Micromega Corporation

uM-FPU V2
Instruction Reference

Overview

The uM-FPU V2.0 coprocessor is connected to a microcontroller using either an SPI or I2C interface. The microcontroller sends instructions and data to the uM-FPU, the uM-FPU executes the instructions, and the microcontroller reads the results. The uM-FPU contains sixteen 32-bit registers, numbered 0 through 15, which are used to store floating point or long integer values. Register 0 is modified by some of the uM-FPU instructions, and should be regarded as a working register. Registers 1 through 15 are available for general use. Instructions are executed in the order that they are sent to the uM-FPU. Arithmetic operations are defined in terms of register A and register B. Register A and register B can be any of the sixteen registers and are selected prior to an operation using the SELECTA and SELECTB instructions, or they are selected as part of the instruction itself. For example:

These two instructions add the floating point value of register 2 to register1.

Opcode	Instruction	Description
01	SELECTA+1	select register 1 as A
62	FADD+2	select register 2 as B, calculate A = A + B

These two instructions calculate the sine of the angle in register 3.

Opcode	Instruction	Description
03	SELECTA+3	select register 3 as A
E5	SIN	calculate A = sin(A)

These three instructions calculate the value of register 7 raised to the power of register 8.

Opcode	Instruction	Description
07	SELECTA+7	select register 7 as A
08	SELECTB+8	select register 8 as B
FE E0	POWER	compute A = A to the power of B

The uM-FPU V2 processor has a 32 byte instruction buffer. Prior to issuing any instruction that reads data from the uM-FPU, the Busy/Ready status must be checked to ensure that all of the instructions have been executed. If more than 32 bytes are required to specify a sequence of operations, the Busy/Ready status must be checked at least every 32 bytes to ensure that the instruction buffer does not overflow. See the datasheet for details about the SPI or I2C interface.

Floating Point Instructions

SELECTA	Select A
SELECTB	Select B
FWRITEA	Select A, and write 32-bit value to A
FWRITEB	Select B, and write 32-bit value to B
FREAD	Read the floating point value from register
READFLOAT	Read floating point value of register A
FSET	Select B, A = B
FADD	Select B, A = A + B
FSUB	Select B, A = A - B
FMUL	Select B, A = A * B
FDIV	Select B, A = A / B
SQRT	A = sqrt(A)
LOG	A = log(A)
LOG10	A = log10(A)
EXP	A = exp(A)
EXP10	A = exp10(A)
FLOOR	A = floor(A)
CEIL	A = ceil(A)
ROUND	A = round(A)
NEGATE	A = -A
ABS	A = |A|
INVERSE	A = 1 / A
MIN	A = minimum of A and B
MAX	A = maximum of A and B
POWER	A = A to the power of B
ROOT	A = the Bth root of A
SIN	A = sin(A)
COS	A = cos(A)
TAN	A = tan(A)
ASIN	A = asin(A)
ACOS	A = acos (A)
ATAN	A = atan(A)
ATAN2	A = atan(A/B)
DEGREES	Convert radians to degrees
RADIANS	Convert degrees to radians
FLOAT	register 0 = float(A)
FIX	register 0 = fix(A)
FRACTION	Load register 0 with the fractional portion of A
FSTATUS	Get the status of A
FCOMPARE	Compare A and B
LOADBYTE	Load register 0 with 8-bit signed integer converted to floating point
LOADUBYTE	Load register 0 with 8-bit unsigned integer converted to floating point
LOADWORD	Load register 0 with 16-bit signed integer converted to floating point
LOADUWORD	Load register 0 with 16-bit unsigned integer converted to floating point
LOADZERO	Load register 0 with zero (long integer or floating point)
LOADONE	Load register 0 with floating point value of 1.0
LOADE	Load register 0 with floating point value of e (2.7182818)
LOADPI	Load register 0 with floating point value of Pi (3.1415927)

Long Integer Instructions

SELECTA	Select A
SELECTB	Select B
LWRITEA	Select A, and write 32-bit value to A
LWRITEB	Select B, and write 32-bit value to B
LREAD	Read long integer value from register
READBYTE	Read lower 8 bits of register A
READWORD	Read lower 16 bits of register A
READLONG	Read long integer value of register A
LSET	Select B, A = B
LADD	Select B, A = A + B
LSUB	Select B, A = A – B
LMUL	Select B, A = A * B
LDIV	Select B, A = A / B, remainder in register 0
LUDIV	Select B, A = A / B (unsigned), remainder in register 0
LNEGATE	A = -A
LABS	A = |A|
LINCA	A = A + 1
LINCB	B = B + 1
LDECA	A = A - 1
LDECB	B = B - 1
LAND	A = A AND B
LOR	A = A OR B
LXOR	A = A XOR B
LNOT	A = NOT A
LTST	A = return status of A AND B
LSHIFT	A = A shift by B bit positions
FIX	register 0 = fix(A)
FLOAT	register 0 = float(A)
LSTATUS	Get the long integer status
LCOMPARE	Compare A and B
LUCOMPARE	Compare A and B (unsigned)
LONGBYTE	Load register 0 with 8-bit signed integer converted to long integer
LONGUBYTE	Load register 0 with 8-bit unsigned integer converted to long integer
LONGWORD	Load register 0 with 16-bit signed integer converted to long integer
LONGUWORD	Load register 0 with 16-bit unsigned integer converted to long integer
LOADZERO	Load register 0 with zero (long integer or floating point)

Left and Right Parentheses

LEFT	Save A register and select new temporary register as A register
RIGHT	Return value in register 0 and restore previous A register

Conversion Instructions

ATOF	Convert ASCII string to floating point value, store in register 0
ATOL	Convert ASCII string to long integer value, store in register 0
FTOA	Convert floating point value to ASCII string and store in string buffer
LTOA	Convert long integer value to ASCII string and store in string buffer
VERSION	Copy version string to the string buffer
READSTR	Read zero terminated string from string buffer

Stored Function Instructions

FUNCTION	Execute user defined function
TABLE	Load A register with 32-bit value from table using register B as index
POLY	Calculate Nth order polynomial
IF_FSTATUSA	Conditional Execution
IF_FSTATUSB	
IF_FCOMPARE	
IF_LSTATUSA	
IF_LSTATUSB	
IF_LCOMPARE	
IF_LUCOMPARE	
IF_LTST	

Miscellaneous Instructions

SYNC	Synchronization
IEEEMODE	Select IEEE floating point format
PICMODE	Select PIC floating point format
XOP	Prefix for extended opcodes
NOP	No operation

Debug Instructions

BREAK	Debug breakpoint
TRACEOFF	Turn debug trace off
TRACEON	Turn debug trace on
TRACESTR	Send debug string to trace buffer
CHECKSUM	Calculate checksum and store in register 0

Further Information

Check the Micromega website at www.micromegacorp.com

ABS **A = |A|**
Opcode: `EC`

Description: Calculates the absolute value of the floating point value in register A, and stores the result in register A.

