Floating Point Assembly – Don't let your precision float away

Mohammed "Saif" Saifullah

SHARE 116 Anaheim, CA
Session 8542, March 1, 2011

EMC² where information lives

Yes, this is the moon.
Our own moon.

Pop Quiz: What is the surface area of the moon? How many countries of this world can fit into that area?

The surface area of a sphere is: $4 * \pi * R^2$

The radius of moon is about 1738.2 KM.

Now, let's plug in the values:

$4 * 3.14159265 * 1738.2 * 1738.2 = 37967268.598162344\ KM^2$

– That would be 37.9 million square kilometers.

Address	Object Code	S/390 Assembly	Reg./ Memory after execution
000036	B375 0010	LZDR R1	FPR1 00000000_00000000
00003A	ED10 C08C 0024	LDE R1,FOUR	FPR1 41400000_00000000
000040	7C10 C090	MDE R1,PIE	FPR1 41C90FDC_00000000
000044	7C10 C094	MDE R1,RADIUS	FPR1 445552DD_F73CD400
000048	7C10 C094	MDE R1,RADIUS	FPR1 47243559_FE390700
00004C	B3C9 0011	CGDR R1,0,R1	GR1 0243559F
000050	5010 C098	ST R1,FIXED	
000054	4E10 C09C	CVD R1,DECIMAL	00000003 7967263C
000088	45B27570	FLOAT DC	X'45B27570'
00008C	41400000	FOUR DC	E'4'
000090	413243F7	PIE DC	E'3.14159265E+0'
000094	436CA333	RADIUS DC	E'1.7382E+3'
000098	00000000	FIXED DC	F'0'
00009C	0000000000000000	DECIMAL DC	2F'0'

Agenda

1. Number Representation for Floating Point (FP) Arithmetic
2. Floating Point – Overview
3. Floating-Point Registers
4. Floating-Point Support Instructions (FPS)
5. Hexadecimal Floating-Point Instructions (HFP)
6. Intro: Binary and Decimal Floating-Point Instructions

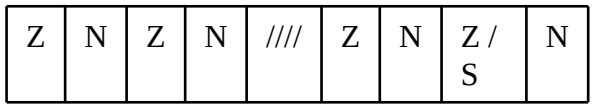
Number Representation

Four Basic Data Formats

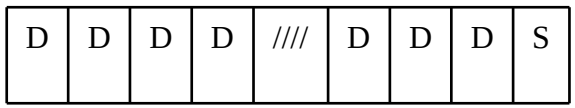
1. Signed Binary
2. Unsigned Binary
3. Unstructured logical data
4. Decimal data

For arithmetic operations, the decimal data is further divided as:

Zoned decimal format:



Packed decimal format:



These formats are very good for commercial applications:

Account balance:	12345.91
Interest rate:	5%
Interest to be paid:	617.2955
In banking terms, the amount will be:	$617.30

However, our Moon calculation is not that straight forward.:

$4 * 3.14159265 * 1738.2 * 1738.2 = 37967268.598162344\ KM^2$

Can be rewritten as:

$4.000 * 3.1416 * 1.7382 * 10^3 * 1.7382 * 10^3 = 37.9\ X\ 10^6$

Signed Binary Integer:

+26 is 0000 0000 0001 0101
-26 is 1111 1111 1110 0110

Unsigned Binary Integer:

199 is 1100 0111
221 is 1101 1101

All unsigned binary numbers are considered positive.

Packed Decimal Integers

+123 is 12 3C
-123 is 12 3D

Floating Point – Overview

Floating-Point Numbers

1. Hexadecimal Floating-Point (HFP) Numbers
2. Binary Floating-Point (BFP) Numbers
3. Decimal Floating-Point (DFP) Numbers

Short HFP Number

One Word

S	Characteristic	6-Digit Fraction
0	1 8	31

Long HPF Number

Two Word

S	Characteristic	14-Digit Fraction	14-Digit Fraction (Continued)
0	1 8	31	32 63

Extended HPF Number

S	High Ord Chst	Leftmost 14-Digit Fraction	Leftmost 14-Digit Fraction (Continued)
0	1 8	31	32 63

Four Word

S	Low Ord Chst	Rightmost 14-Digit Fraction	Rightmost 14-Digit Fraction (Continued)
64	72	95	96 127

The following example is from IBM's PoP:

1.0	$+1/16 \times 16^{1}$	$0\ 100\ 0001\ 0001\ 0000\ 0000\ 0000\ 0000\ 0000_2$
0.5	$+8/16 \times 16^{0}$	$0\ 100\ 0000\ 1000\ 0000\ 0000\ 0000\ 0000\ 0000_2$
1/64	$+4/16 \times 16^{-1}$	$0\ 011\ 1111\ 0100\ 0000\ 0000\ 0000\ 0000\ 0000_2$
0.0	$+0 \times 16^{-64}$	$0\ 000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000_2$
-15.0	$-15/16 \times 16^{1}$	$1\ 100\ 0001\ 1111\ 0000\ 0000\ 0000\ 0000\ 0000_2$
5.4×10^{-79}	$+1/16 \times 16^{-64}$	$0\ 000\ 0000\ 0001\ 0000\ 0000\ 0000\ 0000\ 0000_2$
7.2×10^{75}	$(1-16-6) \times 16^{63}$	$0\ 111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111_2$

EMC² where information lives

Conversion from decimal to HFP:

a) Split the number into decimal integer and decimal fraction:
61.25 = 61 plus 0.25

b) Convert both the components into hexadecimal representation:
61 = 3Cx
0.25 = 0.4x (A shortcut is to multiply by 16)

c) Put them back together as hexadecimal number:
3C.4x = 0.3C4x * 16^2

d) Develop characteristic by adding 64 in the actual location of radix point:
64 + 2 = 66 binary =100 0010

e) Put them together with a sign bit at bit position zero:

S	Char	Fraction
0	1000010	0110 1100 0100 0000 0000 0000

Hexadecimal Floating-Point Numbers

Hexadecimal-floating-point (HFP) operands have formats that provide for exponents that specify powers of the radix 16 and significands that are hexadecimal numbers.

Binary Floating-Point (BFP)

Binary-floating-point (BFP) operands have formats that provide for exponents that specify powers of the radix 2 and significands that are binary numbers.

Decimal Floating-Point (DFP)

Decimal-floating-point (DFP) operands have formats that provide for exponents that specify powers of the radix 10 and significands that are decimal numbers.

Floating-Point Data in Storage and Registers

Other non-number constructs for BFP and DFP

- Sign Bit
- Infinities
- Not-a-Number (NaN)
- Signaling NaNs
- Quiet NaNs
- Payload

Floating–Point Registers (FPR)

Registers and Controls

Floating-Point Registers

All floating-point instructions (FPS, BFP, DFP, and HFP) use the same 16 floating-point registers. The floating-point registers are identified by the numbers 0-15. A datum in the long format uses all the 64 bits. A datum in the extended (128-bit) format occupies a register pair. Register pairs are formed by coupling the 16 registers as follows: 0 and 2, 4 and 6, 8 and 10, 12 and 14, 1 and 3, 5 and 7, 9 and 11, and 13 and 15. Each of the eight pairs is referred to by the number of the lower-numbered register of the pair.

0	2	4	6	8	10	12	14
1	3	5	7	9	11	13	15

Floating-Point-Control (FPC) Register

The floating-point-control (FPC) register is a 32-bit register that contains mask bits, flag bits, a data exception code, IEEE exception trap code, and two rounding-mode fields. The bits of the FPC register are often referred to as, for example, FPC 1.0, meaning bit 0 of byte 1 of the register.

IEEE Invalid Operation

An IEEE invalid operation exception is recognized when, in the execution of an IEEE computational operation, any of the following occurs:

1. An SNaN is encountered in an IEEE computational operation.
2. A QNaN is encountered in an unordered-signaling comparison (COMPARE AND SIGNAL with a QNaN operand).
3. An IEEE difference is undefined (addition of infinities of opposite sign, or subtraction of infinities of like sign).
4. An IEEE product is undefined (zero times infinity).
5. An IEEE quotient is undefined (DIVIDE instruction with both operands zero or both operands infinity).
6. A BFP remainder is undefined (DIVIDE TO INTEGER with a dividend of infinity or a divisor of zero).
7. A BFP square root is undefined (negative nonzero operand).
8. Any other IEEE computational operation whose result is either undefined or not representable in the target format.

Control Instructions

All floating-point-support instructions are subject to the AFP-register-control bit, bit 45 of control register 0. The AFP-register-control bit must be one when an AFP register is specified as an operand location; otherwise, an AFP-register data exception, DXC 1, is recognized. Mnemonics for the floating-point instructions have an R as the last letter when the instruction is in the RR, RRE, or RRF format. Certain letters are used for floating-point instructions to represent operand-format length, as follows:

D	Long
E	Short
X	Extended

Condition Codes for IEEE Instructions

For those operations which set the condition code to indicate the value of an IEEE result, condition codes 0, 1, and 2 are set to indicate that the result is a zero of either sign, less than zero, or greater than zero, respectively.

