



From BREXX to BREXX/370

The Journey

An evolution nobody asked for – but we did it anyway

Peter Jacob

Decommisioned, not Deprecated

**“REXX reads like English,
runs like C,
and debugs like a dream”**

Yes, I know — I’m preaching to the choir.

The House of BREXX



Vasilis Vlachoudis, PhD

Physicist at CERN

Inventor of BREXX

Mike Großmann

by day

Beta Systems Software AG

by night:

co-migrator, chaos-tamer,

and the reason BREXX/370 still talks to us.

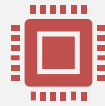


Me — retired, but only from employment,
not from strange ideas



BREXX

Revisiting the Origins



In 2001, **Vasilis Vlachoudis** presented **BREXX** at this conference, introducing a compact, efficient REXX interpreter for various **non-mainframe platforms** — a model of clarity and portability.



His architecture provided a robust starting point.



While **BREXX/370** targets the **IBM mainframe environment**, it still builds directly on the structure and principles he established.



Before we dive into the new features, let's take a quick look at a few of his original foils — the roots of what has now evolved into **BREXX/370**.

Vasilis Vlachoudis

Author of BREXX

Studied Physics at **Aristotle University of Thessaloniki** (1987–1991), where he first encountered REXX on a VM/370 system — an experience that sparked a lifelong interest.

After struggling to find a functional REXX interpreter for **MS-DOS**, he began developing his own from scratch.

During his PhD at **CERN** (1996), he rewrote and optimized the interpreter, resulting in **BREXX**, originally designed for Monte Carlo data processing.

Today, he is a **Senior Physicist at CERN**, leading the **FLUKA** and **FLAIR** projects — while his contribution to REXX lives on through the BREXX project.

The core REXX language and processing remained unchanged

Other functionality and enhancements were provided as MVS-owned add-ons.

His original architecture continues to inspire — and serve as the foundation for BREXX/370.

BRexx Overview

Rexx Symposium 2001
Vasilis.Vlachoudis@cern.ch
CERN SL-EET

From BREXX to BREXX/370: The Journey

Description

- **BRexx is a freeware implementation of Rexx**
- **Quite compliant to the language spec**
- **Written entirely in ANSI-C**
- **Without lex, yacc**
- **Portable to various platforms**
Unix, DOS, MS Windows, Windows CE, BeOS, Amiga, MacOS



The compiler manually handles parsing and tokenising code using custom C code instead of using Lexer, Parser and Generators.

<ftp://ftp.gwdg.de/pub/languages/rexx/brex>

1 May 2001

Now on
<https://github.com/vlachoudis/bREXX>

From BREXX to BREXX/370: The Journey

Advantages

- **Small in size 80-200 kb**

- **Fast!**

– **rexxcps.r on a PIII – 900 MHz**

```
----- REXXCPS 2.1 -- Measuring REXX clauses/second -----  
REXX version is: REXX_BNV R2.0.3 Apr  7 2000  
System is: UNIX  
Averaging: 100 measures of 100 iterations  
  
Performance: 1157435. REXX clauses per second
```


History

- 1991 First attempt
- 1992 V1.0 Buggy
- 1994 V1.3 Quite stable, slow!
- 1998 Started working in
- 1999 V2.0 New code, fast
- 2001 V2.0.3 Current Version

1 May 2001

V. Vlachoudis CERN SL-EET

From BREXX to BREXX/370: The Journey

What changed from V1 to V2

- **V1** interpreting on the fly
 - Compact code
 - Slow
- **V2** compiles the Rexx code and then it interprets the compiled code.
 - Optimized (ie. use of caching for variable access, ...)
 - more than 10 times faster
 - Needs slightly more memory for execution

1 May 2001

V. Vlachoudis CERN SL-EET

Differences with ANSI Rexx [2/2]

- Not implemented in BRexx:

- *CALL ON* <...>
- *NUMERIC DIGITS* [up to 15 digits]
 - All floating point operations are using doubles
- *NUMERIC [FORM | FUZZ]*
- *TRACE RESULTS* = INTERMEDIATES

Differences with ANSI Rexx [1/2]

- Existing in BRexx and not in ANSI:

- **BRexx I/O** specific
 - *open, close, read, write, eof, flush, seek*
[All **ANSI I/O** functions are **supported** also]
- **Math** functions
 - *sin, cos, exp, pow, ...*
- **VM/CMS Stack**
 - “*dir (stack)*”
- **LOAD()** – importing external libraries
- Platform depended functions

Variable Structure

- The variables are stored with a “**hashlist**” pointing to optimized **binary trees**.
- There is a **caching** mechanism for accessing the variables.
- Every procedure with private variable scope, creates a new variable pool.
- Pools are indexed:
 - 0 The main code
 - 1 First procedure
 - 2 Second
 - ...
- **VALUE(name[, [newvalue][, p**

1 May 2001

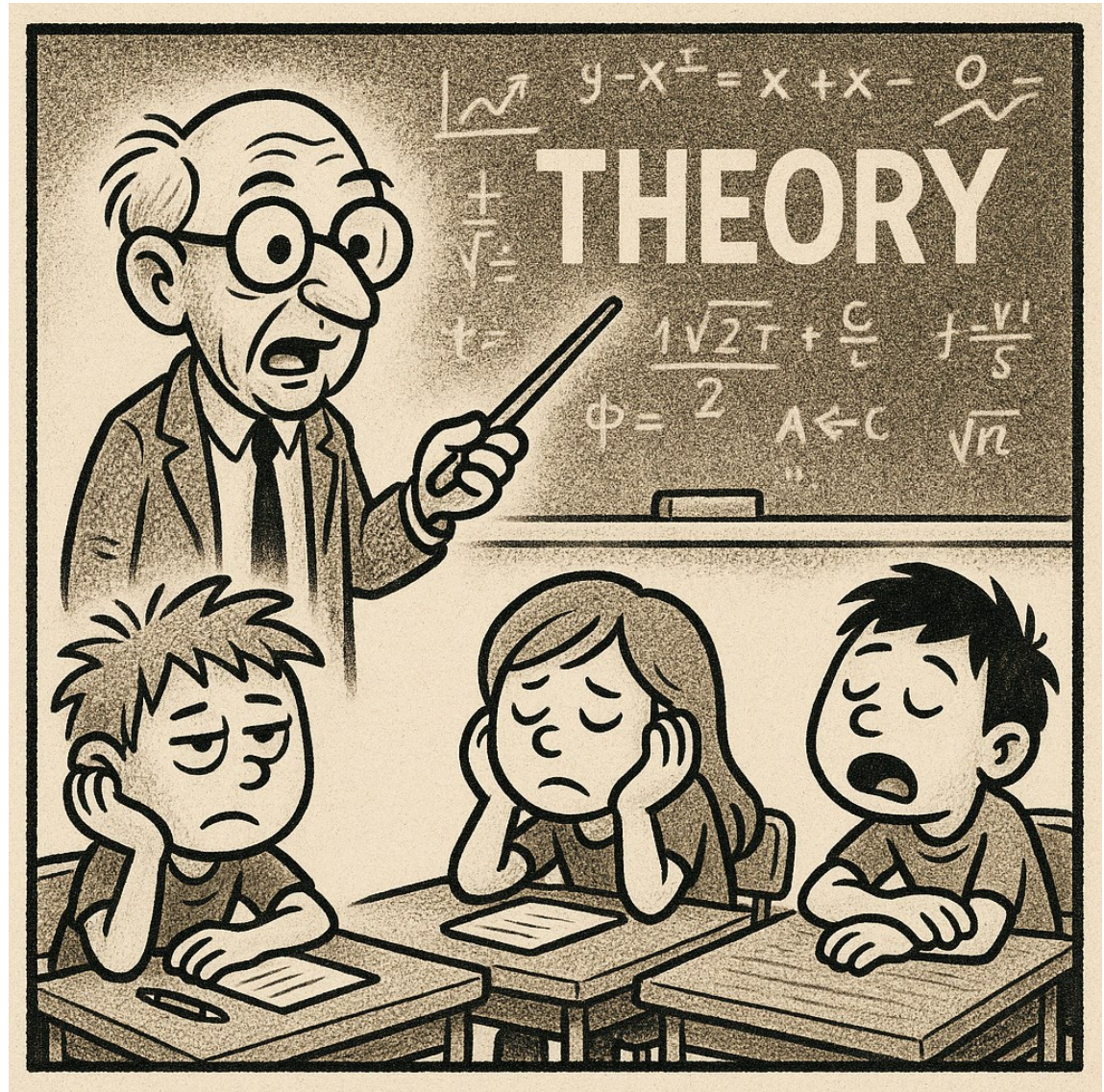
V. Vlachoudis CERN SL

Variable Storage

- Each variable in BRexx is stored as a length prefix string, with 3 different types, integer, real, string depending on the last operation on the variable.
 - a = “2.0” **String**
 - a = “2” **Integer [32 bit]**
 - a = a + 1 **Integer [32 bit]**
 - a = a + 0.1 **Real [64 bit]**
 - substr(a,2,1) **String**
- All the conversions are transparent to the user

From BREXX to BREXX/370: The Journey

Why Mainframes Still Matter



Why Mainframes Still Matter

Aspect	Mainframe (e.g., IBM z/OS)	PC / Distributed Systems
Purpose	High-volume, mission-critical processing	General use / horizontal scaling
Uptime	Designed for near-100% availability (99.999%)	Acceptable downtime; built-in failover needed
Users	Thousands of concurrent users on one machine	Usually 1 user per PC; distributed uses many
Performance	Massive I/O throughput, batch & transaction-heavy	Great at parallel tasks, web, microservices
Security	Centralized, highly secure environments	Varies; more exposed surface across nodes
Longevity	Runs decades-old apps reliably	Shorter lifecycle; frequent upgrades

Performance: It's Not Apples to Apples

It's not about clock speed — it's about what the system is built to do



z16 Mainframe CPU

- Built for throughput & reliability
- Delegates to specialised processors
- Optimised for 24/7 enterprise-scale workloads
- Secure, consistent, highly scalable



Commodity CPU

- Built for speed & versatility
- Handles most tasks internally
- Great for short, bursty workloads
- Distributed, flexible, modular

❏ A z16 doesn't try to win the GHz race — it wins the enterprise marathon.

Performance: It's Not Apples to Apples

- **IBM z16 Processor (Telum chip):**

- 8 cores per chip
- Up to 5.2 GHz

-

- **Commodity CPUs (e.g. Intel Core i9-13900K):**

- 24 cores (8 P + 16 E)
- Up to 5.8 GHz

-

- **Mainframes CPUs are designed for**

- throughput
- system-level reliability,
- massive parallelism across enterprise-scale workloads.
- **z16 Advantages Beyond Raw Speed:**
 - Simultaneous Multi-Threading (SMT) optimised for I/O-heavy workloads
 - On-chip AI inference with sub-millisecond latency
 - Cache and memory architecture tuned for sustained high-load performance
 - Vertical scaling up to 200+ cores across drawers, all managed as a single system

Mainframes: The Central Point of Control



Mainframe CPU – The Conductor

Delegates tasks to assist processors



SAPs: I/O operations



IFLs: Linux workloads



zIIPs/zAAPs: DB2, Java, XML



Crypto cards: encryption

Enables high throughput & reliability



Commodity CPU – The Soloist

- Handles most tasks internally
- Less delegation,
- general-purpose logic
- Optimized for parallel user workloads
- Requires distributed system for scaling
- More moving parts, more management

The Great  IPL
Awakening:

All theory, now
let it roar. Loud.



Time to IPL

From BREXX to BREXX/370 The Journey

Compiler: JCC (MVS 3.8 C Compiler)

- Used for **BREXX** migration
- Built specifically for **MVS 3.8** environments
- **Not a modern development toolchain**
- **No longer actively maintained**

C Language Support

- Supports only **C89 (ISO/IEC 9899:1990)**
- Lacks support for modern C standards (C99/C11)
- Severely **restricts maintainability and modernization**
- Outdated features and tooling introduce **development overhead**

Library Limitations

- **Memory management worked — until it didn't**
- Standard C Library: Incomplete
- Missing or inconsistent libc functions
- No support for POSIX or third-party libraries
- Forces custom, time-consuming implementations



It seemed like a good idea. At the time. In the fog.

