# The Fundamental Instruction Set Operation Codes Support Function Library

Timothy Hall*

**Abstract**

This paper documents the Fundamental Instruction Set Operation Codes (FISOC) Support Function Library that may be used to implement arbitrary extended precision Rational Arithmetic And Conversions (RAC) statistical algorithms in embedded systems, including Field Programmable Gate Arrays and Very High Speed Integrated Circuits. The FISOC support function library consists of a minimal set of low-level register and static memory manipulation commands that provide for all functionality available through Reduced Instruction Set Computing and high-level software (both commercial and maintained shareware), with an emphasis on simplifying and condensing statistical analyses that require exceptionally high levels of precision and errorless calculation results. Examples of common statistics-related calculation algorithms are included that demonstrate the implementation flexibility and practical utility of the library, and a demonstration of the utility of RAC analytic methodology is given through the strategic calculation of the square root.

**Key Words:** Reduced Instruction Set Computing, Rational Arithmetic, Assembly-Level Algorithms, HDL Implementations

## 1. Introduction

The Fundamental Instruction Set Operation Codes (FISOC) is a collection of twenty-five assembly-level operation codes and assembly language directives that are used in the implementation of Rational Arithmetic and Conversions (RAC) routines **[1]**. This set of operators is a minimal set of such commands that may implement the RAC routines in assembly-level code (in the sense that all other instructions may be implemented in terms of the FISOC). While it is possible to further reduce the FISOC to a smaller subset by implementing some of the FISOC operation codes in term of the other operation codes, e.g., subtractions may be implemented as the addition of negated arguments, the resulting code becomes difficult to maintain, as the clarity of its purpose becomes less and less transparent as the number of instructions decreases beyond what is available in the FISOC.

### 1.1 Standard Requirements

Throughout this paper all references to assembly-level instructions and assembly language commands specifically refer to MMIX, the general-purpose assembly-level coding system invented by Dr. Donald E. Knuth of Stanford University **[2]**. All FISOC routines are implemented in the MMIX context, and may be converted, as needed, to other assembly-level contexts.

The FISOC works with BYTE, WYDE, and OCTA memory locations, and expresses TETRA-related operands in terms of high and low WYDE components. Furthermore, the FISOC does not address any MMIX functionality that does not affect registers nor memory values. While input/output routines and extra-FISOC operation codes (such as `NEG`) may be used for acceptance testing, the MMIX-implemented RAC routines comply strictly with the FISOC standard.

*PQI Consulting, P. O. Box 425616, Cambridge, MA, USA 02142-0012 – info@pqic.com

The MMIX implementations of the RAC routines do not include immediate `[I]` nor directional `[B]` indications – these are provided by the assembler during compilation. However, the descriptions in this memorandum do indicate these components for comparisons with MMIX documentation. Probable `[P]` indicators are used both in the descriptions and in the MMIX implemented RAC routines.

### 1.2 Operation Codes

1. **ADD, ADDI, ADDU, LDA, ADDUI (LDA, LDAI)** – Addition, with unsigned addition, and with immediate versions. The **LDA[I]** instructions are aliases for **ADDU[I],** where the Y and Z fields are used the same way in both instructions, and the absence of a Z field in **LDA[I]** is taken as the immediate version with $Z = 0$. The `LDA` assembly language directive is an alias for the `ADDU` operation code.

2. **SUB, SUBI, SUBU, SUBUI** – Subtraction, with unsigned subtraction, and with immediate versions.

3. **MUL, MULI, MULU, MULUI** - Multiplication, with unsigned multiplication, and with immediate versions.

4. **DIV, DIVI, DIVU, DIVUI** - (Integer) Division, with unsigned division, and with immediate versions, where the results are stored in $X (the integer part) and in special register `rR` (the remainder part).

5. **AND** - Logical Bitwise AND, with immediate version.

6. **SET** – Alias for **OR** and **ORI** depending on whether YZ field is a register or an immediate constant. Although **OR[I]** are not part of the FISOC standard, their functionality is provided in the FISOC by this directive.

7. **BZ, BNZ, BP, BNP, BN, BNN, BEV, BOD** – Conditional and probable conditional branches if X field is zero, non-zero, positive, non-positive, negative, non-negative, even, or odd, respectively.

8. **CMP, CMPI, CMPU, CMPUI** –Numerical comparison, with unsigned comparison, and with immediate versions. These comparisons are made as integers, whether signed or unsigned.

9. **IS** – This assembly language directive is an alias for the assignment of a label to a register number or to another label.

10. **GET** – Retrieval of value stored in a special register. There is no immediate version.

11. **PUT, PUTI** – Storage of value in Y field with immediate constant offset, or in a special register.

12. **JMP, JMPB** – Unconditional branch forward (without signifier) and backwards (with B signifier).

13. **LDB, LDBI** – Load BYTE from memory position, with immediate constant offset.

14. **STB, STBI** – Store BYTE at memory position, with immediate constant offset.

15. **LDW, LDWI** – Load WYDE from memory position, with immediate constant offset.

16. **STW, STWI** – Store WYDE at memory position, with immediate constant offset.

17. **LDO, LDOI** – Load OCTA from memory position, with immediate constant offset.

18. **STO, STOI** – Store OCTA at memory position, with immediate constant offset.

19. **PUSHJ, PUSHJB** – Unconditional branch to subroutine forward (without signifier) and backwards (with B signifier).

20. **POP** – Return from subroutine with number of arguments as returned values. The order of the returned values is always $1, $0, $2, $3, ..., relative to the local registers defined in the subroutine.