CONVERT BFP TO HFP

Mnemonic R1,R2 [RRE]

Op Code	////////	R_1	R_2	
0	16	24	28	31

Mnemonic	Op Code	Operands
THDER	‘B358’	Short BFP operand, long HFP result
THDR	‘B359’	Long BFP operand, long HFP result

The second operand (the source operand) is converted from the binary-floating-point (BFP) format to the hexadecimal-floating-point (HFP) format, and the normalized result is placed at the first-operand location. The sign and magnitude of the source operand are tested to determine the setting of the condition code.

CONVERT HFP TO BFP

Mnemonic R1,R2 [RRE]

Op Code	M3	/////	R_1	R_2
0	16	20	24	28 31

Mnemonic	Op Code	Operands
TBEDR	'B350'	Long HFP operand, short BFP result
TBDR	'B351'	Long HFP operand, long BFP result

The second operand (the source operand) is converted from the hexadecimal-floating-point (HFP) format to the binary-floating-point (BFP) format. The result rounded according to the rounding method specified by the M3 field is placed at the first-operand location. The sign and magnitude of the source operand are tested to determine the setting of the condition code.

M3	Effective Rounding Method
0	Round toward 0
1	Round to nearest with ties away from 0
4	Round to nearest with ties to even
5	Round toward 0
6	Round toward +∞
7	Round toward -∞

COPY SIGN

CPSDR R1,R3, R2 [RRF]

'B372'	R_3	/////	R_1	R_2
0	16	20	24	28 31

The second operand is placed at the first-operand location with the sign bit set to the sign of the third operand. The first, second, and third operands are each in a 64-bit floating-point register. The sign bit of the second operand and bits 1-63 of the third operand are ignored.

EXTRACT FPC

EFPC R1 [RRE]

'B38C'	/////	/////	R_1	/////	
0	16	20	24	28	31

The contents of the FPC (floating-point-control) register are placed in bit positions 32-63 of the general register designated by R1. Bit positions 0-31 of the general register remain unchanged.

LOAD

Mnemonic R1,R2 [RR]

Op Code	R_1	R_2	
0	8	12	15

Mnemonic	Op Code	Operands
LER	’38’	Short
LDR	’28’	Long

Mnemonic2 R1,R2 [RRE]

Op Code	////////	R_1	R_2	
0	16	24	28	31

Mnemonic	Op Code	Operands
LXR	‘B365’	Extended

Mnemonic3 R1,D2(X2,B2) [RX]

Op Code	R_1	X_2	B_1	D_2	
0	8	12	16	20	31

Mnemonic3	Op Code	Operands
LE	’78’	Short
LD	’68’	Long

LOAD

Mnemonic4 R1,D2(X2,B2) [RXY]

Op Code	R_1	X_2	B_2	DL_2	DH_2	Op Code
0	8	12	16	20	32	40 47

Mnemonic3	Op Code	Operands
LEY	'ED64'	Short
LDY	'ED65'	Long

The second operand is placed unchanged at the first operand location.

The operation is performed without inspecting the contents of the second operand; no arithmetic exceptions are recognized.

For LXR, the R fields must designate valid floating-point-register pairs; otherwise, a specification exception is recognized.

LOAD COMPLEMENT

LCDFR R1,R2 [RRE]

Op Code	////////	R_1	R_2
0	16	24	28 31

The second operand is placed at the first-operand location with the sign bit inverted. Both the first and second operands are each in a 64-bit floating-point register.

LOAD FPC

LFPC D2(B2) [S]

‘B29D’	B_2	D_2
0	16	20 31

The four-byte second operand in storage is loaded into the FPC (floating-point-control) register.

Bits corresponding to unsupported bit positions in the FPC register must be zero; otherwise, a specification exception is recognized.

LOAD FPC AND SIGNAL

LFAS D2(B2) [S]

'B2BD'	B_2	D_2	
0	16	20	31

First, bits 0-4 of byte 1 of the floating-point-control (FPC) register at the beginning of the operation are preserved to be used as signaling flags. Next, the contents of the source operand are placed in the FPC register; then, the flags in the FPC register are set to the logical OR of the signaling flags and the source flags. Finally, the conditions for simulated- IEEE-exception trap action are examined. The source operand is the second operand in storage.

LOAD FPR FROM GR

LDGR R1,R2 [RRE]

'B3C1'	////////	R_1	R_2
0	16	24	28 31

The second operand is placed at the first-operand location. The second operand is in a general register, and the first operand is in a floating-point register.

LOAD GR FROM FPR

LGGR R1,R2 [RRE]

'B3CD'	////////	R_1	R_2
0	16	24	28 31

The second operand is placed at the first-operand location. The second operand is in a floating-point register, and the first operand is in a general register.

LOAD NEGATIVE

LGGR R1,R2 [RRE]

'B371'	////////	R_1	R_2
0	16	24	28 31

The second operand is placed at the first-operand location with the sign bit set to one. Both the first and second operands are each in a 64-bit floating-point register.