Special case:
- if A is NaN, then the result is NaN

ACOS **A = acos(A)**
Opcode: `FE E6`

Description: Calculates the arc cosine of an angle in the range 0.0 through pi. The initial value is contained in register A, and the result is returned in register A.

Special case:
- if A is NaN or its absolute value is greater than 1, then the result is NaN

ASIN **A = asin(A)**
Opcode: `FE E5`

Description: Calculates the arc sine of an angle in the range of –pi/2 through pi/2. The initial value is contained in register A, and the result in returned in register A.

Special cases:
- if A is NaN or its absolute value is greater than 1, then the result is NaN
- if A is 0.0, then the result is a 0.0
- if A is –0.0, then the result is –0.0

ATAN **A = atan(A)**
Opcode: `FE E7`

Description: Calculates the arc tangent of an angle in the range of –pi/2 through pi/2. The initial value is contained in register A, and the result in returned in register A.

Special cases:
- if A is NaN, then the result is NaN
- if A is 0.0, then the result is a 0.0
- if A is –0.0, then the result is –0.0

ATAN2 **A = atan(A/B)**
Opcode: `FE E8`

Description: Calculates the arc tangent of an angle in the range of –pi/2 through pi/2. The initial value is determined by dividing the value in register A by the value in register B, and the result in returned in register A. This instruction is used to convert rectangular coordinates (A, B) to polar coordinates (r, theta). The value of theta is returned in register A.

Special cases:
- if A or B is NaN, then the result is NaN
- if A is 0.0 and B > 0, then the result is 0.0
- if A > 0 and finite, and B is +inf, then the result is 0.0
- if A is –0.0 and B > 0, then the result is –0.0
- if A < 0 and finite, and B is +inf, then the result is –0.0

• if A is 0.0 and B < 0, then the result is pi
• if A > 0 and finite, and B is –inf, then the result is pi
• if A is –0.0, and B < 0, then the result is –pi
• if A < 0 and finite, and B is –inf, then the result is –pi
• if A > 0, and B is 0.0 or –0.0, then the result is pi/2
• if A is +inf, and B is finite, then the result is pi/2
• if A < 0, and B is 0.0 or –0.0, then the result is –pi/2
• if A is –inf, and B is finite, then the result is –pi/2
• if A is +inf, and B is +inf, then the result is pi/4
• if A is +inf, and B is –inf, then the result is 3*pi/4
• if A is –inf, and B is +inf, then the result is –pi/4
• if A is –inf, and B is –inf, then the result is –3*pi/4

ATOF **Convert a zero terminated ASCII string to floating point**

Opcode: `F9 nn nn … 00` (where `nn` and `00` are the bytes of the string)

Description: Converts a zero terminated ASCII string to a 32-bit floating point number, stores the result in register 0, and selects register 0 as register B. The string to convert is sent immediately following the opcode. The string can be normal number format (e.g. 1.56, -0.5) or exponential format (e.g. 10E6). Conversion will stop at the first invalid character, but data will continue to be read until a zero terminator is encountered.

Example:

```
F9 32 2E 35 34 00          (string 2.54)  stores the value 2.54 in register 0
F9 31 46 33 00             (string 1E3)   stores the value 1000.0 in register 0
```

ATOL **Convert a zero terminated ASCII string to long integer**

Opcode: `FB nn nn … 00` (where `nn` and `00` are the bytes of the string)

Description: Converts a zero terminated ASCII string to a 32-bit long integer, stores the result in register 0, and selects register 0 as register B. The string to convert is sent immediately following the opcode. Conversion will stop at the first invalid character, but data will continue to be read until a zero terminator is encountered.

Example:

```
FB 35 30 30 30 30 30 00 (string 500000)  stores the value 500000 in register 0
FB 35 45 00             (string -5)      stores the value -5 in register 0
```

BREAK **Debug breakpoint**

Opcode: `FE FB`

Description: Used in conjunction with the built-in debugger. If the debugger is enabled, a breakpoint occurs and the debug monitor is entered. If debug mode is not selected, this instruction is ignored.

CEIL **A = ceil(A)**

Opcode: `E9`

Description: Calculates the floating point value equal to the nearest integer that is greater than or equal to the floating point value in register A. The result is stored in register A.

Special cases:
• if A is NaN, then the result is NaN
• if A is +infinity or -infinity, then the result is +infinity or -infinity
• if A is 0.0 or –0.0, then the result is 0.0 or –0.0
• if A is less than zero but greater than –1.0, then the result is –0.0

CHECKSUM **Calculate a checksum for uM-FPU code**

Opcode: `FE FA`

Description: A checksum is calculated for the uM-FPU code and stored in register 0. This is used as a diagnostic test for confirming the state of a uM-FPU chip.

COS **A = cos(A)**

Opcode: `E6`

Description: Calculates the cosine of the angle (in radians) in register A and stored the result in register A.

Special case:
- if A is NaN or an infinity, then the result is NaN

DEGREES **Convert radians to degrees**

Opcode: `EE`

Description: The floating point value in register A is converted from radians to degrees and the result is stored in register A.

Special case:
- if A is NaN, then the result is NaN

EXP **A = exp(A)**

Opcode: `E3`

Description: Calculates the value of e (2.7182818) raised to the power of the floating point value in register A. The result is stored in register A.

Special cases:
- if A is NaN, then the result is NaN
- if A is +infinity or greater than 88, then the result is +infinity
- if A is –infinity or less than -88, then the result is 0.0

EXP10 **A = exp10(A)**

Opcode: `E4`

Description: Calculates the value of 10 raised to the power of the floating point value in register A. The result is stored in A.

Special cases:
- if Ais NaN, then the result is NaN
- if A is +infinity or greater than 38, then the result is +infinity
- if A is –infinity or less than -38, then the result is 0.0

FADD **A = A + B**

Opcode: `6x` (where `x` specifies register B)

Description: The floating point value in register B is added to the floating point value in register A and the result is stored in register A. The lower 4 bits of the opcode are used to select register B.

Special cases:
- if either value is NaN, then the result is NaN
- if one value is +infinity and the other is –infinity, then the result is NaN
- if one value is +infinity and the other is not –infinity, then the result is +infinity

• if one value is -infinity and the other is not +infinity, then the result is -infinity

---

FCOMPARE **Compare A and B**

Opcode: `F3`

Returns: `nn` (where `nn` is the status byte)

Description: Compares the floating point values in registers A and B. The status byte must be read immediately following this instruction. The status byte is set as follows:

BIT	7	6	5	4	3	2	1	0
	1	-	-	-	-	N	S	Z

Bit 2	Not-a-Number	Set if either value is not a valid number
Bit 1	Sign	Set if A < B
Bit 0	Zero	Set if A = B
		If neither Bit 0 or Bit 1 is set, A > B

FDIV **A = A / B**

Opcode: `9x` (where `x` specifies register B)

Description: The floating point value in register A is divided by the floating point value in register B and the result is stored in register A. The lower 4 bits of the opcode are used to select register B.