21. **SETH** – Assign WYDE value in YZ field to highest position (16 highest-order bit positions) in X field register.

22. **SL, SLI, [SLU, SLUI]** – Shift left within a register with immediate versions, with all zeros as right fill. Note that unsigned versions of shift left are redundant with the signed versions.

23. **SR, SRI, SRU, SRUI** – Shift right within a register with immediate versions, with left fill policy determined by signed (depending on the sign of the Y field – its leading bit) or unsigned (zeros) versions.

24. **TRAP** – General purpose control of input/output to the user interface.

25. **SWYM** - The "no op" instruction; it does nothing. Acronym stands for "Sympathize With Your Machinery."

### 1.3 Sufficiency With MMIX

The FISOC standard functions cover the following MMIX instructions: (00) `TRAP`, (18-27) `MUL[U][I]`, `DIV[U][I]`, `ADD[U][I]`, `SUB[U][I]`, (30-33) `CMP[U][I]`, (38-3F) `SL[U][I]`, `SR[U][I]`, (40-5F) `[P]BN[B]`, `[P]BZ[B]`, `[P]BP[B]`, `[P]BOD[B]`, `[P]BNN[B]`, `[P]BNZ[B]`, `[P]BNP[B]`,`[P]BEV[B]`, (80-87) `LDB[U][I]`, `LDW[U][I]`, (8C-8F) `LDO[U][I]`, (A0-A7) `STB[U][I]`, `STW[U][I]`, (AC-AF) `STO[U][I]`, (C8-C9) `AND[I]`, (E0) `SETH`, (F0-F3) `JMP[B]`, `PUSHJ[B}`, (F6-F8) `PUT[I]`, `POP`, (FD-FE) `SWYM`, and `GET`.

All other MMIX operation codes may be implemented in terms of the FISOC operation codes. A sample of this universality is demonstrated in the following listings. This means the FISOC standard is sufficient for implementing any MMIX command/instruction.

1. (01-17) **Floating Point Calculations** – These are equivalent to the corresponding RAC functions where floating point numbers are expressed as signed rational numbers, with the exception of the following special cases that handle exceptions (`NAN`) and tests for tolerances (the epsilon value given in the special register `rE`). These considerations are part of the RAC conversions and the use of 0 as the sign value.

2. (61) **CSNI $X,$Y,Z – Conditional Set If Negative With Immediate**

```
CMPI    $W,$Y,0
BNN     $W,@+4*2
SET     $X,Z
```

3. (72) **ZSZ $X,$Y,$Z – Zero Or Set If Zero With Register**

```
CMPI    $W,$Y,0              JMP     @+4*2
BNZ     $W,@+4*3             SET     $X,0
SET     $X,$Z
```

4. (90-91/B0-B1) **LDSF[I]/STSF[I] $X,$Y,$Z/Z – Load/Store Short Float With Register/Immediate**

   The FISOC standard does not use IEEE-754 (et seq.) floating point numbers in its calculations. Furthermore, the input/output interface is restricted to signed decimal numbers as alternate forms of signed rational numbers. Only XNUM and RAC structures are exclusively used within the MMIX code.

5. (C2) **ORN $X,$Y,$Z – Logical OR NOT With Register**

   The first ten instructions place ones in all positions of register `$K`. The instructions with asterisks perform the logical `OR NOT` operation.

```
SETH    $K,#FFFF     SRUI    $KK,32       *SUBU   $Y,$K,$Y
SETH    $KK,#FFFF    ADDU    $K,$K,$KK    *AND    $X,$Y,$Z
SRUI    $KK,16       SETH    $KK,#FFFF    *SUBU   $X,$K,$X
ADDU    $K,$K,$KK    SRUI    $KK,48
SETH    $KK,#FFFF    ADDU    $K,$K,$KK
```

*Proof.*

| | | | | |
|---|---|---|---|---|
| Start | $\$X = \varnothing$<br>$\$Y = 0$<br>$\$Z = 0$ | $\$X = \varnothing$<br>$\$Y = 0$<br>$\$Z = 1$ | $\$X = \varnothing$<br>$\$Y = 1$<br>$\$Z = 0$ | $\$X = \varnothing$<br>$\$Y = 1$<br>$\$Z = 1$ |
| `*SUBU $Y,$K,$Y` | $\$X = \varnothing$<br>$\$Y' = 1$<br>$\$Z = 0$ | $\$X = \varnothing$<br>$\$Y' = 1$<br>$\$Z = 1$ | $\$X = \varnothing$<br>$\$Y' = 0$<br>$\$Z = 0$ | $\$X = \varnothing$<br>$\$Y' = 0$<br>$\$Z = 1$ |
| `*AND $X,$Y,$Z` | $\$X' = 0$<br>$\$Y' = 1$<br>$\$Z = 0$ | $\$X' = 1$<br>$\$Y' = 1$<br>$\$Z = 1$ | $\$X' = 0$<br>$\$Y' = 0$<br>$\$Z = 0$ | $\$X' = 0$<br>$\$Y' = 0$<br>$\$Z = 1$ |
| `*SUBU $X,$K,$X`<br>Results = $X" = ORN | $\$X" = 1$<br>$\$Y' = 1$<br>$\$Z = 0$ | $\$X" = 0$<br>$\$Y' = 1$<br>$\$Z = 1$ | $\$X" = 1$<br>$\$Y' = 0$<br>$\$Z = 0$ | $\$X" = 1$<br>$\$Y' = 0$<br>$\$Z = 1$ |