LOAD POSITIVE

LPDFR R1,R2 [RRE]

'B370'	////////	R_1	R_2
0	16	24	28 31

The second operand is placed at the first-operand location with the sign bit set to zero. Both the first and second operands are each in a 64-bit floating-point register.

LOAD ZERO

Mnemonic R1 [RRE]

Op Code	////////	R_1	//////
0	16	24	28 31

Mnemonic	Op Code	Operands
LZER	'B374'	Short
LZDR	'B375'	Long
LZXR	'B376'	Extended

All bits of the first operand are set to zeros.
For LZXR, the R1 field must designate a valid floating-point-register pair; otherwise, a specification exception is recognized.

PERFORM FLOATING-POINT OPERATION

PFPO [E]

'010A'
0 15

The operation specified by the function code in general register 0 is performed and the condition code is set to indicate the result. When there are no exceptional conditions, condition code 0 is set. When an IEEE nontrap exception is recognized, condition code 1 is set. When an IEEE trap exception with alternate action is recognized, condition code 2 is set. A 32-bit return code is placed in bits 32-63 of general register 1; bits. Powerful instruction see PoP. One usage is to convert among different FP formats.

SET BFP ROUNDING MODE

SRNM D2(B2) [S]

'B299'	B_2	D_2	
0	16	20	31

The BFP rounding-mode bits are set from the second-operand address.
The second-operand address is not used to address data; instead, the BFP rounding-mode bits in the FPC register are set with bits 62 and 63 of the address.
Bits other than 62 and 63 of the second-operand address are ignored.

SET DFP ROUNDING MODE

SRNMT D2(B2) [S]

'B2B9'	B_2	D_2	
0	16	20	31

The DFP rounding-mode bits are set from the second-operand address.
The second-operand address is not used to address data; instead, the DFP rounding-mode bits in the FPC register are set with bits 61-63 of the address. Bits other than 61-63 of the second-operand address are ignored.

SET FPC

SFPC R1 [RRE]

'B284'	//////	R_1	//////
0	16	24	28 31

The contents of bit positions 32-63 of the general register designated by R1 are placed in the FPC (floating-point-control) register.
All of bits 32-63 corresponding to unsupported bit positions in the FPC register must be zero; otherwise, a specification exception is recognized. For purposes of this checking, a bit position is considered to be unsupported only if it is either unassigned or assigned to a facility which is not installed in any architectural mode of the configuration.

SET FPC AND SIGNAL

SFASR R1 [RRE]

'B385'	//////	R_1	//////
0	16	24	28 31

First, bits 0-4 of byte 1 of the floating-point-control (FPC) register at the beginning of the operation are preserved to be used as signaling flags. Next, the contents of the source operand are placed in the FPC register; then, the flags in the FPC register are set to the logical OR of the signaling flags and the source flags. Finally, the conditions for simulated-IEEE-exception trap action are examined. The source operand is in bits 32-63 of the general register designated by R1.

STORE

Mnemonic R1,D2(X2,B2) [RX]

Op Code	R_1	X_2	B_1	D_2	
0	8	12	16	20	31

Mnemonic	Op Code	Operands
STE	'70'	Short
STD	'60'	Long

Mnemonic R1,D2(X2,B2) [RXY]

Op Code	R_1	X_2	B_2	DL_2	DH_2	Op Code	
0	8	12	16	20	32	40	47

Mnemonic	Op Code	Operands
STEY	'ED66'	Short
STDY	'ED67'	Long

The first operand is placed unchanged in storage at the second-operand location. The displacement for STE and STD is treated as a 12-bit unsigned binary integer. The displacement for STEY and STDY is treated as a 20-bit signed binary integer.

STORE FPC

STFPC D2(B2) [S]

'B29C'	B_2	D_2	
0	16	20	31

The contents of the FPC (floating-point-control) register are placed in storage at the second-operand location. The operand is four bytes in length. All 32 bits of the FPC register are stored.

Q&A

Hexadecimal Floating-Point Instructions (HFP)

Normalization

A quantity can be represented with the greatest precision by an HFP number of a given fraction length when that number is normalized. A normalized HFP number has a nonzero leftmost hexadecimal fraction digit. If one or more leftmost fraction digits are zeros, the number is said to be un-normalized. Un-normalized numbers are normalized by shifting the fraction left, one digit at a time, until the leftmost hexadecimal digit is nonzero and reducing the characteristic by the number of hexadecimal digits shifted.

HFP Data Formats

HFP numbers have a 32-bit (short) format, a 64-bit (long) format, or a 128-bit (extended) format. Numbers in the short and long formats may be designated as operands both in storage and in the floating-point registers, whereas operands having the extended format can be designated only in the floating-point registers. In all formats, the first bit (bit 0) is the sign bit (S). The next seven bits are the characteristic. In the short and long formats, the remaining bits constitute the fraction, which consists of six or 14 hexadecimal digits, respectively.

