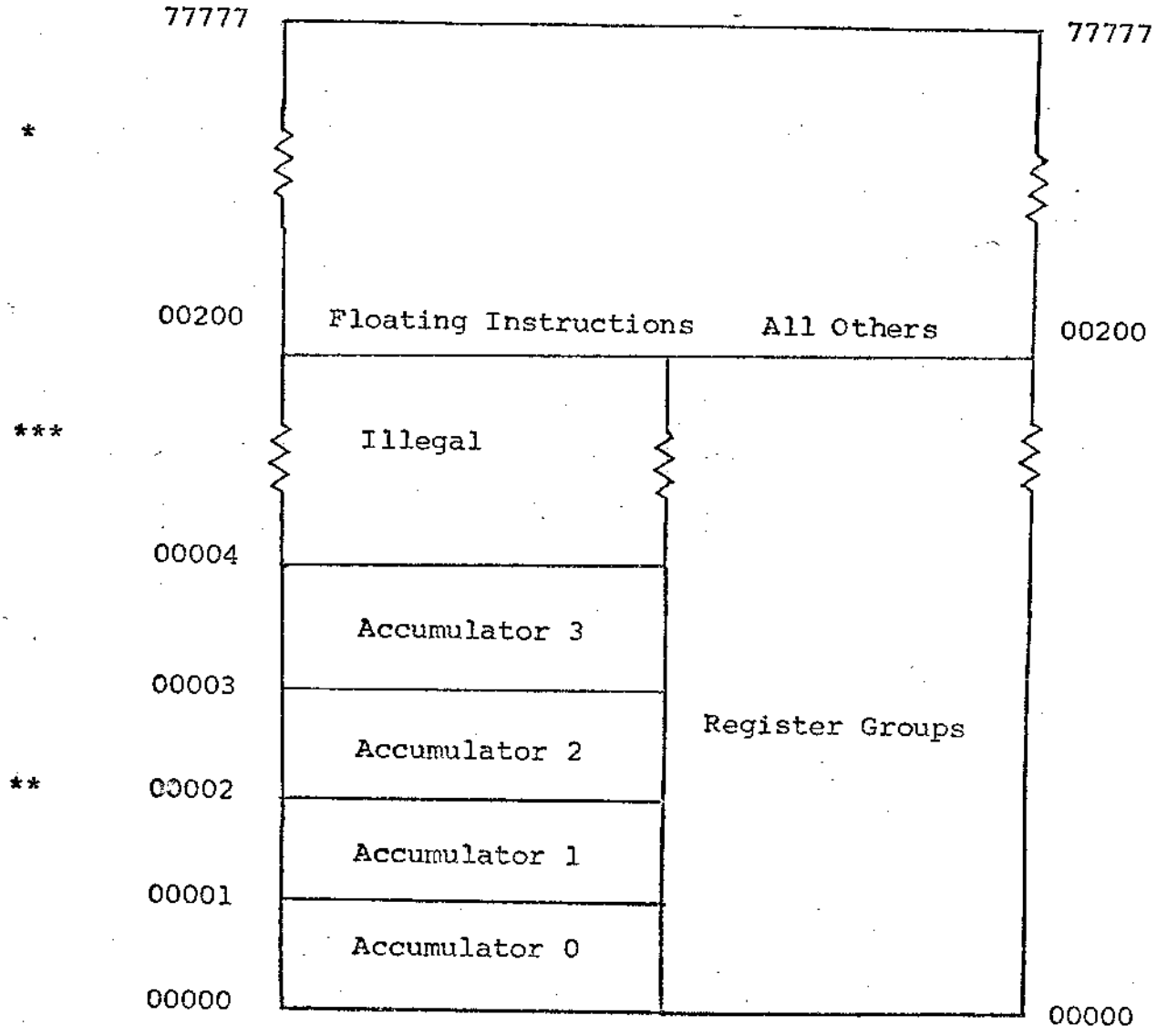
There are four floating-point accumulators, each 64 bits long. Floating-point double-precision operations use all 64 bits of the accumulators while single-precision operations use the high-order 32 bits (0-31). The floating-point accumulators may be addressed only by the floating-point instructions. No other instruction class can access them.

