

Implementing Exceptions in Z79Forth/A

François Laagel, IEEE

Introduction to Z79Forth

- Hardware (HD6309 based SBC) and software platform for learning Forth and hardware experimentation.
- HW design started at the end of 2018 and lasted about 3 months.
- Forth is the OS and mostly EEPROM resident.
- Code originally written for the Z80 in 1983 (EDTASM+ by Microsoft).
- Either a 79-STANDARD sub-set or an extended ANS94 core word set.
- 32 KB SRAM, 8 KB EEPROM, mass storage over CompactFlash (64 MB).
- Asynchronous serial line for the console (USB or RS232 based).
- USB powered.
- Has been ported to other platforms (Scott Vitale, David J. Richards).

Z79Forth/A as a Minimalist Implementation

- Produces machine code directly (no VM).
- Return addresses are stored in the 6309 system stack (via **Sreg**).
- The 6309 **Ureg** points directly to the data stack.
- Error handler can perform *introspection*.
- No multi-tasking support (single user variable set).
- No files, vocabularies or floating point support.
- Has a dedicated control flow stack.
- Has a dedicated exception stack.

Methodology

Confidence is what you have before you understand the problem. *Woody Allen*

- Prototyping user level exceptions in Forth.
- Design with Gemini (concepts identification, exceptions typology).
- Step by step implementation (6309 assembly).
- Observability concerns.
- Testing/troubleshooting and standard conformance.
- Application level verification.
- Packaging, documentation changes and source control updates.
- Public release on Github.

Constraints Limited EEPROM capacity.

Goals Standard conformance, quality/predictability.

Forth Standard Exception Concepts

- The Exception word set: **CATCH**, **THROW**, **ABORT** and **ABORT"**
- Some incidence on **EVALUATE** and **LOAD**.
- Exception numbers have the same size as a *cell*.
- Negative exceptions are mostly *system generated*.
- Positive exceptions are *application defined*.
- An *error condition* is signalled by throwing an exception.
- Can be nested, so an *exception stack* is needed.
- Entries on the exception stack are called *frames*.
- An exception frame has all the elements describing a saved *process context*.
- The Forth Standard suggests a *reference implementation*, which is not **normative**.

```
( Exceptions )
variable handler
: catch ( xt -- n )
  fp@ >r sp@ >r handler @ >r rp@ handler ! execute
  r> handler ! rdrop rdrop 0 ;
: throw ( n -- )
  dup if handler @ rp! r> handler !
    r> swap >r sp! drop r> r> fp! else drop then ;
```

Brad Nelson's Reference Implementation (SVFIG)

User Level Exceptions Prototyping (1/2)

- Dedicated exception stack primitives: **>E** and **E>**.

```
\ Pop one cell from the exception stack.  
: E> ( -- n )  
  ESP @ DUP ESbottom = IF  
  ." Estack UDF"  ABORT  
  THEN  
  
  DUP @  
  SWAP CELL+ ESP ! ;  
  
\ Push one cell to the exception stack.  
: >E ( n -- )  
  ESP @ DUP EStop = IF  
  ." Estack OVF"  ABORT  
  THEN  
  
  1 CELLS - DUP ESP ! ! ;
```

- A stack frame has 3 cells: the system stack pointer, the data stack pointer and the return stack pointer.