Short HFP Number

One Word

S	Characteristic	6-Digit Fraction
0	1 8	31

Long HPF Number

Two Word

S	Characteristic	14-Digit Fraction	14-Digit Fraction (Continued)
0	1 8	31	32 63

Extended HPF Number

S	High Ord Chst	Leftmost 14-Digit Fraction	Leftmost 14-Digit Fraction (Continued)
0	1 8	31	32 63

Four Word

S	Low Ord Chst	Rightmost 14-Digit Fraction	Rightmost 14-Digit Fraction (Continued)
64	72	95	96 127

ADD NORMALIZED

Mnemonic R1,R2 [RR]

Op Code	R_1	R_2	
0	16	20	31

Mnemonic	Op Code	Operands
AER	'3A'	Short HFP
ADR	'2A'	Long HPF
AXR	'36'	Extended HPF

Mnemonic R1,D2(X2,B2) [RX]

Op Code	R_1	X_2	B_2	D_2	
0	8	12	16	20	31

Mnemonic	Op Code	Operands
AE	'74'	Short HFP
AD	'6A'	Long HFP

The second operand is added to the first operand, and the normalized sum is placed at the first-operand location.

Addition of two HFP numbers includes characteristic comparison, fraction alignment, and signed fraction addition. The characteristics of the two operands are compared, and the fraction accompanying the smaller characteristic is aligned with the other fraction.

ADD UNNORMALIZED

Mnemonic R1,R2 [RR]

Op Code	R_1	R_2	
0	16	20	31

Mnemonic	Op Code	Operands
AUR	'3E'	Short HFP
AWR	'2E'	Long HPF

Mnemonic R1,D2(X2,B2) [RX]

Op Code	R_1	X_2	B_2	D_2	
0	8	12	16	20	31

Mnemonic	Op Code	Operands
AU	'7E'	Short HFP
AW	'6E'	Long HFP

The second operand is added to the first operand, and the un-normalized sum is placed at the first-operand location.

COMPARE

Mnemonic R1,R2 [RR]

Op Code	R_1	R_2	
0	16	20	31

Mnemonic	Op Code	Operands
CER	'39'	Short HFP
CDR	'29'	Long HPF

Mnemonic R1,R2 [RRE]

Op Code	//////	R_1	R_2	
0	16	24	28	31

Mnemonic	Op Code	Operands
CXR	'B369'	Extended HFP

Mnemonic R1,D2(X2,B2) [RX]

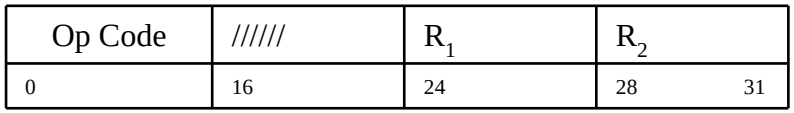
Op Code	R_1	X_2	B_2	D_2	
0	8	12	16	20	31

Mnemonic	Op Code	Operands
CE	'79'	Short HFP
CD	'69'	Long HFP

The first operand is compared with the second operand, and the condition code is set to indicate the result. The comparison is algebraic and follows the procedure for normalized subtraction, except that the difference is discarded after setting the condition code and both operands remain unchanged. When the difference, including the guard digit, is zero, the operands are equal.

CONVERT FROM FIXED

Mnemonic R1,R2 [RRE]



Mnemonic	Op Code	Operands
CEFR	'B3B4'	32-bit binary-integer operand, Short HFP result
CDFR	'B3B5'	32-bit binary-integer operand, Long HFP result
CXFR	'B3B6'	32-bit binary-integer operand, Extended HFP result
CEGR	'B3C4'	64-bit binary-integer operand, Short HFP result
CDGR	'B3C5'	64-bit binary-integer operand, Long HFP result
CXGR	'B3C4'	64-bit binary-integer operand, Extended HFP result

The fixed-point second operand is converted to the HFP format, and the normalized result is placed at the first-operand location. A nonzero result is normalized. A zero result is made a positive true zero. Very small number is rounded to 1 or zero depending on the rounding mode. Very large number gives Condition Code 3.

The second operand is a signed binary integer that is located in the general register designated by R2. A 32-bit operand is in bit positions 32-63 of the register.