■

6. (DC-DD) **MOR[I] $X,$Y,$Z/Z – Multiple OR With Register/Immediate**

   If $y_{ij}$ is byte $i$ and bit $j$ within that byte of register `$Y`, with a corresponding meaning of $z_{ij}$ for `$Z/Z`, then the assignment to `$X` as a result of the MOR[I] operation code is given by

$$\begin{pmatrix} y_{00} & y_{10} & \cdots & y_{70} \\ y_{01} & y_{11} & \cdots & y_{71} \\ \vdots & \vdots & \ddots & \vdots \\ y_{07} & y_{17} & \cdots & y_{77} \end{pmatrix} \begin{pmatrix} z_{00} & z_{10} & \cdots & z_{70} \\ z_{01} & z_{11} & \cdots & z_{71} \\ \vdots & \vdots & \ddots & \vdots \\ z_{07} & z_{17} & \cdots & z_{77} \end{pmatrix}$$

where the bit products are `MUL[U][I]` operations and the additions are logical `OR[I]` operations (since `OR[I]` may in turn be expressed as FISOC operation codes). Therefore, `MOR[I]` may be expressed as FISOC operation codes.

## 2. Programming Considerations

The following information addresses the programming considerations necessary for implementing the RAC routines in MMIX.

1. Since BYTE is the smallest basic unit of memory location available in MMIX, all individual decimal digit representations (using four bits[1] per digit) appear as $\#0X$, where $X$ may be $0 - 9$.

2. Since OCTA is the largest basic unit of memory locations (and the only register size), considerable additional code is needed in MMIX to manipulate the position of loaded and stored data in memory locations. This differs from C array structures where the individual bytes are addressed as offsets from a base position, regardless of crossing OCTA or any other memory location boundary. This consideration makes the use of shift operators (SLU and SRU – and their signed variations) necessary in the FISOC.

3. Many ANSI C, MATLAB, and MAPLE implementation structures are not directly available in MMIX (and therefore in the FISOC standard). For example, a `for` loop requires the use of three registers (independently of all others used within the loop) – one for the loop index, one to hold comparison results, and one to hold the limiting value for the loop index. While some registers used in the loop body may substitute for one or more of these registers depending on availability, such "economies" usually render the code less transparent and therefore more difficult to maintain. Furthermore, several branches and jump operations are needed to control the flow of the logical code to the beginning of the loop depending on the tests performed and/or conditions found in the loop body. Finally, loops within loops (commonly needed to index vectors and matrices) require a hierarchy of such reserved registers to make sure inapplicable values are not inadvertently stored in registers that are used prematurely.

4. Many of the RAC routines call other RAC routines as subroutines. These subroutine RAC routines may themselves call other RAC routines as sub-subroutines. While each RAC routine uses a unique namespace to prevent misuse of identically named labels within each routine, proprietary memory space is needed for each RAC routine to ensure needed results are not overwritten or otherwise misused during subroutine processing. This is especially the case for iterative methods, such as the square root routine, where twenty-five XNUM structures are needed to hold all the intermediate values.

[1] Since there are ten decimal digits, three binary positions is not sufficient to represent all possible integers ($2^3 < 10$), Likewise, five binary positions would be a waste of memory space when four binary position will suffice, i.e.,

$$\min_n \{2^n > 10\} = 4$$

Furthermore, even four binary digits has some wasted memory space, e.g., $\#0A - \#FF$ are unused. In fact, only a nybble is needed to store an individual decimal digit.

However, the analytical methods used for RAC arithmetic must allow for the arithmetic result of two single-decimal digit arguments to be expressed in the same format as any single digit decimal number. Since the maximum sum of these numbers is $18$ and the minimum difference is $-18$, and the maximum product is $81$, then a full byte of memory space is needed to represent even a single decimal digit.

## 3. Example FISOC Implementations Of The RAC Routines

The following sections document the MMIX implementations of several common RAC routines according to the FISOC standard. While an XNUM structure may be of any length (subject to implementation storage limits), the XNUM structure implemented herein represents a signed $512$ decimal digit integer, so that the range of representable numbers is $\pm 10^{512} - 1$, and the smallest absolute increment in any RAC representation is therefore $\left(10^{512} - 1\right)^{-1} \approx 10^{-512}$.

Comment lines start with a semicolon (**;**), which must be in the first column of the line. The dollar sign (**$**) followed by a positive integer between $0$ and $255$ (inclusive) are the names of $64$-bit registers available in MMIX.

