

EXCVATE: Spoofing Exceptions and Solving Constraints to Test Exception Handling in Numerical Libraries

Jackson Vanover^{*}, James Demmel[†], Xiaoye Sherry Li[‡], Cindy Rubio-González^{*}

^{*}University of California, Davis

[†]University of California, Berkeley

[‡]Lawrence Berkeley National Laboratory



U.S. Department of Energy, Office of Science, Advanced Scientific Computing Research, under award DE-SC0020286,
and the National Science Foundation under award CCF-1750983.

Numerical code should be correct

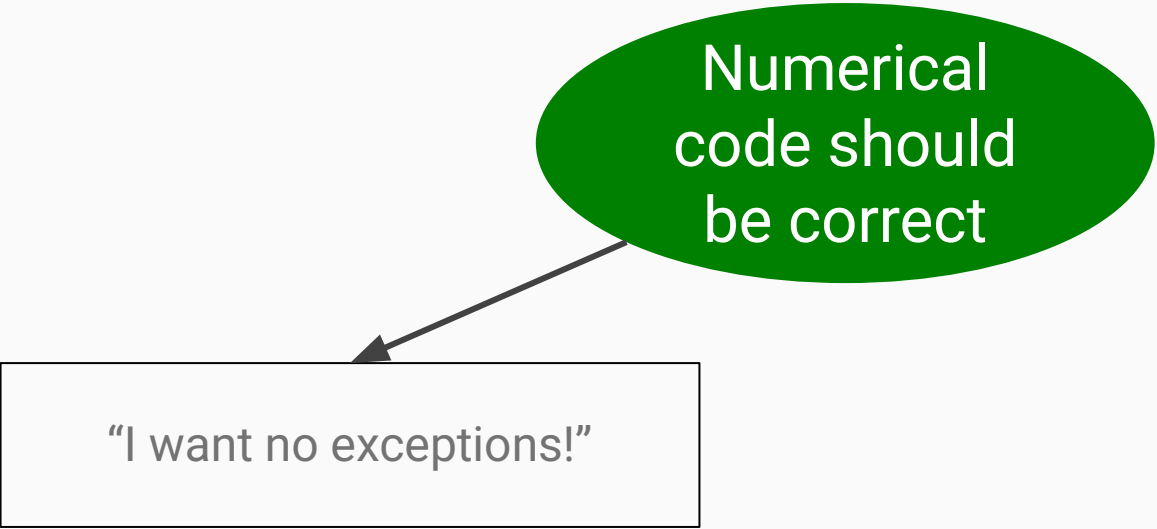
Numerical code should be correct
but floating-point arithmetic is inherently error-prone.

Numerical code should be correct
but floating-point arithmetic is inherently error-prone.

**OVERFLOW
DIVIDE-BY-ZERO
INVALID
INEXACT
UNDERFLOW**

Numerical
code should
be correct

Numerical
code should
be correct



```
graph TD; A([Numerical code should be correct]) --> B["I want no exceptions!"]
```

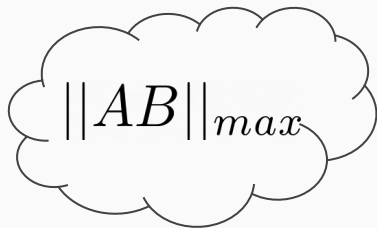
“I want no exceptions!”

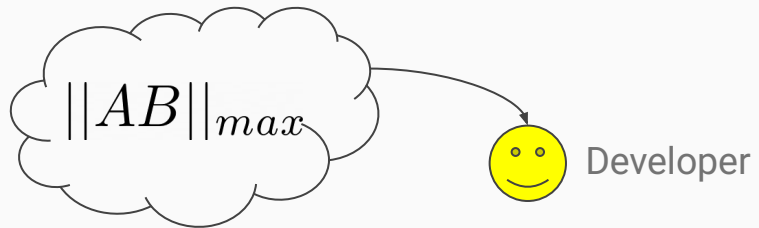
```
graph TD; A([Numerical code should be correct]) --> B["I want no exceptions!"]; A --> C["If there is an exception, I want to know about it!"]
```

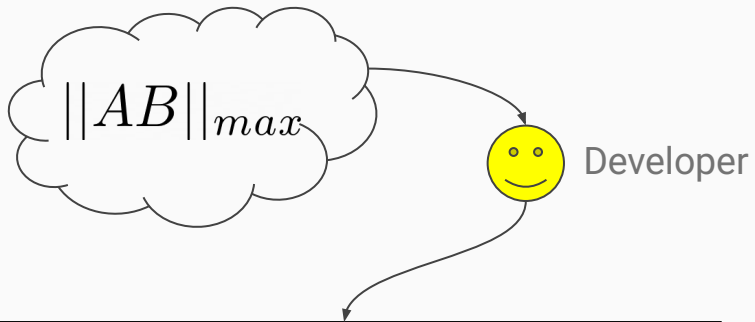
Numerical
code should
be correct

"I want no exceptions!"

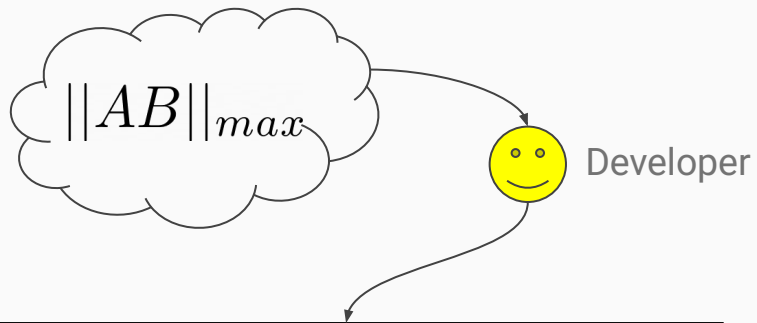
"If there is an exception,
I want to know about it!"


$$||AB||_{max}$$



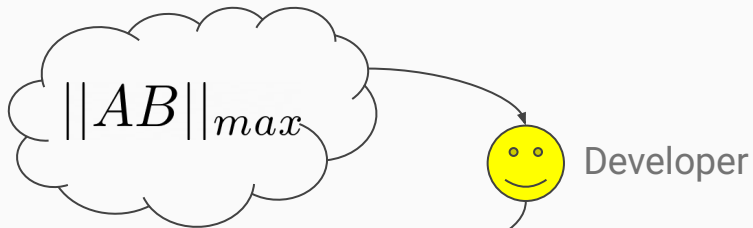


```
function mmMaxNorm(dim, A, B)
    ...
    result = 0.0
    do i = 1, dim
        do j = 1, dim
            acc = 0.0
            do k = 1, dim
                acc = acc + A(i,k) * B(k,j)
            end do
            result = max(result, abs(acc))
        end do
    end do
    ...
end function mmMaxNorm
```



```
function mmMaxNorm(dim, A, B)
    ...
    result = 0.0
    do i = 1, dim
        do j = 1, dim
            acc = 0.0
            do k = 1, dim
                acc = acc + A(i,k) * B(k,j)
            end do
            result = max(result, abs(acc))
        end do
    end do
    ...
end function mmMaxNorm
```

$$A = \begin{bmatrix} 0.0 & 0.0 \\ -3.40282 \times 10^{38} & -2.4245 \end{bmatrix}$$
$$B = \begin{bmatrix} 0.0 & 3.06254 \times 10^{38} \\ 0.0 & -3.08694 \times 10^{38} \end{bmatrix}$$



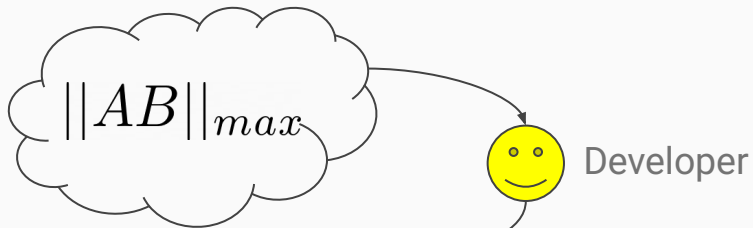
```
function mmMaxNorm(dim, A, B)
...
result = 0.0
do i = 1, dim
  do j = 1, dim
    acc = 0.0
    do k = 1, dim
      acc = acc + A(i,k) * B(k,j)
    end do
    result = max(result, abs(acc))
  end do
end do
...
end function mmMaxNorm
```

$$A = \begin{bmatrix} 0.0 & 0.0 \\ -3.40282 \times 10^{38} & -2.4245 \end{bmatrix}$$

$$B = \begin{bmatrix} 0.0 & 3.06254 \times 10^{38} \\ 0.0 & -3.08694 \times 10^{38} \end{bmatrix}$$

gfortran

bin1



```
function mmMaxNorm(dim, A, B)
...
result = 0.0
do i = 1, dim
  do j = 1, dim
    acc = 0.0
    do k = 1, dim
      acc = acc + A(i,k) * B(k,j)
    end do
    result = max(result, abs(acc))
  end do
end do
...
end function mmMaxNorm
```

$$A = \begin{bmatrix} 0.0 & 0.0 \\ -3.40282 \times 10^{38} & -2.4245 \end{bmatrix}$$
$$B = \begin{bmatrix} 0.0 & 3.06254 \times 10^{38} \\ 0.0 & -3.08694 \times 10^{38} \end{bmatrix}$$

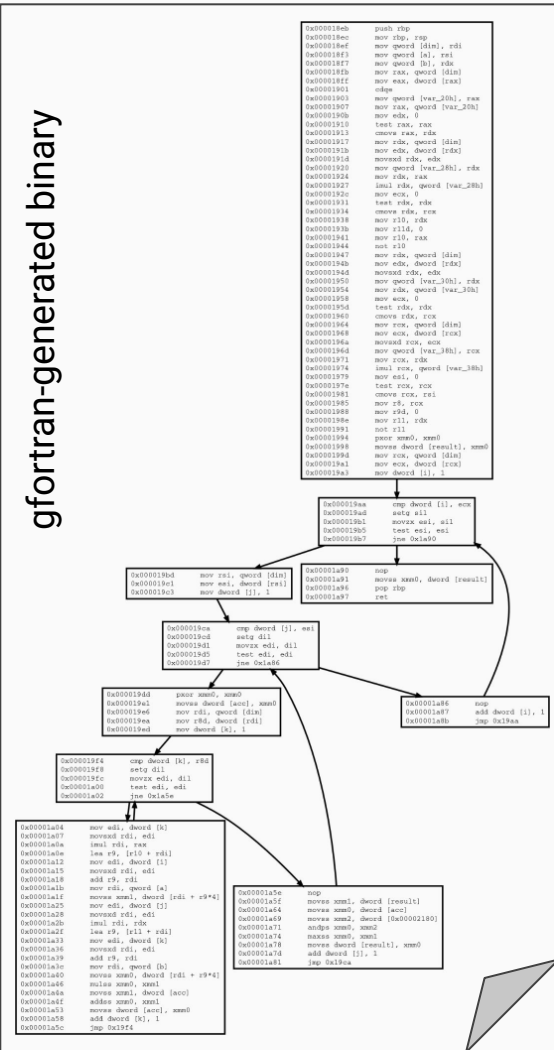
gfortran

bin1

0.0

INCORRECT

gfortran-generated binary



The eighth execution of this add instruction triggers an “Invalid” exception, generating a NaN that is ultimately lost.

```

0x00001a04    mov edi, dword [k]
0x00001a07    movsxd rdi, edi
0x00001a0a    imul rdi, rax
0x00001a0e    lea r9, [r10 + rdi]
0x00001a12    mov edi, dword [i]
0x00001a15    movsxd rdi, edi
0x00001a18    add r9, rdi
0x00001a1b    mov rdi, qword [a]
0x00001a1f    movss xmm1, dword [rdi + r9*4]
0x00001a25    mov edi, dword [j]
0x00001a28    movsxd rdi, edi
0x00001a2b    imul rdi, rdx
0x00001a2f    lea r9, [r11 + rdi]
0x00001a33    mov edi, dword [k]
0x00001a36    movsxd rdi, edi
0x00001a39    add r9, rdi
0x00001a3c    mov rdi, qword [b]
0x00001a40    movss xmm0, dword [rdi + r9*4]
0x00001a46    mulss xmm0, xmm1
0x00001a4a    movss xmm1, dword [acc]
0x00001a4f    addss xmm0, xmm1
0x00001a53    movss dword [acc], xmm0
0x00001a58    add dword [k], 1
0x00001a5c    jmp 0x19f4