CONVERT TO FIXED

Mnemonic R1,M1,R2 [RRF]

Op Code	M_1	//////	R_1	R_2
0	16	20	24	28 31

Mnemonic	Op Code	Operands
CEER	'B3B8'	Short HFP operand, 32-bit binary-integer result
CFDR	'B3B9'	Long HFP operand, 32-bit binary-integer result
CFXR	'B3BA'	Extended HFP operand, 32-bit binary-integer result
CGER	'B3C8'	Short HFP operand, 64-bit binary-integer result
CGDR	'B3C9'	Long HFP operand, 64-bit binary-integer result
CGXR	'B3CA'	Extended HFP operand, 64-bit binary-integer result

The HFP second operand is rounded to an integer value and then converted to the fixed-point format. The result is placed at the first-operand location. The result is a signed binary integer that is placed in the general register designated by R1. A 32-bit result replaces bits 32-63 of the register, and bits 0-31 of the register remain unchanged.

The second operand is rounded to an integer value by rounding as specified by the modifier in the M3 field:

M3	Effective Rounding Method
0	Round toward 0
1	Round to nearest with ties away from 0
4	Round to nearest with ties to even
5	Round toward 0
6	Round toward +∞
7	Round toward -∞

DIVIDE

Mnemonic R1,R2 [RR]

Op Code	R_1	R_2	
0	16	20	31

Mnemonic	Op Code	Operands
DER	'3D'	Short HFP
DDR	'2D'	Long HPF

Mnemonic2 R1,R2 [RRE]

Op Code	//////	R_1	R_2	
0	16	24	28	31

Mnemonic	Op Code	Operands
DXR	'B22D'	Extended HFP

Mnemonic3 R1,D2(X2,B2) [RX]

Op Code	R_1	X_2	B_2	D_2	
0	8	12	16	20	31

Mnemonic	Op Code	Operands
DE	'7D'	Short HFP
DD	'6D'	Long HFP

The first operand (the dividend) is divided by the second operand (the divisor), and the normalized quotient is placed at the first-operand location. No remainder is preserved.

HALVE

Mnemonic R1,R2 [RR]

Op Code	R_1	R_2	
0	16	20	31

Mnemonic	Op Code	Operands
HER	'34'	Short HFP
HDR	'24'	Long HPF

The second operand is divided by 2, and the normalized quotient is placed at the first-operand location.

LOAD AND TEST

Mnemonic R1,R2 [RR]

Op Code	R_1	R_2	
0	16	20	31

Mnemonic	Op Code	Operands
LTER	'32'	Short HFP
LTDR	'22'	Long HPF

Mnemonic2 R1,R2 [RRE]

Op Code	//////	R_1	R_2	
0	16	24	28	31

Mnemonic	Op Code	Operands
LTXR	'B362'	Extended HFP

The second operand is placed at the first-operand location, and its sign and magnitude are tested to determine the setting of the condition code. The condition code is set the same as for a comparison of the second operand with zero.

LOAD COMPLEMENT

Mnemonic R1,R2 [RR]

Op Code	R_1	R_2	
0	16	20	31

Mnemonic	Op Code	Operands
LCER	’33	Short HFP
LCDR	‘23’	Long HPF

Mnemonic2 R1,R2 [RRE]

Op Code	//////	R_1	R_2	
0	16	24	28	31

Mnemonic	Op Code	Operands
LCXR	‘B363’	Extended HFP

The second operand is placed at the first-operand location with the sign bit inverted.
The sign bit is inverted even if the operand is zero. For all operand lengths, the source fraction is placed unchanged in the result.

LOAD FP INTEGER

Mnemonic R1,R2 [RRE]

Op Code	//////	R_1	R_2
0	16	24	28 31

Mnemonic	Op Code	Operands
LIER	‘B377’	Short HFP
LIDR	‘B37F’	Long HFP
LIXR	‘B367’	Extended HFP

The second operand is truncated (rounded toward zero) to an integer value in the same floating-point format and the normalized result is placed at the first-operand location. A nonzero result is normalized. A zero result is made a positive true zero.

LOAD LENGTHENED

Mnemonic R1,R2 [RRE]

Op Code	//////	R_1	R_2
0	16	24	28 31

Mnemonic	Op Code	Operands
LDER	'B324'	Short HFP operand 2, Long HFP operand 1
LXDR	'B325'	Long HFP operand 2, Extended HFP operand 1
LXER	'B326'	Short HFP operand 2, Extended HFP operand 1

Mnemonic R1,D2(X2,b2) [RXE]

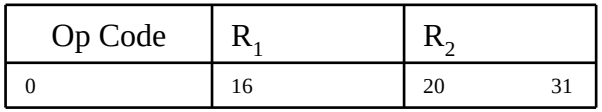
Op Code	R_1	X_2	B_2	D_2	//////	Op Code
0	8	12	16	20	22	40 47

Mnemonic	Op Code	Operands
LDE	'ED24'	Short HFP operand 2, Long HFP operand 1
LXD	'ED25'	Long HFP operand 2, Extended HFP operand 1
LXE	'ED26'	Short HFP operand 2, Extended HFP operand 1

The second operand is extended to a longer format, and the result is placed at the first-operand location.