### 3.1 Floor Function

```
; Calculates the floor of the ratio of two XNUM structures
; rIN/rID and stores the results in the RAC structure rON/rOD
rIN         IS      $0
rID         IS      $1
rON         IS      $2
rOD         IS      $3
rXXN        IS      $4
rXXD        IS      $5
rONE        IS      $6
rZER        IS      $7
rRO         IS      $8
rRT         IS      $9
rCMP        IS      $10
rJ          IS      $11
rTMPA       IS      $12
rTMPB       IS      $13
rTMPC       IS      $14
rTMPD       IS      $15
rTMPE       IS      $16

Start       GET     rJ,:rJ
            SET     rXXN,:FQA
            SET     rXXD,:FQB
            SET     rONE,:FQC
            SET     rZER,:FQD
            SET     rRO,:FQE
            SET     rRT,:FQF
            STB     :PosO,rONE,0
            STB     :PosO,rZER,0
            STW     :PosO,rONE,2
            STW     :PosO,rZER,2
            SET     rTMPA,0
            ADDU    rTMPB,:rLMT,7
            STB     rTMPA,rZER,rTMPB
            STB     :PosO,rONE,rTMPB
            SUBU    rTMPB,rTMPB,1
Zero        STB     rTMPA,rONE,rTMPB
            STB     rTMPA,rZER,rTMPB
            SUBU    rTMPB,rTMPB,1
            CMP     rCMP,rTMPB,8
            PBNN    rCMP,Zero
```

```
42              SET     rTMPB,rIN
43              SET     rTMPC,rID
44              SET     rTMPD,rXXN
45              SET     rTMPE,rXXD
46              PUSHJ   rTMPA,:pqicstat:RAC:pqicRREC:Start
47              LDB     rTMPA,rXXN,0
48              CMP     rCMP,rTMPA,0
49              BNZ     rCMP,NonI
50              SET     rTMPB,rXXN
51              SET     rTMPC,rON
52              PUSHJ   rTMPA,:pqicstat:RAC:pqicXCPX:Start
53              SET     rTMPB,rONE
54              SET     rTMPC,rOD
55              PUSHJ   rTMPA,:pqicstat:RAC:pqicXCPX:Start
56              JMP     Quit
57 NonI         BP      rCMP,NonN
58              STB     :NegO,rON,0
59              STB     :PosO,rOD,0
60              JMP     @+4*3
61 NonN         STB     :PosO,rON,0
62              STB     :PosO,rOD,0
63              SET     rTMPB,rXXN
64              SET     rTMPC,rXXD
65              SET     rTMPD,rRO
66              SET     rTMPE,rRT
67              PUSHJ   rTMPA,:pqicstat:RAC:pqicXDIV:Start
68              SET     rTMPB,rRT
69              SET     rTMPC,rZER
70              PUSHJ   rTMPA,:pqicstat:RAC:pqicXCMP:Start
71              CMP     rCMP,rTMPA,0
72              BNZ     rCMP,RndOff
73              SET     rTMPB,rRO
74              SET     rTMPC,rON
75              PUSHJ   rTMPA,:pqicstat:RAC:pqicXCPX:Start
76              JMP     Finish
77 RndOff       LDB     rTMPA,rON,0
78              CMP     rCMP,rTMPA,0
79              BN      rCMP,DecrN
80              SET     rTMPB,rRO
81              SET     rTMPC,rON
82              PUSHJ   rTMPA,:pqicstat:RAC:pqicXCPX:Start
83              JMP     Finish
84 DecrN        SET     rTMPB,rRO
85              SET     rTMPC,rONE
86              SET     rTMPD,rRT
87              PUSHJ   rTMPA,:pqicstat:RAC:pqicXSUB:Start
88              SET     rTMPB,rRT
89              SET     rTMPC,rON
90              PUSHJ   rTMPA,:pqicstat:RAC:pqicXCPX:Start
91 Finish       SET     rTMPB,rONE
92              SET     rTMPC,rOD
93              PUSHJ   rTMPA,:pqicstat:RAC:pqicXCPX:Start
94 Quit         PUT     :rJ,rJ
95              POP     0,0
```