Special cases:
• if either value is NaN, then the result is NaN
• if both values are zero or both values are infinity, then the result is NaN
• if B is zero and A is not zero, then the result is infinity
• if B is infinity, then the result is zero

FIX **register 0 = fix(A)**

Opcode: `F2`

Description: Converts the floating point value in register A to a long integer value and stores the result in register 0.

Special cases:
• if A is NaN, then the result is zero
• if A is +infinity or greater than the maximum signed long integer, then the result is the maximum signed long integer (decimal: 2147483647, hex: $7FFFFFFF)
• if A is –infinity or less than the minimum signed long integer, then the result is the minimum signed long integer (decimal: -2147483648, hex: $80000000)

FLOAT **register 0 = float(A)**

Opcode: `F1`

Description: Converts the long integer value in register A to a floating point value and stores the result in register 0.

FLOOR **A = floor(A)**

Opcode: `E8`

Description: Calculates the floating point value equal to the nearest integer that is less than or equal to the floating point value in register A. The result is stored in register A.

Special cases:
- if A is NaN, then the result is NaN
- if A is +infinity or -infinity, then the result is +infinity or -infinity
- if A is 0.0 or –0.0, then the result is 0.0 or –0.0

FMUL **A = A * B**

Opcode: `8x` (where `x` specifies register B)

Description: The floating point value in register A is multiplied by the floating point value in register B and the result is stored in register A. The lower 4 bits of the opcode are used to select register B.

Special cases:
- if either value is NaN, or one value is zero and the other is infinity, then the result is NaN
- if either values is infinity and the other is nonzero, then the result is infinity

---

FRACTION **Load register 0 with the fractional part of A**

Opcode: `FE E4`

Description: Register 0 is loaded with the fractional part the floating point value in register A.

Special cases:
- if A is NaN or infinity, then the result is NaN

FREAD **Read floating point value from register**

Opcode: `4x` (where `x` specifies the register)
Returns: `nn nn nn nn` (where `nn` are data bytes, MSB first)

Description: Returns the floating point value of the register selected by the lower 4 bits of the opcode. The four bytes of the 32-bit floating point value must be read immediately following this instruction. If the PIC data format has been selected (using the PICMODE instruction), the IEEE 754 format floating point value is converted to PIC format before being sent.

FSET **A = B**

Opcode: `5x` (where `x` specifies register B)

Description: Sets the value of register A to the value of register B. The lower 4 bits of the opcode are used to select register B.

FSTATUS **Get the floating point status of A**

Opcode: `FD`
Returns: `nn` (where `nn` is the status byte)

Description: Get the status of the floating point value in register A. The status byte must be read following this instruction. The status byte is set as follows:

BIT	7	6	5	4	3	2	1	0
	1	-	-	-	I	N	S	Z

Bit 3	Infinity	Set if the value is an infinity
Bit 2	Not-a-Number	Set if the value is not a valid number
Bit 1	Sign	Set if the value is negative
Bit 0	Zero	Set if the value is zero

FSUB **A = A – B**

Opcode: `7x` (where x specifies register B)

Description: The floating point value in register B is subtracted from the floating point value in register A and the result is stored in register A. The lower 4 bits of the opcode are used to select register B.

Special cases:
- if either value is NaN, then the result is NaN
- if both values are infinity and the same sign, then the result is NaN
- if the A value is +infinity and the B value not +infinity, then the result is +infinity
- if the A value is -infinity and the B value not -infinity, then the result is -infinity
- if the A value is not an infinity and the B value is an infinity, then the result is an infinity of the opposite sign as the B value

---

FTOA **Convert floating point value to ASCII string and store in string buffer**

Opcode: `FA nn` (where nn is the format byte)

Description: The floating point value in register A is converted to an ASCII string and stored in the string buffer. The byte immediately following the opcode is the format byte and determines the format of the converted value.

If the format byte is zero, as many digits as necessary will be used to represent the number with up to eight significant digits. Very large or very small numbers are represented in exponential notation. The length of the displayed value is variable and can be from 3 to 12 characters in length. The special cases of NaN (Not a Number), +infinity, -infinity, and -0.0 are handled. Examples of the ASCII strings produced are as follows:

```
1.0              NaN              0.0
10e20            Infinity         -0.0
3.1415927        -Infinity        1.0
-52.333334       -3.5e-5          0.01
```

If the format byte is non-zero, it is interpreted as a decimal number. The tens digit specifies the maximum length of the converted string, and the ones digit specifies the number of decimal points. The maximum number of digits for the formatted conversion is 9, and the maximum number of decimal points is 6. If the floating point value is too large for the format specified, asterisks will be stored. If the number of decimal points is zero, no decimal point will be displayed. Examples of the display format are as follows:

Value in register A	Format byte	Display format
123.567	61 (6.1)	123.6
123.567	62 (6.2)	123.57
123.567	42 (4.2)	*.**
0.9999	20 (2.0)	1
0.9999	31 (3.1)	1.0

This instruction is normally followed by a **READSTR** instruction to read the string.

FUNC **Execute user defined function**

Opcode:
`FE 0x` (where x specifies the lower 4 bits of function numbers 0 to 15)
`FE 1x` (where x specifies the lower 4 bits of function numbers 16 to 31)
`FE 2x` (where x specifies the lower 4 bits of function numbers 32 to 47)
`FE 3x` (where x specifies the lower 4 bits of function numbers 48 to 63)

Description: The specified user function is executed from uM-FPU flash memory. The lower 6 bits of the

opcode are used to select the user function. If the selected user function has not been defined, register 0 will be set to NaN and the instruction will terminate. User functions are programmed by the user using the debug monitor (see the uM-FPU datasheet). Functions are defined as a pre-defined series of uM-FPU instructions, and can modify any register. Register B is set to register 0 after all user functions.

FWRITEA **Select A, and write floating point value to A**

Opcode: `2x nn nn nn nn` (where `x` specifies register A, and `nn` are the data bytes, MSB first)

Description: A floating point value is stored in register A. The lower 4 bits of the opcode are used to select register A, and the four bytes immediately following the opcode contain the 32-bit floating point value. If the PIC data format has been selected (using the PICMODE instruction), the PIC format floating point value is converted to IEEE 754 format before being stored in register A.