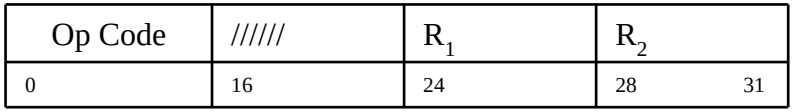
LOAD NEGATIVE

Mnemonic R1,R2 [RR]



Mnemonic	Op Code	Operands
LNER	'31'	Short HFP
LDDR	'21'	Long HPF

Mnemonic2 R1,R2 [RRE]



Mnemonic	Op Code	Operands
LNXR	'B361'	Extended HFP

The second operand is placed at the first-operand location with the sign bit made one.

LOAD POSITIVE

Mnemonic R1,R2 [RR]

Op Code	R_1	R_2	
0	8	12	15

Mnemonic	Op Code	Operands
LPER	'30'	Short HFP
LPDR	'20'	Long HPF

Mnemonic2 R1,R2 [RRE]

Op Code	//////	R_1	R_2	
0	16	24	28	31

Mnemonic	Op Code	Operands
LPXR	'B360'	Extended HFP

The second operand is placed at the first-operand location with the sign bit made zero.

LOAD ROUNDED

Mnemonic R1,R2 [RR]

Op Code	R_1	R_2	
0	8	12	15

Mnemonic	Op Code	Operands
LEDR	'35'	Long HFP operand 2, Short HFP operand 1
LDXR	'25'	Extended Long HPF operand 2, Long operand 1

Mnemonic2 R1,R2 [RRE]

Op Code	//////	R_1	R_2	
0	16	24	28	31

Mnemonic	Op Code	Operands
LEXR	'B366'	Extended HFP 2, Short HFP operand 1

The second operand is placed at the first-operand location rounded to the length of first operand.

MULTIPLY

Mnemonic R1,R2 [RR]

Op Code	R_1	R_2	
0	8	12	15

Mnemonic	Op Code	Operands
MDR	'2C'	Long HFP
MXR	'26'	Extended HFP
MDER	'3C'	Short HFP multiplier and multiplicand, Long HFP product
MXDR	'27'	Long HFP multiplier and multiplicand, Extended HFP product

Mnemonic2 R1,R2 [RRE]

Op Code	//////	R_1	R_2	
0	16	24	28	31

Mnemonic	Op Code	Operands
MEER	'B337'	Short HFP

Mnemonic R1,D2(X2,b2) [RXE]

Op Code	R_1	X_2	B_2	D_2	//////	Op Code	
0	8	12	16	20	22	40	47

Mnemonic	Op Code	Operands
MEE	'ED37'	Short HFP

MULTIPLY

Mnemonic3 R1,D2(X2,B2) [RX]

Op Code	R_1	X_2	B_2	D_2	
0	8	12	16	20	31

Mnemonic	Op Code	Operands
MD	'6C'	Long HFP
MDE	'7C'	Short HFP multiplier and multiplicand, Long HFP product
MXD	'67'	Long HFP multiplier and multiplicand, Extended HFP product

The normalized product of the second operand (the multiplier) and the first operand (the multiplicand) is placed at the first-operand location.

MULTIPLY AND ADD

Mnemonic R1,R2 [RRF]

Op Code	R_1	//////	R_3	R_2	
0	16	20	24	28	31

Mnemonic	Op Code	Operands
MAER	'B32E'	Short HFP
MADR	'B33E'	Long HFP

Mnemonic R1,R2,D2(X2,b2) [RXF]

Op Code	R_1	X_2	B_2	D_2	R_1	//////	Op Code	
0	8	12	16	20	32	36	40	47

Mnemonic	Op Code	Operands
MAE	'ED2E'	Short HFP
MAD	'ED3E'	Long HFP

The third operand is multiplied by the second operand, and then the first operand is added to from the product. The sum is placed at the first-operand location.

MULTIPLY AND SUBTRACT

Mnemonic R1,R2 [RRF]

Op Code	R_1	//////	R_3	R_2	
0	16	20	24	28	31

Mnemonic	Op Code	Operands
MSER	'B32F'	Short HFP
MSDR	'B33F'	Long HFP

Mnemonic R1,R2,D2(X2,b2) [RXF]

Op Code	R_1	X_2	B_2	D_2	R_1	//////	Op Code	
0	8	12	16	20	32	36	40	47

Mnemonic	Op Code	Operands
MSE	'ED2F'	Short HFP
MSD	'ED3F'	Long HFP

The third operand is multiplied by the second operand, and then the first operand is subtracted to from the product. The difference is placed at the first-operand location.