### 3.2 RAC Multiplication

```
; Calculates the product of two RAC structures rUN/rUD and
; rVN/rVD and stores the results in RAC structure rRN/rRD
rUN         IS      $0
rUD         IS      $1
rVN         IS      $2
rVD         IS      $3
rRN         IS      $4
rRD         IS      $5
rTAN        IS      $6
rTAD        IS      $7
rTBN        IS      $8
rTBD        IS      $9
rKK         IS      $10
rNN         IS      $11
rMM         IS      $12
rRR         IS      $13
rJJ         IS      $14
rCMP        IS      $15
rJ          IS      $16
rTMPA       IS      $17
rTMPB       IS      $18
rTMPC       IS      $19
rTMPD       IS      $20
rTMPE       IS      $21

Start       GET     rJ,:rJ
            SET     rTAN,:RMA
            SET     rTAD,:RMB
            SET     rTBN,:RMC
            SET     rTBD,:RMD
            SET     rKK,:RME
            SET     rNN,:RMF
            SET     rMM,:RMG
            SET     rRR,:RMH
            SET     rJJ,:RMI
            SET     rTMPA,0
            SET     rTMPB,8
            ADDU    rTMPC,:rLMT,7
Zero        STB     rTMPA,rKK,rTMPB
            ADDU    rTMPB,rTMPB,1
            CMP     rCMP,rTMPB,rTMPC
            PBNP    rCMP,Zero
            STB     :PosO,rKK,0
            STW     :PosO,rKK,2
            SET     rTMPB,rUN
            SET     rTMPC,rKK
            PUSHJ   rTMPA,:pqicstat:RAC:pqicXCMP:Start
            CMP     rCMP,rTMPA,0
            BZ      rCMP,PZero
            SET     rTMPB,rVN
            SET     rTMPC,rKK
            PUSHJ   rTMPA,:pqicstat:RAC:pqicXCMP:Start
            CMP     rCMP,rTMPA,0
            BZ      rCMP,PZero
            SET     rTMPB,rUN
```

```
56          SET     rTMPC,rUD
57          SET     rTMPD,rTAN
58          SET     rTMPE,rTAD
59          PUSHJ   rTMPA,:pqicstat:RAC:pqicRREC:Start
60          SET     rTMPB,rVN
61          SET     rTMPC,rVD
62          SET     rTMPD,rTBN
63          SET     rTMPE,rTBD
64          PUSHJ   rTMPA,:pqicstat:RAC:pqicRREC:Start
65          LDB     rTMPA,rTAN,0
66          LDB     rTMPB,rTBN,0
67          MUL     rCMP,rTMPA,rTMPB
68          STB     :PosO,rTAN,0
69          STB     :PosO,rTBN,0
70          SET     rTMPB,rTAN
71          SET     rTMPC,rTBD
72          SET     rTMPD,rKK
73          PUSHJ   rTMPA,:pqicstat:RAC:pqicGCD:Start
74          SET     rTMPB,rTAD
75          SET     rTMPC,rTBN
76          SET     rTMPD,rRR
77          PUSHJ   rTMPA,:pqicstat:RAC:pqicGCD:Start
78          SET     rTMPB,rTBN
79          SET     rTMPC,rRR
80          SET     rTMPD,rNN
81          SET     rTMPE,rJJ
82          PUSHJ   rTMPA,:pqicstat:RAC:pqicXDIV:Start
83          SET     rTMPB,rTAN
84          SET     rTMPC,rKK
85          SET     rTMPD,rMM
86          SET     rTMPE,rJJ
87          PUSHJ   rTMPA,:pqicstat:RAC:pqicXDIV:Start
88          SET     rTMPB,rMM
89          SET     rTMPC,rNN
90          SET     rTMPD,rJJ
91          PUSHJ   rTMPA,:pqicstat:RAC:pqicXMUL:Start
92          SET     rTMPB,rJJ
93          SET     rTMPC,rNN
94          PUSHJ   rTMPA,:pqicstat:RAC:pqicXCPX:Start
95          SET     rTMPB,rTBD
96          SET     rTMPC,rKK
97          SET     rTMPD,rTAN
98          SET     rTMPE,rJJ
99          PUSHJ   rTMPA,:pqicstat:RAC:pqicXDIV:Start
100         SET     rTMPB,rTAN
101         SET     rTMPC,rKK
102         PUSHJ   rTMPA,:pqicstat:RAC:pqicXCPX:Start
103         SET     rTMPB,rTAD
104         SET     rTMPC,rRR
105         SET     rTMPD,rTAN
106         SET     rTMPE,rJJ
107         PUSHJ   rTMPA,:pqicstat:RAC:pqicXDIV:Start
108         SET     rTMPB,rTAN
109         SET     rTMPC,rRR
110         PUSHJ   rTMPA,:pqicstat:RAC:pqicXCPX:Start
111         SET     rTMPB,rRR
112         SET     rTMPC,rKK
```

```
113             SET     rTMPD,rMM
114             PUSHJ   rTMPA,:pqicstat:RAC:pqicXMUL:Start
115             SET     rTMPB,rNN
116             SET     rTMPC,rMM
117             SET     rTMPD,rTAN
118             SET     rTMPE,rTAD
119             PUSHJ   rTMPA,:pqicstat:RAC:pqicRRED:Start
120             SET     rTMPB,rTAN
121             SET     rTMPC,rTAD
122             SET     rTMPD,rRN
123             SET     rTMPE,rRD
124             PUSHJ   rTMPA,:pqicstat:RAC:pqicRREC:Start
125             STB     rCMP,rRN,0
126             JMP     Quit
127 PZero       SET     rTMPB,rKK
128             SET     rTMPC,rRN
129             PUSHJ   rTMPA,:pqicstat:RAC:pqicXCPX:Start
130             ADDU    rTMPB,:rLMT,7
131             STB     :PosO,rKK,rTMPB
132             SET     rTMPB,rKK
133             SET     rTMPC,rRD
134             PUSHJ   rTMPA,:pqicstat:RAC:pqicXCPX:Start
135 Quit        PUT     :rJ,rJ
136             POP     0,0
```