FWRITEB **Select B, and write floating point value to B**

Opcode: `3x nn nn nn nn` (where `x` specifies register A, and `nn` are the data bytes, MSB first)

Description: A floating point value is stored in register B. The lower 4 bits of the opcode are used to select register B, and the four bytes immediately following the opcode specify the 32-bit floating point value. If the PIC data format has been selected (using the PICMODE instruction), the PIC format floating point value is converted to IEEE 754 format before being stored in register B.

IEEEMODE **Select IEEE floating point format**

Opcode: `FE F8`

Description: Selects the IEEE 754 floating point format for the FREAD, FWRITEA, FWRITEB, and READFLOAT instructions. This is the default mode on reset and only needs to be changed if the PICMODE instruction has been used.

IF_FCOMPARE Conditional memory function, floating point compare of A and B

Opcode: `FE 82 tt cc nn … nn` (where `tt` is the test conditions, `cc` is size of code block, and `nn` are the bytes of the conditional code block)

Description: This opcode is only valid within a user function stored in the uM-FPU flash memory. If the result of a floating point compare of the values in register A and B matches the test conditions, the block of code that follows is executed, otherwise the block of code is skipped.

IF_FSTATUSA Conditional memory function, floating point status of A

Opcode: `FE 80 tt cc nn … nn` (where `tt` is the test conditions, `cc` is size of code block, and `nn` are the bytes of the conditional code block)

Description: This opcode is only valid within a user function stored in the uM-FPU flash memory. If the floating point status of register A matches the test conditions, the block of code that follows is executed, otherwise the block of code is skipped.

IF_FSTATUSB Conditional memory function, floating point status of B

Opcode: `FE 81 tt cc nn … nn` (where `tt` is the test conditions, `cc` is size of code block, and `nn` are the bytes of the conditional code block)

Description: This opcode is only valid within a user function stored in the uM-FPU flash memory. If the floating point status of register B matches the test conditions, the block of code that follows is executed, otherwise the block of code is skipped.

IF_LCOMPARE Conditional memory function, signed long compare of A and B

Opcode: `FE 85 tt cc nn … nn` (where `tt` is the test conditions, `cc` is size of code block, and `nn` are the bytes of the conditional code block)

Description: This opcode is only valid within a user function stored in the uM-FPU flash memory. If the result of a signed long integer compare of the values in register A and B matches the test conditions, the block of code that follows is executed, otherwise the block of code is skipped.

IF_LSTATUSA Conditional memory function, long integer status of A

Opcode: `FE 83 tt cc nn … nn` (where `tt` is the test conditions, `cc` is size of code block, and `nn` are the bytes of the conditional code block)

Description: This opcode is only valid within a user function stored in the uM-FPU flash memory. If the long integer status of register A matches the test conditions, the block of code that follows is executed, otherwise the block of code is skipped.

IF_ LSTATUSB Conditional memory function, long integer status of B

Opcode: `FE 84 tt cc nn … nn` (where `tt` is the test conditions, `cc` is size of code block, and `nn` are the bytes of the conditional code block)

Description: This opcode is only valid within a user function stored in the uM-FPU flash memory. If the long integer status of register B matches the test conditions, the block of code that follows is executed, otherwise the block of code is skipped.

IF_LTST Conditional memory function, bitwise AND of A and B

Opcode: `FE 87 tt cc nn … nn` (where `tt` is the test conditions, `cc` is size of code block, and `nn` are the bytes of the conditional code block)

Description: This opcode is only valid within a user function stored in the uM-FPU flash memory. If the bitwise `AND` of the value in register A and the value in register B matches the test conditions, the block of code that follows is executed, otherwise the block of code is skipped.

IF_LUCOMPARE Conditional memory function, unsigned long compare of A and B

Opcode: `FE 86 tt cc nn … nn` (where `tt` is the test conditions, `cc` is size of code block, and `nn` are the bytes of the conditional code block)

Description: This opcode is only valid within a user function stored in the uM-FPU flash memory. If the result of an unsigned long integer compare of the values in register A and B matches the test conditions, the block of code that follows is executed, otherwise the block of code is skipped.

INVERSE **A = 1 / A.**
Opcode: `ED`

Description: The inverse of the floating point value in register A is stored in register A.

Special cases:
- if A is NaN, then the result is NaN
- if A is zero, then the result is infinity
- if A is infinity, then the result is zero

LABS **A = |A|**
Opcode: `FE ED`

Description: The absolute value of the long integer value in register A is stored in register A.

LADD **A = A + B**
Opcode: `Ax` (where `x` specifies register B)

Description: The long integer value in register B is added to the long integer value in register A and the result is stored in register A. The lower 4 bits of the opcode are used to select register B.

LAND **A = A AND B**

Opcode: `FE 98`

Description: The bitwise AND of the values in register A and B is calculated and stored in register A.

LCOMPARE **Compare A and B**
Opcode: `FE E9`
Returns: `nn` (where `nn` is the status byte)

Description: Compares the signed long integer values in registers A and B. The status byte must be read immediately following this instruction. The status byte is set as follows:

BIT	7	6	5	4	3	2	1	0
	1	-	-	-	-	-	S	Z

Bit 1	Sign	Set if A < B
Bit 0	Zero	Set if A = B
		If neither Bit 0 or Bit 1 is set, A > B

LDECA **A = A - 1**

Opcode: `FE 96`

Description: The long integer value in register A is decremented by one.

LDECB **B = B - 1**
Opcode: `FE 97`

Description: The long integer value in register B is decremented by one.

LDIV **A = A / B**

Opcode: `Dx` (where `x` specifies register B)

Description: The long integer value in register A is divided by the long integer value in register B and the result is stored in register A. The remainder of the division is stored in register 0. The lower 4 bits of the opcode are used to select register B.

Special cases: • if B is zero, the result is the largest positive long integer ($3FFFFFFF)

LEFT **Left Parenthesis**

Opcode: `FE EE`

Returns: `none`

Description: The left parenthesis command saves the current register A selection, allocates the next temporary register, and selects the new temporary register as register A. Used together with the right parenthesis command to allocate temporary registers, and to change the order of a calculation. There are five temporary registers, so parentheses can be nested up to five levels.

Special cases: • the maximum number of temporary registers is five. If the maximum number is exceeded, the value of register A is set to NaN ($7FC00000).

LINCA **A = A + 1**

Opcode: `FE 94`

Description: The long integer value in register A is incremented by one.

LINCB **B = B + 1**

Opcode: `FE 95`

Description: The long integer value in register B is incremented by one.

LMUL **A = A * B**

Opcode: `Cx` (where `x` specifies register B)

Description: The long integer value in register A is multiplied by the long integer value in register B and the result is stored in register A. The lower 4 bits of the opcode are used to select register B.

LNEGATE **A = -A**

Opcode: `FE EC`

Description: The negative of the long integer value in register A is stored in register A.

LNOT **A = NOT A**

Opcode: `FE 9B`

Description: The bitwise complement of the value in register A is stored in register A.