User Level Exceptions Prototyping (2/2)

```
: CATCH ( xt -- n | 0 ) \ Build up exception frame.
s CELL+ >E \ Save S register ('Sreg0')
sp@ CELL+ >E \ Data stack pointer backup ('sp0')
rp@ >E \ We omit the currently stacked 'xt'

EXECUTE

\ Normal execution in the absence of exceptions. We have to
\ assume that return stack integrity rules are followed.
E> DROP \ Drop rp@
E> DROP \ Drop sp@
E> DROP \ Drop saved Sreg
0 ; \ "No problemo" indication
```

```
: THROW ( n -- )
?DUP IF \ An exception was triggered
\ Restore saved context from the exception stack.
E> rp! \ Restore the return stack pointer

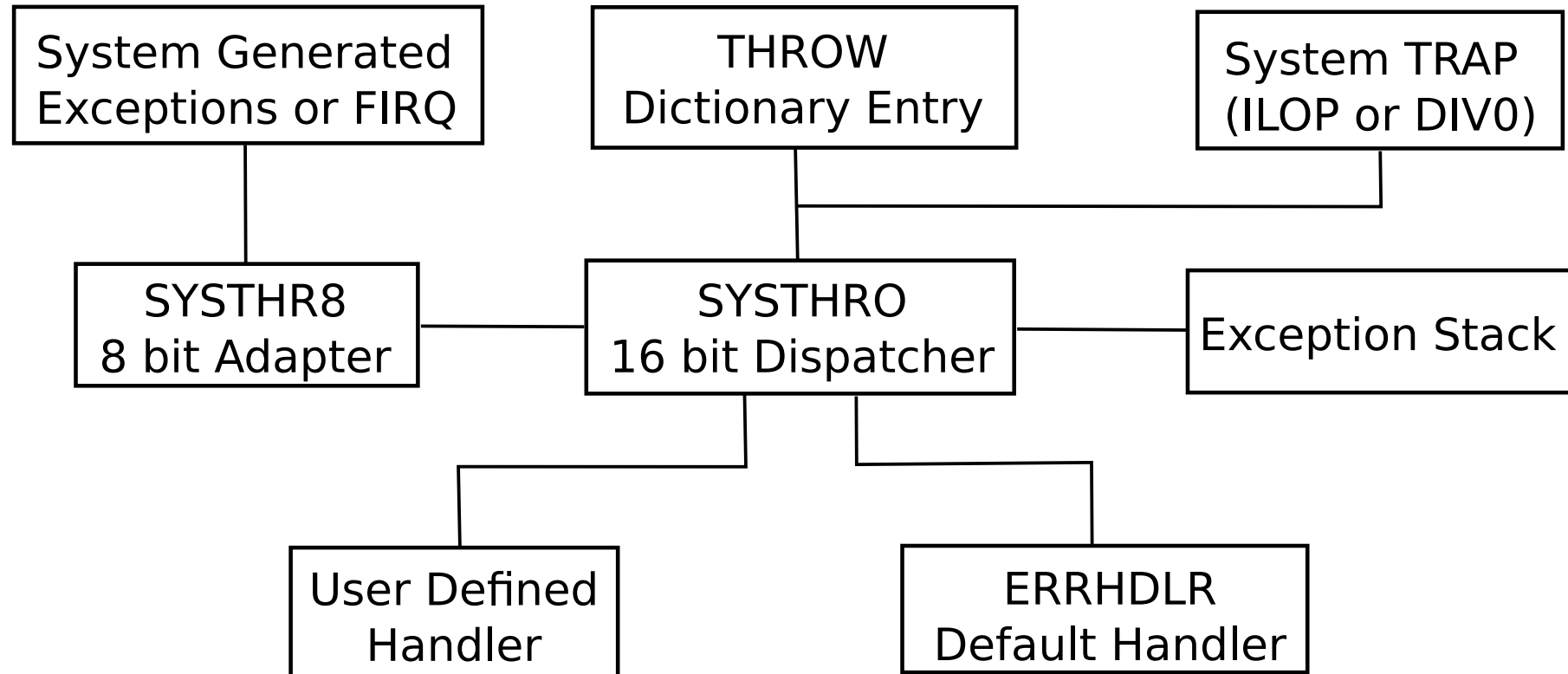
>R \ E: Sreg0\sp0, R: n
E> sp! \ Restore the data stack pointer
R> \ Stack up 'n' to the data stack
E> \ E: -, S: n\Sreg0
\ Need to restore the S register from the TOS and then
\ simply RTS like a good boy. Using inline code.
[ jsrNPOP, \ TFR X,S
$1F14 , ]
THEN ; \ Implied RTS sitting right here.
```

Implementation—Taming the Error Handler

- An 8 bit affair. *Abstraction* needed.
- A cell based interface was required for **THROW**.
- Negative error codes handling capability.
- Need to align error codes to standard specified exception numbers.

```
* Error condition codes passed through Breg to SYSTHR8.
EABORT equ -1 STD -- ABORT
EABRTQ equ -2 STD -- ABORT"
EDSOVF equ -3 STD -- DS overflow
EDSUDF equ -4 STD -- DS underflow
ERSOVF equ -5 STD -- RS overflow
ERSUDF equ -6 STD -- RS underflow
ESSEGV equ -9 STD -- Illegal argument (CHAR FIND PICK EXECUTE)
EOORNG equ -11 STD -- OoR error
EUNDEF equ -13 STD -- ? / Undefined word
ESTATE equ -14 STD -- Incorrect STATE
ENONAM equ -16 STD -- Missing word (POSTPONE : CREATE)
ENAMLN equ -19 STD -- Name too long
ERONLY equ -20 STD -- R0 word
ENOSUP equ -21 STD -- IO error or TRAP (DIV0 ILOP)
EILCST equ -22 STD -- Illegal construct
EUIINTR equ -28 STD -- SIGINT
ENOCRE equ -31 STD -- Not CREATED
* Allocated against the standard specification. These are not catchable--i.e.
* we branch directly to ERRHDLR (under the 16 bit interface). They're not
* recoverable anyway.
EIN00R equ -32 >IN OoR
EESOVF equ -33 ES overflow
```

Exception Dispatching



- The error handler becomes a *backend* invoked only for uncaught exceptions.
- We can have a cell based interface and keep the old 8 bit API.