Floating-point quantities may be either single-precision (32 bits, 2 PDP-X words) or double-precision (64 bits, 4 PDP-X words). The addresses of single-precision floating-point quantities in storage must be even (effective address bit 15 a zero) or an address exception error will occur (bit 5 of the PSW is set). The addresses of double-precision floating-point quantities in storage must be divisible by four (effective address bits 14 and 15 zero) or an address exception error will occur. The effective addresses of floating-point instructions must be greater than 177_8 unless they are used to address the floating-point accumulators. In that case, the effective address of either a single-precision or a double-precision floating-point instruction must be 0, 1, 2, 3.

The floating-point accumulators have addresses of 0,1, 2, 3 in the floating-point address space (memory as seen by the floating-point instructions). They may be addressed either by the generated effective address or by the AC-field of the instruction (instruction bits 4, 5).

In each of the 12 floating-point instructions, the high-order AC-bit (instruction bit 3) is used to specify either a single-precision operation (0) or a double-precision operation (1).

The effective-address space of PDP-X is thus:



* Odd addresses illegal; for double-precision instructions (instruction bit 3 = 1), address must be divisible by four.

** 0, 1, 2, 3 used to address accumulators in double and single precision instructions (in effective address).

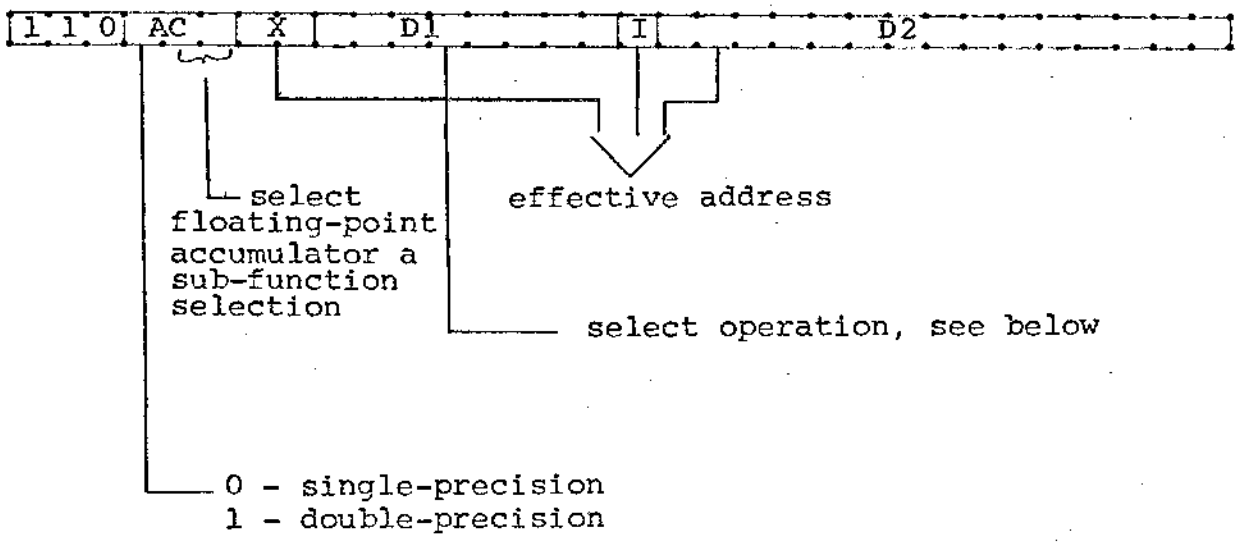
*** Effective addresses such that $3_8 < EFA < 200_8$ are illegal.

2. (Continued)

A quantity is represented with the greatest precision when it is normalized. A normalized floating-point number has a non-zero high-order hexadecimal fraction digit. If one or more high-order fraction digits are zero, the number is said to be unnormalized. The process of normalization consists of shifting the fraction left (4 bits at a time) until the high-order hexadecimal digit is non-zero and reducing the exponent by the number of hexadecimal digits shifted. A zero fraction cannot be normalized and its associated exponent, therefore, remains unchanged when normalization is called for. Since normalization applies to hexadecimal digits, the three high-order bits of a normalized number may be zero.

A number with a zero exponent, zero fraction and plus sign is called a true zero. A true zero may arise as the result of an arithmetic operation because of the particular magnitudes of the operands. A true zero is forced when the fraction becomes zero during an arithmetic operation (FNA, FUA, FNS, FDIV, FMUL). Zero fractions and true zero's participate as normal numbers in all operations. The sign of a sum, difference, product or quotient with zero fraction is positive. The sign of a zero fraction resulting from other operations is established by the rules of algebra from the operand signs.