LOADBYTE **Load register 0 with 8-bit signed integer converted to floating point**
Opcode: `F4 nn` (where `nn` is the data byte)

Description: Loads register 0 with the 8-bit signed integer value following the opcode, converts it to a floating point value, and selects register 0 as register B.

LOADE **Load register 0 with floating point value of e (2.7182818)**
Opcode: `FE F2`

Description: Loads register 0 with the floating point value of e (2.7182818), and selects register 0 as register B.

LOADONE **Load register 0 with One.**
Opcode: `FE F1`

Description: Loads register 0 with the floating point value 1.0, and selects register 0 as register B.

LOADPI **Load register 0 with value of Pi.**
Opcode: `FE F3`

Description: Loads register 0 with the floating point value of pi (3.1415927), and selects register 0 as register B.

LOADUBYTE **Load register 0 with 8-bit unsigned integer converted to floating point**
Opcode: `F5 nn` (where `nn` is the data byte)

Description: Loads register 0 with the 8-bit unsigned integer value following the opcode, converts it to a floating point value, and selects register 0 as register B.

LOADUWORD **Load register 0 with 16-bit unsigned integer converted to floating point**
Opcode: `F7 nn nn` (where `nn` are the data bytes, MSB first)

Description: Loads register 0 with the 16-bit unsigned integer value following the opcode, converts it to a floating point value, and selects register 0 as register B.

LOADWORD **Load register 0 with 16-bit signed integer converted to floating point**
Opcode: `F6 nn nn` (where `nn` are the data bytes, MSB first)

Description: Loads register 0 with the 16-bit signed integer value following the opcode, converts it to a floating point value, and selects register 0 as register B.

LOADZERO **Load register 0 with Zero.**
Opcode: `FE F0`

Description: Loads register 0 with a value of zero, and selects register 0 as register B. Used to load a floating point zero or a long integer zero.

LOG **A = log(A)**
Opcode: `E1`

Description: Calculates the natural log of the floating point value in register A. The result is stored in register A. The number e (2.7182818) is the base of the natural system of logarithms.

Special cases:
- if the value is NaN or less than zero, then the result is NaN
- if the value is +infinity, then the result is +infinity
- if the value is 0.0 or –0.0, then the result is -infinity

LOG10 **A = log10(A)**
Opcode: `E2`

Description: Calculates the base 10 logarithm of the floating point value in register A. The result is stored in register A.

Special cases:
- if the value is NaN or less than zero, then the result is NaN
- if the value is +infinity, then the result is +infinity
- if the value is 0.0 or –0.0, then the result is -infinity

LONGBYTE **Load register 0 with 8-bit signed integer converted to long integer**
Opcode: `FE F4 nn` (where nn is the data byte)

Description: Loads register 0 with the 8-bit signed integer value following the opcode, converts it to a long integer value, and selects register 0 as register B.

LONGUBYTE **Load register 0 with 8-bit unsigned integer converted to long integer.**
Opcode: `FE F5 nn` (where nn is the data byte)

Description: Loads register 0 with the 8-bit unsigned integer value following the opcode, converts it to a long integer value, and selects register 0 as register B.

LONGUWORD **Load register 0 with 16-bit unsigned integer converted to long integer.**
Opcode: `FE F7 nn nn` (where nn are the data bytes, MSB first)

Description: Loads register 0 with the 16-bit unsigned integer value following the opcode, converts it to a long integer value, and selects register 0 as register B.

LONGWORD **Load register 0 with 16-bit signed integer converted to long integer**
Opcode: `FE F6 nn nn` (where nn are the data bytes, MSB first)

Description: Loads register 0 with the 16-bit signed integer value following the opcode, converts it to a long integer value, and selects register 0 as register B.

LOR **A = A OR B**
Opcode: `FE 99`

Description: The bitwise OR of the values in register A and B is calculated and stored in register A.

LREAD **Get the long integer value of a register.**

Opcode: `FE Cx` (where `x` specifies the register)
Returns: `nn nn nn nn` (where `nn` are the data bytes, MSB first)

Description: Returns the long integer value from the register selected by the lower 4 bits of the opcode. The four bytes of the 32-bit long integer value must be read immediately following this instruction.

LSET **A = B**

Opcode: `5x` (where `x` specifies register B)

Description: Sets the value of register A to the value of register B. The lower 4 bits of the opcode are used to select register B.

LSHIFT **A = A shifted by B bit positions**

Opcode: `FE 9D`

Description: The value in register A is shifted by the number of bit positions specified by the long integer value in register B. Register A is shifted left if the value in B is positive and right if the value is negative.

Special cases:
- if B = 0, no shift occurs
- if B > 32 or B < –32, the result is zero

LSTATUS **Get the long integer status of A**

Opcode: `FE EB`
Returns: `nn` (where `nn` is the status byte)

Description: Get the status of the long integer value in register A. The status byte must be read immediately following this instruction. The status byte is set as follows:

BIT	7	6	5	4	3	2	1	0
	1	-	-	-	-	-	S	Z

Bit 1	Sign	Set if the value is negative
Bit 0	Zero	Set if the value is zero

LSUB **A = A – B**

Opcode: `Bx` (where `x` specifies register B)

Description: The long integer value in register B is subtracted from the long integer value in register A and the result is stored in register A. The lower 4 bits of the opcode are used to select register B.

LTOA **Convert long integer value to ASCII string and store in string buffer**

Opcode: `FC nn` (where `nn` is the format byte)

Description: The long integer value in register A is converted to an ASCII string and stored in the string buffer. The byte immediately following the opcode is the format byte and determines the format of the converted value.

If the format byte is zero, the length of the converted string is variable and can range from 1 to 11 characters in length. Examples of the converted string are as follows:

1
500000
-3598390

If the format byte is non-zero, it is interpreted as a decimal number. A value between 0 and 15 specifies the length of the converted string. The converted string is right justified. If 100 is added to the format value the value is converted as an unsigned long integer, otherwise it is converted as an signed long integer. If the value is larger than the specified width, asterisks are stored. If the length is specified as zero, the string will be as long as necessary to represent the number. Examples of the converted string are as follows:

Value in register A	Format byte	Display format
-1	10	(signed 10)	-1
-1	110	(unsigned 10)	4294967295
-1	4	(signed 4)	-1
-1	104	(unsigned 4)	****
0	4	(signed 4)	0
0	0	(unformatted)	0
1000	6	(signed 6)	1000

The maximum length of the string is 15. This instruction is normally followed by a **READSTR** instruction to read the string.