```

gfortran-generated binary

[illegible]

```

graph TD
    Top["0x000019aa cmp dword [i], ecx  
0x000019ad setg al  
0x000019b1 movzx esi, al  
0x000019b5 test esi, esi  
0x000019b7 jne 0x1a90"]
    Bottom["0x000019bd mov esi, qword [di]  
0x000019c1 mov esi, dword [esi]  
0x000019c3 mov dword [j], 1"]
    Top -- "jne 0x1a90" --> Bottom
    Bottom -- "jbe 0x1a90" --> Top
  
```

```

0x000019ca    cmp dword [edi], 0
0x000019cd    setg dil
0x000019d1    movzx edi, dil
0x000019d5    test edi, edi
0x000019d7    jne 0x001a68

0x000019d8    movzx eax, eax
0x000019e1    movsd dword [aecx], eax
0x000019e6    mov rdi, qword [din]
0x000019ea    mov r8d, dword [rdi]
0x000019ed    mov dword [k], 1

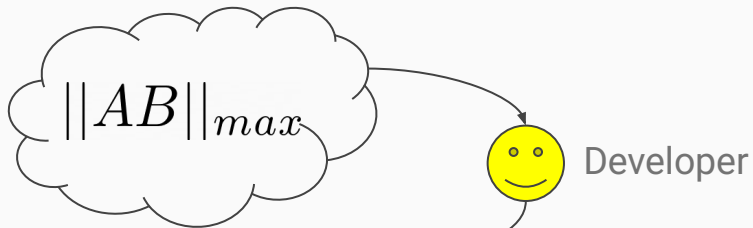
```

```
0x000019f4    cmp dword [k], r8d
0x000019f8    setg dil
0x000019fc    movzx edi, dil
0x00001a00    test edi, edi
0x00001a02    jne 0x1a5e
```

[illegible]

0x000001a6	nop
0x000001a7	add
0x000001a8	jmp

```
0x00001a5e nop
0x00001a5f movss xmm1, dword [result]
0x00001a64 movss xmm0, dword [acc]
0x00001a69 movss xmm2, dword [0x00002180]
0x00001a71 andps xmm0, xmm2
0x00001a74 maxss xmm0, xmm1
0x00001a78 movss dword [result], xmm0
0x00001a7d add dword [i], 1
0x00001a81 jmp 0x19ca
```



```
function mmMaxNorm(dim, A, B)
...
result = 0.0
do i = 1, dim
  do j = 1, dim
    acc = 0.0
    do k = 1, dim
      acc = acc + A(i,k) * B(k,j)
    end do
    result = max(result, abs(acc))
  end do
end do
...
end function mmMaxNorm
```

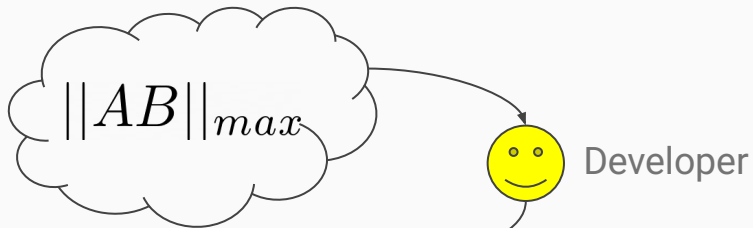
$$A = \begin{bmatrix} 0.0 & 0.0 \\ -3.40282 \times 10^{38} & -2.4245 \end{bmatrix}$$
$$B = \begin{bmatrix} 0.0 & 3.06254 \times 10^{38} \\ 0.0 & -3.08694 \times 10^{38} \end{bmatrix}$$

gfortran

bin1

0.0

INCORRECT



```
function mmMaxNorm(dim, A, B)
...
result = 0.0
do i = 1, dim
  do j = 1, dim
    acc = 0.0
    do k = 1, dim
      acc = acc + A(i,k) * B(k,j)
    end do
    result = max(result, abs(acc))
  end do
end do
...
end function mmMaxNorm
```

$$A = \begin{bmatrix} 0.0 & 0.0 \\ -3.40282 \times 10^{38} & -2.4245 \end{bmatrix}$$

$$B = \begin{bmatrix} 0.0 & 3.06254 \times 10^{38} \\ 0.0 & -3.08694 \times 10^{38} \end{bmatrix}$$

gfortran

bin1

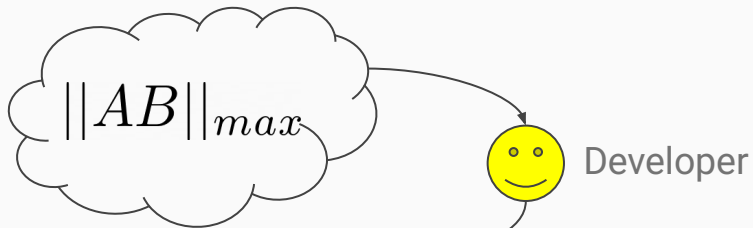
0.0

INCORRECT

gfortran
-fdefault-real-8

bin2

(increase FP precision to 64-bit)



```
function mmMaxNorm(dim, A, B)
...
result = 0.0
do i = 1, dim
  do j = 1, dim
    acc = 0.0
    do k = 1, dim
      acc = acc + A(i,k) * B(k,j)
    end do
    result = max(result, abs(acc))
  end do
end do
...
end function mmMaxNorm
```

$$A = \begin{bmatrix} 0.0 & 0.0 \\ -3.40282 \times 10^{38} & -2.4245 \end{bmatrix}$$

$$B = \begin{bmatrix} 0.0 & 3.06254 \times 10^{38} \\ 0.0 & -3.08694 \times 10^{38} \end{bmatrix}$$

gfortran

bin1

0.0

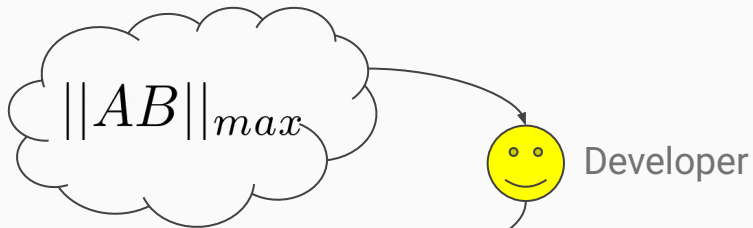
INCORRECT

gfortran
-fdefault-real-8

bin2

1.04212×10^{77}

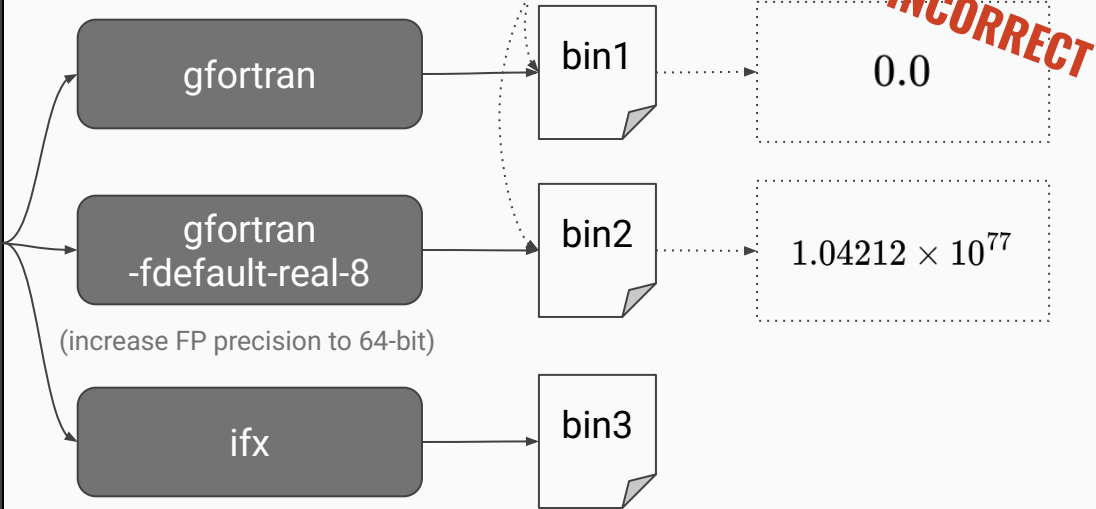
(increase FP precision to 64-bit)

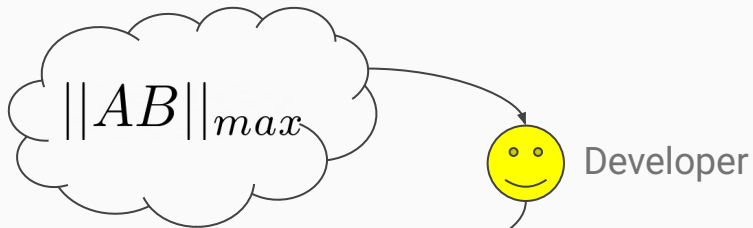


```
function mmMaxNorm(dim, A, B)
...
result = 0.0
do i = 1, dim
  do j = 1, dim
    acc = 0.0
    do k = 1, dim
      acc = acc + A(i,k) * B(k,j)
    end do
    result = max(result, abs(acc))
  end do
end do
...
end function mmMaxNorm
```

$$A = \begin{bmatrix} 0.0 & 0.0 \\ -3.40282 \times 10^{38} & -2.4245 \end{bmatrix}$$

$$B = \begin{bmatrix} 0.0 & 3.06254 \times 10^{38} \\ 0.0 & -3.08694 \times 10^{38} \end{bmatrix}$$

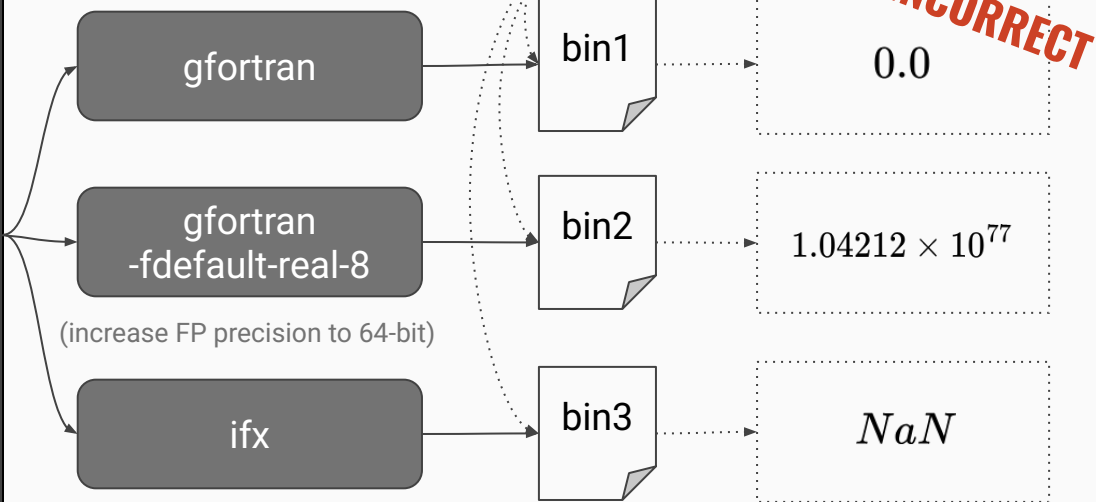




```
function mmMaxNorm(dim, A, B)
...
result = 0.0
do i = 1, dim
  do j = 1, dim
    acc = 0.0
    do k = 1, dim
      acc = acc + A(i,k) * B(k,j)
    end do
    result = max(result, abs(acc))
  end do
end do
...
end function mmMaxNorm
```

$$A = \begin{bmatrix} 0.0 & 0.0 \\ -3.40282 \times 10^{38} & -2.4245 \end{bmatrix}$$

$$B = \begin{bmatrix} 0.0 & 3.06254 \times 10^{38} \\ 0.0 & -3.08694 \times 10^{38} \end{bmatrix}$$



Numerical
code should
be correct

"I want no exceptions!"

"If there is an exception,
I want to know about it!"

gfortran

bin1

0.0

INCORRECT

gfortran
-fdefault-real-8

bin2

1.04212×10^{77}

(increase FP precision to 64-bit)

ifx

bin3

NaN

Numerical
code should
be correct

"I want no exceptions!"