From BREXX to BREXX/370: the Journey

First Release 1. April 2019

Several Releases since then, the current release V2R5M3

Delivered Documentation:

- **Installation document**
- **BREXX/370 User's Guide**
- **Formatted Screens User's Guide**
- **VSAM User's Guide**
- **BREXX Arrays**

Releases available at: <https://github.com/mvslovers/brex370/releases>

Original BREXX available at: <https://github.com/vlachoudis/brex>

Performance MVS % Windows

```
----- REXXCPS 2.1 -- Measuring REXX clauses/second -----  
REXX version is: BREXX/370 V2R5M3 L03 (Sep 24 2024)  
System is: MVS  
Averaging: 100 measures of 100 iterations
```

Performance: 43077 REXX clauses per second

BREXX Statistics

```
Instructions 8041038  
Elapsed Time 232.267015 secs  
Instructions 34619.801696/secs
```

```
----- REXXCPS 2.1 -- Measuring REXX clauses/second -  
REXX version is: BREXX/370 V2R5M3 (Jan 29 2025)  
System is: UNIX  
Averaging: 100 measures of 100 iterations
```

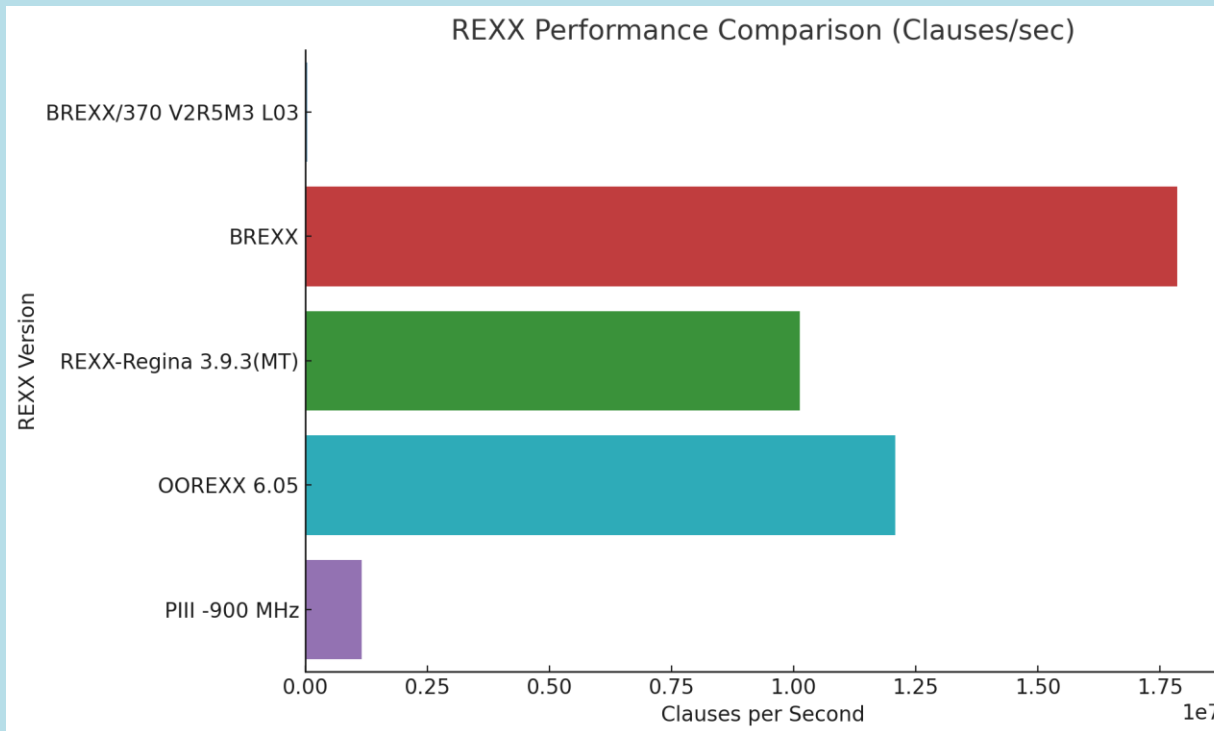
Performance: 17847138 REXX clauses per second

BREXX Statistics

```
Instructions 8041038  
Elapsed Time 1.016517 secs  
Instructions 7910514.510575/secs
```

REXX Performance Comparison

REXX Version	System	Date	Clauses/sec	Elapsed Time (s)	Instructions/sec
BREXX/370 V2R5M3 L03	MVS	Sep 24, 2024	43,077	232.267	34,619.80
BREXX	UNIX	Apr 16, 2025	17,847,138	-	-
REXX-Regina 3.9.3(MT)	WIN64	Oct 5, 2019	10,122,503	-	-
OOREXX 6.05	WindowsNT	Jun 25 2024	12,077,295		
PIII -900 MHz	Unix	Apr 7, 2000	1,157,435		



The moment has come. It's showtime.
Destiny awaits.



The moment has come. It's showtime.
Destiny awaits

```
Hercules Version : 4.5.0.10820-SDI-DEV-g5198616
Host name       : eltri
Host OS        : Linux-4.4.0-210-generic #242-Ubuntu SMP Fri Apr 10
Host Architecture : x86_64
Processors      : MP=2
LPAR Name       : HERCULES
Device number   : 0:00C0
```

```

                *****      ****      *****
                **      **      **      **      **
                **      **      **      **      **
                **      **      **      **
                **      **      **      **
                **      **      **      **
                **      **      **      **
                **      **      **      **
                **      **      **      **
                **      **      **      **
                **      **      **      **
                **      **      **      **
                **      **      **      **
                **      **      **      **
                *****      ****      *****

ZZZZzz /,'.-'-' _',,---,,_
      |,4- ) )-,_ , ( ( ' '- '
      '---' '(_/--' `-' )_)

The MVS 3.8j
Tur(n)key System

                *****      ****      *****
                **      **      **      **      **
                **      **      **      **
                **      **      **      **
                **      **      **      **
                **      **      **      **
                **      **      **      **
                **      **      **      **
                **      **      **      **
                **      **      **      **
                **      **      **      **
                **      **      **      **
                **      **      **      **
                **      **      **      **
                **      **      **      **
                *****      ****      *****
```

```
TK3 created by Volker Bandke      vbandke@bsp-gmbh.com
TK4- update by Juergen Winkelmann  winkelman@id.ethz.ch
      see TK4-.CREDITS for complete credits
```

MA A

The cursor blinks. The system awakens!

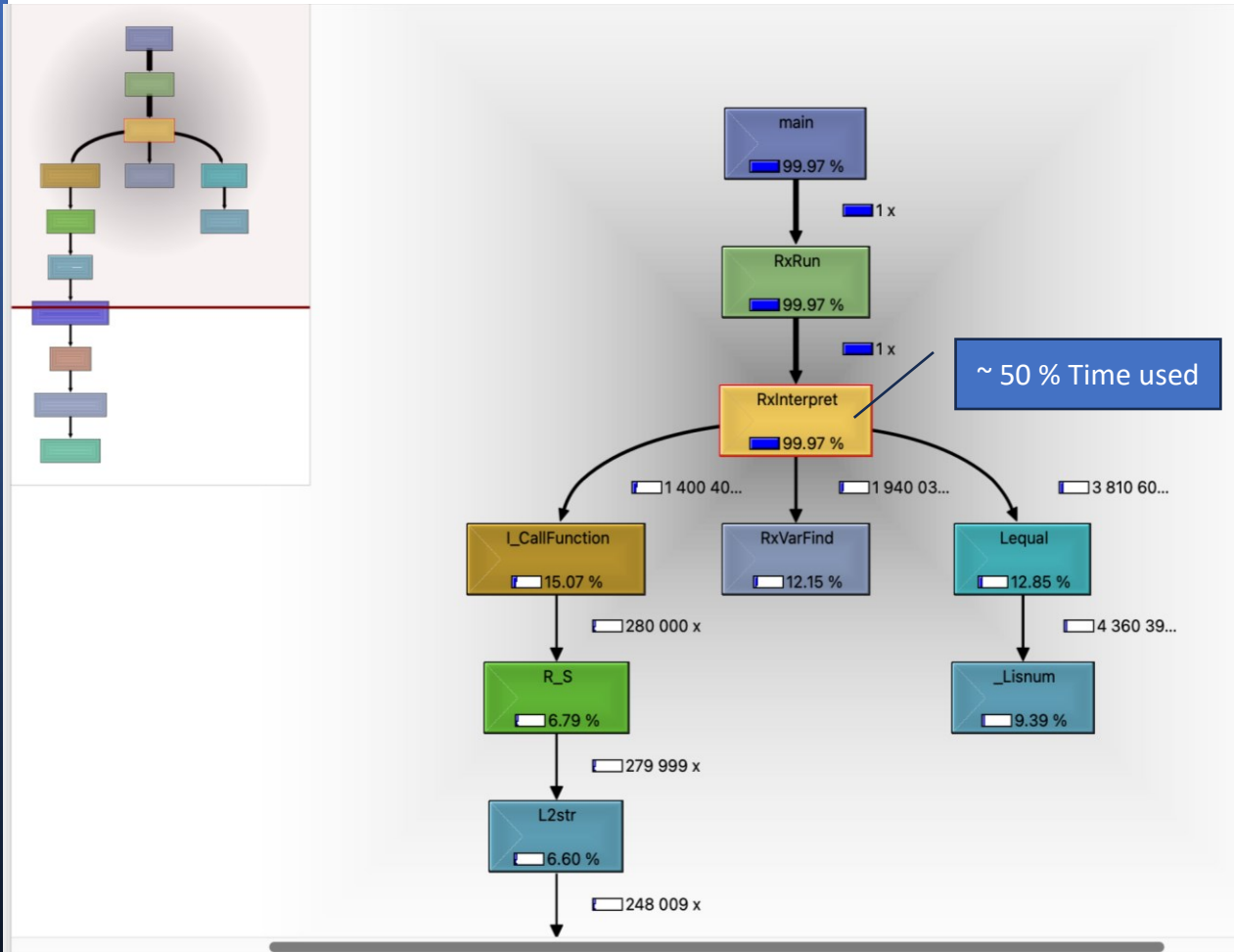
REXX code is executed via a streamlined token stream