All floating-point instructions are long form, EOP class instructions. Since all floating-point operands are either two or four PDP-X words, immediate mode is meaningless. Since the floating-point accumulators are addressable in the effective address, accumulator-accumulator instructions are possible.



D1	mnem	Definition
130	F(S)LD, F(D)LD	Floating-point LoaD accumulator. The floating-point accumulator selected by the AC-bits (instruction bits 4 and 5) is loaded with the single or double precision word located at the effective address. (These words remain unchanged.) If a single-precision operation is specified, the effective double-word is loaded into accumulator bits 0-31 and bits 32-63 of the accumulator are cleared. The condition codes remain unchanged. Examples: FSLD 2, A Single-precision load accumulator 2 with the contents of A, A + 1 (A must be even). FDLD 3, 2 Double-precision load accumulator 3 with the contents of accumulator 2. FSLD 2, 6 Illegal, effective address is less than 200_8 and greater than 3_8. FDLD 1, B Double-precision load accumulator 1 with the contents of B, B + 1, B + 2, B + 3 (B must be divisible by four).
131	F(S)LN, F(D)LN	Floating-point LoaD Negative. The floating-point accumulator selected by the AC-bits (instruction bits 4 and 5) is loaded with the negative of the single or double precision quantity located at the effective address. This operation is the same as FLD except that bit 0 of the selected accumulator is complemented after it is loaded.

D1	mnem	Definition
132	F(S)ST, F(D)ST	Floating-point STore accumulator. The floating-point accumulator selected by the AC-bits (instruction bits 4 and 5) replaces the single or double precision floating-point word located by the effective address. The selected accumulator remains unchanged. If a single processor operation is specified, accumulator bits 0-31 replace the two PDP-X words specified by the effective address. If a double-precision operation is specified, accumulator bits 0-63 replace the four PDP-X words specified by the effective address. The condition code remains unchanged.

Examples:

FDST 2, A

Double-precision store accumulator 2 into locations A, A + 1, A + 2, A + 3 (A must be divisible by four).

FSST 2, 100

Illegal; effective address less than 200_8 and greater than 3_8.

FSST 2, 3

Single-precision store accumulator 2 into accumulator 3. AC_3 bits 32-63 remain unchanged.

133	F(S)NA, F(D)NA	Floating-point Normalized Add. The normalized sum of the floating-point accumulator selected by the AC-bits (instruction bits 4 and 5) and the single or double precision word located by the effective address replaces the selected accumulator. The effective words remain unchanged (unless they are also modified in the AC-field). If a single

D1	mnem	Definition
133 (Cont.)		precision operation is specified, only the 32 high-order bits of the accumulator participate. Floating-point addition consists of exponent comparison and a fraction addition. The exponents of the two operands are compared and the fraction with the smaller characteristic right shifted with its exponent increased by one for each hexadecimal digit of shift, until the exponents agree. The fractions are then added algebraically to form an intermediate sum. If an overflow carry occurs, the intermediate sum is right shifted one hexadecimal digit and the exponent increased by one.* The single precision intermediate sum consists of seven hecadecimal digits and possible carry. The low-order digit is a guard digit retained from the fraction which was shifted right. There is one 1 guard digit which participates in the addition. If no right shift occurred, the guard digit is 0. The double-precision intermediate sum consists of 14 hexadecimal digits and a possible carry. No guard digit is retained. The intermediate sum is left-shifted as necessary to form a normalized fraction; vacated low-order digit positions are filled with zeros and the exponent is reduced by the amount of shift.* The sign of the sum is derived from the rules of algebra. The sign of a sum with zero result fraction is always positive. *if this causes exponent overflow or underflow, the arithmetic error bit (bit 2 of the PSW) is set and a trap may occur if enabled.