LTST **Return the status of A AND B**

Opcode: `FE 9C`

Description: Returns a status byte based on the result of a bitwise AND of the values in registers A and B. (The values of the A and B registers are not changed.) The status byte must be read immediately following this instruction. The status byte is set as follows:

Bit 1	Sign	Set if the value is negative
Bit 0	Zero	Set if the value is zero

LUCOMPARE **Compare A and B (unsigned)**

Opcode: `FE EA`
Returns: `nn` (where `nn` is the status byte)

Description: Compares the unsigned long integer values in registers A and B. The status byte must be read immediately following this instruction. The status byte is set as follows:

BIT	7	6	5	4	3	2	1	0
	1	-	-	-	-	-	S	Z

Bit 1	Sign	Set if A < B
Bit 0	Zero	Set if A = B
		If neither Bit 0 or Bit 1 is set, A > B

LUDIV **A = A / B (unsigned)**

Opcode: `FE Dx` (where `x` specifies register B)

Description: The unsigned long integer value in register A is divided by the unsigned long integer value in register B and the result is stored in register A. The remainder of the division is stored in register 0. The lower 4 bits of the opcode are used to select register B.

Special cases: • if B is zero, the result is the largest positive long integer ($3FFFFFFF)

LWRITEA **Load register A with long integer value**

Opcode: `FE Ax nn nn nn nn` (where x specifies register A, and nn are the data bytes, MSB first)

Description: A long integer value is stored in register A. The lower 4 bits of the opcode are used to select register A, and the four bytes immediately following the opcode contain the 32-bit long integer value.

LWRITEB **Load register B with long integer value**

Opcode: `FE Bx nn nn nn nn` (where x specifies register A, and nn are the data bytes, MSB first)

Description: A long integer value is stored in register B. The lower 4 bits of the opcode are used to select register B, and the four bytes immediately following the opcode contain the 32-bit long integer value.

LXOR **A = A XOR B**

Opcode: `FE 9A`

Description: The bitwise XOR of the values in register A and B is calculated and stored in register A.

MAX **A = maximum of A and B**

Opcode: `FE E3`

Description: The maximum floating point value of registers A and B is stored in register A.

Special cases: • if either value is NaN, then the result is NaN

MIN **A = minimum of A and B**

Opcode: `FE E2`

Description: The minimum floating point value of registers A and B is stored in register A.

Special cases: • if either value is NaN, then the result is NaN

NEGATE **A = -A**

Opcode: `EB`

Description: The negative of the floating point value in register A is stored in register A.

Special case: • if the value is NaN, then the result is NaN

NOP **No operation**

Opcode: `FF`

Description: No operation.

PICMODE **Select PIC floating point format**

Opcode: `FE 89 nn nn … nn` (where nn are the bytes of the conditional code blocks)

Description: Selects the alternate PIC floating point mode using by many PIC compilers. All internal data on the uM-FPU is stored in IEEE 754 format, but when the uM-FPU is in PIC mode an automatic conversion is done by the FREAD, FWRITEA, FWRITEB, and READFLOAT instructions so the PIC program can use floating point data in the alternate format. Normally this instruction would be issued immediately after the reset as part of the initialization code. The IEEEMODE instruction can be used to revert to standard IEEE 754 floating point mode..

POLY **A = nth order polynomial**

Opcode: `FE 89 nn yy yy zz zz …` (where nn is the order of the polynomial, followed by the `yyyyzzzz` coefficient of each term)

Description: This opcode is only valid within a user function stored in the uM-FPU flash memory. The value of the specified polynomial is calculated and stored in register A. The general form of the polynomial is:

$$y = A_0 + A_1x^1 + A_2x^2 + \ldots A_nx^n$$

The value of n is the order of the polynomial and is stored in the first byte following the opcode. The value of x is the initial value of register A. The coefficient values A0, A1, A2, … An are stored as a series of four byte floating point values in order from N to 0. If a given term in the polynomial is not needed, a zero is stored for that value.

Example: The polynomial 3x + 5 would be represented as follows:

```
FE 89 01 40 A0 00 00 40 40 00 00
```

Where:	`FE 89`	opcode
`01`	order of the polynomial	
`40 40 00 00`		floating point constant 3.0
`40 A0 00 00`		floating point constant 5.0

POWER **A = A raised to the power of B**

Opcode: `FE E0`

Description: The floating point value in register A is raised to the power of the floating point value in register B and stored in register A.

Special cases:
- if B is 0.0 or –0.0, then the result is 1.0
- if B is 1.0, then the result is the same as the A value
- if B is NaN, then the result is Nan
- if A is NaN and B is nonzero, then the result is NaN
- if |A| > 1 and B is +infinite, then the result is +infinity
- if |A| < 1 and B is -infinite, then the result is +infinity
- if |A| > 1 and B is -infinite, then the result is 0.0
- if |A| < 1 and B is +infinite, then the result is 0.0
- if |A| = 1 and B is infinite, then the result is NaN
- if A is 0.0 and B > 0, then the result is 0.0
- if A is +infinity and B < 0, then the result is 0.0
- if A is 0.0 and B < 0, then the result is +infinity

• if A is +infinity and B > 0, then the result is +infinity
• if A is -0.0 and B > 0 but not a finite odd integer, then the result is 0.0
• if the A is -infinity and B < 0 but not a finite odd integer, then the result is 0.0
• if A is -0.0 and the B is a positive finite odd integer, then the result is –0.0
• if A is -infinity and B is a negative finite odd integer, then the result is –0.0
• if A is -0.0 and B < 0 but not a finite odd integer, then the result is +infinity
• if A is -infinity and B > 0 but not a finite odd integer, then the result is +infinity
• if A is -0.0 and B is a negative finite odd integer, then the result is –infinity
• if A is -infinity and B is a positive finite odd integer, then the result is –infinity
• if A < 0 and B is a finite even integer,
then the result is equal to |A| to the power of B
• if A < 0 and B is a finite odd integer,
then the result is equal to the negative of |A| to the power of B
• if A < 0 and finite and B is finite and not an integer, then the result is NaN

RADIANS **Convert degrees to radians**

Opcode: `EF`

Description: The floating point value in register A is converted from degrees to radians and the result is stored in register A.

Special case: • if the value is NaN, then the result is NaN

READBYTE **Read the lower 8-bits of register A**

Opcode: `FE 90`
Returns: `nn` (where `nn` is the data byte)

Description: Returns the lower 8 bits of register A. The byte containing the 8-bit long integer value must be read immediately following the instruction.

READFLOAT **Read the floating point value of register A**

Opcode: `FE 93`
Returns: `nn nn nn nn` (where `nn` are the data bytes, MSB first)

Description: Returns the floating point value of register A. The four bytes of the 32-bit floating point value must be read immediately following this instruction. If the PIC data format has been selected (using the PICMODE instruction), the IEEE 754 format floating point value is converted to PIC format before being sent.

READLONG **Read the long integer value of register A**

Opcode: `FE 92`
Returns: `nn nn nn nn` (where `nn` are the data bytes, MSB first)

Description: Returns the 32-bit long integer value of register A. The four bytes of the 32-bit long integer value must be read immediately following this instruction.