Actual Implementation—CATCH

CATCH	fcb	\$5	ANSI (Exception)
	fcc	'CATCH'	(i*x xt -- j*x 0 i*x n)
	fdb	THROW	
	RFCS		
	jsr	MIN1PST	Make sure at least 'xt' is stacked up
* Save exception frame.			
	orcc	#FFLAG	Disable FIRQ
	leax	2,u	
	bsr	EPUSH	sp@ CELL+ >E (omit 'xt')
	ldx	RSP	
	bsr	EPUSH	rp@ >E
	tfr	s,x	Return address pointer if uncaught
	bsr	EPUSH	Sreg >E
	andcc	^FFLAG	Re-enable FIRQ
	RFXT	jsr,EXCT+10	xt for EXECUTE
* Normal execution in the absence of exceptions.			
	orcc	#FFLAG	Disable FIRQ
	bsr	EPOP	E> DROP (original Sreg)
	bsr	EPOP	E> DROP (original rp@)
	bsr	EPOP	E> DROP (original sp@)
	andcc	^FFLAG	Re-enable FIRQ
	tfr	0,x	
	jmp	NPUSH	"No problemo" indication to DS

Actual Implementation—THROW

```

THROW    fcb      5           ANSI (Exception)
         fcc      'THROW'     ( k*x n -- k*x | i*x n )
         fdb      EBUFS
         RFCS
         jsr      NPOP
         tfr      x,d
         tstb     0           0 exception number?
         bne      SYSTHRO     if not, we continue execution in SYSTHRO
         rts                               just EXIT if 'n' is 0

```

■ * Compatibility entry point for 8 bit system generated exceptions.

* Note: Breg never is zero if we enter via this interface.

SYSTHR8 sex Sign extension Breg to Dreg

* Exception number is in D. Route it to ERRHDLR or to the user code just

* following the latest call to CATCH, depending on the state of the exception

* stack. Preserve X and D before resorting to ERRHDLR (unhandled exception).

```

SYSTHRO  pshs     x
         ldx      ESP          The exception stack pointer
         cmpx     #ESTBOT
         puls     x           This does not affect CReg
         lbeq     ERRHDLR     Stack is empty, invoke the default handler
* At this point we need to return control just after the latest call to CATCH.
         orcc     #FFLAG      Disable FIRQ
         bsr      EPOP        Does not affect Dreg or Yreg
         tfr      x,s         Restore Sreg
         bsr      EPOP
         stx      RSP         E> rp!
         bsr      EPOP
         tfr      x,u         E> sp!
         andcc    #^FFLAG     Re-enable FIRQ
         tfr      d,x
         jmp      NPUSH       Exception number to the data stack

```

ABORT" Concerns (1/2)

Run-time:

$(i * x \ x^1 \ -- \ | \ i * x) (R: j * x \ -- \ | \ j * x)$

Remove x^1 from the stack. If any bit of x^1 is not zero, perform the function of **-2** **THROW**, displaying *ccc* if there is no exception frame on the exception stack.

The Standard Specification

- If TOS in NZ, a string (**caddr\length**) is pushed on the data stack and -2 is thrown.
- If that exception is caught, it is supposed to be handled by the application.
- If it is uncaught, the string is displayed and the semantics of ABORT is performed.

```
: ABORT"                                \ ANSI (Exception)
  ( Compilation: "ccc<quote>" -- )
  ( Runtime: i*x x1 -- | i*x ) ( R: j*x -- | j*x )
  POSTPONE IF
  POSTPONE S"
  -2 POSTPONE LITERAL
  POSTPONE THROW
  POSTPONE THEN ; IMMEDIATE RESTRICT
```

The Z79Forth/A Implementation

ABORT" Concerns (2/2)

If ABORT" is caught and there was something on the data stack, there will still be something (same depth + 1) on the data stack but there is no way for a user defined handler to tell, in a predictable manner, what it is...

```
Gforth 0.7.9 20231019
Authors: Anton Ertl, Bernd Paysan, Jens Wilke et al., for more type `authors'
Copyright © 2023 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <https://gnu.org/licenses/gpl.html>
Gforth comes with ABSOLUTELY NO WARRANTY; for details type `license'
Type `help' for basic help
: foo ABORT" Gottya!" ; ok
1 ' foo CATCH DROP HEX .S <1> -2 ok 1
```

```
b=16|<1> #-2 |o:Forth Forth Root Forth
```

```
Z79Forth/AE 6309 ANS Forth System
20260820 (C) Francois Laugel 2019
```

```
RAM OK: 32 KB
SanDisk SDCFJ-128
Block #1 loaded
: foo ABORT" Gottya!" ; ok
1 ' foo CATCH DROP HEX .S 19DD ok
$19DD .' F00+000C ok
see foo
19D1 BDE817 jsr NPOP
19D4 2603 bne *+05
19D6 7E19F6 jmp F00+0025
19D9 7E19E4 jmp F00+0013
19DC 07 fcb $07
19DD 47 fcb $47
19DE 6F fcb $6F
19DF 74 fcb $74
19E0 74 fcb $74
19E1 79 fcb $79
19E2 61 fcb $61
19E3 21 fcb $21
19E4 8E19DC lxi F00+000B
19E7 BDE808 jsr NPUSH
19EA BDF7E2 jsr COUNT
19ED 8EFFFFE lxi ???
19F0 BDE808 jsr NPUSH
19F3 BDEA5B jsr THROW
19F6 39 rts ok
```

```
CTRL-A Z for help | 38400 8N1 | NOR | Minicom 2.8 | VT102 | Offline | ttyS0
```