"If there is an exception,
I want to know about it!"

gfortran

bin1

0.0

INCORRECT

gfortran
-fdefault-real-8

bin2

1.04212×10^{77}

CORRECT

(increase FP precision to 64-bit)

ifx

bin3

NaN

Numerical
code should
be correct

"I want no exceptions!"

"If there is an exception,
I want to know about it!"

gfortran

bin1

0.0

INCORRECT

gfortran
-fdefault-real-8

bin2

1.04212×10^{77}

CORRECT

(increase FP precision to 64-bit)

ifx

bin3

NaN

CORRECT

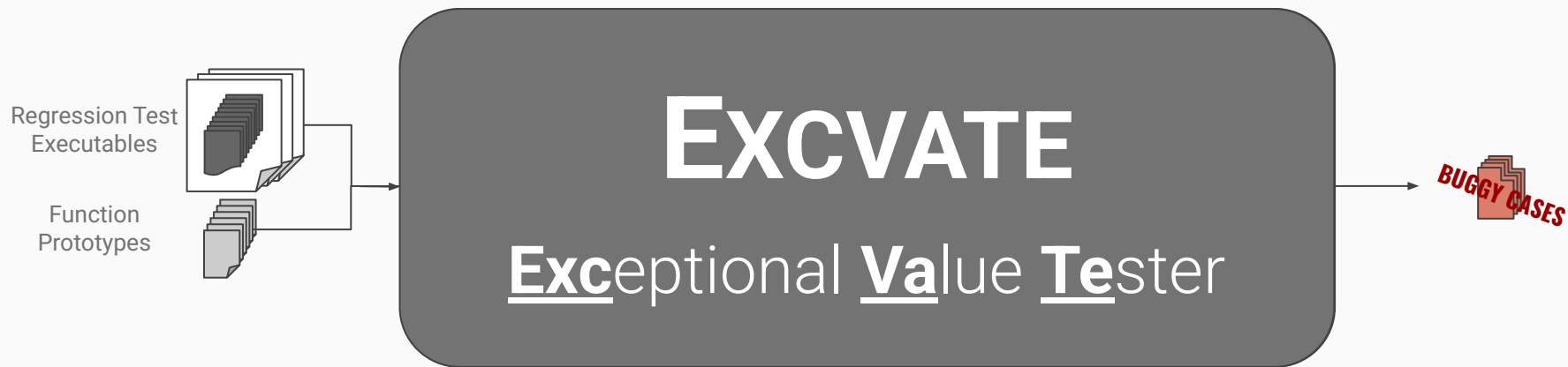
Exception-Handling Specification:

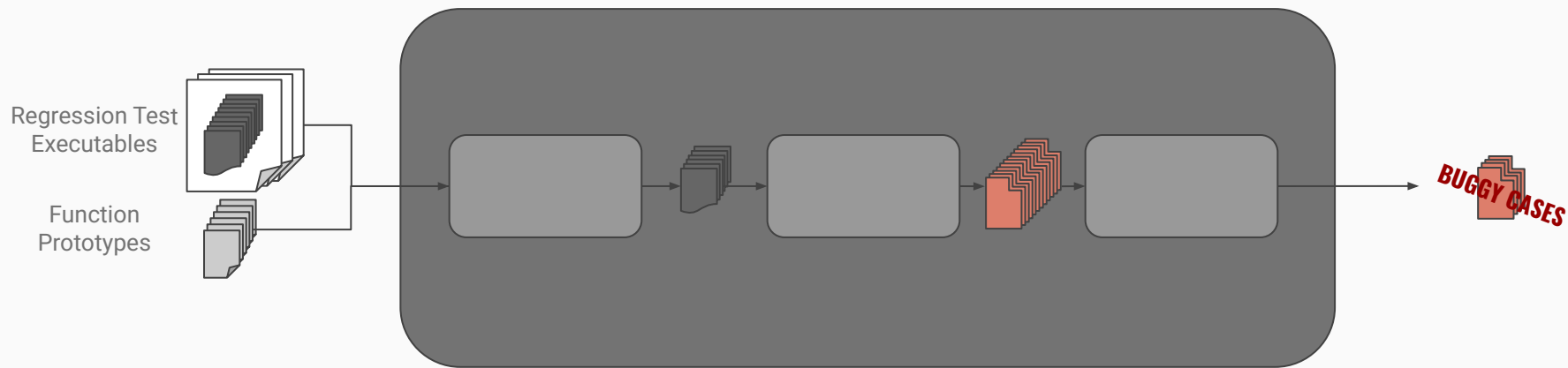
When an exceptional value is input to a routine or created internally, it should ***propagate*** until it reaches the routine output or an appropriate error handler.

Exception-Handling Specification:

When an exceptional value is input to a routine or created internally, it should ***propagate*** until it reaches the routine output or an appropriate error handler.

This is the basis for the specification adopted by the reference implementations of LAPACK/BLAS [1]. **-> How can we test this?**





EXCVATE is composed of three components
that address three key challenges...

Challenge 1: Existing input generation techniques are not well-suited to testing exception handling

Challenge 1: Existing input generation techniques are not well-suited to testing exception handling

Pre-existing works use a variety of input generation techniques to yield floating-point inputs that trigger exceptions.

- e.g., fuzzing [1], symbolic execution [2,3], Bayesian optimization [4]

[1] A. Tran, I. Laguna, G. Gopalakrishnan. "FPBoxer: Efficient Input-Generation for Targeting Floating-Point Exceptions in GPU Programs".

[2] E.T. Barr, T. Vo, V. Le, Z. Su. "Automatic Detection of Floating-Point Exceptions".

[3] X. Wu, L. Li, J. Zhang. "Symbolic Execution with Value-Range Analysis for Floating-Point Exception Detection".

[4] I. Laguna, G. Gopalakrishnan. "Finding Inputs that Trigger Floating-Point Exceptions in GPUs via Bayesian Optimization".

Challenge 1: Existing input generation techniques are not well-suited to testing exception handling

Pre-existing works use a variety of input generation techniques to yield floating-point inputs that trigger exceptions.

- e.g., fuzzing [1], symbolic execution [2,3], Bayesian optimization [4]

They are designed with the “I want no exceptions” mindset; they only seek to trigger exceptions...

[1] A. Tran, I. Laguna, G. Gopalakrishnan. “FPBoxer: Efficient Input-Generation for Targeting Floating-Point Exceptions in GPU Programs”.

[2] E.T. Barr, T. Vo, V. Le, Z. Su. “Automatic Detection of Floating-Point Exceptions”.

[3] X. Wu, L. Li, J. Zhang. “Symbolic Execution with Value-Range Analysis for Floating-Point Exception Detection”.

[4] I. Laguna, G. Gopalakrishnan. “Finding Inputs that Trigger Floating-Point Exceptions in GPUs via Bayesian Optimization”.

Challenge 1: Existing input generation techniques are not well-suited to testing exception handling

Pre-existing works use a variety of input generation techniques to yield floating-point inputs that trigger exceptions.

- e.g., fuzzing [1], symbolic execution [2,3], Bayesian optimization [4]

They are designed with the “I want no exceptions” mindset; they only seek to trigger exceptions... but most code should handle exceptions correctly!

- IEEE754 mandates propagation of exceptional-values in most cases
- Developers often do a good job!

[1] A. Tran, I. Laguna, G. Gopalakrishnan. “FPBoxer: Efficient Input-Generation for Targeting Floating-Point Exceptions in GPU Programs”.

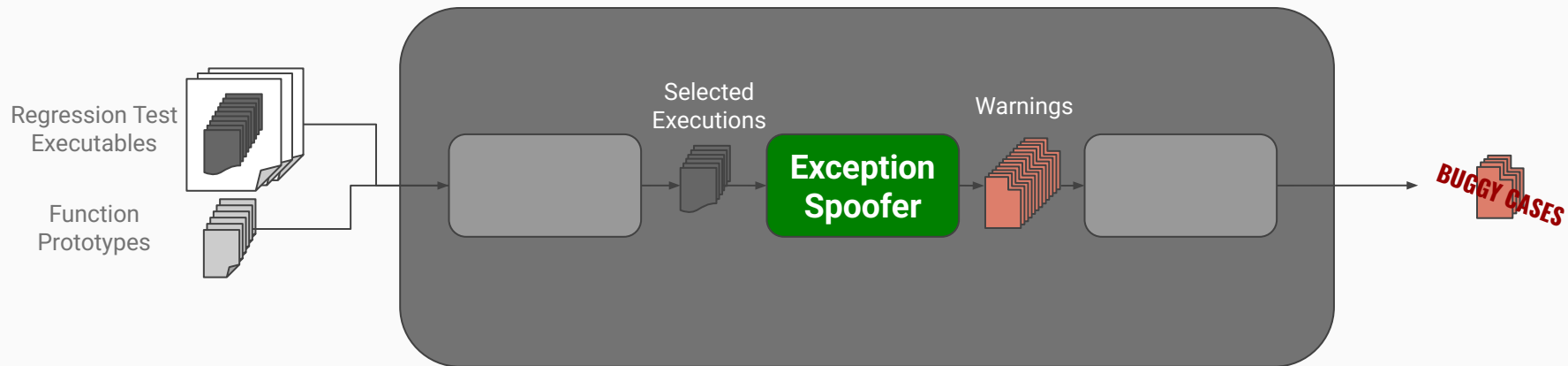
[2] E.T. Barr, T. Vo, V. Le, Z. Su. “Automatic Detection of Floating-Point Exceptions”.

[3] X. Wu, L. Li, J. Zhang. “Symbolic Execution with Value-Range Analysis for Floating-Point Exception Detection”.

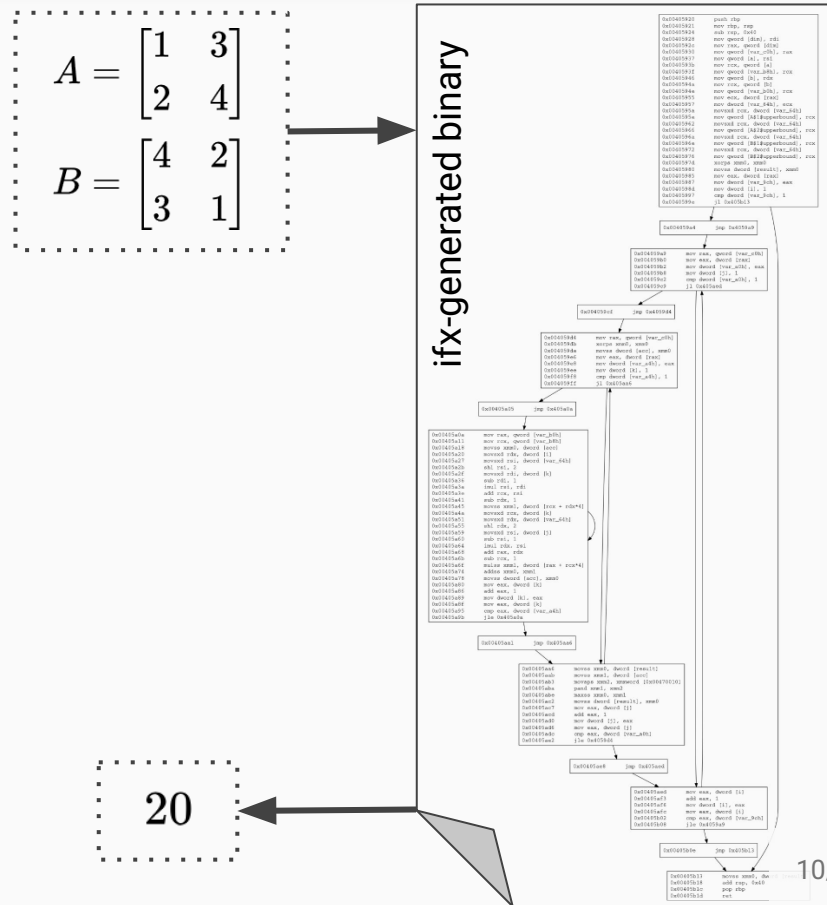
[4] I. Laguna, G. Gopalakrishnan. “Finding Inputs that Trigger Floating-Point Exceptions in GPUs via Bayesian Optimization”.