~ 100 base instructions

```
preload.c x testR.rexx x compile.c x rexx.c x interpre.c x x2c.c x soundex.c x x2b.c x d2c.c x ldefs.h x
914 switch (*(Rxcip++)) {
915     /*
916     * [mnemonic] <type>[value]...
917     * type:    b = byte
918     *          w = word
919     *          p = pointer
920     */
921     /* START A NEW COMMAND
922     case OP_NEWCLAUSE:
923         ullInstrCount++;
924         DEBUGDISPLAY( a: "NEWC
925         if (_trace) TraceClaus
926     #ifdef WCE
927     ...
928     #endif
929         goto main_loop;
930
931     /* POP = NO OPERAT
932     case OP_NOP:
933         DEBUGDISPLAY( a: "NOP
934         goto main_loop;
935
936     /* PUSH p[lit]
937     /* push a litteral
938     case OP_PUSH:
939         RxStckTop++;
940         STACKTOP = (PLstr)*(d
941         INCWORD(Rxcip);
942         CHKERROR;
943         DEBUGDISPLAY( a: "PUSH
944         goto chk4trace;
945
946     /* PUSH TMP */
947     case OP_PUSHTMP:
948         RxStckTop++;
949         STACKTOP = &_tmpstr[Rx
950         CHKERROR;
951         DEBUGDISPLAY( a: "PUSH
952         goto main_loop;
953
954     /* PATCH w[rel] b[code]
955     /* patch CODE string to
956     case OP_PATCH:
957         w = *(CWORD *)Rxcip; INCW
958
959     /* POP FROM STACK ELEMENTS */
960     case OP_POP:
961         DEBUGDISPLAY( a: "POP");
962         RxStckTop -= *(Rxcip++);
963         goto main_loop;
964
965     /* DUP b[rel]
966     /* duplicate RELative st
967     case OP_DUP:
968         RxStckTop++;
969         STACKTOP = RxStck[RxStckTop-
970         CHKERROR;
971         DEBUGDISPLAY( a: "DUP");
972         goto main_loop;
973
974     /* COPY */
975     /* copy (Lstrcpy) top it
976     /* to previous one
977     case OP_COPY:
978         DEBUGDISPLAY( a: "COPY");
979         Lstrcpy( to: STACKP( i: 1), STAC
980         RxStckTop -= 2;
981         goto main_loop;
982
983     /* COPY2TMP */
984     /* if top item is not a --
985     /* to a tmp var then cop
986     /* value to a tmp var
987     case OP_COPY2TMP:
988         /* copy to temporary only if
989         if (STACKTOP != &(_tmpstr[Rx
990         Lstrcpy(&(_tmpstr[RxStck
991         STACKTOP = &(_tmpstr[RxS
992         DEBUGDISPLAY( a: "COPY2TMP");
993         goto main_loop;
994
995     /* PATCH w[rel] b[code]
996     /* patch CODE string to
997     case OP_PATCH:
998         w = *(CWORD *)Rxcip; INCW
999
1000     /* LOAD p[leaf]
1001     /* push a VARIABLE to stack */
1002     case OP_LOAD:
1003         INCSTACK; /* make space */
1004         PLEAF(litleaf); /* get variable ptr */
1005         DEBUGDISPLAY( a: "LOAD", b: &(litleaf->key));
1006
1007         inf = (IdentInfo*)(litle
1008
1009         /* check to see if we ha
1010         if (inf->id == Rx_id) {
1011             leaf = inf->leaf[0];
1012             STACKTOP = LEAFVAL(l
1013         } else {
1014             leaf = RxVarFind(Var
1015             if (found)
1016                 STACKTOP = LEAFV
1017             else {
1018                 if (inf->stem) {
1019                     /* Lstrcpy t
1020                     Lstrcpy(&(_t
1021                     STACKTOP = &
1022                     if (leaf==NU
1023                         _proc[_r
1024                         RxSignal
1025                 } else {
1026                     /* signal no
1027                     if (_proc[_r
1028                         RxSignal
1029                     STACKTOP = &
1030                     RxSignal
1031                     STACKTOP = &
1032                 }
1033             }
1034         }
1035         goto chk4trace;
1036
1037     case OP_NEG:
1038         DEBUGDISPLAY( a: "NEG");
1039         Lneg( to: STACKP( i: 1), STACKTOP);
1040         RxStckTop--;
1041         goto chk4trace;
1042
1043     case OP_INC:
1044         DEBUGDISPLAY( a: "INC");
1045         Linc(RxStck[RxStckTop--]);
1046         goto chk4trace;
1047
1048     case OP_DEC:
1049         DEBUGDISPLAY( a: "DEC");
1050         Ldec(RxStck[RxStckTop--]);
1051         goto chk4trace;
1052
1053     case OP_ADD:
1054         DEBUGDISPLAY2( a: "ADD");
1055         Ladd( to: STACKP( i: 2), A: STACKP( i: 1), S
1056         RxStckTop -= 2;
1057         goto chk4trace;
1058
1059     case OP_SUB:
1060         DEBUGDISPLAY2( a: "SUB");
1061         Lsub( to: STACKP( i: 2), A: STACKP( i: 1), S
1062         RxStckTop -= 2;
1063         goto chk4trace;
1064
1065     case OP_MUL:
1066         DEBUGDISPLAY2( a: "MUL");
1067         Lmult( to: STACKP( i: 2), A: STACKP( i: 1), S
1068         RxStckTop -= 2;
1069         goto chk4trace;
1070
1071     case OP_DIV:
1072         DEBUGDISPLAY2( a: "DIV");
1073         Ldiv( to: STACKP( i: 2), A: STACKP( i: 1), S
1074         RxStckTop -= 2;
1075         goto chk4trace;
1076
1077     case OP_MOD:
1078         DEBUGDISPLAY2( a: "MOD");
1079         Lmod( to: STACKP( i: 2), A: STACKP( i: 1), S
1080         RxStckTop -= 2;
1081         goto chk4trace;
1082
1083     case OP_AND:
1084         DEBUGDISPLAY2( a: "AND");
1085         Land( to: STACKP( i: 2), A: STACKP( i: 1), S
1086         RxStckTop -= 2;
1087         goto chk4trace;
1088
1089     case OP_OR:
1090         DEBUGDISPLAY2( a: "OR");
1091         Lor( to: STACKP( i: 2), A: STACKP( i: 1), S
1092         RxStckTop -= 2;
1093         goto chk4trace;
1094
1095     case OP_XOR:
1096         DEBUGDISPLAY2( a: "XOR");
1097         Lxor( to: STACKP( i: 2), A: STACKP( i: 1), S
1098         RxStckTop -= 2;
1099         goto chk4trace;
1100
1101     case OP_NOT:
1102         DEBUGDISPLAY2( a: "NOT");
1103         Lnot( to: STACKP( i: 2), A: STACKP( i: 1), S
1104         RxStckTop -= 2;
1105         goto chk4trace;
1106
1107     case OP_SHL:
1108         DEBUGDISPLAY2( a: "SHL");
1109         Lshl( to: STACKP( i: 2), A: STACKP( i: 1), S
1110         RxStckTop -= 2;
1111         goto chk4trace;
1112
1113     case OP_SHR:
1114         DEBUGDISPLAY2( a: "SHR");
1115         Lshr( to: STACKP( i: 2), A: STACKP( i: 1), S
1116         RxStckTop -= 2;
1117         goto chk4trace;
1118
1119     case OP_SAL:
1120         DEBUGDISPLAY2( a: "SAL");
1121         Lsal( to: STACKP( i: 2), A: STACKP( i: 1), S
1122         RxStckTop -= 2;
1123         goto chk4trace;
1124
1125     case OP_SAR:
1126         DEBUGDISPLAY2( a: "SAR");
1127         Lsar( to: STACKP( i: 2), A: STACKP( i: 1), S
1128         RxStckTop -= 2;
1129         goto chk4trace;
1130
1131     case OP_RL:
1132         DEBUGDISPLAY2( a: "RL");
1133         Lrl( to: STACKP( i: 2), A: STACKP( i: 1), S
1134         RxStckTop -= 2;
1135         goto chk4trace;
1136
1137     case OP_RR:
1138         DEBUGDISPLAY2( a: "RR");
1139         Lrr( to: STACKP( i: 2), A: STACKP( i: 1), S
1140         RxStckTop -= 2;
1141         goto chk4trace;
1142
1143     case OP_RLL:
1144         DEBUGDISPLAY2( a: "RLL");
1145         Lrll( to: STACKP( i: 2), A: STACKP( i: 1), S
1146         RxStckTop -= 2;
1147         goto chk4trace;
1148
1149     case OP_RRL:
1150         DEBUGDISPLAY2( a: "RRL");
1151         Lrrl( to: STACKP( i: 2), A: STACKP( i: 1), S
1152         RxStckTop -= 2;
1153         goto chk4trace;
1154
1155     case OP_ROR:
1156         DEBUGDISPLAY2( a: "ROR");
1157         Lror( to: STACKP( i: 2), A: STACKP( i: 1), S
1158         RxStckTop -= 2;
1159         goto chk4trace;
1160
1161     case OP_RORL:
1162         DEBUGDISPLAY2( a: "RORL");
1163         Lrorl( to: STACKP( i: 2), A: STACKP( i: 1), S
1164         RxStckTop -= 2;
1165         goto chk4trace;
1166
1167     case OP_RORR:
1168         DEBUGDISPLAY2( a: "RORR");
1169         Lrorr( to: STACKP( i: 2), A: STACKP( i: 1), S
1170         RxStckTop -= 2;
1171         goto chk4trace;
1172
1173     case OP_RORLL:
1174         DEBUGDISPLAY2( a: "RORLL");
1175         Lrorll( to: STACKP( i: 2), A: STACKP( i: 1), S
1176         RxStckTop -= 2;
1177         goto chk4trace;
1178
1179     case OP_RORRL:
1180         DEBUGDISPLAY2( a: "RORRL");
1181         Lrorrl( to: STACKP( i: 2), A: STACKP( i: 1), S
1182         RxStckTop -= 2;
1183         goto chk4trace;
1184
1185     case OP_RORRR:
1186         DEBUGDISPLAY2( a: "RORRR");
1187         Lrorrr( to: STACKP( i: 2), A: STACKP( i: 1), S
1188         RxStckTop -= 2;
1189         goto chk4trace;
1190
1191     case OP_RORRLL:
1192         DEBUGDISPLAY2( a: "RORRLL");
1193         Lrorrll( to: STACKP( i: 2), A: STACKP( i: 1), S
1194         RxStckTop -= 2;
1195         goto chk4trace;
1196
1197     case OP_RORRLR:
1198         DEBUGDISPLAY2( a: "RORRLR");
1199         Lrorrlr( to: STACKP( i: 2), A: STACKP( i: 1), S
1200         RxStckTop -= 2;
1201         goto chk4trace;
1202
1203     case OP_RORRRLL:
1204         DEBUGDISPLAY2( a: "RORRRLL");
1205         Lrorrrll( to: STACKP( i: 2), A: STACKP( i: 1), S
1206         RxStckTop -= 2;
1207         goto chk4trace;
1208
1209     case OP_RORRRR:
1210         DEBUGDISPLAY2( a: "RORRRR");
1211         Lrorrrr( to: STACKP( i: 2), A: STACKP( i: 1), S
1212         RxStckTop -= 2;
1213         goto chk4trace;
1214
1215     case OP_RORRRLLR:
1216         DEBUGDISPLAY2( a: "RORRRLLR");
1217         Lrorrrllr( to: STACKP( i: 2), A: STACKP( i: 1), S
1218         RxStckTop -= 2;
1219         goto chk4trace;
1220
1221     case OP_RORRRRL:
1222         DEBUGDISPLAY2( a: "RORRRRL");
1223         Lrorrrrl( to: STACKP( i: 2), A: STACKP( i: 1), S
1224         RxStckTop -= 2;
1225         goto chk4trace;
1226
1227     case OP_RORRRRLR:
1228         DEBUGDISPLAY2( a: "RORRRRLR");
1229         Lrorrrrlr( to: STACKP( i: 2), A: STACKP( i: 1), S
1230         RxStckTop -= 2;
1231         goto chk4trace;
1232
1233     case OP_RORRRRLL:
1234         DEBUGDISPLAY2( a: "RORRRRLL");
1235         Lrorrrrll( to: STACKP( i: 2), A: STACKP( i: 1), S
1236         RxStckTop -= 2;
1237         goto chk4trace;
1238
1239     case OP_RORRRRLLR:
1240         DEBUGDISPLAY2( a: "RORRRRLLR");
1241         Lrorrrrllr( to: STACKP( i: 2), A: STACKP( i: 1), S
1242         RxStckTop -= 2;
1243         goto chk4trace;
1244
1245     case OP_RORRRRRL:
1246         DEBUGDISPLAY2( a: "RORRRRRL");
1247         Lrorrrrrl( to: STACKP( i: 2), A: STACKP( i: 1), S
1248         RxStckTop -= 2;
1249         goto chk4trace;
1250
1251     case OP_RORRRRRLR:
1252         DEBUGDISPLAY2( a: "RORRRRRLR");
1253         Lrorrrrrlr( to: STACKP( i: 2), A: STACKP( i: 1), S
1254         RxStckTop -= 2;
1255         goto chk4trace;
1256
1257     case OP_RORRRRR:
1258         DEBUGDISPLAY2( a: "RORRRRR");
1259         Lrorrrrr( to: STACKP( i: 2), A: STACKP( i: 1), S
1260         RxStckTop -= 2;
1261         goto chk4trace;
1262
1263     case OP_RORRRRRLL:
1264         DEBUGDISPLAY2( a: "RORRRRRLL");
1265         Lrorrrrrll( to: STACKP( i: 2), A: STACKP( i: 1), S
1266         RxStckTop -= 2;
1267         goto chk4trace;
1268
1269     case OP_RORRRRRLLR:
1270         DEBUGDISPLAY2( a: "RORRRRRLLR");
1271         Lrorrrrrllr( to: STACKP( i: 2), A: STACKP( i: 1), S
1272         RxStckTop -= 2;
1273         goto chk4trace;
1274
1275     case OP_RORRRRRRL:
1276         DEBUGDISPLAY2( a: "RORRRRRRL");
1277         Lrorrrrrrl( to: STACKP( i: 2), A: STACKP( i: 1), S
1278         RxStckTop -= 2;
1279         goto chk4trace;
1280
1281     case OP_RORRRRRRLR:
1282         DEBUGDISPLAY2( a: "RORRRRRRLR");
1283         Lrorrrrrrlr( to: STACKP( i: 2), A: STACKP( i: 1), S
1284         RxStckTop -= 2;
1285         goto chk4trace;
1286
1287     case OP_RORRRRRRLL:
1288         DEBUGDISPLAY2( a: "RORRRRRRLL");
1289         Lrorrrrrrll( to: STACKP( i: 2), A: STACKP( i: 1), S
1290         RxStckTop -= 2;
1291         goto chk4trace;
1292
1293     case OP_RORRRRRRLLR:
1294         DEBUGDISPLAY2( a: "RORRRRRRLLR");
1295         Lrorrrrrrllr( to: STACKP( i: 2), A: STACKP( i: 1), S
1296         RxStckTop -= 2;
1297         goto chk4trace;
1298
1299     case OP_RORRRRRRRL:
1300         DEBUGDISPLAY2( a: "RORRRRRRRL");
1301         Lrorrrrrrrl( to: STACKP( i: 2), A: STACKP( i: 1), S
1302         RxStckTop -= 2;
1303         goto chk4trace;
1304
1305     case OP_RORRRRRRRLR:
1306         DEBUGDISPLAY2( a: "RORRRRRRRLR");
1307         Lrorrrrrrrlr( to: STACKP( i: 2), A: STACKP( i: 1), S
1308         RxStckTop -= 2;
1309         goto chk4trace;
1310
1311     case OP_RORRRRRRR:
1312         DEBUGDISPLAY2( a: "RORRRRRRR");
1313         Lrorrrrrrr( to: STACKP( i: 2), A: STACKP( i: 1), S
1314         RxStckTop -= 2;
1315         goto chk4trace;
1316
1317     case OP_RORRRRRRRLL:
1318         DEBUGDISPLAY2( a: "RORRRRRRRLL");
1319         Lrorrrrrrrll( to: STACKP( i: 2), A: STACKP( i: 1), S
1320         RxStckTop -= 2;
1321         goto chk4trace;
1322
1323     case OP_RORRRRRRRLLR:
1324         DEBUGDISPLAY2( a: "RORRRRRRRLLR");
1325         Lrorrrrrrrllr( to: STACKP( i: 2), A: STACKP( i: 1), S
1326         RxStckTop -= 2;
1327         goto chk4trace;
1328
1329     case OP_RORRRRRRRRL:
1330         DEBUGDISPLAY2( a: "RORRRRRRRRL");
1331         Lrorrrrrrrrl( to: STACKP( i: 2), A: STACKP( i: 1), S
1332         RxStckTop -= 2;
1333         goto chk4trace;
1334
1335     case OP_RORRRRRRRRLR:
1336         DEBUGDISPLAY2( a: "RORRRRRRRRLR");
1337         Lrorrrrrrrrlr( to: STACKP( i: 2), A: STACKP( i: 1), S
1338         RxStckTop -= 2;
1339         goto chk4trace;
1340
1341     case OP_RORRRRRRRRLL:
1342         DEBUGDISPLAY2( a: "RORRRRRRRRLL");
1343         Lrorrrrrrrrll( to: STACKP( i: 2), A: STACKP( i: 1), S
1344         RxStckTop -= 2;
1345         goto chk4trace;
1346
1347     case OP_RORRRRRRRRLLR:
1348         DEBUGDISPLAY2( a: "RORRRRRRRRLLR");
1349         Lrorrrrrrrrllr( to: STACKP( i: 2), A: STACKP( i: 1), S
1350         RxStckTop -= 2;
1351         goto chk4trace;
1352
1353     case OP_RORRRRRRRRRL:
1354         DEBUGDISPLAY2( a: "RORRRRRRRRRL");
1355         Lrorrrrrrrrrl( to: STACKP( i: 2), A: STACKP( i: 1), S
1356         RxStckTop -= 2;
1357         goto chk4trace;
1358
1359     case OP_RORRRRRRRRRLR:
1360         DEBUGDISPLAY2( a: "RORRRRRRRRRLR");
1361         Lrorrrrrrrrrlr( to: STACKP( i: 2), A: STACKP( i: 1), S
1362         RxStckTop -= 2;
1363         goto chk4trace;
1364
1365     case OP_RORRRRRRRRR:
1366         DEBUGDISPLAY2( a: "RORRRRRRRRR");
1367         Lrorrrrrrrrr( to: STACKP( i: 2), A: STACKP( i: 1), S
1368         RxStckTop -= 2;
1369         goto chk4trace;
1370
1371     case OP_RORRRRRRRRRLL:
1372         DEBUGDISPLAY2( a: "RORRRRRRRRRLL");
1373         Lrorrrrrrrrrll( to: STACKP( i: 2), A: STACKP( i: 1), S
1374         RxStckTop -= 2;
1375         goto chk4trace;
1376
1377     case OP_RORRRRRRRRRLLR:
1378         DEBUGDISPLAY2( a: "RORRRRRRRRRLLR");
1379         Lrorrrrrrrrrllr( to: STACKP( i: 2), A: STACKP( i: 1), S
1380         RxStckTop -= 2;
1381         goto chk4trace;
1382
1383     case OP_RORRRRRRRRRRL:
1384         DEBUGDISPLAY2( a: "RORRRRRRRRRRL");
1385         Lrorrrrrrrrrrl( to: STACKP( i: 2), A: STACKP( i: 1), S
1386         RxStckTop -= 2;
1387         goto chk4trace;
1388
1389     case OP_RORRRRRRRRRRLR:
1390         DEBUGDISPLAY2( a: "RORRRRRRRRRRLR");
1391         Lrorrrrrrrrrrlr( to: STACKP( i: 2), A: STACKP( i: 1), S
1392         RxStckTop -= 2;
1393         goto chk4trace;
1394
1395     case OP_RORRRRRRRRRRLL:
1396         DEBUGDISPLAY2( a: "RORRRRRRRRRRLL");
1397         Lrorrrrrrrrrrll( to: STACKP( i: 2), A: STACKP( i: 1), S
1398         RxStckTop -= 2;
1399         goto chk4trace;
1400
1401     case OP_RORRRRRRRRRRLLR:
1402         DEBUGDISPLAY2( a: "RORRRRRRRRRRLLR");
1403         Lrorrrrrrrrrrllr( to: STACKP( i: 2), A: STACKP( i: 1), S
1404         RxStckTop -= 2;
1405         goto chk4trace;
1406
1407     case OP_RORRRRRRRRRRRL:
1408         DEBUGDISPLAY2( a: "RORRRRRRRRRRRL");
1409         Lrorrrrrrrrrrrl( to: STACKP( i: 2), A: STACKP( i: 1), S
1410         RxStckTop -= 2;
1411         goto chk4trace;
1412
1413     case OP_RORRRRRRRRRRRLR:
1414         DEBUGDISPLAY2( a: "RORRRRRRRRRRRLR");
1415         Lrorrrrrrrrrrrlr( to: STACKP( i: 2), A: STACKP( i: 1), S
1416         RxStckTop -= 2;
1417         goto chk4trace;
1418
1419     case OP_RORRRRRRRRRRR:
1420         DEBUGDISPLAY2( a: "RORRRRRRRRRRR");
1421         Lrorrrrrrrrrrr( to: STACKP( i: 2), A: STACKP( i: 1), S
1422         RxStckTop -= 2;
1423         goto chk4trace;
1424
1425     case OP_RORRRRRRRRRRRLL:
1426         DEBUGDISPLAY2( a: "RORRRRRRRRRRRLL");
1427         Lrorrrrrrrrrrrll( to: STACKP( i: 2), A: STACKP( i: 1), S
1428         RxStckTop -= 2;
1429         goto chk4trace;
1430
1431     case OP_RORRRRRRRRRRRLLR:
1432         DEBUGDISPLAY2( a: "RORRRRRRRRRRRLLR");
1433         Lrorrrrrrrrrrrllr( to: STACKP( i: 2), A: STACKP( i: 1), S
1434         RxStckTop -= 2;
1435         goto chk4trace;
1436
1437     case OP_RORRRRRRRRRRRRL:
1438         DEBUGDISPLAY2( a: "RORRRRRRRRRRRRL");
1439         Lrorrrrrrrrrrrrl( to: STACKP( i: 2), A: STACKP( i: 1), S
1440         RxStckTop -= 2;
1441         goto chk4trace;
1442
1443     case OP_RORRRRRRRRRRRRLR:
1444         DEBUGDISPLAY2( a: "RORRRRRRRRRRRRLR");
1445         Lrorrrrrrrrrrrrlr( to: STACKP( i: 2), A: STACKP( i: 1), S
1446         RxStckTop -= 2;
1447         goto chk4trace;
1448
1449     case OP_RORRRRRRRRRRRRLL:
1450         DEBUGDISPLAY2( a: "RORRRRRRRRRRRRLL");
1451         Lrorrrrrrrrrrrrll( to: STACKP( i: 2), A: STACKP( i: 1), S
1452         RxStckTop -= 2;
1453         goto chk4trace;
1454
1455     case OP_RORRRRRRRRRRRRLLR:
1456         DEBUGDISPLAY2( a: "RORRRRRRRRRRRRLLR");
1457         Lrorrrrrrrrrrrrllr( to: STACKP( i: 2), A: STACKP( i: 1), S
1458         RxStckTop -= 2;
1459         goto chk4trace;
1460
1461     case OP_RORRRRRRRRRRRRRL:
1462         DEBUGDISPLAY2( a: "RORRRRRRRRRRRRRL");
1463         Lrorrrrrrrrrrrrrl( to: STACKP( i: 2), A: STACKP( i: 1), S
1464         RxStckTop -= 2;
1465         goto chk4trace;
1466
1467     case OP_RORRRRRRRRRRRRRLR:
1468         DEBUGDISPLAY2( a: "RORRRRRRRRRRRRRLR");
1469         Lrorrrrrrrrrrrrrlr( to: STACKP( i: 2), A: STACKP( i: 1), S
1470         RxStckTop -= 2;
1471         goto chk4trace;
1472
1473     case OP_RORRRRRRRRRRRRR:
1474         DEBUGDISPLAY2( a: "RORRRRRRRRRRRRR");
1475         Lrorrrrrrrrrrrrr( to: STACKP( i: 2), A: STACKP( i: 1), S
1476         RxStckTop -= 2;
1477         goto chk4trace;
1478
1479     case OP_RORRRRRRRRRRRRRLL:
1480         DEBUGDISPLAY2( a: "RORRRRRRRRRRRRRLL");
1481         Lrorrrrrrrrrrrrrll( to: STACKP( i: 2), A: STACKP( i: 1), S
1482         RxStckTop -= 2;
1483         goto chk4trace;
1484
1485     case OP_RORRRRRRRRRRRRRLLR:
1486         DEBUGDISPLAY2( a: "RORRRRRRRRRRRRRLLR");
1487         Lrorrrrrrrrrrrrrllr( to: STACKP( i: 2), A: STACKP( i: 1), S
1488         RxStckTop -= 2;
1489         goto chk4trace;
1490
1491     case OP_RORRRRRRRRRRRRRRL:
1492         DEBUGDISPLAY2( a: "RORRRRRRRRRRRRRRL");
1493         Lrorrrrrrrrrrrrrrl( to: STACKP( i: 2), A: STACKP( i: 1), S
1494         RxStckTop -= 2;
1495         goto chk4trace;
1496
1497     case OP_RORRRRRRRRRRRRRRLR:
1498         DEBUGDISPLAY2( a: "RORRRRRRRRRRRRRRLR");
1499         Lrorrrrrrrrrrrrrrlr( to: STACKP( i: 2), A: STACKP( i: 1), S
1500         RxStckTop -= 2;
1501         goto chk4trace;
1502
1503     case OP_RORRRRRRRRRRRRRRLL:
1504         DEBUGDISPLAY2( a: "RORRRRRRRRRRRRRRLL");
1505         Lrorrrrrrrrrrrrrrll( to: STACKP( i: 2), A: STACKP( i: 1), S
1506         RxStckTop -= 2;
1507         goto chk4trace;
1508
1509     case OP_RORRRRRRRRRRRRRRLLR:
1510         DEBUGDISPLAY2( a: "RORRRRRRRRRRRRRRLLR");
1511         Lrorrrrrrrrrrrrrrllr( to: STACKP( i: 2), A: STACKP( i: 1), S
1512         RxStckTop -= 2;
1513         goto chk4trace;
1514
1515     case OP_RORRRRRRRRRRRRRRRL:
1516         DEBUGDISPLAY2( a: "RORRRRRRRRRRRRRRRL");
1517         Lrorrrrrrrrrrrrrrrl( to: STACKP( i: 2), A: STACKP( i: 1), S
1518         RxStckTop -= 2;
1519         goto chk4trace;
1520
1521     case OP_RORRRRRRRRRRRRRRRLR:
1522         DEBUGDISPLAY2( a: "RORRRRRRRRRRRRRRRLR");
1523         Lrorrrrrrrrrrrrrrrlr( to: STACKP( i: 2), A: STACKP( i: 1), S
1524         RxStckTop -= 2;
1525         goto chk4trace;
1526
1527     case OP_RORRRRRRRRRRRRRRR:
1528         DEBUGDISPLAY2( a: "RORRRRRRRRRRRRRRR");
1529         Lrorrrrrrrrrrrrrrr( to: STACKP( i: 2), A: STACKP( i: 1), S
1530         RxStckTop -= 2;
1531         goto chk4trace;
1532
1533     case OP_RORRRRRRRRRRRRRRRLL:
1534         DEBUGDISPLAY2( a: "RORRRRRRRRRRRRRRRLL");
1535         Lrorrrrrrrrrrrrrrrll( to: STACKP( i: 2), A: STACKP( i: 1), S
1536         RxStckTop -= 2;
1537         goto chk4trace;
1538
1539     case OP_RORRRRRRRRRRRRRRRLLR:
1540         DEBUGDISPLAY2( a: "RORRRRRRRRRRRRRRRLLR");
1541         Lrorrrrrrrrrrrrrrrllr( to: STACKP( i: 2), A: STACKP( i: 1), S
1542         RxStckTop -= 2;
1543         goto chk4trace;
1544
1545     case OP_RORRRRRRRRRRRRRRRRL:
1546         DEBUGDISPLAY2( a: "RORRRRRRRRRRRRRRRRL");
1547         Lrorrrrrrrrrrrrrrrrl( to: STACKP( i: 2), A: STACKP( i: 1), S
1548         RxStckTop -= 2;
1549         goto chk4trace;
1550
1551     case OP_RORRRRRRRRRRRRRRRRLR:
1552         DEBUGDISPLAY2( a: "RORRRRRRRRRRRRRRRRLR");
1553         Lrorrrrrrrrrrrrrrrrlr( to: STACKP( i: 2), A: STACKP( i: 1), S
1554         RxStckTop -= 2;
1555         goto chk4trace;
1556
1557     case OP_RORRRRRRRRRRRRRRRRLL:
1558         DEBUGDISPLAY2( a: "RORRRRRRRRRRRRRRRRLL");
1559         Lrorrrrrrrrrrrrrrrrll( to: STACKP( i: 2), A: STACKP( i: 1), S
1560         RxStckTop -= 2;
1561         goto chk4trace;
1562
1563     case OP_RORRRRRRRRRRRRRRRRLLR:
1564         DEBUGDISPLAY2( a: "RORRRRRRRRRRRRRRRRLLR");
1565         Lrorrrrrrrrrrrrrrrrllr( to: STACKP( i: 2), A: STACKP( i: 1), S
1566         RxStckTop -= 2;
1567         goto chk4trace;
1568
1569     case OP_RORRRRRRRRRRRRRRRRRL:
1570         DEBUGDISPLAY2( a: "RORRRRRRRRRRRRRRRRRL");
1571         Lrorrrrrrrrrrrrrrrrrl( to: STACKP( i: 2), A: STACKP( i: 1), S
1572         RxStckTop -= 2;
1573         goto chk4trace;
1574
1575     case OP_RORRRRRRRRRRRRRRRRRLR:
1576         DEBUGDISPLAY2( a: "RORRRRRRRRRRRRRRRRRLR");
1577         Lrorrrrrrrrrrrrrrrrrlr( to: STACKP( i: 2), A: STACKP( i: 1), S
1578         RxStckTop -= 2;
1579         goto chk4trace;
1580
1581     case OP_RORRRRRRRRRRRRRRRRR:
1582         DEBUGDISPLAY2( a: "RORRRRRRRRRRRRRRRRR");
1583         Lrorrrrrrrrrrrrrrrrr( to: STACKP( i: 2), A: STACKP( i: 1), S
1584         RxStckTop -= 2;
1585         goto chk4trace;
1586
1587     case OP_RORRRRRRRRRRRRRRRRRLL:
1588         DEBUGDISPLAY2( a: "RORRRRRRRRRRRRRRRRRLL");
1589         Lrorrrrrrrrrrrrrrrrrll( to: STACKP( i: 2), A: STACKP( i: 1), S
1590         RxStckTop -= 2;
1591         goto chk4trace;
1592
1593     case OP_RORRRRRRRRRRRRRRRRRLLR:
1594         DEBUGDISPLAY2( a: "RORRRRRRRRRRRRRRRRRLLR");
1595         Lrorrrrrrrrrrrrrrrrrllr( to: STACKP( i: 2), A: STACKP( i: 1), S
1596         RxStckTop -= 2;
1597         goto chk4trace;
1598
1599     case OP_RORRRRRRRRRRRRRRRRRRL:
1600         DEBUGDISPLAY2( a: "RORRRRRRRRRRRRRRRRRRL");
1601         Lrorrrrrrrrrrrrrrrrrrl( to: STACKP( i: 2), A: STACKP( i: 1), S
1602         RxStckTop -= 2;
1603         goto chk4trace;
1604
1605     case OP_RORRRRRRRRRRRRRRRRRRLR:
1606         DEBUGDISPLAY2( a: "RORRRRRRRRRRRRRRRRRRLR");
1607         Lrorrrrrrrrrrrrrrrrrrlr( to: STACKP( i: 2), A: STACKP( i: 1), S
1608         RxStckTop -= 2;
1609         goto chk4trace;
1610
1611     case OP_RORRRRRRRRRRRRRRRRRRLL:
1612         DEBUGDISPLAY2( a: "RORRRRRRRRRRRRRRRRRRLL");
1613         Lrorrrrrrrrrrrrrrrrrrll( to: STACKP( i: 2), A: STACKP( i: 1), S
1614         RxStckTop -= 2;
1615         goto chk4trace;
1616
1617     case OP_RORRRRRRRRRRRRRRRRRRLLR:
1618         DEBUGDISPLAY2( a: "RORRRRRRRRRRRRRRRRRRLLR");
1619         Lrorrrrrrrrrrrrrrrrrrllr( to: STACKP( i: 2), A: STACKP( i: 1), S
1620         RxStckTop -= 2;
1621         goto chk4trace;
1622
1617     case OP_RORRRRRRRRRRRRRRRRRRRL:
1618         DEBUGDISPLAY2( a: "RORRRRRRRRRRRRRRRRRRRL");
1619         Lrorrrrrrrrrrrrrrrrrrl( to: STACKP( i: 2), A: STACKP( i: 1), S
1620         RxStckTop -= 2;
1621         goto chk4trace;
1622
1623     case OP_RORRRRRRRRRRRRRRRRRRRLR:
1624         DEBUGDISPLAY2( a: "RORRRRRRRRRRRRRRRRRRRLR");
1625         Lrorrrrrrrrrrrrrrrrrrlr( to: STACKP( i: 2), A: STACKP( i: 1), S
1626         RxStckTop -= 2;
1627         goto chk4trace;
1628
1629     case OP_RORRRRRRRRRRRRRRRRRRR:
1630         DEBUGDISPLAY2( a: "RORRRRRRRRRRRRRRRRRRR");
1631         Lrorrrrrrrrrrrrrrrrrrr( to: STACKP( i: 2), A: STACKP( i: 1), S
1632         RxStckTop -= 2;
1633         goto chk4trace;
1634
1635     case OP_RORRRRRRRRRRRRRRRRRRRLL:
1636         DEBUGDISPLAY2( a: "RORRRRRRRRRRRRRRRRRRRLL");
1637         Lrorrrrrrrrrrrrrrrrrrrll( to: STACKP( i: 2), A: STACKP( i: 1), S
1638         RxStckTop -= 2;
1639         goto chk4trace;
1640
1639     case OP_RORRRRRRRRRRRRRRRRRRRLLR:
1640         DEBUGDISPLAY2( a: "RORRRRRRRRRRRRRRRRRRRLLR");
1641         Lrorrrrrrrrrrrrrrrrrrrllr( to: STACKP( i: 2), A: STACKP( i: 1), S
1642         RxStckTop -= 2;
1643         goto chk4trace;
1644
1641     case OP_RORRRRRRRRRRRRRRRRRRRRL:
1642         DEBUGDISPLAY2( a: "RORRRRRRRRRRRRRRRRRRRRL");
1643         Lrorrrrrrrrrrrrrrrrrrrrl( to: STACKP( i: 2), A: STACKP( i: 1), S
1644         RxStckTop -= 2;
1645         goto chk4trace;
1646
1645     case OP_RORRRRRRRRRRRRRRRRRRRRLR:
1646         DEBUGDISPLAY2( a: "RORRRRRRRRRRRRRRRRRRRRLR");
1647         Lrorrrrrrrrrrrrrrrrrrrrlr( to: STACKP( i: 2), A: STACKP( i: 1), S
1648         RxStckTop -= 2;
1649         goto chk4trace;
1647
1648     case OP_RORRRRRRRRRRRRRRRRRRRRLL:
1649         DEBUGDISPLAY2( a: "RORRRRRRRRRRRRRRRRRRRRLL");
1650         Lrorrrrrrrrrrrrrrrrrrrrll( to: STACKP( i: 2), A: STACKP( i: 1), S
1651         RxStckTop -= 2;
1652         goto chk4trace;
1653
1649     case OP_RORRRRRRRRRRRRRRRRRRRRLLR:
1650         DEBUGDISPLAY2( a: "RORRRRRRRRRRRRRRRRRRRRLLR");
1651         Lrorrrrrrrrrrrrrrrrrrrrllr( to: STACKP( i: 2), A: STACKP( i: 1), S
1652         RxStckTop -= 2;
1653         goto chk4trace;
1654
1651     case OP_RORRRRRRRRRRRRRRRRRRRRRL:
1652         DEBUGDISPLAY2( a: "RORRRRRRRRRRRRRRRRRRRRRL");
1653         Lrorrrrrrrrrrrrrrrrrrrrl( to: STACKP( i: 2), A: STACKP( i: 
```