D1	mnem	Definition
133 (Cont.)		Condition code bits are set as follows: bit 0 remains unchanged 1 set for a negative result, cleared otherwise 2 set for a non-zero fraction, cleared otherwise
134	F(S)UA, F(D)UA	Floating-point Un-normalized Add. This operation is identical to FNA (above) except that no post-normalization of the intermediate result occurs (exponent overflow may occur as in FNA).
135	F(S)NS, F(D)NS	Floating-point Normalized Subtract. The normalized difference of the floating-point accumulator selected by the AC-bits (instruction bits 3 and 4) and the single or double precision word located by the effective address replaces the selected accumulator. FNS is identical to FNA except that the sign of the floating-point word located by the effective address is inverted before addition (exponent overflow or underflow may occur as in FNA).
136	F(S)MP, F(D)MP	Floating-point MultiPlication. The normalized algebraic product of the multiplier (specified by the effective address) and the multiplicand (specified by the low-order AC-bits of the instruction) replaces the multiplicand. In single-precision, the operands are 32 bits long (i.e., the low-order bits of the accumulator are ignored). In double-precision, the operands are 64 bits long. Floating-point multiplication is performed in five steps: a. The operands are pre-normalized (if necessary) and their exponents modified accordingly.

D1	mnem	Definition
136 (Cont.)		b. The exponents are added and the sum minus 100_8 becomes the exponent of the intermediate product.
		c. The operand fractions are multiplied together to form the fraction of the intermediate product. In both single and double precision, the product fraction is calculated to 14_{10} hexadecimal digits.
		d. The intermediate product is normalized (if necessary) and the intermediate characteristic is reduced by one for each left shift.
		e. The final product is stored. In double-precision operations, the product replaces bits 0-63 of the multiplicand. In single-precision operations, the product replaces bits 0-31 of the multiplicand. Bits 32-63 remain unchanged.
		When all 14_{10} result fraction digits are zero, the product sign and exponent bits are set to zero.
		Exponent underflow or overflow may occur. The operation is completed but bit 2 of the PSW is set and may cause a program trap.
		The condition code remains unchanged.

D1	mnem	Definition
140	F(S)DV, F(D)DV	Floating-point DiVision. The normalized algebraic quotient of the dividend (specified by the low-order AC-bits of the instruction) and the divisor (specified by the effective address) replaces the dividend. In single-precision, the operands are 32 bits long (i.e., the low-order bits of the accumulator are ignored and 2 words are fetched starting at the effective address). In double-precision, the operands are 64 bits long. Floating-point division is performed in four steps: a. The operands are pre-normalized (if necessary) and their exponents modified accordingly. b. The difference between the dividend and divisor exponents plus 100_8 becomes the exponent of the intermediate quotient. c. Dividend fraction is divided by the divisor fraction. The intermediate quotient fraction need not be normalized but a right shift may be necessary. The intermediate quotient exponent is adjusted for this shift (if taken). d. The quotient fraction is truncated to single-precision (if a single-precision operation) and the quotient is stored. In single precision operations, accumulator bits 32-63 remain unchanged. Exponent underflow or overflow may occur. The operation is completed and bit 2 of the PSW is set and may cause a program trap. If the divisor is zero, bit 2 of the PSW is set and the operation is terminated. When the dividend fraction is zero, the quotient will be a true zero. The condition code remains unchanged.

D1	mnem	Definition
141	F(S)CM, F(D)CM	Floating-point CoMpare. The specified accumulator and the effective word are algebraically compared. Neither accumulator nor effective word are changed, but condition code bits 1 and 2 are set according to the result. condition code bit 0 remains unchanged 1 set if; accumulator <effective word cleared if; accumualtor ≥effective word 2 set if; accumulator ≠effective word cleared if; accumulator=effective word In single-precision, the low-order bits of the two operands are ignored. The comparison is algebraic, taking into account the sign, exponent and fraction of each operand. An exponent inequality is not decisive for magnitude determination since the fractions may be different.

D1	mnem	AC	Definition
142			Floating-point Modify Group. The two low-order AC-bits are used to select one of four possible instructions. The high-order AC-bit (instruction bit 3) is used to select single or double precision mode.
142	F(S)TS, F(D)TS	0	Floating-point TeSt. The floating-point quantity selected by the effective address is tested and condition code bits 1 and 2 set accordingly. A number is said to be zero if it is a true zero or if the fraction is zero. condition code bit 0 unchanged 1 set if <0 cleared if $\geqslant 0$ 2 set if $\neq 0$ cleared if $= 0$
142	F(S)NG, F(D)NG	1	Floating-point NeGate. The floating-point quantity selected by the effective address is negated. The operation is performed by the inversion of the sign bit (bit 0). Condition code bits 1 and 2 are set as in FTS.
142	F(S)AB, F(D)AB	2	Floating-point ABsolute value. The floating-point quantity selected by the effective address is made positive. The operation is performed by clearing the sign bit (bit 0). Condition code bits 1 and 2 are set as in FTS. (Note, condition code bit 1 is never set as the result of this operation since the result cannot be negative.)