Challenge 1: Existing input generation techniques are not well-suited to testing exception handling

Approach: spoof exceptions via binary rewriting to cheaply find handling failures



Approach: spoof exceptions via binary rewriting to cheaply find handling failures



Approach: spoof exceptions via binary rewriting to cheaply find handling failures

There are only two instructions in the ifx binary that could encounter “Invalid” exceptions.

$$A = \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix}$$
$$B = \begin{bmatrix} 4 & 2 \\ 3 & 1 \end{bmatrix}$$

```

:
sub rsi, 1
imul rdx, rsi
add rax, rdx
sub rcx, 1
mulss xmm1, dword [rax + rcx*4]
addss xmm0, xmm1
movss dword [rbp-0x98], xmm0
mov eax, dword [rbp-0x8c]
add eax, 1
mov dword [rbp-0x8c], eax
mov eax, dword [rbp-0x8c]
cmp eax, dword [rbp-0xa4]
jle 0x405a0a

```

20

ifx-generated binary

```

0x00459320      push rbp
0x00459321      mov rbp, rbp
0x00459324      mov rbp, 5600
0x00459325      mov qword [rbp+0], 0
0x00459326      mov rax, qword [rdi]
0x00459330      mov qword [rax+0], rax
0x00459331      mov qword [rax+8], rax
0x00459332      mov qword [rax+16], rax
0x00459333      mov rax, qword [rax+24]
0x00459334      mov qword [rax+0], rax
0x00459335      mov rax, dword [rax+4]
0x00459336      mov rax, qword [rax+8]
0x00459337      mov rax, qword [rax+16]
0x00459338      mov rax, qword [rax+24]
0x00459339      mov rax, qword [rax+32]
0x0045933A      mov rax, qword [rax+40]
0x0045933B      mov rax, qword [rax+48]
0x0045933C      mov rax, qword [rax+56]
0x0045933D      mov rax, qword [rax+64]
0x0045933E      mov rax, qword [rax+72]
0x0045933F      mov rax, qword [rax+80]
0x00459340      mov rax, qword [rax+88]
0x00459341      mov rax, qword [rax+96]
0x00459342      mov rax, qword [rax+104]
0x00459343      mov rax, qword [rax+112]
0x00459344      mov rax, qword [rax+120]
0x00459345      mov rax, qword [rax+128]
0x00459346      mov rax, qword [rax+136]
0x00459347      mov rax, qword [rax+144]
0x00459348      mov rax, qword [rax+152]
0x00459349      mov rax, qword [rax+160]
0x0045934A      mov rax, qword [rax+168]
0x0045934B      mov rax, qword [rax+176]
0x0045934C      mov rax, qword [rax+184]
0x0045934D      mov rax, qword [rax+192]
0x0045934E      mov rax, qword [rax+200]
0x0045934F      mov rax, qword [rax+208]
0x00459350      mov rax, qword [rax+216]
0x00459351      mov rax, qword [rax+224]
0x00459352      mov rax, qword [rax+232]
0x00459353      mov rax, qword [rax+240]
0x00459354      mov rax, qword [rax+248]
0x00459355      mov rax, qword [rax+256]
0x00459356      mov rax, qword [rax+264]
0x00459357      mov rax, qword [rax+272]
0x00459358      mov rax, qword [rax+280]
0x00459359      mov rax, qword [rax+288]
0x0045935A      mov rax, qword [rax+296]
0x0045935B      mov rax, qword [rax+304]
0x0045935C      mov rax, qword [rax+312]
0x0045935D      mov rax, qword [rax+320]
0x0045935E      mov rax, qword [rax+328]
0x0045935F      mov rax, qword [rax+336]
0x00459360      mov rax, qword [rax+344]
0x00459361      mov rax, qword [rax+352]
0x00459362      mov rax, qword [rax+360]
0x00459363      mov rax, qword [rax+368]
0x00459364      mov rax, qword [rax+376]
0x00459365      mov rax, qword [rax+384]
0x00459366      mov rax, qword [rax+392]
0x00459367      mov rax, qword [rax+400]
0x00459368      mov rax, qword [rax+408]
0x00459369      mov rax, qword [rax+416]
0x0045936A      mov rax, qword [rax+424]
0x0045936B      mov rax, qword [rax+432]
0x0045936C      mov rax, qword [rax+440]
0x0045936D      mov rax, qword [rax+448]
0x0045936E      mov rax, qword [rax+456]
0x0045936F      mov rax, qword [rax+464]
0x00459370      mov rax, qword [rax+472]
0x00459371      mov rax, qword [rax+480]
0x00459372      mov rax, qword [rax+488]
0x00459373      mov rax, qword [rax+496]
0x00459374      mov rax, qword [rax+504]
0x00459375      mov rax, qword [rax+512]
0x00459376      mov rax, qword [rax+520]
0x00459377      mov rax, qword [rax+528]
0x00459378      mov rax, qword [rax+536]
0x00459379      mov rax, qword [rax+544]
0x0045937A      mov rax, qword [rax+552]
0x0045937B      mov rax, qword [rax+560]
0x0045937C      mov rax, qword [rax+568]
0x0045937D      mov rax, qword [rax+576]
0x0045937E      mov rax, qword [rax+584]
0x0045937F      mov rax, qword [rax+592]
0x00459380      mov rax, qword [rax+600]
0x00459381      mov rax, qword [rax+608]
0x00459382      mov rax, qword [rax+616]
0x00459383      mov rax, qword [rax+624]
0x00459384      mov rax, qword [rax+632]
0x00459385      mov rax, qword [rax+640]
0x00459386      mov rax, qword [rax+648]
0x00459387      mov rax, qword [rax+656]
0x00459388      mov rax, qword [rax+664]
0x00459389      mov rax, qword [rax+672]
0x0045938A      mov rax, qword [rax+680]
0x0045938B      mov rax, qword [rax+688]
0x0045938C      mov rax, qword [rax+696]
0x0045938D      mov rax, qword [rax+704]
0x0045938E      mov rax, qword [rax+712]
0x0045938F      mov rax, qword [rax+720]
0x00459390      mov rax, qword [rax+728]
0x00459391      mov rax, qword [rax+736]
0x00459392      mov rax, qword [rax+744]
0x00459393      mov rax, qword [rax+752]
0x00459394      mov rax, qword [rax+760]
0x00459395      mov rax, qword [rax+768]
0x00459396      mov rax, qword [rax+776]
0x00459397      mov rax, qword [rax+784]
0x00459398      mov rax, qword [rax+792]
0x00459399      mov rax, qword [rax+800]
0x0045939A      mov rax, qword [rax+808]
0x0045939B      mov rax, qword [rax+816]
0x0045939C      mov rax, qword [rax+824]
0x0045939D      mov rax, qword [rax+832]
0x0045939E      mov rax, qword [rax+840]
0x0045939F      mov rax, qword [rax+848]
0x004593A0      mov rax, qword [rax+856]
0x004593A1      mov rax, qword [rax+864]
0x004593A2      mov rax, qword [rax+872]
0x004593A3      mov rax, qword [rax+880]
0x004593A4      mov rax, qword [rax+888]
0x004593A5      mov rax, qword [rax+896]
0x004593A6      mov rax, qword [rax+904]
0x004593A7      mov rax, qword [rax+912]
0x004593A8      mov rax, qword [rax+920]
0x004593A9      mov rax, qword [rax+928]
0x004593AA      mov rax, qword [rax+936]
0x004593AB      mov rax, qword [rax+944]
0x004593AC      mov rax, qword [rax+952]
0x004593AD      mov rax, qword [rax+960]
0x004593AE      mov rax, qword [rax+968]
0x004593AF      mov rax, qword [rax+976]
0x004593B0      mov rax, qword [rax+984]
0x004593B1      mov rax, qword [rax+992]
0x004593B2      mov rax, qword [rax+1000]
0x004593B3      mov rax, qword [rax+1008]
0x004593B4      mov rax, qword [rax+1016]
0x004593B5      mov rax, qword [rax+1024]
0x004593B6      mov rax, qword [rax+1032]
0x004593B7      mov rax, qword [rax+1040]
0x004593B8      mov rax, qword [rax+1048]
0x004593B9      mov rax, qword [rax+1056]
0x004593BA      mov rax, qword [rax+1064]
0x004593BB      mov rax, qword [rax+1072]
0x004593BC      mov rax, qword [rax+1080]
0x004593BD      mov rax, qword [rax+1088]
0x004593BE      mov rax, qword [rax+1096]
0x004593BF      mov rax, qword [rax+1104]
0x004593C0      mov rax, qword [rax+1112]
0x004593C1      mov rax, qword [rax+1120]
0x004593C2      mov rax, qword [rax+1128]
0x004593C3      mov rax, qword [rax+1136]
0x004593C4      mov rax, qword [rax+1144]
0x004593C5      mov rax, qword [rax+1152]
0x004593C6      mov rax, qword [rax+1160]
0x004593C7      mov rax, qword [rax+1168]
0x004593C8      mov rax, qword [rax+1176]
0x004593C9      mov rax, qword [rax+1184]
0x004593CA      mov rax, qword [rax+1192]
0x004593CB      mov rax, qword [rax+1200]
0x004593CC      mov rax, qword [rax+1208]
0x004593CD      mov rax, qword [rax+1216]
0x004593CE      mov rax, qword [rax+1224]
0x004593CF      mov rax, qword [rax+1232]
0x004593D0
```

```

0x00401004: jmp 0x00401009
0x00401009: mov eax, speed [var_0006]
0x0040100C: mov eax, dword [rax]
0x0040100E: mov dword [var_0006], eax
0x00401010: mov dword [i], 1
0x00401012: cmp dword [var_0006], 1
0x00401015: jz 0x00401020

```

```

0x0040030E      jmp 0x000944

```

```

01 mov eax, dword [var_n0h]
02 scasd eax, word
03 movzx dword [eax], word
04 mov eax, dword [eax]
05 mov dword [var_n0h], eax
06 mov dword [i], 1
07 cmp dword [var_n0h], 1
08 jf 0x0040030E