Where time is consumed

(CALGRIND)



Now Off-Road

Entering the wilderness...



BREXX's Distinct Characteristics



Good performance



Pre-compiled (tokenising at run time)

Split into elementary tokens
Tokens placed into a **Token Stream** (array)
Variables and STEMs resolved via binary tree lookup
Compiled into stack operations using Reverse Polish Notation (RPN)



Execution runs the **Token Stream**



Benefits / Disadvantages

The Token Stream is live – and it may grow as needed
Labels must be unique over the Token Stream
Call of procedures (labels) work within the stack
This allows call-back procedures
Variables unique and accessible/maintainable in all
Procedures (without PROCEDURE statement)



Pre-compiled % Compiled

BREXX approach



BREXX takes a direct, integrated approach to compilation



No separate lexer, tokeniser, or parser stages



No abstract syntax tree (AST)



Tokenisation directly drives bytecode generation



Simplified structure supports minimal tooling and fast execution



Pre-compiled % Compiled

CREXX approach



- Lexer (Lexical Analysis)



- Tokeniser (Tokens)



- Parser (Syntactic Analysis)



- Grammar (REXX)



- Abstract Syntax Tree (AST)



- Code Generator

CREXX Compilation Process

Lexer (Lexical Analysis):

- Converts raw source code into lexemes (e.g., `count = 10;` → `[count]`, `[=]`, `[;]`).
- Uses rules like regular expressions to identify lexemes.

tokeniser (Tokenization):

- Converts lexemes into structured tokens (e.g., `IDENTIFIER(count)`, `EQUALS(=)`, `NUMBER(10)`, `SEMICOLON(;)`).