### 3.3 XNUM Division

```
1  ; Calculates the integer quotient and remainder of the ratio of two
2  ; XNUM structures rUIN/rUDIV and stores the quotient in XNUM structure
3  ; rDQ and stores the remainder in XNUM structure rDR; the sign of the
4  ; Quotient is the ratio of the signs of rUIN and rUDIV, and the sign
5  ; Of the remainder is the sign of rUIN
6  rUIN        IS      $0
7  rUDIV       IS      $1
8  rDQ         IS      $2
9  rDR         IS      $3
10 rFLG        IS      $4
11 rLMT        IS      $5
12 rMRK        IS      $6
13 rST         IS      $7
14 rCMP        IS      $8
15 rJ          IS      $9
16 rTMPA       IS      $10
17 rTMPB       IS      $11
18 rTMPC       IS      $12
19 rTMPD       IS      $13
20 rTMPE       IS      $14
21
22 Start       GET     rJ,:rJ
23             ADDU    rLMT,:rLMT,8
24             SET     rTMPA,8
25 ZeroOut     STO     :PosZ,:DVQ,rTMPA
26             STO     :PosZ,:DVT,rTMPA
27             STO     :PosZ,:DVN,rTMPA
28             STO     :PosZ,:DVM,rTMPA
29             ADDU    rTMPA,rTMPA,8
30             CMP     rCMP,rTMPA,rLMT
```

```
31              PBN     rCMP,ZeroOut
32              STB     :PosO,:DVQ,0
33              SET     rTMPB,rUIN
34              SET     rTMPC,:DVQ
35              PUSHJ   rTMPA,:pqicstat:RAC:pqicXCMP:Start
36              CMP     rCMP,rTMPA,0
37              BNZ     rCMP,Cont
38              STB     :PosO,:DVQ,0
39              STW     :PosO,:DVQ,2
40              SET     rTMPB,:DVQ
41              SET     rTMPC,rDQ
42              PUSHJ   rTMPA,:pqicstat:RAC:pqicXCPX:Start
43              SET     rTMPB,:DVQ
44              SET     rTMPC,rDR
45              PUSHJ   rTMPA,:pqicstat:RAC:pqicXCPX:Start
46              JMP     Quit
47 Cont         STB     :PosO,:DVQ,0
48              LDW     rTMPA,rUIN,2
49              LDW     rTMPB,rUDIV,2
50              CMP     rCMP,rTMPA,rTMPB
51              BN      rCMP,kkp
52              BP      rCMP,Regular
53              SUBU    rTMPC,rLMT,rTMPA
54 FindSz       LDB     rTMPD,rUIN,rTMPC
55              LDB     rTMPE,rUDIV,rTMPC
56              CMP     rCMP,rTMPD,rTMPE
57              BN      rCMP,kkp
58              BP      rCMP,Regular
59              ADDU    rTMPC,rTMPC,1
60              CMP     rCMP,rTMPC,rLMT
61              BN      rCMP,FindSz
62              LDB     rTMPA,rUIN,0
63              LDB     rTMPB,rUDIV,0
64              MUL     rTMPA,rTMPA,rTMPB
65              STB     rTMPA,:DVQ,0
66              STW     :PosO,:DVQ,2
67              SUBU    rTMPD,rLMT,1
68              STB     :PosO,:DVQ,rTMPD
69              STB     :PosO,:DVT,0
70              STW     :PosO,:DVT,2
71              SET     rTMPB,:DVQ
72              SET     rTMPC,rDQ
73              PUSHJ   rTMPA,:pqicstat:RAC:pqicXCPX:Start
74              SET     rTMPB,:DVT
75              SET     rTMPC,rDR
76              PUSHJ   rTMPA,:pqicstat:RAC:pqicXCPX:Start
77              JMP     Quit
78 kkp          STB     :PosO,:DVQ,0
79              STW     :PosO,:DVQ,2
80              SET     rTMPB,:DVQ
81              SET     rTMPC,rDQ
82              PUSHJ   rTMPA,:pqicstat:RAC:pqicXCPX:Start
83              SET     rTMPB,rUIN
84              SET     rTMPC,rDR
85              PUSHJ   rTMPA,:pqicstat:RAC:pqicXCPX:Start
86              JMP     Quit
87 Regular      LDB     rTMPA,rUIN,0
```

```
88              LDB     rTMPB,rUDIV,0
89              MUL     rTMPA,rTMPA,rTMPB
90              STB     rTMPA,:DVQ,0
91              STW     :PosZ,:DVQ,2
92              STB     :PosO,:DVN,0
93              STB     :PosO,:DVM,0
94              LDW     rTMPA,rUIN,2
95              LDW     rTMPB,rUDIV,2
96              STW     rTMPB,:DVM,2
97              STW     rTMPB,:DVN,2
98              SUBU    rTMPA,rTMPA,rTMPB
99              ADDU    rMRK,rTMPA,1
100             SUBU    rTMPA,rLMT,rTMPB
101             LDW     rTMPB,rUIN,2
102             SUBU    rTMPB,rLMT,rTMPB
103 FillNM      LDB     rTMPC,rUIN,rTMPB
104             STB     rTMPC,:DVN,rTMPA
105             LDB     rTMPC,rUDIV,rTMPA
106             STB     rTMPC,:DVM,rTMPA
107             ADDU    rTMPA,rTMPA,1
108             ADDU    rTMPB,rTMPB,1
109             CMP     rCMP,rTMPA,rLMT
110             BN      rCMP,FillNM
111 CheckM      CMP     rCMP,rMRK,0
112             BNP     rCMP,F1
113 WhileO      SET     rTMPB,:DVN
114             SET     rTMPC,:DVM
115             PUSHJ   rTMPA,:pqicstat:RAC:pqicXCMP:Start
116             CMP     rCMP,rTMPA,0
117             BNN     rCMP,jjp
118             CMP     rCMP,rMRK,1
119             BZ      rCMP,MrkM
120             SUBU    rMRK,rMRK,1
121             SET     rTMPB,:DVT
122             SET     rTMPC,:DVN
123             SET     rTMPD,1
124             PUSHJ   rTMPA,:pqicstat:RAC:pqicXEQT:Start
125             SUBU    rTMPA,rLMT,rMRK
126             LDB     rTMPB,rUIN,rTMPA
127             SUBU    rTMPD,rLMT,1
128             STB     rTMPB,:DVT,rTMPD
129             SET     rTMPB,:DVT
130             SET     rTMPC,:DVN
131             PUSHJ   rTMPA,:pqicstat:RAC:pqicXCPX:Start
132             JMP     jjp
133 MrkM        SUBU    rTMPD,rLMT,1
134             STB     :PosZ,:DVQ,rTMPD
135             SET     rMRK,0
136             JMP     CheckM
137 jjp         LDW     rTMPA,:DVQ,2
138             CMP     rCMP,rTMPA,0
139             BNZ     rCMP,@+4*2
140             STW     rMRK,:DVQ,2
141             LDW     rTMPA,:DVM,2
142             SUBU    rTMPA,rLMT,rTMPA
143             LDB     rTMPB,:DVN,rTMPA
144             LDB     rTMPD,:DVM,rTMPA
```

```
145             SUBU    rTMPA,rTMPA,1
146             LDB     rTMPC,:DVN,rTMPA
147             MULU    rTMPC,rTMPC,:nTEN
148             ADDU    rTMPC,rTMPC,rTMPB
149             DIVU    rST,rTMPC,rTMPD
150             SET     rFLG,1
151 WLoop       CMP     rCMP,rFLG,0
152             BNP     rCMP,CheckM
153             LDA     rTMPB,:SCRH
154             STO     rST,rTMPB,0
155             SET     rTMPC,:DVT
156             PUSHJ   rTMPA,:pqicstat:RAC:pqicNLDX:Start
157             SET     rTMPB,:DVT
158             SET     rTMPC,:DVM
159             SET     rTMPD,:DVX
160             PUSHJ   rTMPA,:pqicstat:RAC:pqicXMUL:Start
161             SET     rTMPB,:DVN
162             SET     rTMPC,:DVX
163             SET     rTMPD,:DVT
164             PUSHJ   rTMPA,:pqicstat:RAC:pqicXSUB:Start
165             SET     rTMPB,:DVT
166             SET     rTMPC,:DVM
167             PUSHJ   rTMPA,:pqicstat:RAC:pqicXCMP:Start
168             CMP     rCMP,rTMPA,0
169             BNN     rCMP,EndW
170             LDB     rTMPA,:DVT,0
171             CMP     rCMP,rTMPA,0
172             BP      rCMP,@+4*3
173             SUBU    rST,rST,1
174             JMP     WLoop
175             SUBU    rTMPA,rLMT,rMRK
176             STB     rST,:DVQ,rTMPA
177             CMP     rCMP,rMRK,1
178             BNP     rCMP,RX
179             SET     rTMPB,:DVN
180             SET     rTMPC,:DVT
181             SET     rTMPD,1
182             PUSHJ   rTMPA,:pqicstat:RAC:pqicXEQT:Start
183             SUBU    rTMPA,rLMT,rMRK
184             ADDU    rTMPA,rTMPA,1
185             SUBU    rTMPB,rLMT,1
186             LDB     rTMPC,rUIN,rTMPA
187             STB     rTMPC,:DVN,rTMPB
188             JMP     RXC
189 RX          SET     rTMPB,:DVT
190             SET     rTMPC,:DVN
191             PUSHJ   rTMPA,:pqicstat:RAC:pqicXCPX:Start
192 RXC         SUBU    rMRK,rMRK,1
193             SET     rFLG,0
194             JMP     WLoop
195 EndW        ADDU    rST,rST,1
196             JMP     WLoop
197 F1          SET     rTMPB,:DVQ
198             PUSHJ   rTMPA,:pqicstat:RAC:pqicXLEN:Start
199             STW     rTMPA,:DVQ,2
200             SET     rTMPB,:DVN
201             PUSHJ   rTMPA,:pqicstat:RAC:pqicXLEN:Start
```

```
202             STW     rTMPA,:DVN,2
203             SET     rTMPB,:DVQ
204             SET     rTMPC,rDQ
205             PUSHJ   rTMPA,:pqicstat:RAC:pqicXCPX:Start
206             SET     rTMPB,:DVN
207             SET     rTMPC,rDR
208             PUSHJ   rTMPA,:pqicstat:RAC:pqicXCPX:Start
209             LDB     rTMPA,rUIN,0
210             STB     rTMPA,rDR,0
211 Quit        PUT     :rJ,rJ
212             POP     0,0
```