```

[illegible]

```

graph TD
    A[0x004000a1] --> B[0x004000a5]
    B --> C[0x004000a6]
    B --> D[0x004000a7]
    B --> E[0x004000a8]
    B --> F[0x004000a9]
    B --> G[0x004000aa]
    B --> H[0x004000ab]
    B --> I[0x004000ac]
    B --> J[0x004000ad]
    B --> K[0x004000ae]
    B --> L[0x004000af]
    B --> M[0x004000b0]
    B --> N[0x004000b1]
    B --> O[0x004000b2]
    B --> P[0x004000b3]
    B --> Q[0x004000b4]
    B --> R[0x004000b5]
    B --> S[0x004000b6]
    B --> T[0x004000b7]
    B --> U[0x004000b8]
    B --> V[0x004000b9]
    B --> W[0x004000ba]
    B --> X[0x004000bb]
    B --> Y[0x004000bc]
    B --> Z[0x004000bd]
    B --> AA[0x004000be]
    B --> AB[0x004000bf]
    B --> AC[0x004000c0]
    B --> AD[0x004000c1]
    B --> AE[0x004000c2]
    B --> AF[0x004000c3]
    B --> AG[0x004000c4]
    B --> AH[0x004000c5]
    B --> AI[0x004000c6]
    B --> AJ[0x004000c7]
    B --> AK[0x004000c8]
    B --> AL[0x004000c9]
    B --> AM[0x004000ca]
    B --> AN[0x004000cb]
    B --> AO[0x004000cc]
    B --> AP[0x004000cd]
    B --> AQ[0x004000ce]
    B --> AR[0x004000cf]
    B --> AS[0x004000d0]
    B --> AT[0x004000d1]
    B --> AU[0x004000d2]
    B --> AV[0x004000d3]
    B --> AW[0x004000d4]
    B --> AX[0x004000d5]
    B --> AY[0x004000d6]
    B --> AZ[0x004000d7]
    B --> BA[0x004000d8]
    B --> BB[0x004000d9]
    B --> BC[0x004000da]
    B --> BD[0x004000db]
    B --> BE[0x004000dc]
    B --> BF[0x004000dd]
    B --> BG[0x004000de]
    B --> BH[0x004000df]
    B --> BI[0x004000e0]
    B --> BJ[0x004000e1]
    B --> BK[0x004000e2]
    B --> BL[0x004000e3]
    B --> BM[0x004000e4]
    B --> BN[0x004000e5]
    B --> BO[0x004000e6]
    B --> BP[0x004000e7]
    B --> BQ[0x004000e8]
    B --> BR[0x004000e9]
    B --> BS[0x004000ea]
    B --> BT[0x004000eb]
    B --> BU[0x004000ec]
    B --> BV[0x004000ed]
    B --> BW[0x004000ee]
    B --> BX[0x004000ef]
    B --> BY[0x004000f0]
    B --> BZ[0x004000f1]
    B --> C0[0x004000f2]
    B --> C1[0x004000f3]
    B --> C2[0x004000f4]
    B --> C3[0x004000f5]
    B --> C4[0x004000f6]
    B --> C5[0x004000f7]
    B --> C6[0x004000f8]
    B --> C7[0x004000f9]
    B --> C8[0x004000fa]
    B --> C9[0x004000fb]
    B --> CA[0x004000fc]
    B --> CB[0x004000fd]
    B --> CC[0x004000fe]
    B --> CD[0x004000ff]
    B --> CE[0x00400100]
    B --> CF[0x00400101]
    B --> CG[0x00400102]
    B --> CH[0x00400103]
    B --> CI[0x00400104]
    B --> CJ[0x00400105]
    B --> CK[0x00400106]
    B --> CL[0x00400107]
    B --> CM[0x00400108]
    B --> CN[0x00400109]
    B --> CO[0x0040010a]
    B --> CP[0x0040010b]
    B --> CQ[0x0040010c]
    B --> CR[0x0040010d]
    B --> CS[0x0040010e]
    B --> CT[0x0040010f]
    B --> CU[0x00400110]
    B --> CV[0x00400111]
    B --> CW[0x00400112]
    B --> CX[0x00400113]
    B --> CY[0x00400114]
    B --> CZ[0x00400115]
    B --> D0[0x00400116]
    B --> D1[0x00400117]
    B --> D2[0x00400118]
    B --> D3[0x00400119]
    B --> D4[0x0040011a]
    B --> D5[0x0040011b]
    B --> D6[0x0040011c]
    B --> D7[0x0040011d]
    B --> D8[0x0040011e]
    B --> D9[0x0040011f]
    B --> DA[0x00400120]
    B --> DB[0x00400121]
    B --> DC[0x00400122]
    B --> DD[0x00400123]
    B --> DE[0x00400124]
    B --> DF[0x00400125]
    B --> DG[0x00400126]
    B --> DH[0x00400127]
    B --> DI[0x00400128]
    B --> DJ[0x00400129]
    B --> DK[0x0040012a]
    B --> DL[0x0040012b]
    B --> DM[0x0040012c]
    B --> DN[0x0040012d]
    B --> DO[0x0040012e]
    B --> DP[0x0040012f]
    B --> DQ[0x00400130]
    B --> DR[0x00400131]
    B --> DS[0x00400132]
    B --> DT[0x00400133]
    B --> DU[0x00400134]
    B --> DV[0x00400135]
    B --> DW[0x00400136]
    B --> DX[0x00400137]
    B --> DY[0x00400138]
    B --> DZ[0x00
```

```

0x0107a08      jmp 0x07a0d
0x0107a0d      mov     eax,
0x0107a0e      add     eax,
0x0107a0f      mov     dword
0x0107a10      mov     eax,
0x0107a11      cmp     eax,
0x0107a12      jle     0x07

```

```

0x0040100e  jmp 0x00401013
0x00401013  movax  eax, 0
0x00401018  add  eax, 0x0040101c
0x0040101c  pop  ebp

```

Approach: spoof exceptions via binary rewriting to cheaply find handling failures

There are only two instructions in the ifx binary that could encounter “Invalid” exceptions.

These two are in a block that is executed eight times -> $2 \times 8 = 16$ possible exception sites.

$$A = \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix}$$
$$B = \begin{bmatrix} 4 & 2 \\ 3 & 1 \end{bmatrix}$$

```
...  
sub rsi, 1  
imul rdx, rsi  
add rax, rdx  
sub rcx, 1  
mulss xmm1, dword [rax + rcx*4]  
addss xmm0, xmm1  
movss dword [rbp-0x98], xmm0  
mov eax, dword [rbp-0x8c]  
add eax, 1  
mov dword [rbp-0x8c], eax  
mov eax, dword [rbp-0x8c]  
cmp eax, dword [rbp-0xa4]  
jle 0x405a0a
```

ifx-generated binary

20

Approach: spoof exceptions via binary rewriting to cheaply find handling failures

There are only two instructions in the ifx binary that could encounter “Invalid” exceptions.

These two are in a block that is executed eight times -> $2 \times 8 = 16$ possible exception sites.

So, replay the function execution 16 times, **spoofing** a different Invalid exception each time.

$$A = \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix}$$
$$B = \begin{bmatrix} 4 & 2 \\ 3 & 1 \end{bmatrix}$$

```
    :  
    sub rsi, 1  
    imul rdx, rsi  
    add rax, rdx  
    sub rcx, 1  
    mulss xmm1, dword [rax + rcx*4]  
    addss xmm0, xmm1  
    movss dword [rbp-0x98], xmm0  
    mov eax, dword [rbp-0x8c]  
    add eax, 1  
    mov dword [rbp-0x8c], eax  
    mov eax, dword [rbp-0x8c]  
    cmp eax, dword [rbp-0xa4]  
    jle 0x405a0a
```

ifx-generated binary

20

Approach: spoof exceptions via binary rewriting to cheaply find handling failures

There are only two instructions in the ifx binary that could encounter “Invalid” exceptions.

These two are in a block that is executed eight times -> $2 \times 8 = 16$ possible exception sites.

So, replay the function execution 16 times, **spoofing** a different Invalid exception each time.

Table of Test Results

	1	2	3	4	5	6	7	8
mul								
add								

Iteration in which EXCVATE spoofs an exception

$$A = \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix}$$
$$B = \begin{bmatrix} 4 & 2 \\ 3 & 1 \end{bmatrix}$$

```
sub rsi, 1
imul rdx, rsi
add rax, rdx
sub rcx, 1
mulss xmm1, dword [rax + rcx*4]
addss xmm0, xmm1
movss dword [rbp-0x98], xmm0
mov eax, dword [rbp-0x8c]
add eax, 1
mov dword [rbp-0x8c], eax
mov eax, dword [rbp-0x8c]
cmp eax, dword [rbp-0xa4]
jle 0x405a0a
```

ifx-generated binary

20

Approach: spoof exceptions via binary rewriting to cheaply find handling failures

There are only two instructions in the ifx binary that could encounter “Invalid” exceptions.

These two are in a block that is executed eight times $\rightarrow 2 \times 8 = 16$ possible exception sites.

So, replay the function execution 16 times, **spoofing** a different Invalid exception each time.

Table of Test Results

mul	20	20						
add								
	1	2	3	4	5	6	7	8

Iteration in which EXCVATE
spoofs an exception

$$A = \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix}$$
$$B = \begin{bmatrix} 4 & 2 \\ 3 & 1 \end{bmatrix}$$

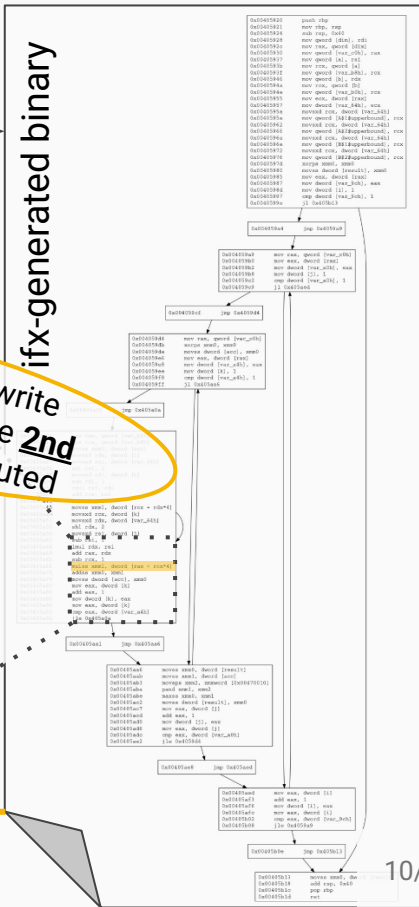
Replay and overwrite xmm1 with NaN the 2nd time mulss is executed

```

sub rsi, 1
imul rdx, rsi
add rax, rdx
sub rcx, 1
mulss xmm1, dword [rax + rcx]
addss xmm0, xmm1
movss dword [rbp-0x98], xmm0
mov eax, dword [rbp-0x8c]
add eax, 1
mov dword [rbp-0x8c], eax
mov eax, dword [rbp-0x8c]
cmp eax, dword [rbp-0xa4]
jle 0x405a0a

```

ifx-generated binary



Approach: spoof exceptions via binary rewriting to cheaply find handling failures

There are only two instructions in the ifx binary that could encounter “Invalid” exceptions.

These two are in a block that is executed eight times $\rightarrow 2 \times 8 = 16$ possible exception sites.

So, replay the function execution 16 times, **spoofing** a different Invalid exception each time.

Table of Test Results

mul	20	20	20					
add								
	1	2	3	4	5	6	7	8

Iteration in which EXCVATE
spoofs an exception

$$A = \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix}$$
$$B = \begin{bmatrix} 4 & 2 \\ 3 & 1 \end{bmatrix}$$

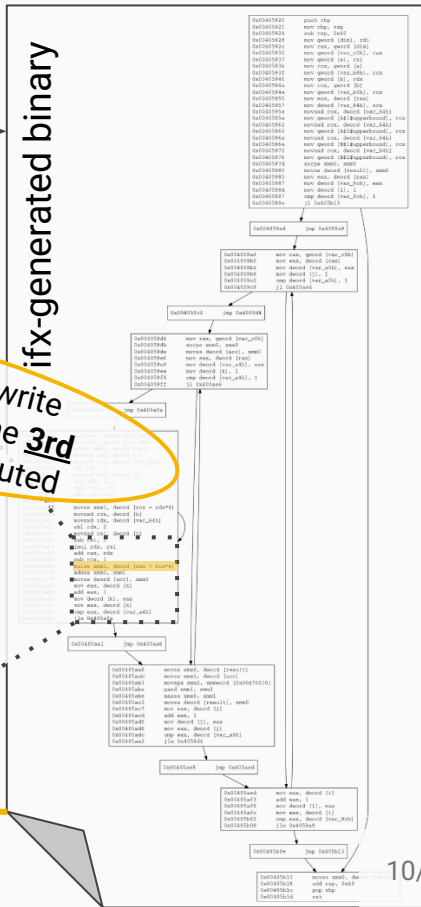
Replay and overwrite xmm1 with NaN the 3rd time mulss is executed

```

:
sub rsi, 1
imul rdx, rsi
add rax, rdx
sub rcx, 1
mulss xmm1, dword [rax + rcx*4]
addss xmm0, xmm1
movss dword [rbp-0x98], xmm0
mov eax, dword [rbp-0x8c]
add eax, 1
mov dword [rbp-0x8c], eax
mov eax, dword [rbp-0x8c]
cmp eax, dword [rbp-0xa4]
jle 0x405a0a

```

ifx-generated binary



Approach: spoof exceptions via binary rewriting to cheaply find handling failures

There are only two instructions in the ifx binary that could encounter “Invalid” exceptions.

These two are in a block that is executed eight times $\rightarrow 2 \times 8 = 16$ possible exception sites.

So, replay the function execution 16 times, **spoofing** a different Invalid exception each time.