A Case for **EVALUATE** (1/2)

```
: EVALUATE ( i*x c-addr u -- j*x )  
  SAVE-INPUT N>R  
  ( c-addr u) #TIB 2!  >IN OFF  LINE OFF  BLK OFF  -1 'SOURCE-ID !  
  ['] INTERPRET CATCH  NR> RESTORE-INPUT DROP  THROW ;
```

SwiftForth x86_64 4.1.10 Definition

A Case for **EVALUATE** (2/2)

```

LOAD    fcb    4          ANSI (Block)
        fcc    'LOAD'      ( i*x u -- j*x )
        fdb    TERPRET
        RFCS
        jsr    NPOP        ZFLAG is set by NPOP
        bne    LOAD1
        rts
LOAD1   jsr    SAVINP      Block 0 is _not_ loadable
        stx    UBLK        SAVE-INPUT. X is preserved
        ldd    #BLKSIZ     Update BLK with the LOAD argument
        std    ISLEN       1024 bytes
        clrd              Set input stream length
        sta    SRCID       Not invoked in EVALUATE context
LOAD2   std    UT0IN       Clear >IN
        std    ISEADDR     End of input stream address (excluded)
* Map the new BLK in, interpret code from there.
        RFXT    ldx,#TERPRET+12 XT for INTERPRET
        jsr    NPUSH       to the data stack
        RFXT    jsr,CATCH+8   ['] INTERPRET CATCH
        jsr    RSTINP      RESTORE-INPUT
        jsr    BKIN2PT     Map BLK in (if needed) and update BSBFADR
* We need to throw back the caught exception (if any) before returning.
        RFXT    jmp,THROW+8

EVAL    fcb    8          ANSI (Core/Block)
        fcc    'EVALUATE'  ( i * x c-addr u -- j * x )
        fdb    LOAD
        RFCS
        jsr    MIN2PST     Need at least 2 parameters on the data stack
* The whole thing looks like LOAD1 except we do not need to map a block in.
        jsr    SAVINP      SAVE-INPUT. X and Y are preserved
        UCNPOP            U to X
        stx    ISLEN       Set ISLEN from U
        UCNPOP            C-ADDR to X
        stx    BSBFADR     Set BSBFADR from C-ADDR
        lda    #$FF
        sta    SRCID       -1 (byte) to SRCID. Invoked in EVALUATE context
        clrd              Future value for >IN
        std    UBLK        Target block number is zero
        bra    LOAD2       Interpret, restore input context and proceed

```

Observability Notes

- Serial line input is interrupt driven in Z79Forth. When ^C is detected (in the FIRQ handler), control will be redirected to a small piece of code which:
 - Sets the return address to SYSTHRO.
 - Preserves the original user code return address in the system stack.
 - Sets the exception number to EUINTR (via Dreg).
- The perils of tail call optimization. **CATCH** should not be allowed to be terminal. A few changes to ; were needed.

```
* Optimization: replace the last JSR by a JMP, if possible.
    leay    -3,x          Yreg has HERE - 3, Dreg has JSRLAST
    cmpr    d,y
    bne     @rtsreq
    ldd     1,y           Address of the last word called
    RFXT    cmpd,#CATCH+8 Observability: we definitely do
    beq     @rtsreq       not want CATCH to be terminal
* Tail call optimization applies (Case #3).
    lda     #JMPOPC
    sta     ,y
    bra     @finalz
@rtsreq lda     #RTSOPC      RTS inherent
    sta     ,x+
@finalz stx     DICEND      Update HERE
    IFNE    RELFEAT
    RFXT    jsr,MONITOR+10  XT for MONITOR. All : words are candidates
*                               for integrity check by ICHECK.
    ENDC
    rts    RELFEAT
```

Conclusions

- Exception support in Forth is very useful and can greatly simplify your code.
- An application should not hesitate to re-throw an exception that it does not own. The default handler might provide more useful information.
- Firmware supported exceptions now is an integral part of Z79Forth/AE.
- Standard conformance is essential.

Q&A