Parser (Syntactic Analysis):

- Builds syntactic structure from tokens using grammar rules.
- Example: `assignment_statement` → `identifier '=' expression ';'.`

Abstract Syntax Tree (AST):

- Simplified tree representation of syntactic structure.
- Example: Assignment $\begin{array}{c} \text{Identifier: count} \\ \text{Value: 10.} \end{array}$

Code Generator:

- Converts AST into CREXX assembly code.
- Example: `count = 10` → `MOV [memory_address], 10.`

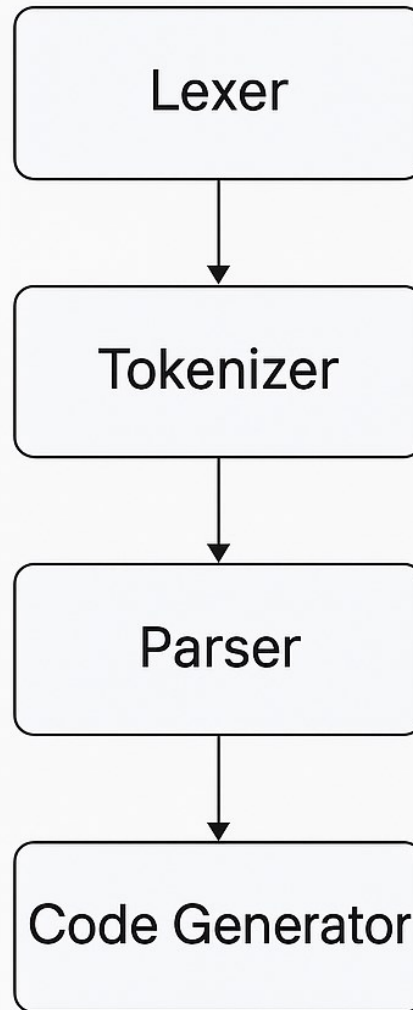
Complete Workflow:

- Source Code → Lexer → tokeniser → Parser → AST → Code Generator → Target Code.

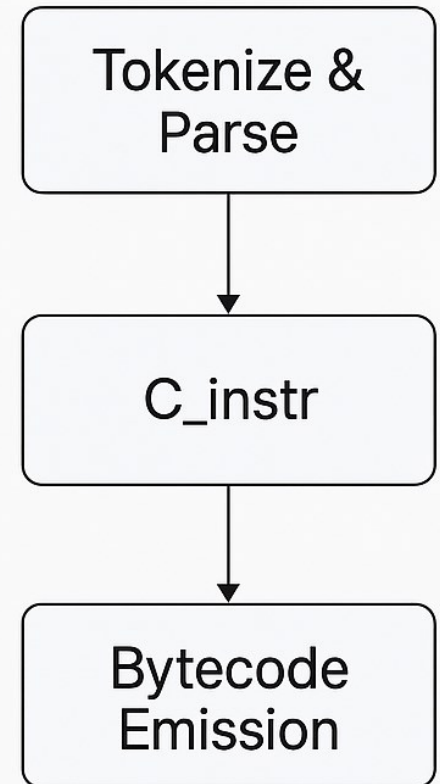
BREXX/CREXX

Diverging Designs
Shared Goals

Traditional Compiler



BREXX Compiler



And now back
from something
completely
different

To
BREXX/370 Layer
Concept



BREXX Layer Concept

SYSEXEC (User REXX)

User-written functions

RXLIB Layer

BREXX Library Functions

BREXX Kernel

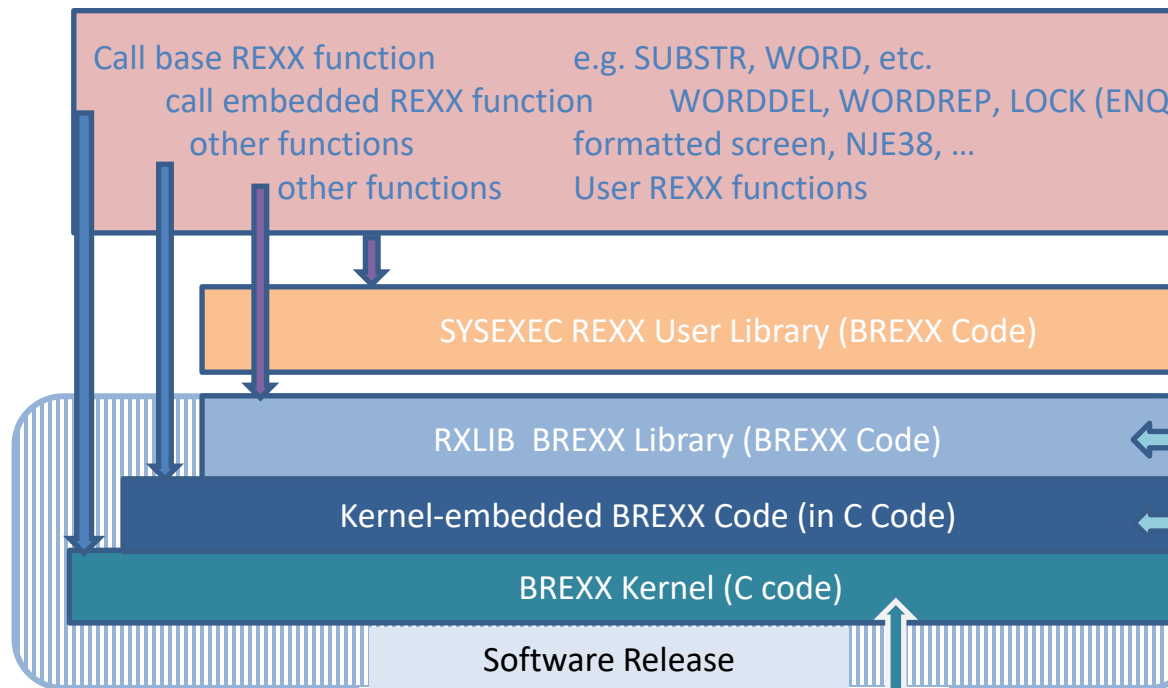
Base REXX Functions
(SUBSTR, WORD)

C-embedded core features

Operating System (MVS)

Files, Screens, NJE38

BREXX Layer Concept



String Functions

ABBREV(*information*,*info*[,*length*])

tests whether *info* is an abbreviation of *information*; returns "1" on true, else returns "0". If *length* is specified then searching takes place only for the first *length* characters.

```
abbrev("billy", "bill") /* 1 */
abbrev("billy", "billa") /* 0 */
abbrev("billy", "billa", 3) /* 1 */
```

CENTRE(string,length[,pad])
CENTER(string,length[,pad])

returns **string** centered in a padded string of length **length**

```
center("rexx",8) /* rexx */
center("rexx",8,"-") /* --rexx-- */
```

CHANGESTR(target,string,replace)

```
changestr("aa","aabbccaabbcc")
```

COMPARE(string1,string2[,pad])

```
compare('bill','bill')  /* 0 */
```

```
compare("aa", "aa") /* 0 */
compare("bi", "bi") /* 0 */
compare("bi-", "bi-") /* 5 */
```

COUNTSTR(target,string)
counts all the appearances of **target** in **string**

```
countstr("aa","aabbccaabbccaa")/*
```

`COPIES(string,n)`
returns **n** concatenated copies of string

```
copies('Vim',3) / * VimVimVim
```

```

BREXX.RXLIB on PUB010
-----
Command ==>

  NAME          TTR      VV  MM  CREATED      CHANGED      INIT      SIZE      MOD  ID
  FMTLIST       04F2001  01.62  21-07-08  21-08-01  12:50:48  1002    1017  0  PEJ
  FMTLIST0      024401  01.29  21-02-01  21-07-09  16:45:59  980     1001  0  PEJ
  FMTMENU       000207  01.29  20-02-03  20-02-07  11:31:47  63      53  0  PEJ
  FMTMON        050901  01.99  21-06-21  21-08-06  07:52:12  165     289  0  PEJ
  FMTMON0       03D901  01.01  21-07-18  21-07-18  06:47:32  298     299  0  PEJ
  FSSPFI        048901  01.99  19-07-31  21-07-26  06:57:14  29      523  0  PEJ
  FSSCREEN      050F05  01.54  19-07-31  21-10-25  10:42:42  93      154  0  PEJ
  FSSDASH       048809  01.07  21-07-20  21-07-26  08:53:15  46      57  0  PEJ
  FSSMENU       034603  01.99  19-08-10  21-07-12  07:32:13  37      207  0  PEJ
  FSSICKY       041305  01.40  21-07-09  21-07-20  14:42:06  18      46  0  PEJ
  LISTALC       000108  01.21  19-02-04  19-07-15  07:54:41  59      75  0  PEJ
  LISTCAT       000A01  01.06  21-02-17  21-02-22  17:29:51  28      80  0  PEJ
  MTTSCAN       001D03  01.02  21-06-24  21-06-24  07:50:08  194     89  0  PEJ
  MVCSB         000805  01.10  19-05-25  20-09-23  08:59:24  14      24  0  PEJ
  N3E38CMD      002A03  01.19  21-03-20  21-07-01  07:03:43  28      25  0  PEJ
  N3E38DIR      038B01  01.99  20-07-04  21-07-14  18:03:31  186     476  0  PEJ
  N3E38DSN      000A09  01.26  20-06-28  21-05-23  08:01:04  25      31  0  PEJ
  PDSDIR        000401  01.41  19-03-04  20-07-06  09:19:49  127     180  0  PEJ
  PDSRESET      000705  01.19  19-03-20  20-08-14  07:49:18  17      63  0  PEJ
  PERFORM       000707  01.08  19-04-17  20-08-14  07:51:51  29      31  0  PEJ
  QUOTE         000109  01.04  19-03-30  19-04-08  11:43:54  14      20  0  PEJ
  RA2E          001203  01.12  19-04-08  21-06-16  07:59:51  43      26  0  PEJ
  READALL       050E01  01.30  19-04-06  21-08-08  07:10:51  7      66  0  PEJ
  RE2A          001205  01.10  19-04-08  21-06-16  08:00:01  38      26  0  PEJ

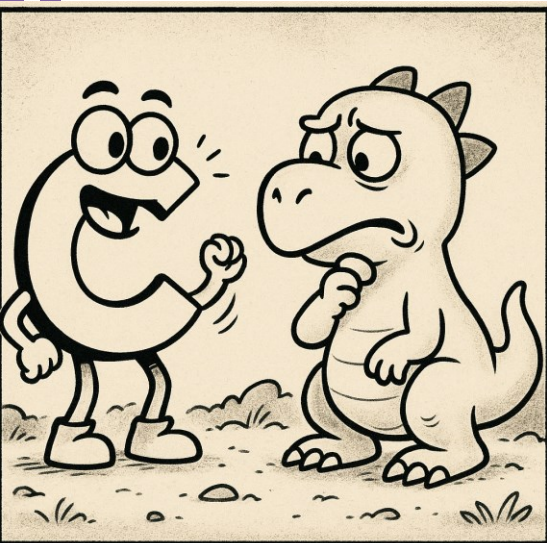
```

[illegible]

Transparent integration of REXX-written core utilities

BREXX/370

Embedded Kernel Functions



C now speaks REXX. Not fluently—but enough to get into trouble.

Embedded REXX Functions

BREXX supports embedded kernel functions written in REXX code strings, which are transparently loaded and behave like native built-in functions.

- Defined in C via ``RxPreLoad()`` as string-based REXX procedures
- Automatically injected at runtime when referenced
- Appear native to the interpreter and user
- Allows definition of new procedures on the fly
- No visible distinction from compiled commands

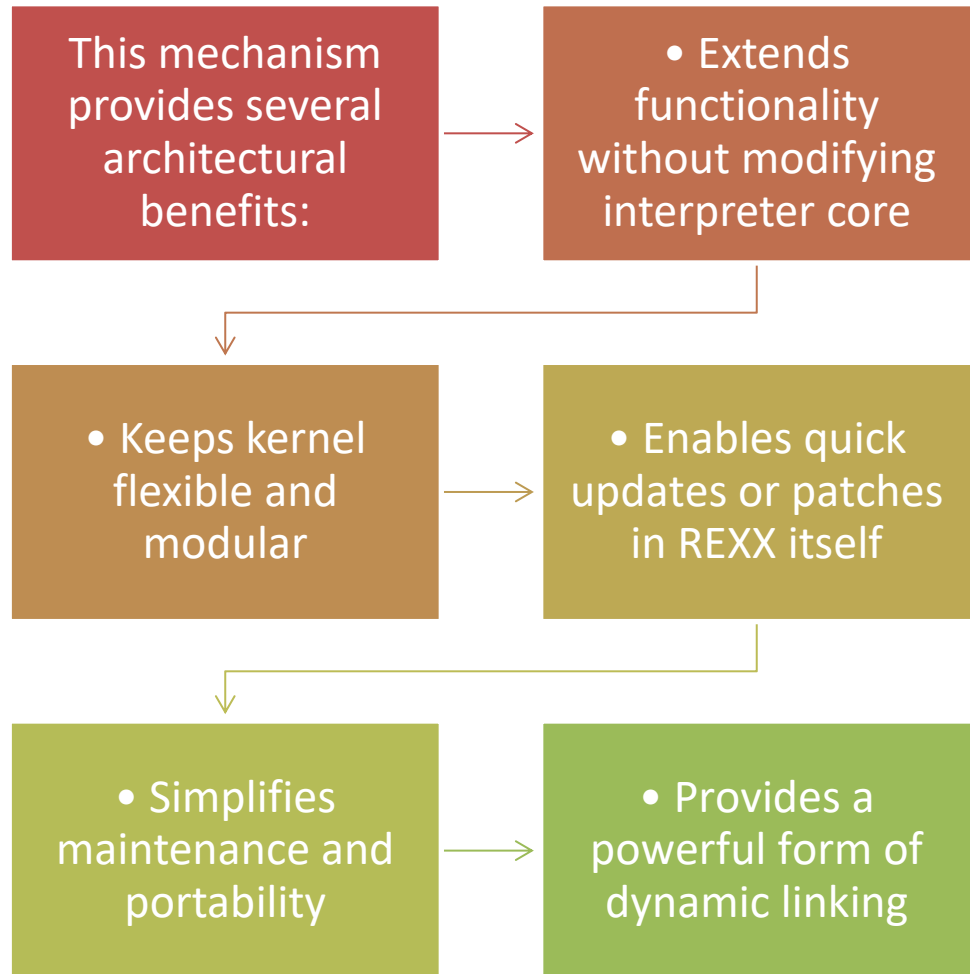
Examples of Embedded Kernel Functions

These REXX functions are defined in `preload.c` and transparently loaded:

- PEEKA, PEEKU, PEEKN – memory inspection
- BASE64ENC, BASE64DEC – base64 utilities
- STEMCOPY, STEM2LL, LL2STEM – stem/list conversions
- DATETIME, EPOCH2DATE – date/time formatting
- JOBINFO, SYSVAR – system environment access
- MOD, ROOT, CLRSCRN – general-purpose utilities
- etc. ...