Table of Test Results

mul	20	20	20	20				
add								
	1	2	3	4	5	6	7	8

Iteration in which EXCVATE
spoofs an exception

$$A = \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix}$$
$$B = \begin{bmatrix} 4 & 2 \\ 3 & 1 \end{bmatrix}$$

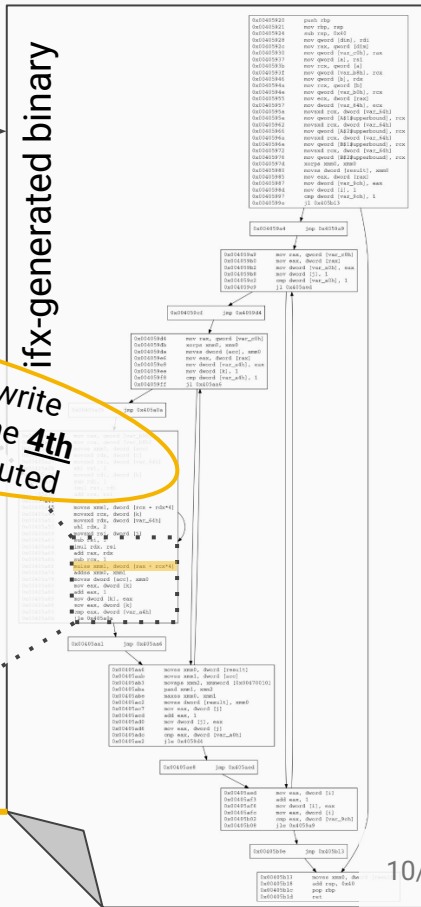
Replay and overwrite
xmm1 with NaN the 4th
time mulss is executed

```

sub rsi, 1
imul rdx, rsi
add rax, rdx
sub rcx, 1
mulss xmm1, dword [rax + rcx]
addss xmm0, xmm1
movss dword [rbp-0x98], xmm0
mov eax, dword [rbp-0x8c]
add eax, 1
mov dword [rbp-0x8c], eax
mov eax, dword [rbp-0x8c]
cmp eax, dword [rbp-0xa4]
jle 0x405a0a

```

ifx-generated binary



Approach: spoof exceptions via binary rewriting to cheaply find handling failures

There are only two instructions in the ifx binary that could encounter “Invalid” exceptions.

These two are in a block that is executed eight times $\rightarrow 2 \times 8 = 16$ possible exception sites.

So, replay the function execution 16 times, **spoofing** a different Invalid exception each time.

Table of Test Results

mul	20	20	20	20	8			
add								
	1	2	3	4	5	6	7	8

Iteration in which EXCVATE
spoofs an exception

$$A = \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix}$$
$$B = \begin{bmatrix} 4 & 2 \\ 3 & 1 \end{bmatrix}$$

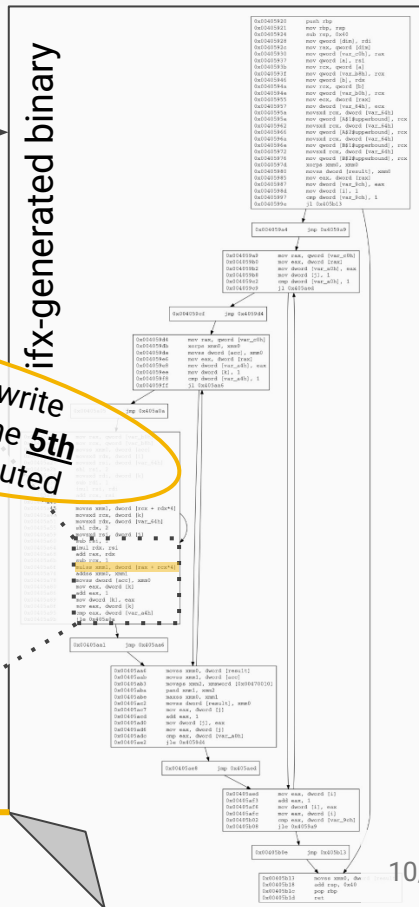
Replay and overwrite
xmm1 with NaN the 5th
time mulss is executed

```

:
sub rsi, 1
imul rdx, rsi
add rax, rdx
sub rcx, 1
mulss xmm1, dword [rax + rcx]
addss xmm0, xmm1
movss dword [rbp-0x98], xmm0
mov eax, dword [rbp-0x8c]
add eax, 1
mov dword [rbp-0x8c], eax
mov eax, dword [rbp-0x8c]
cmp eax, dword [rbp-0xa4]
jle 0x405a0a

```

ifx-generated binary



Approach: spoof exceptions via binary rewriting to cheaply find handling failures

There are only two instructions in the ifx binary that could encounter “Invalid” exceptions.

These two are in a block that is executed eight times $\rightarrow 2 \times 8 = 16$ possible exception sites.

So, replay the function execution 16 times, **spoofing** a different Invalid exception each time.

Table of Test Results

mul	20	20	20	20	8	8		
add								
	1	2	3	4	5	6	7	8

Iteration in which EXCVATE
spoofs an exception

$$A = \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix}$$
$$B = \begin{bmatrix} 4 & 2 \\ 3 & 1 \end{bmatrix}$$

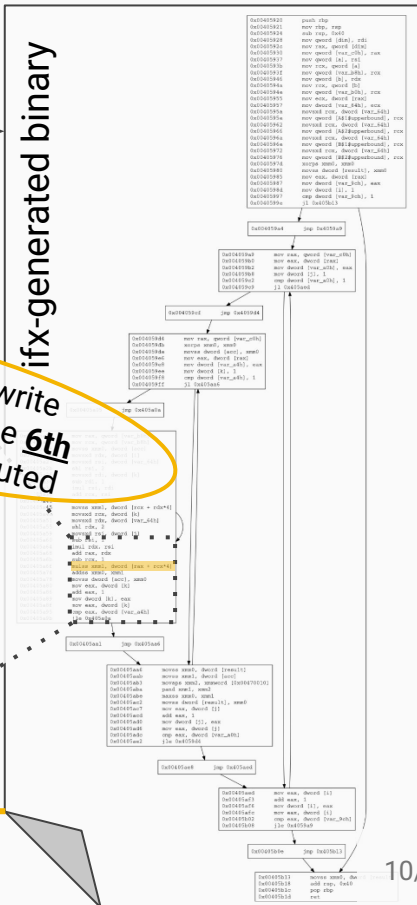
Replay and overwrite xmm1 with NaN the 6th time mulss is executed

```

sub rsi, 1
imul rdx, rsi
add rax, rdx
sub rcx, 1
mulss xmm1, dword [rax + rcx]
addss xmm0, xmm1
movss dword [rbp-0x98], xmm0
mov eax, dword [rbp-0x8c]
add eax, 1
mov dword [rbp-0x8c], eax
mov eax, dword [rbp-0x8c]
cmp eax, dword [rbp-0xa4]
jle 0x405a0a

```

ifx-generated binary



Approach: spoof exceptions via binary rewriting to cheaply find handling failures

There are only two instructions in the ifx binary that could encounter “Invalid” exceptions.

These two are in a block that is executed eight times $\rightarrow 2 \times 8 = 16$ possible exception sites.

So, replay the function execution 16 times, **spoofing** a different Invalid exception each time.

Table of Test Results

mul	20	20	20	20	8	8	NaN	
add								
	1	2	3	4	5	6	7	8

Iteration in which EXCVATE
spoofs an exception

$$A = \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix}$$
$$B = \begin{bmatrix} 4 & 2 \\ 3 & 1 \end{bmatrix}$$

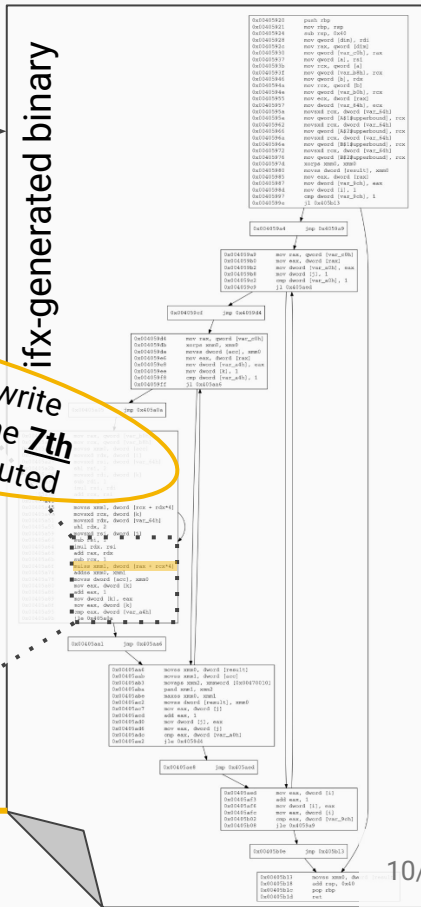
Replay and overwrite xmm1 with NaN the 7th time mulss is executed

```

sub rsi, 1
imul rdx, rsi
add rax, rdx
sub rcx, 1
mulss xmm1, dword [rax + rcx]
addss xmm0, xmm1
movss dword [rbp-0x98], xmm0
mov eax, dword [rbp-0x8c]
add eax, 1
mov dword [rbp-0x8c], eax
mov eax, dword [rbp-0x8c]
cmp eax, dword [rbp-0xa4]
jle 0x405a0a

```

ifx-generated binary



Approach: spoof exceptions via binary rewriting to cheaply find handling failures

There are only two instructions in the ifx binary that could encounter “Invalid” exceptions.

These two are in a block that is executed eight times $\rightarrow 2 \times 8 = 16$ possible exception sites.

So, replay the function execution 16 times, **spoofing** a different Invalid exception each time.

Table of Test Results

mul	20	20	20	20	8	8	NaN	NaN
add								
	1	2	3	4	5	6	7	8

Iteration in which EXCVATE
spoofs an exception

$$A = \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix}$$
$$B = \begin{bmatrix} 4 & 2 \\ 3 & 1 \end{bmatrix}$$

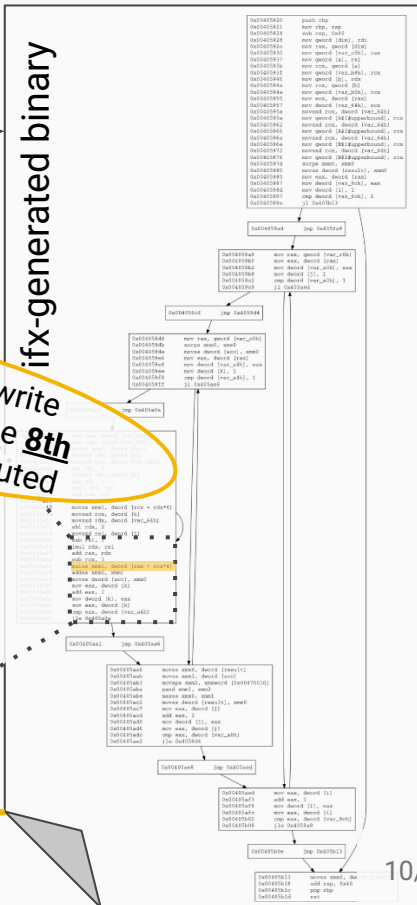
Replay and overwrite
xmm1 with NaN the 8th
time mulss is executed

```

sub rsi, 1
imul rdx, rsi
add rax, rdx
sub rcx, 1
mulss xmm1, dword [rax + rcx]
addss xmm0, xmm1
movss dword [rbp-0x98], xmm0
mov eax, dword [rbp-0x8c]
add eax, 1
mov dword [rbp-0x8c], eax
mov eax, dword [rbp-0x8c]
cmp eax, dword [rbp-0xa4]
jle 0x405a0a

```

ifx-generated binary



Approach: spoof exceptions via binary rewriting to cheaply find handling failures

There are only two instructions in the ifx binary that could encounter “Invalid” exceptions.

These two are in a block that is executed eight times -> $2 \times 8 = 16$ possible exception sites.

So, replay the function execution 16 times, **spoofing** a different Invalid exception each time.