READSTR **Reads a zero terminated string from the string buffer**

Opcode: `F8`
Returns: `nn nn … 00` (where `nn` and `00` are the bytes of the string)

Description: Returns the zero terminated string in the string buffer. Data bytes must be read immediately following this instruction and continue until a zero byte is read. This instruction is typically used after an FTOA, LTOA or VERSION instruction.

READWORD **Read the lower 16-bits of register A**

Opcode: `FE 91`

Returns: `nn nn` (where `nn` are the data bytes, MSB first)

Description: Returns the lower 16 bits of register A. The two bytes containing the 16-bit long integer value must be read immediately following this instruction.

RIGHT **Right Parenthesis**

Opcode: `FE EF`

Description: The right parenthesis command copies the value of register A (the current temporary register) to register 0, and selects register 0 as register B. If the right parenthesis is the outermost parenthesis, the register A selection from before the first left parenthesis is restored, otherwise the previous temporary register is selected as register. Used together with the left parenthesis command to allocate temporary registers, and to change the order of a calculation. There are five temporary registers, so parentheses can be nested up to five levels.

Special case:
- if no left parenthesis is currently outstanding, then the value of register 0 is set to NaN. ($7FC00000).

ROOT **A = the Bth root of A**

Opcode: `FE E1`

Description: Calculates the n^{th} root of the floating point value in register A and stores the result in register A. Where the value n is equal to the floating point value in register B. It is equivalent to raising A to the power of (1/B).

Special cases:
- see the description of the POWER instruction for the special cases of (1/B)
- if B is infinity, then (1/B) is zero
- if B is zero, then (1/B) is infinity

ROUND **A = round(A)**

Opcode: `EA`

Description: The floating point value equal to the nearest integer to the floating point value in register A is stored in register A.

Special cases:
- if the value is NaN, then the result is NaN
- if the value is +infinity or -infinity, then the result is +infinity or -infinity
- if the value is 0.0 or –0.0, then the result is 0.0 or –0.0

SELECTA **Select A**

Opcode: `0x` (where `x` specifies register A)

Description: The lower 4 bits of the opcode are used to select register A.

SELECTB **Select B**

Opcode: `1x` (where `x` specifies register B)

Description: The lower 4 bits of the opcode are used to select register B.

SIN **A = sin(A)**

Opcode: `E5`

Description: Calculates the sine of the angle (in radians) in register A and stored the result in register A.

Special cases:
- if A is NaN or an infinity, then the result is NaN
- if A is 0.0, then the result is 0.0
- if A is –0.0, then the result is –0.0

SQRT **A = sqrt(A)**

Opcode: `E0`

Description: Calculates the square root of the floating point value in register A and stored the result in register A.

Special cases:
- if the value is NaN or less than zero, then the result is NaN
- if the value is +infinity, then the result is +infinity
- if the value is 0.0 or –0.0, then the result is 0.0 or –0.0

SYNC **Synchronization**

Opcode: `F0`
Returns: `5C`

Description: A sync character (`0x5C`) is sent in reply. This instruction is typically used after a reset to verify communications.

TABLE **A = value from table indexed by B**

Opcode: `FE 88 nn yy yy zz zz …` (where `nn` is the size of the table, followed by the `yyyzzzz` table values)

Description: This opcode is only valid within a user function stored in the uM-FPU flash memory. The value of the item in the table, indexed by register B, is stored in register A. The first byte after the opcode specifies the size of the table, followed by groups of four bytes representing the 32-bit values for each item in the table. This instruction can be used to load either floating point values or long integer values. The long integer value in register B is used as an index into the table, with the first table entry having index 0.

Special cases:
- if B <= 0, then the result is item 0
- if B > maximum size of table, then the result is the last item in the table

---

TAN **A = tan(A)**

Opcode: `E7`

Description: Calculates the tangent of the angle (in radians) in register A and stored the result in register A.

Special cases:
- if A is NaN or an infinity, then the result is NaN
- if A is 0.0, then the result is 0.0
- if A is –0.0, then the result is –0.0

TRACEOFF **Turn debug trace off**

Opcode: `FE FC`

Description: Used with the built-in debugger. If the debugger is not enabled, this instruction is ignored. If the debugger is enabled, debug tracing will be turned on. The debug terminal will display a trace of all instructions executed until tracing is turned off.

TRACEON **Turn debug trace on**

Opcode: `FE FD`

Description: Used with the built-in debugger. If the debugger is not enabled, this instruction is ignored. If the debugger is enabled, debug tracing will be turned off.

TRACESTR **Display debug trace message**

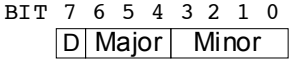
Opcode: `FE FE nn nn … 00` (where `nn` and `00` are the bytes of the string)

Description: Used with the built-in debugger. If the debugger is not enabled, this instruction is ignored. If the debugger is enabled, a message will be displayed on the debug terminal. The zero terminated ASCII string to be displayed is sent immediately following the opcode.

VERSION **Copy the version string to the string buffer**

Opcode: `FE FF`

Description: The uM-FPU version string is copied to the string buffer. And the version code is copied to register 0. The version code is represented as follows:



Bit 7 Debug Flag Set if debug mode is enabled
Bit 4-6 Major Version
Bit 0-3 Minor Version

To read the version string, this instruction is followed by a READSTR instruction.

XOP **Extended opcode**

Opcode: `FE`

Description: The first byte of all two byte opcodes is XOP. Many software interface routines are designed to only handle 8-bit data, so extended opcodes, which are 16-bit opcodes, are sent by sending an XOP followed by the second half of the opcode. For example, the LOADPI instruction would be sent as XOP, LOADPI (where XOP is defined as `FE`, and LOADPI is defined as `F3`).