Embedded Kernel Functions

Design Advantages



Example LISTVOLS volumes of a MVS System

Deep inside the BREXX Reactor Core, this REXX module continues to deliver time-tested power.":

```
} else if (strcmp((const char *) LSTR(rxf->name), "LISTVOLS") == 0) {  
  RxPreLoad(rxf,"LISTVOLS: Procedure  expose volumes.; trace off; parse upper arg option; call privilege('on'); call outtrap('dev.');"   
  "ADDRESS COMMAND 'CP DEVLIST'; call outtrap('off'); call privilege('off'); bi=0; do volj=1 to dev.0;"   
  "parse upper value word(dev.volj,4) with dasd '/' 'vol'.'dev; if dasd<>'DASD' then iterate; bi=bi+1;"   
  "unit=word(dev.volj,3); buffer.bi=vol' 'unit' 'dev; end; bi=bi+1; buffer.0=bi; buffer.bi=bi-1' Volumes found";   
  "if abbrev('FMTLIST',option,3)>0 then call fmtlist ,,'Volume  Unit Device'; else if abbrev('LIST',option,3)>0 then do;"   
  "say 'Volume  Unit Device'; do i=1 to buffer.0; say buffer.i; end; end; else do; buffer.0=buffer.0-1;"   
  "do i=0 to buffer.0; volumes.i=buffer.i; end; end; return;");
```

The moment has come. It's showtime.
Destiny awaits.



BREXX/370

Added Features part I

ADDRESS FSS Formatted Screen Services
Menus, Screens, Lists

VSAM KSDS support read and write VSAM records

ADDRESS LINK/LINKMVS/LINKPGM/ATTACH
Call external programs

ADDRESS MVS Interface to certain REXX environments
as VSAM and EXECIO

ADDRESS TSO Interface to the TSO commands, e.g.
LISTCAT, ALLOC, FREE, etc.

ADDRESS COMMAND
Interface to the Host system of your
MVS system. Typically Hercules or
VM370

BREXX/370

Added Features part II

TCP/IP Interface

Building TCP servers as well as TCP clients

Interface to NJE38

communicating with other MVS and CMS sites: messages and datasets

Operator Interface

sending operator commands

OUTTRAP

hijack write to terminal (say ...)

Receiving Master Trace Table events

Dynamic Dataset Service

CREATE/ALLOCATE/FREE/OPEN (new) Datasets

Create and execute REXX scripts on the fly

Global and Profile Variables

read/write variables across procedures

New REXX Functions

many, many

The moment has come. It's showtime.
Destiny awaits.



MVS 3.8

special functions

RXMVS adds a large set of specialized functions for BREXX that interface directly with **IBM MVS** and **TSO (Time Sharing Option)** environments

These functions are **not part of standard REXX**, but are tightly bound to:
MVS-style datasets and file structures (PDS, PS)

MVS system information blocks (CVT, PSA, SMCA)

TSO-specific features (console access, SVC calls, user IDs)

System Privilege Management (via Privilege, SYSVAR, etc.)

JES/NJE environment integration (e.g., SYSNJVER, RxNjeGetNetId)

ENQ/DEQ Services to guarantee transaction control over databases

Low-level data and memory access (e.g., dumpIt, outtrap, updateIOP)

BREXX MVS Integration Overview

BREXX includes extensions tailored to IBM MVS systems, adding powerful scripting capabilities for system interaction, dataset access, security, and console control.

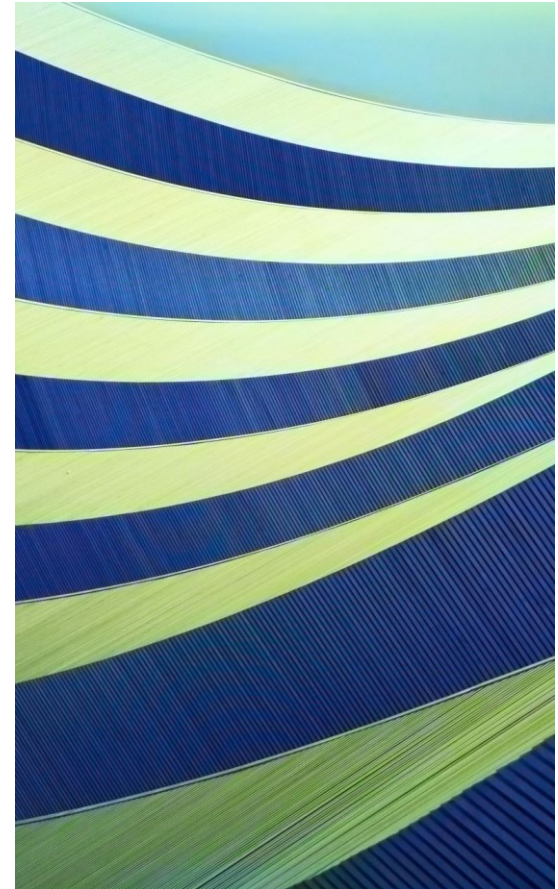
- Access to datasets (PS, PDS), directory entries, record counts

- Use of SVCs and TSO console commands

- Low-level ENQ/DEQ locking mechanisms

- Dynamic privilege control and environment detection (Hercules, VM/370)

- Integration with NJE, SMF, RACF environments



BREXX/370 Array Extensions

BREXX vs. BREXX/370 – Variable Management

BREXX:

- Variables in BREXX are organized using a **binary tree structure**.
- While flexible, this approach introduces **significant overhead** in:
 - **Memory usage** (due to metadata and pointer structures)
 - **Performance**, especially during **intensive iteration** or when dealing with a **large number of variables**
- Each variable access requires tree traversal, slowing down execution.

BREXX/370:

- Variables in BREXX are organized using a **binary tree structure**.
- Introduces **native array support**, allowing **direct indexed access** to data.
- Arrays eliminate the need for binary tree traversal, providing:
 - **Faster variable access**
 - **Reduced memory overhead**
 - Improved performance in **highly iterative or data-heavy tasks**

✓ **Key Advantage:** Arrays in BREXX/370 significantly optimize storage and execution speed, making it more efficient for large-scale or performance-critical REXX applications.

High-Performance Array Support in BREXX

BREXX introduces internal string, integer and float arrays, bypassing traditional REXX variable pools for faster data processing.

- Arrays are stored outside the normal symbol table / variable pool

- String array: ``sarray[]``, with indexes via ``sindex`` and ``sarrayhi[]``

- Integer and Float arrays: optimized for numeric lookup, manipulation, and batch updates

- Operations supported: sort, search, batch load/store

- Greatly reduces CPU load for large-scale operations vs. stem variables

- Designed for MVS workloads and system-level scripting efficiency

sarray[]: High-Speed String Arrays

``sarray[]`` is a fast, low-overhead string array structure designed for heavy-duty scripting.

- Defined as ``char *sarray[sarraymax]``, outside normal REXX variable pools
- Directly accessed by index, avoiding stem variable performance limits
- ``sarrayhi[]`` tracks the highest index used in each array
- Optional ``sindex[]`` provides sorted views or lookup optimization
- Used for fast batch processing, logging, or system dataset parsing
- Ideal for MVS workloads where classic REXX is too slow

sarray[]: Memory Management

- SCREATE – Create a new sarray instance
- SRESIZE – Resize an existing sarray
- SFREE – Free the memory associated with sarray
- SARRAY – Return info/status of an sarray

sarray[]: Element Access & Assignment

- SSET – Set a value at a specific index
- SGET – Retrieve a value from a specific index
- SSUBSTR – Get a substring from an entry
- SWORD – Extract a word from an entry
- SLSTR – Convert list-style string into sarray

Example: High-Speed sarray[] Access

```
RFEEDIT  BREXX.V2R5M3.SAMPLES($SORT) - 1.00
Command ==>
***** Autosave***** Top of Data *****
000001 smax=5000      /* Number of random entries */
000002 /* -----
000003  * QuickSORT
000004  * -----
000005  */
000006 srt=setupArray('QuickSort')
000007 call slist srt,1,10
000008 elp=time('e')
000009 call SQSORT srt,"QUICKSORT"
000010 elp=Trunc(time('e')-elp,3)
000011 say "quicksort time "elp" seconds, items: "smax
000012 call slist srt,1,10
000013 /* -----
000014  * ShellSORT
000015  * -----
000016  */
000017 srt=setupArray('ShellSort')
000018 call slist srt,1,10
000019 elp=time('e')
000020 call SHSORT srt,"SHELLSORT"
000021 elp=Trunc(time('e')-elp,3)
000022 say "shellsort time "elp" seconds, items: "smax
000023 call slist srt,1,10
000024 exit _
000025 /* -----
000026  * Some Subroutines
000027  * -----
000028  */
000029 setupArray:
000030 parse arg stype
000031 srt=Screate(smax)
000032 do i=1 to smax
000033     call sset(srt,,right(RANDOM(0,10000),6,'0')) 'stype)
000034 end
000035 return srt
```


Example: High-Speed sarray[] Access

Entries of Source Array: 1

Entry	Data
-------	------

00001	002037 QuickSort
00002	007084 QuickSort
00003	001651 QuickSort
00004	002001 QuickSort
00005	006034 QuickSort
00006	009235 QuickSort
00007	005196 QuickSort
00008	000417 QuickSort
00009	003595 QuickSort
00010	004499 QuickSort

10 Entries

quicksort time 0.121 seconds, items: 5000

Entries of Source Array: 1

Entry	Data
-------	------

00001	000004 QuickSort
00002	000005 QuickSort
00003	000006 QuickSort
00004	000008 QuickSort
00005	000010 QuickSort
00006	000010 QuickSort
00007	000010 QuickSort
00008	000011 QuickSort
00009	000015 QuickSort
00010	000016 QuickSort

10 Entries

Entries of Source Array: 2

Entry	Data
-------	------

00001	009592 ShellSort
00002	000440 ShellSort
00003	003683 ShellSort
00004	006024 ShellSort
00005	004442 ShellSort
00006	003596 ShellSort
00007	000618 ShellSort
00008	004510 ShellSort
00009	000006 ShellSort
00010	008781 ShellSort

10 Entries

shellsort time 0.306 seconds, items: 5000

Entries of Source Array: 2

Entry	Data
-------	------

00001	000002 ShellSort
00002	000003 ShellSort
00003	000003 ShellSort
00004	000003 ShellSort
00005	000003 ShellSort
00006	000004 ShellSort
00007	000004 ShellSort
00008	000006 ShellSort
00009	000006 ShellSort
00010	000010 ShellSort

10 Entries

```
quicksort  time 5.655 seconds, items: 5000
shellsort  time 39.862 seconds, items: 5000
heapsort   time 12.749 seconds, items: 5000
Bubble Sort is not recommended for items > 250.
***
```