Table of Test Results

mul	20	20	20	20	8	8	NaN	NaN
add								
	1	2	3	4	5	6	7	8

Iteration in which EXCVATE spoofs an exception

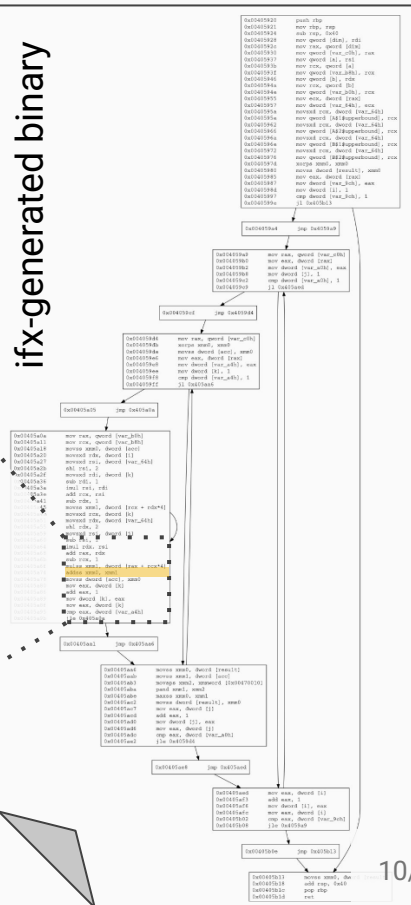
$$A = \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix}$$

$$B = \begin{bmatrix} 4 & 2 \\ 3 & 1 \end{bmatrix}$$

```

:
sub rsi, 1
imul rdx, rsi
add rax, rdx
sub rcx, 1
mulss xmm1, dword [rax + rcx*4]
addss xmm0, xmm1
movss dword [rbp-0x98], xmm0
mov eax, dword [rbp-0x8c]
add eax, 1
mov dword [rbp-0x8c], eax
mov eax, dword [rbp-0xa4]
cmp eax, dword [rbp-0xa4]
jle 0x405a0a
    
```

ifx-generated binary



Approach: spoof exceptions via binary rewriting to cheaply find handling failures

There are only two instructions in the ifx binary that could encounter “Invalid” exceptions.

These two are in a block that is executed eight times -> $2 \times 8 = 16$ possible exception sites.

So, replay the function execution 16 times, **spoofing** a different Invalid exception each time.

Table of Test Results

mul	20	20	20	20	8	8	NaN	NaN
	1	2	3	4	5	6	7	8
add	20	20	20	20	8	8	NaN	NaN
	1	2	3	4	5	6	7	8

Iteration in which EXCVATE spoofs an exception

$$A = \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix}$$
$$B = \begin{bmatrix} 4 & 2 \\ 3 & 1 \end{bmatrix}$$

```
sub rsi, 1
imul rdx, rsi
add rax, rdx
sub rcx, 1
mulss xmm1, dword [rax + rcx*4]
addss xmm0, xmm1
movss dword [rbp-0x98], xmm0
mov eax, dword [rbp-0x8c]
add eax, 1
mov dword [rbp-0x8c], eax
mov eax, dword [rbp-0x8c]
cmp eax, dword [rbp-0xa4]
jle 0x405a0a
```

ifx-generated binary

Challenge 2: How do we “reify” a spoofed exception that results in an exception-handling failure?

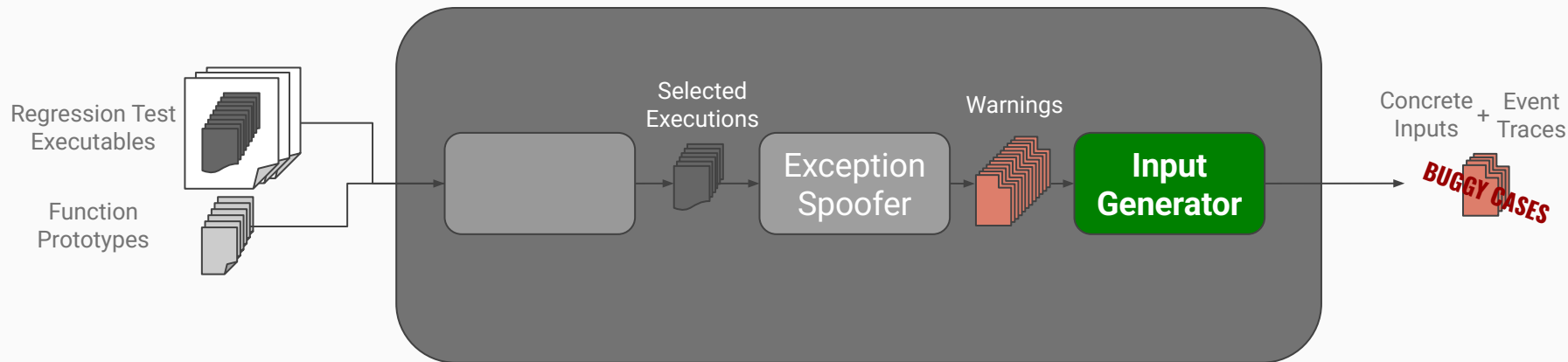
Challenge 2: How do we “reify” a spoofed exception that results in an exception-handling failure?

For a **true buggy case**, EXCVATE must create an input that:

1. triggers the spoofed Invalid exception
2. preserves the control flow that resulted in failed exception handling

Challenge 2: How do we “reify” a spoofed exception that results in an exception-handling failure?

Approach: encode the desired behavior (exception + control flow) into an SMT query

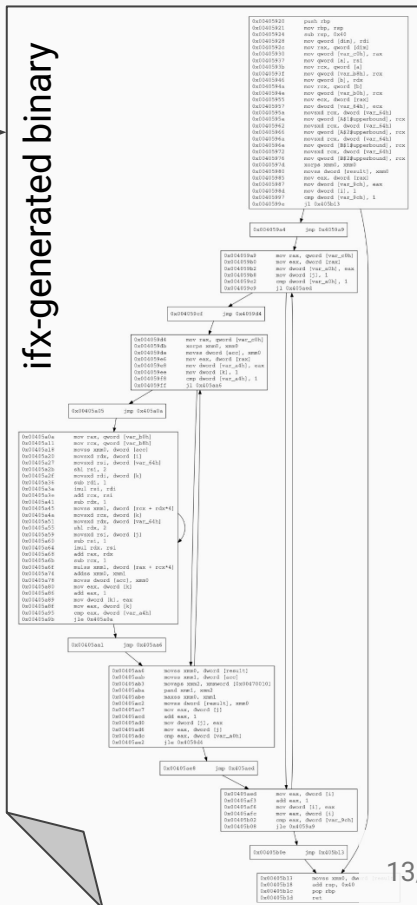


Approach: encode the desired behavior (control flow + exception) into an SMT query

Let's check the *first warning* which resulted from spoofing an Invalid exception in the first execution of the multiply instruction.

$$A = \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix}$$
$$B = \begin{bmatrix} 4 & 2 \\ 3 & 1 \end{bmatrix}$$

ifx-generated binary



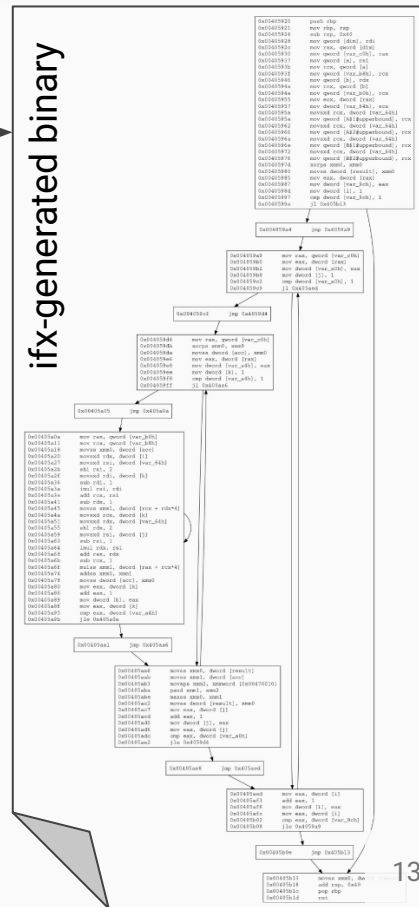
Approach: encode the desired behavior (control flow + exception) into an SMT query

Let's check the *first warning* which resulted from spoofing an Invalid exception in the first execution of the multiply instruction.

$$A = \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix}$$
$$B = \begin{bmatrix} 4 & 2 \\ 3 & 1 \end{bmatrix}$$

$$A_{sym} = \begin{bmatrix} a_0 & a_2 \\ a_1 & a_3 \end{bmatrix}$$

$$B_{sym} = \begin{bmatrix} b_0 & b_2 \\ b_1 & b_3 \end{bmatrix}$$



Approach: encode the desired behavior (control flow + exception) into an SMT query

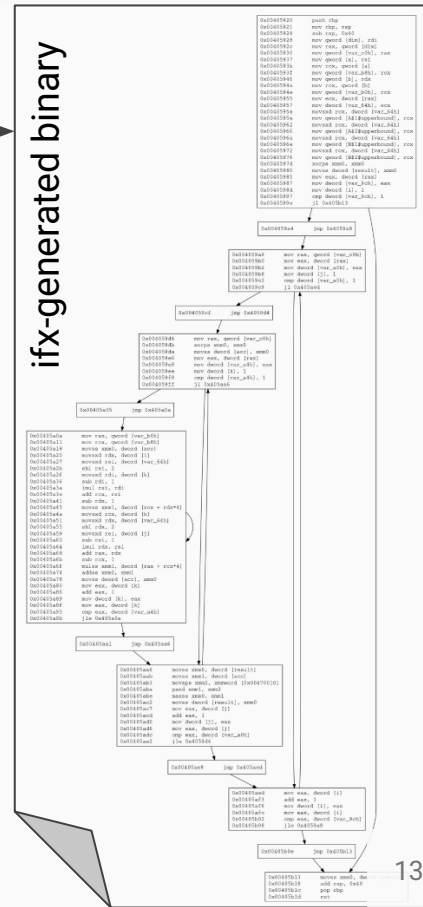
Let's check the *first warning* which resulted from spoofing an Invalid exception in the first execution of the multiply instruction.

$$A = \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix}$$
$$B = \begin{bmatrix} 4 & 2 \\ 3 & 1 \end{bmatrix}$$

$$A_{sym} = \begin{bmatrix} a_0 & a_2 \\ a_1 & a_3 \end{bmatrix}$$
$$B_{sym} = \begin{bmatrix} b_0 & b_2 \\ b_1 & b_3 \end{bmatrix}$$

Active Symbols

```
a_0 : 0x7f9dea0970a4
a_1 : 0x7f9dea0970a8
a_2 : 0x7f9dea0970ac
a_3 : 0x7f9dea0970b0
b_0 : 0x7f9dea0970b4
b_1 : 0x7f9dea0970b8
b_2 : 0x7f9dea0970bc
b_3 : 0x7f9dea0970c0
```



Approach: encode the desired behavior (control flow + exception) into an SMT query

Let's check the *first warning* which resulted from spoofing an Invalid exception in the first execution of the multiply instruction.

```
result = 0.0
do i = 1, dim
  do j = 1, dim
    acc = 0.0
    do k = 1, dim
      acc = acc + A(i,k) * B(k,
    end do
    result = max(result, abs(acc))
  end do
end do
```

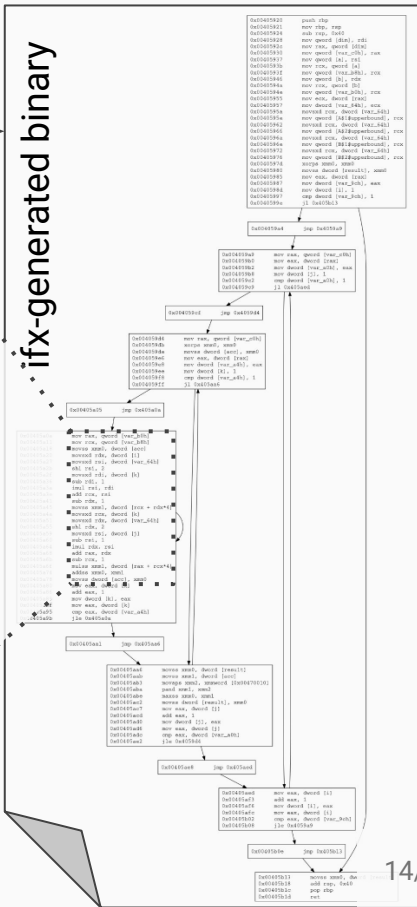
Active Symbols

```
a_0 : 0x7f9dea0970a4
a_1 : 0x7f9dea0970a8
a_2 : 0x7f9dea0970ac
a_3 : 0x7f9dea0970b0
b_0 : 0x7f9dea0970b4
b_1 : 0x7f9dea0970b8
b_2 : 0x7f9dea0970bc
b_3 : 0x7f9dea0970c0
```

$$A = \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix}$$
$$B = \begin{bmatrix} 4 & 2 \\ 3 & 1 \end{bmatrix}$$

```
movss xmm0, dword ptr [rbp-0x98]
:
movss xmm1, dword ptr [rcx+rdx*4]
:
mulss xmm1, dword ptr [rax+rcx*4]
addss xmm0, xmm1
movss dword ptr [rbp-0x98], xmm0
```

fx-generated binary



Approach: encode the desired behavior (control flow + exception) into an SMT query