MULTIPLY AND ADD UNNORMALIZED

Mnemonic R1,R2 [RRF]

'Op Code	R_1	//////	R_3	R_2	
0	16	20	24	28	31

Mnemonic	Op Code	Operands
MAYR	'B32A'	Long HFP sources, extended HFP result
MAYHR	'B33C'	Long HFP sources, high-order part of extended HFP result
MAYLR	'B338'	Long HFP sources, low-order part of extended HFP result

Mnemonic R1,R2,D2(X2,b2) [RXF]

Op Code	R_1	X_2	B_2	D_2	R_1	//////	Op Code	
0	8	12	16	20	32	36	40	47

Mnemonic	Op Code	Operands
MAY	'ED3A'	Long HFP sources, extended HFP result
MAYH	'ED3C'	Long HFP sources, high-order part of extended result
MAYL	'ED38'	Long HFP sources, low-order part of extended result

The second and third HFP operands are multiplied, forming an intermediate product; the first operand (addend) is then added algebraically to the intermediate product to form an intermediate sum; the intermediate-sum fraction is truncated on the left or on the right, if need be, to form an intermediate extended result. The operands, intermediate values, and results are not normalized to eliminate leading hexadecimal zeros.

MULTIPLY UNNORMALIZED

Mnemonic R1,R2 [RRF]

'Op Code	R_1	//////	R_3	R_2	
0	16	20	24	28	31

Mnemonic	Op Code	Operands
MYR	'B33B'	Long HFP multiplier & multiplicand, extended HFP product
MYHR	'B33D'	Long HFP multiplier & multiplicand, high-order part of extended HFP product
MYLR	'B339'	Long HFP multiplier & multiplicand, low-order part of extended HFP product

Mnemonic R1,R2,D2(X2,b2) [RXF]

Op Code	R_1	X_2	B_2	D_2	R_1	//////	Op Code	
0	8	12	16	20	32	36	40	47

Mnemonic	Op Code	Operands
MY	'ED3B'	Long HFP multiplier & multiplicand, extended HFP product
MYH	'ED3D'	Long HFP multiplier & multiplicand, high-order part of extended product
MYL	'ED39'	Long HFP multiplier & multiplicand, low-order part of extended product

The second and third HFP operands are multiplied, forming an intermediate product, which, in turn, is used to form an intermediate extended result. The intermediate extended result is placed in the floating-point-register designated by the R1 field. The operands, intermediate values, and results are not normalized to eliminate leading hexadecimal zeros.

SQUARE ROOT

Mnemonic R1,R2 [RRF]

'Op Code	//////	R_1	R_2
0	16	24	28 31

Mnemonic	Op Code	Operands
SQER	'B245'	Short HFP
SQDR	'B244'	Long HFP
SQXR	'B336'	Extended HFP

Mnemonic R1,D2(X2,b2) [RXF]

Op Code	R_1	X_2	B_2	D_2	/////	Op Code
0	8	12	16	20	32	40 47

Mnemonic	Op Code	Operands
SQE	'ED34'	Short HFP
SQD	'ED35'	Long HFP

The normalized and rounded square root of the second operand is placed at the first-operand location. When the fraction of the second operand is zero, the sign and characteristic of the second operand are ignored, and the operation is completed by placing a positive true zero at the first-operand location.

SUBTRACT NORMALIZED

Mnemonic R1,R2 [RR]

Op Code	R_1	R_2	
0	16	20	31

Mnemonic	Op Code	Operands
SER	'3B'	Short HFP
SDR	'2B'	Long HPF
SXR	'37'	Extended HPF

Mnemonic R1,R2 [RRF]

'Op Code	R_1	//////	R_3	R_2	
0	8	12	16	28	31

Mnemonic	Op Code	Operands
SE	'7B'	Short HFP
SD	'6B'	Long HFP

The second operand is subtracted from the first operand, and the normalized difference is placed at the first-operand location.

SUBTRACT UNNORMALIZED

Mnemonic R1,R2 [RR]

Op Code	R_1	R_2	
0	16	20	31

Mnemonic	Op Code	Operands
SUR	'3F'	Short HFP
SWR	'2F'	Long HPF

Mnemonic R1,D2,(X2,B2) [RX]

'Op Code	R_1	X_2	B_2	D_2	
0	8	12	16	28	31

Mnemonic	Op Code	Operands
SU	'7F'	Short HFP
SW	'6F'	Long HFP

The second operand is subtracted from the first operand, and the unnormalized difference is placed at the first-operand location.
The execution of SUBTRACT UNNORMALIZED is identical to that of ADD UNNORMALIZED, except that the second operand participates in the operation with its sign bit inverted.

Q&A

THANK YOU