STEM sort



SARRAY sort



STEM sort: timeless in all the wrong ways.
SARRAY sort: so fast, you'll need a new hobby.

sarray[]: Transformation & Modification

- SCHANGE – Replace occurrences of a substring

- SREVERSE – Reverse the order of entries

- SSWAP – Swap two entries by index

- SCLC – Clear array content

- SCOPY – Copy one sarray to another

- SINSERT – Insert a new entry

- SDEL – Delete an entry by index

- SPASTE – Append entries from another source

sarray[]: Search & Selection



- SSEARCH – Search for an entry



- SSELECT – Select entries by pattern or value



- SKEEP – Keep matching entries

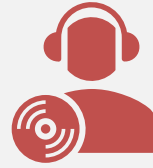


- SKEEPAND – Keep entries matching multiple conditions



- SDROP – Drop matching entries

sarray[]: Sorting & Counting



- SQSORT – Quick sort for entries



- SHSORT – Shell sort (likely stable)



- SCOUNT – Count elements or pattern matches

The moment has come. It's showtime.
Destiny awaits.



sarray[]: Set Operations

- SMERGE – Merge sorted sarrays

- SINTERSECT – Intersection of arrays

- SDIFFERENCE – Difference (A not in B)

- SUNIFY – Remove duplicates

sarray[]: Load & Store



- SREAD – Load sarray from file or source



- SWRITE – Write sarray to file or target



- S2LL – Convert to linked list



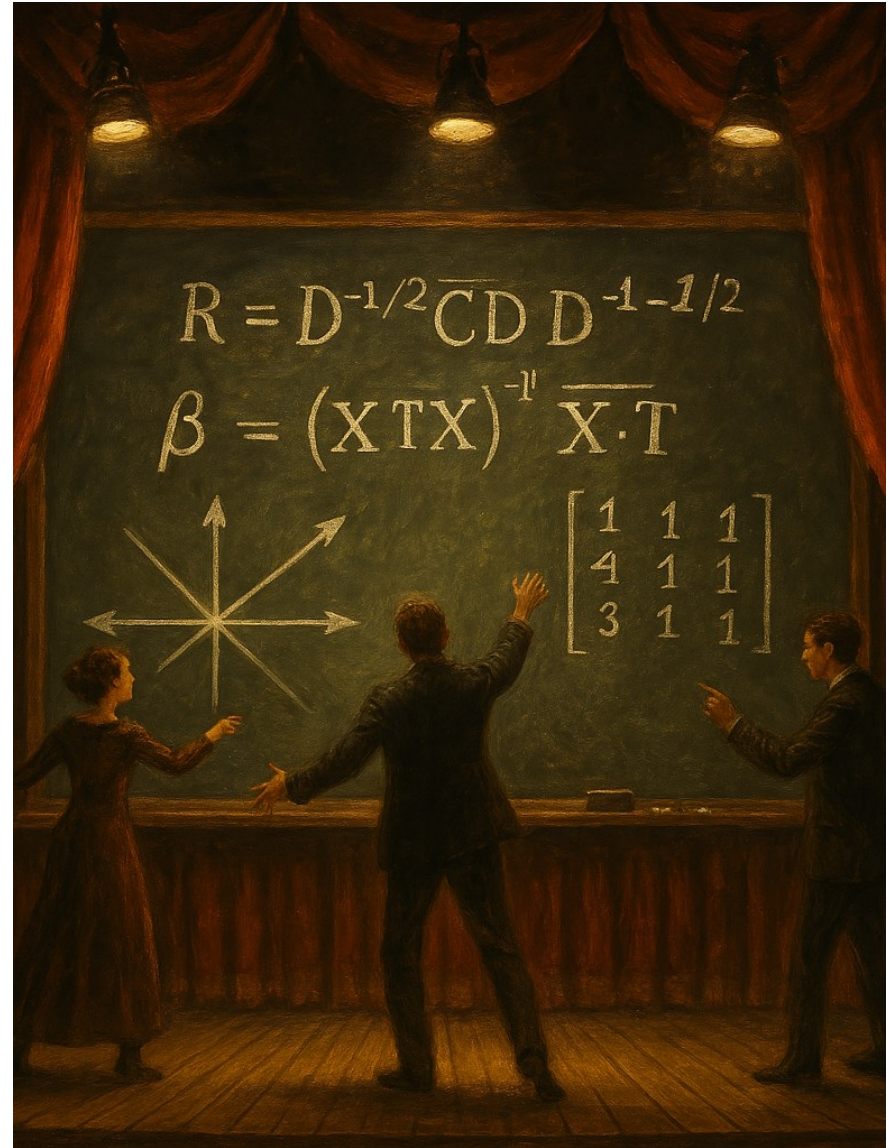
- S2IARRAY – Convert to integer array



- SLIST – Return sarray as a stem/list

Exploring the Unexpected
Passion of Linear Algebra in
BREXX

I Came for the Vectors, Stayed for the Drama



Why Matrix Operations Matter

+
.
0



Matrix Operations in BREXX are foundational tools for implementing many statistical and analytical methods:



Correlation Analysis – based on covariance and matrix algebra



Regression Analysis – uses matrix inversion and multiplication



Factor Analysis – relies on eigenvalue decomposition and matrix transformations



Principal Component Analysis (PCA) – transforms data into eigenvector space



Data Normalisation, aggregation, and reduction techniques



Enables advanced statistical modelling within REXX scripts

The moment has come. It's showtime.
Destiny awaits.



Matrix Operations: Creation & Access

+

.

0

MCREATE – Create a new matrix

MGET – Retrieve value at a given
row/column

MSET – Set value at a given row/column

MDELCOL – Delete a column

MDELRROW – Delete a row

Matrix Operations: Manipulation

- MCOPY – Copy one matrix to another

- MTRANSPOSE – Transpose matrix (flip rows/columns)

- MNORMALISE – Normalize data across rows or columns

- MINVERT – Invert the matrix (if square and invertible)

Matrix Operations: Arithmetic

- MMULTIPLY – Multiply two matrices

- MADD – Add two matrices

- MSUBTRACT – Subtract matrices

- MPROD – Element-wise product

- MSQR – Square each element

- MSCALAR – Multiply matrix by a scalar

Linear Algebra:

Correlation & Regression

Mathematical formulas using matrix operations

$$R = (\sqrt{D})^{-1} C (\sqrt{D})^{-1}$$

where: C = Covariance matrix

D = Diagonal matrix of variances

$$D = \text{diag}(\text{Var}(X))$$

Linear regression coefficients β :

$$\beta = (X^T X)^{-1} X^T$$

where: X = Matrix of input variables (predictors)

Y = Output vector (target variable)

β = vector of regression coefficients

And now, before we're escorted out by an angry mob with torches, we shall summarise... and leg it.



BREXX/370

Summary



Rooted in Proven Design

Built upon the original BREXX interpreter by Vasilis Vlachoudis — a compact, portable implementation of REXX that set the foundation.



Adapted for Mainframe Environments

BREXX/370 brings this design to the IBM MVS 3.8 platform, extending core functionality while remaining true to its architectural simplicity.



System-Level Integration

Deep integration with MVS subsystems: VSAM, NJE38, formatted screen I/O, and more — enabling seamless scripting for complex system interactions.



Expanded Capabilities

Enhanced with array support, embedded functions, and matrix operations — enabling data analysis, transformation, and advanced scripting.



Sustainable and Documented

Continuously maintained, with full documentation, source availability, and a transparent development approach.

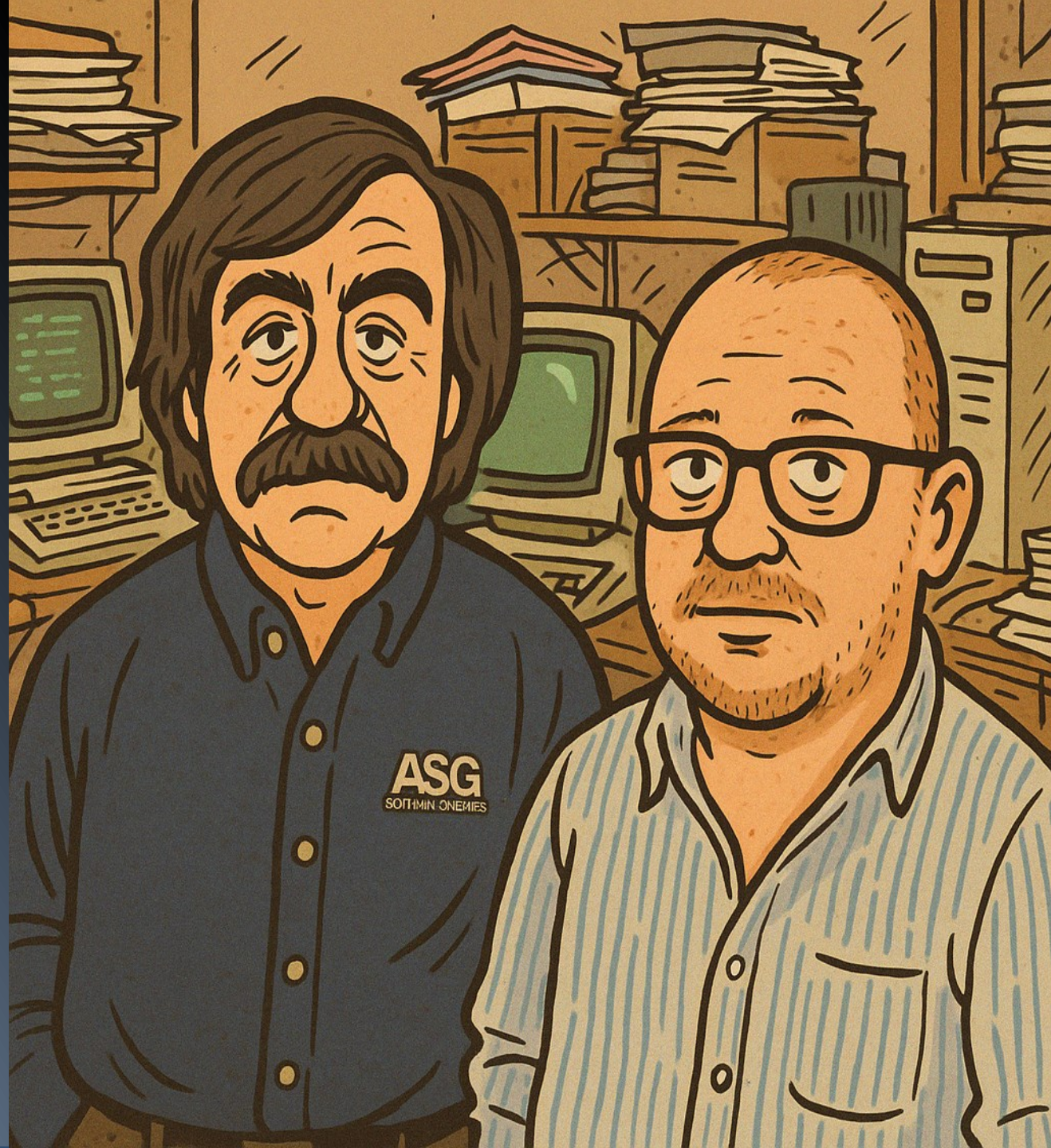


Same Language, New Ground

BREXX/370 continues the REXX tradition: readable, robust, and remarkably adaptable — now fully at home on the mainframe.

The BREXX/370 Development Team

Mike & Peter



REXX reads like English, runs like C, and debugs like a dream

So long,
and thanks
for all the
STEMs!

From DOS to MVS — BREXX never stopped listening

NAME

brexx370—stubbornly portable, surprisingly powerful
REXX interpreter for IBM mainframes

SYNOPSIS

brexx370 [--retro] [---nostalgia] [--macro=love]
script.rexx

DESCRIPTION

BREXX/370 brings the minimalist elegance of BREXX to
the world of MVS 3.8 and beyond.
It interprets REXX scripts with speed, style, and
suspiciously few lines of code.
Originally designed for platforms no one admits using
anymore, it now runs on systems your grandchildren
won't believe ever existed.

FEATURES

- Inline compilation using `_CodeAddByte()` — because
we could
- Control flow patching via `CODEFIXUP` macros —
trust the process
- Parses like it's 1889, runs like it's 2025
- No AST. No regrets.

BUGS

- May cause feelings of nostalgia
- Cannot compute your life decisions
- Occasionally too efficient for its own good

AUTHOR Written by people who looked at IBM's architecture