Appendix A
uM-FPU V2 Instruction Summary

Opcode Name	Data Type	Opcode	Arguments	Returns	B Reg	Description
SELECTA		0x				Select A register
SELECTB		1x			x	Select B register
FWRITEA	Float	2x	yyyy zzzz			Select A register, Write floating point value to A register
FWRITEB	Float	3x	yyyy zzzz		x	Select B register, Write floating point value to B register
FREAD	Float	4x		yyyy zzzz		Read register
FSET/LSET	Either	5x				Select B register, A = B
FADD	Float	6x			x	Select B register, A = A + B
FSUB	Float	7x			x	Select B register, A = A - B
FMUL	Float	8x			x	Select B register, A = A * B
FDIV	Float	9x			x	Select B register, A = A / B
LADD	Long	Ax			x	Select B register, A = A + B
LSUB	Long	Bx			x	Select B register, A = A -B
LMUL	Long	Cx			x	Select B register, A = A * B
LDIV	Long	Dx			x	Select B register, A = A / B Remainder stored in register 0
SQRT	Float	E0				A = sqrt(A)
LOG	Float	E1				A = ln(A)
LOG10	Float	E2				A = log(A)
EXP	Float	E3				A = e ** A
EXP10	Float	E4				A = 10 ** A
SIN	Float	E5				A = sin(A) radians
COS	Float	E6				A = cos(A) radians
TAN	Float	E7				A = tan(A) radians
FLOOR	Float	E8				A = nearest integer <= A
CEIL	Float	E9				A = nearest integer >= A
ROUND	Float	EA				A = nearest integer to A
NEGATE	Float	EB				A = -A
ABS	Float	EC				A = |A|
INVERSE	Float	ED				A = 1 / A
DEGREES	Float	EE				Convert radians to degrees A = A / (PI / 180)
RADIANS	Float	EF				Convert degrees to radians A = A * (PI / 180)
SYNC		F0		5C		Synchronization
FLOAT	Long	F1			0	Copy A to register 0 Convert long to float
FIX	Float	F2			0	Copy A to register 0 Convert float to long
FCOMPARE	Float	F3		ss		Compare A and B (floating point)
LOADBYTE	Float	F4	bb		0	Write signed byte to register 0 Convert to float
LOADUBYTE	Float	F5	bb		0	Write unsigned byte to register 0 Convert to float
LOADWORD	Float	F6	wwww		0	Write signed word to register 0 Convert to float
LOADUWORD	Float	F7	wwww		0	Write unsigned word to register 0 Convert to float

READSTR		F8		aa … 00		Read zero terminated string from string buffer
ATOF	Float	F9	aa … 00		0	Convert ASCII to float Store in register 0
FTOA	Float	FA	ff			Convert float to ASCII Store in string buffer
ATOL	Long	FB	aa … 00		0	Convert ASCII to long Store in register 0
LTOA	Long	FC	ff			Convert long to ASCII Store in string buffer
FSTATUS	Float	FD		ss		Get floating point status of A
XOP		FE				Extended opcode prefix (extended opcodes are listed below)
NOP		FF				No Operation
FUNCTION		FE0n FE1n FE2n FE3n			0	User defined functions 0-15 User defined functions 16-31 User defined functions 32-47 User defined functions 48-63
IF_FSTATUSA	Float	FE80	ss			Execute user function code if FSTATUSA conditions match
IF_FSTATUSB	Float	FE81	ss			Execute user function code if FSTATUSB conditions match
IF_FCOMPARE	Float	FE82	ss			Execute user function code if FCOMPARE conditions match
IF_LSTATUSA	Long	FE83	ss			Execute user function code if LSTATUSA conditions match
IF_LSTATUSB	Long	FE84	ss			Execute user function code if LSTATUSB conditions match
IF_LCOMPARE	Long	FE85	ss			Execute user function code if LCOMPARE conditions match
IF_LUCOMPARE	Long	FE86	ss			Execute user function code if LUCOMPARE conditions match
IF_LTST	Long	FE87	ss			Execute user function code if LTST conditions match
TABLE	Either	FE88				Table Lookup (user function)
POLY	Float	FE89				Calculate n^{th} degree polynomial (user function)
READBYTE	Long	FE90		bb		Get lower 8 bits of register A
READWORD	Long	FE91		wwww		Get lower 16 bits of register A
READLONG	Long	FE92		yyyy zzzz		Get long integer value of register A
READFLOAT	Float	FE93		yyyy zzzz		Get floating point value of register A
LINCA	Long	FE94				A = A + 1
LINCB	Long	FE95				B = B + 1
LDECA	Long	FE96				A = A - 1
LDECB	Long	FE97				B = B - 1
LAND	Long	FE98				A = A AND B
LOR	Long	FE99				A = A OR B
LXOR	Long	FE9A				A = A XOR B
LNOT	Long	FE9B				A = NOT A
LTST	Long	FE9C	ss			Get the status of A AND B
LSHIFT	Long	FE9D				A = A shifted by B bit positions
LWRITEA	Long	FEAx	yyyy zzzz			Write register and select A
LWRITEB	Long	FEBx	yyyy zzzz		x	Write register and select B
LREAD	Long	FECx		yyyy zzzz		Read register
LUDIV	Long	FEDx			x	Select B register, A = A / B (unsigned) Remainder stored in register 0
POWER	Float	FEE0				A = A raised to the power of B
ROOT	Float	FEE1				A = the Bth root of A

MIN	Float	FEE2				A = minimum of A and B
MAX	Float	FEE3				A = maximum of A and B
FRACTION	Float	FEE4			0	Load Register 0 with the fractional part of A
ASIN	Float	FEE5				A = asin(A) radians
ACOS	Float	FEE6				A = acos(A) radians
ATAN	Float	FEE7				A = atan(A) radians
ATAN2	Float	FEE8				A = atan(A/B)
LCOMPARE	Long	FEE9		ss		Compare A and B (signed long integer)
LUCOMPARE	Long	FEEA		ss		Compare A and B (unsigned long integer)
LSTATUS	Long	FEEB		ss		Get long status of A
LNEGATE	Long	FEEC				A = -A
LABS	Long	FEED				A = |A|
LEFT		FEEE				Left parenthesis
RIGHT		FEEF			0	Right parenthesis
LOADZERO	Float	FEF0			0	Load Register 0 with Zero
LOADONE	Float	FEF1			0	Load Register 0 with 1.0
LOADE	Float	FEF2			0	Load Register 0 with e
LOADPI	Float	FEF3			0	Load Register 0with pi
LONGBYTE	Long	FEF4	bb		0	Write signed byte to register 0 Convert to long
LONGUBYTE	Long	FEF5	bb		0	Write unsigned byte to register 0 Convert to long
LONGWORD	Long	FEF6	wwww		0	Write signed word to register 0 Convert to long
LONGUWORD	Long	FEF7	wwww		0	Write unsigned word to register 0 Convert to long
IEEEMODE		FEF8				Set IEEE mode (default)
PICMODE		FEF9				Set PIC mode
CHECKSUM		FEFA			0	Calculate checksum for uM-FPU code
BREAK		FEFB				Debug breakpoint
TRACEOFF		FEFC				Turn debug trace off
TRACEON		FEFD				Turn debug trace on
TRACESTR		FEFE	aa … 00			Send debug string to trace buffer
VERSION		FEFF				Copy version string to string buffer

Notes:

Data Type	data type required by opcode
Opcode	hexadecimal opcode value
Arguments	additional data required by opcode
Returns	data returned by opcode
B Reg	value of B register after opcode executes
x	register number (0-15)
n	function number (0-63)
yyyy	most significant 16 bits of 32-bit value
zzzz	least significant 16 bits of 32-bit value
ss	status byte
bb	8-bit value
wwww	16-bit value
aa … 00	zero terminated ASCII string