## 4. The Improved Precision Of A Reciprocal Square Root

An interesting feature of calculating the (positive) square root of a RAC, a result commonly needed in statistical calculations, is found in the choice between calculating its value directly or through its reciprocal. What at first appears to be an uncontroversial choice is actually a demonstration of the utility of using rational arithmetic.

Let $\frac{r}{s}$ be the RAC result for calculating $\sqrt{\frac{a}{b}}$ to within tolerance $tol$, for $a > b > 0$. If the calculation were exact,[2] we would have

$$\frac{r}{s} = \sqrt{\frac{a}{b}} \implies \frac{s}{r} = \sqrt{\frac{b}{a}}$$

so that the use of $\frac{s}{r}$ is also an errorless calculation of $\sqrt{\frac{b}{a}}$.

Now suppose the absolute difference between $\frac{r}{s}$ and $\sqrt{\frac{a}{b}}$ is $\varepsilon$, where[3] $0 < \varepsilon < \sqrt{\frac{a}{b}} - \sqrt{\frac{b}{a}}$ and $a > b > 0$, i.e., we have

$$0 < \left| \frac{r}{s} - \sqrt{\frac{a}{b}} \right| = \varepsilon < \sqrt{\frac{a}{b}} - \sqrt{\frac{b}{a}}, a > b$$

If $\frac{r}{s} > \sqrt{\frac{a}{b}}$, then we have

$$0 < \frac{r}{s} - \sqrt{\frac{a}{b}} = \varepsilon < \sqrt{\frac{a}{b}} - \sqrt{\frac{b}{a}}, a > b$$

and

$$\sqrt{\frac{b}{a}} - \frac{s}{r} > 0$$

which means

$$\frac{r}{s} = \frac{\varepsilon\sqrt{b} + \sqrt{a}}{\sqrt{b}} = \frac{\varepsilon\sqrt{\frac{b}{a}} + 1}{\sqrt{\frac{b}{a}}}$$

[2]This result is only possible when $\frac{a}{b}$ is a perfect square of a rational number. However, the purpose here is to show that an errorless calculation of any rational number is an errorless calculation of its reciprocal.