Dl	mnem	AC	Definition
142	F(S)CL, F(D)CL	3	Floating-point CLear. The floating-point number specified by the effective address is cleared to a true zero. Condition code bits 1 and 2 are set as in FTS. (Note, condition codes 1 and 2 are always cleared by this instruction since the result is positive and is zero.)

Floating-Point Operations Summary:

For the convenience of the programmer, two mnemonics are provided for each floating-point instruction. One mnemonic indicates a single-precision operation (i.e., bit 3 = 0) and the other indicates a double-precision operation (i.e., bit 3 = 1).

These instructions are summarized in the attached table.

Double-Precision Floating-Point Instructions

mnem	Definition	OP	D1(octal	AC(binary)	CC
FDAB	Floating-point Double-precision ABsolute value	6	142	110	1,2
FDCL	Floating-point Double-precision CLear	6	142	111	1,2
FDCM	Floating-point Double-precision CoMpare	6	141	1XX	1,2
FDDV	Floating-point Double-precision DiVide	6	140	1XX	---
FDLD	Floating-point Double-precision LoaD	6	130	1XX	---
FDLN	Floating-point Double-precision Load Negative	6	131	1XX	---
FDMP	Floating-point Double-precision MultiPly	6	136	1XX	---
FDNA	Floating-point Double-precision Normalized Add	6	133	1XX	1,2
FDNG	Floating-point Double-precision NeGate	6	142	101	1,2
FDNS	Floating-point Double-precision Normalized Subtract	6	135	1XX	1,2
FDST	Floating-point Double-precision STore	6	132	1XX	---
FDTS	Floating-point Double-precision TeSt	6	142	100	1,2
FDUA	Floating-point Double-precision Unnormalized Add	6	134	1XX	1,2

Single-Precision Floating-Point Instructions

mnem	Definition	OP	D1(octal)	AC(binary)	CC
FSAB	Floating-point Single-precision ABsolute value	6	142	010	1,2
FSCL	Floating-point Single-precision CLear	6	142	011	1,2
FSCM	Floating-point Single-precision CoMpare	6	141	0XX	1,2
FSDV	Floating-point Single-precision DiVide	6	140	0XX	---
FSLD	Floating-point Single-precision LoaD	6	130	0XX	---
FSLN	Floating-point Single-precision Load Negative	6	131	0XX	---
FSMP	Floating-point Single-precision MultiPly	6	136	0XX	---
FSNA	Floating-point Single-precision Normalized Add	6	133	0XX	1,2
FSNG	Floating-point Single-precision NeGate	6	142	001	1,2
FSNS	Floating-point Single-precision Normalized Subtract	6	135	0XX	1,2
FSST	Floating-point Single-precision STore	6	132	0XX	1,2
FSTS	Floating-point Single-precision TeSt	6	141	000	1,2
FSUA	Floating-point Unnormalized Add	6	134	0XX	1,2

Double-Precision Instructions

mnem	Definition	OP	Dl(octal)	AC(octal)	CC
DADD	Double-precision ADD	6	122	---	0,1,2
DCLR	Double-precision CLeaR	6	124	7	1,2
DCMP	Double-precision CoMpare	6	125	---	1,2
DCOM	Double-precision COMplement	6	124	1	1,2
DDIV	Double-precision DIVide	6	127	---	1,2
DLDA	Double-precision LoaD Accumulator	6	120	---	1,2
DLR	Double-precision Left Rotate	6	124	5	0,1,2
DMUL	Double-precision MULtiply	6	126	---	1,2
DNC	Double-precision iNCrement	6	124	2	1,2
DNEG	Double-precision NEGate	6	124	3	1,2
DRR	Double-precision Right Rotate	6	124	4	0,1,2
DSTA	Double-precision STore Accumulator	6	121	---	---
DSUB	Double-precision SUBtract	6	123	---	0,1,2
DTST	Double-precision TeST	6	124	0	1,2