[3]The calculations in this section show that the upper bound of $\varepsilon$ needs to be $\sqrt{\frac{a}{b}} - \sqrt{\frac{b}{a}}$, rather than an arbitrary small value, to ensure the conclusion when $\frac{r}{s}$ underestimates $\sqrt{\frac{a}{b}}$. As a practical matter, when $a$ is significantly larger than $b$, we have $\sqrt{\frac{a}{b}} - \sqrt{\frac{b}{a}}$ is significantly larger than 1. Since $\varepsilon > 0$ is meant to be small (representing the number of significant digits of $\frac{r}{s}$ representing $\sqrt{\frac{a}{b}}$), then this condition only becomes questionable when $a$ is only slightly larger than $b$, i.e., when $\sqrt{\frac{a}{b}}$ is close to 1. It shall be the responsibility of the implementing analyst to check this condition before accepting the results from the RAC square root module.

$$\frac{s}{r} = \frac{\sqrt{\frac{b}{a}}}{\varepsilon\sqrt{\frac{b}{a}} + 1}$$

However, we also have

$$\left(\varepsilon\sqrt{\frac{b}{a}} + 1\right)\left(-\varepsilon + \sqrt{\frac{b}{a}}\right) = \left(1 - \varepsilon^2\right)\sqrt{\frac{b}{a}} - \varepsilon\left(1 - \frac{b}{a}\right) < \sqrt{\frac{b}{a}} \tag{1}$$

since $\frac{b}{a} < 1$ and $\varepsilon > 0$, which means

$$\frac{s}{r} = \frac{\sqrt{\frac{b}{a}}}{\varepsilon\sqrt{\frac{b}{a}} + 1} > -\varepsilon + \sqrt{\frac{b}{a}}$$

or

$$0 < \sqrt{\frac{b}{a}} - \frac{s}{r} < \varepsilon$$

Likewise, if $\frac{r}{s} < \sqrt{\frac{a}{b}}$, then we have

$$0 < -\left(\frac{r}{s} - \sqrt{\frac{a}{b}}\right) = \varepsilon < 1, a > b$$

and

$$\frac{s}{r} - \sqrt{\frac{b}{a}} > 0$$

which means

$$\frac{r}{s} = \frac{-\varepsilon\sqrt{b} + \sqrt{a}}{\sqrt{b}} = \frac{1 - \varepsilon\sqrt{\frac{b}{a}}}{\sqrt{\frac{b}{a}}}$$

$$\frac{s}{r} = \frac{\sqrt{\frac{b}{a}}}{1 - \varepsilon\sqrt{\frac{b}{a}}}$$

However, we also have

$$\left(1 - \varepsilon\sqrt{\frac{b}{a}}\right)\left(\varepsilon + \sqrt{\frac{b}{a}}\right) = \sqrt{\frac{b}{a}} + \varepsilon\left(1 - \frac{b}{a} - \varepsilon\sqrt{\frac{b}{a}}\right) > \sqrt{\frac{b}{a}} \tag{2}$$

since

$$\begin{aligned} 0 < \varepsilon < \sqrt{\frac{a}{b}} - \sqrt{\frac{b}{a}} \quad &\Longrightarrow \quad 0 < \varepsilon\sqrt{\frac{b}{a}} < 1 - \frac{b}{a} \\ &\Longrightarrow \quad 0 < \frac{b}{a} < 1 - \varepsilon\sqrt{\frac{b}{a}} \end{aligned}$$

which means

$$\frac{s}{r} = \frac{\sqrt{\frac{b}{a}}}{1 - \varepsilon\sqrt{\frac{b}{a}}} < \varepsilon + \sqrt{\frac{b}{a}}$$

or

$$0 < \frac{s}{r} - \sqrt{\frac{b}{a}} < \varepsilon$$

This shows

$$0 < \left| \frac{s}{r} - \sqrt{\frac{b}{a}} \right| < \varepsilon$$

which means the absolute difference of $\frac{s}{r}$ from $\sqrt{\frac{b}{a}}$ is necessarily less than the absolute difference of $\frac{r}{s}$ from $\sqrt{\frac{a}{b}}$.

Note that (1) and (2) give the exact extent to which the error of $\left| \frac{s}{r} - \sqrt{\frac{b}{a}} \right|$ differs from the error of $\left| \frac{r}{s} - \sqrt{\frac{a}{b}} \right|$ in both over- and under-estimation circumstances, respectively.

There is necessarily less error calculating the square root of a number less than $1$ (under the FISOC-compliant RAC methods) than when the number is greater than $1$. This follows from the fact that the square root of a rational number between $0$ and $1$ is also between $0$ and $1$ (which bounds the absolute error to a rapidly decreasing value with each approximating iteration – see **[3]**). This provides for a simple solution when the operand of the square root is significantly large: Calculate the (positive) square root of the reciprocal of the operand as a RAC, and then use the reciprocal of that RAC as the final answer. Whatever tolerance was used, the final answer will be closer (in absolute value) to the exact value than would have been found if the reciprocals had not been used (for the same tolerance).

## REFERENCES

**[1]** Hall, T. (2017), "Statistical Calculations Through Rational Arithmetic And Conversions," *Proceedings of the 2017 Joint Statistical Meetings*, Section on Statistical Computing, Alexandria, VA: American Statistical Association, pp. 876-893.

**[2]** Knuth, D. E. (2003), *MMIXware: A RISC Computer for the Third Millennium*, Berlin: Springer Verlag, Lecture Notes in Computer Science Series.

**[3]** Hall, T. (2015), "PQIC Mathematical Notes, Operations Research Series, Number 2: PQICSTAT Rational Arithmetic And Conversion Operators," Cambridge, MA, USA: Proprietary PQIC Technical Memorandum.