Let's check the *first warning* which resulted from spoofing an Invalid exception in the first execution of the multiply instruction.

$$A = \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix}$$
$$B = \begin{bmatrix} 4 & 2 \\ 3 & 1 \end{bmatrix}$$

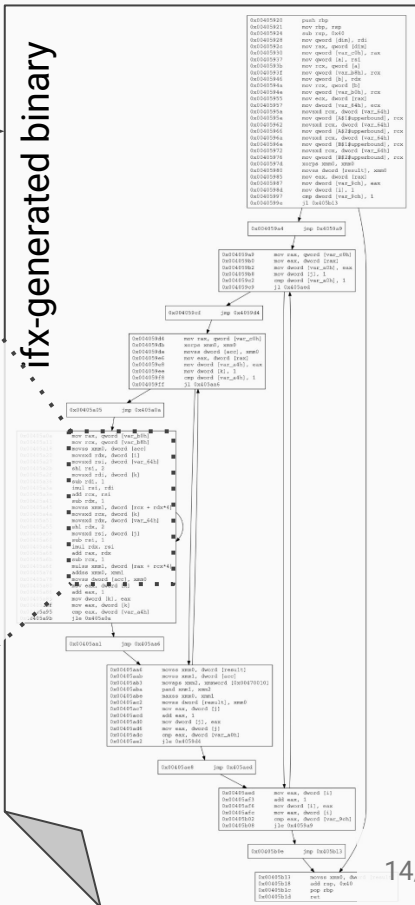
```
result = 0.0
do i = 1, dim
  do j = 1, dim
    acc = 0.0
    do k = 1, dim
      acc = acc + A(i,k) * B(k,j)
    end do
    result = max(result, abs(acc))
  end do
end do
```

Active Symbols

```
a_0 : 0x7f9dea0970a4
a_1 : 0x7f9dea0970a8
a_2 : 0x7f9dea0970ac
a_3 : 0x7f9dea0970b0
b_0 : 0x7f9dea0970b4
b_1 : 0x7f9dea0970b8
b_2 : 0x7f9dea0970bc
b_3 : 0x7f9dea0970c0
```

```
... -> load acc
movss xmm0, dword ptr [rbp-0x98]
...
movss xmm1, dword ptr [rcx+rdx*4]
...
mulss xmm1, dword ptr [rax+rcx*4]
addss xmm0, xmm1
movss dword ptr [rbp-0x98], xmm0
...
```

ifx-generated binary



Approach: encode the desired behavior (control flow + exception) into an SMT query

Let's check the *first warning* which resulted from spoofing an Invalid exception in the first execution of the multiply instruction.

```
result = 0.0
do i = 1, dim
  do j = 1, dim
    acc = 0.0
    do k = 1, dim
      acc = acc + A(i,k) * B(k,
    end do
    result = max(result, abs(acc))
  end do
end do
```

Active Symbols

```
a_0 : 0x7f9dea0970a4
a_1 : 0x7f9dea0970a8
a_2 : 0x7f9dea0970ac
a_3 : 0x7f9dea0970b0
b_0 : 0x7f9dea0970b4
b_1 : 0x7f9dea0970b8
b_2 : 0x7f9dea0970bc
b_3 : 0x7f9dea0970c0
```

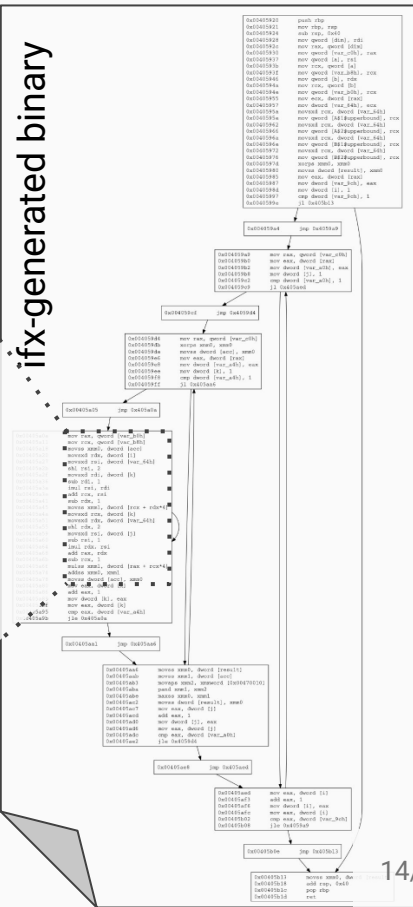
$$A = \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix}$$
$$B = \begin{bmatrix} 4 & 2 \\ 3 & 1 \end{bmatrix}$$

```

:      -> load acc
movss xmm0, dword ptr [rbp-0x98]
:
:      -> load A(i,k)
movss xmm1, dword ptr [rcx+rdx*4]
:
mulss xmm1, dword ptr [rax+rcx*4]
addss xmm0, xmm1
movss dword ptr [rbp-0x98], xmm0
:

```

fx-generated binary



Approach: encode the desired behavior (control flow + exception) into an SMT query

Let's check the *first warning* which resulted from spoofing an Invalid exception in the first execution of the multiply instruction.

$$A = \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix}$$
$$B = \begin{bmatrix} 4 & 2 \\ 3 & 1 \end{bmatrix}$$

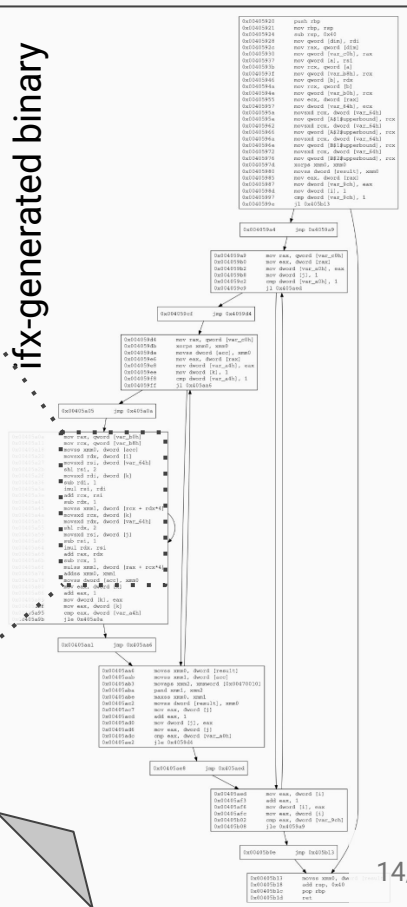
```
result = 0.0
do i = 1, dim
  do j = 1, dim
    acc = 0.0
    do k = 1, dim
      acc = acc + A(i,k) * B(k,j)
    end do
    result = max(result, abs(acc))
  end do
end do
```

Active Symbols

```
a_0 : 0x7f9dea0970a4
a_1 : 0x7f9dea0970a8
a_2 : 0x7f9dea0970ac
a_3 : 0x7f9dea0970b0
b_0 : 0x7f9dea0970b4
b_1 : 0x7f9dea0970b8
b_2 : 0x7f9dea0970bc
b_3 : 0x7f9dea0970c0
```

```
... -> load acc
movss xmm0, dword ptr [rbp-0x98]
...
... -> load A(i,k)
movss xmm1, dword ptr [rcx+rdx*4]
...
... -> multiply with B(k,j)
mulss xmm1, dword ptr [rax+rcx*4]
addss xmm0, xmm1
movss dword ptr [rbp-0x98], xmm0
...
```

ifx-generated binary



Approach: encode the desired behavior (control flow + exception) into an SMT query

Let's check the *first warning* which resulted from spoofing an Invalid exception in the first execution of the multiply instruction.

$$A = \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix}$$
$$B = \begin{bmatrix} 4 & 2 \\ 3 & 1 \end{bmatrix}$$

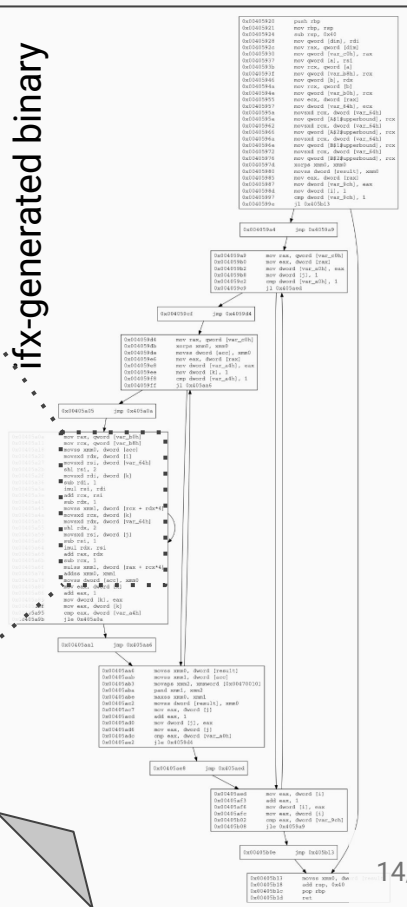
```
result = 0.0
do i = 1, dim
  do j = 1, dim
    acc = 0.0
    do k = 1, dim
      acc = acc + A(i,k) * B(k,j)
    end do
    result = max(result, abs(acc))
  end do
end do
```

Active Symbols

```
a_0 : 0x7f9dea0970a4
a_1 : 0x7f9dea0970a8
a_2 : 0x7f9dea0970ac
a_3 : 0x7f9dea0970b0
b_0 : 0x7f9dea0970b4
b_1 : 0x7f9dea0970b8
b_2 : 0x7f9dea0970bc
b_3 : 0x7f9dea0970c0
```

```
... -> load acc
movss xmm0, dword ptr [rbp-0x98]
...
... -> load A(i,k)
movss xmm1, dword ptr [rcx+rdx*4]
...
... -> multiply with B(k,j)
mulss xmm1, dword ptr [rax+rcx*4]
addss xmm0, xmm1
movss dword ptr [rbp-0x98], xmm0
...
... -> accumulate and save acc
```

ifx-generated binary



Approach: encode the desired behavior (control flow + exception) into an SMT query

Let's check the *first warning* which resulted from spoofing an Invalid exception in the first execution of the multiply instruction.

$$A = \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix}$$
$$B = \begin{bmatrix} 4 & 2 \\ 3 & 1 \end{bmatrix}$$

```

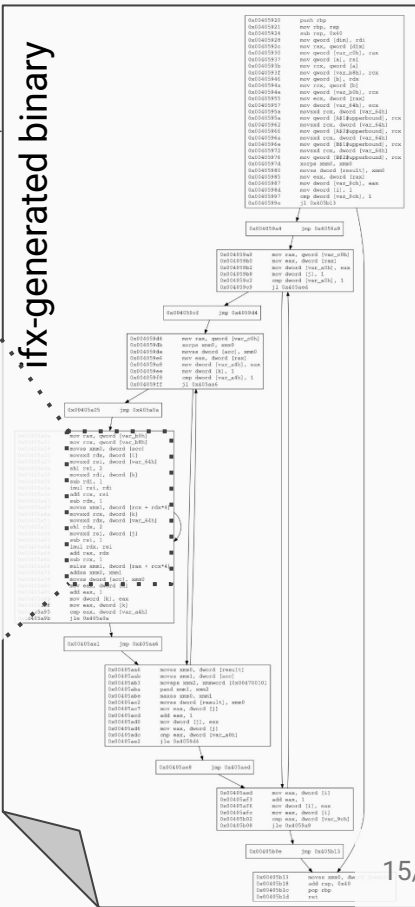
:      -> load acc
movss xmm0, dword ptr [rbp-0x98]
:
:      -> load A(i,k)
movss xmm1, dword ptr [rcx+rdx*4]
:
:      -> multiply with B(k,j)
mulss xmm1, dword ptr [rax+rcx*4]
addss xmm0, xmm1
movss dword ptr [rbp-0x98], xmm0
:      -> accumulate and save acc

```

Active Symbols

```
a_0 : 0x7f9dea0970a4
a_1 : 0x7f9dea0970a8
a_2 : 0x7f9dea0970ac
a_3 : 0x7f9dea0970b0
b_0 : 0x7f9dea0970b4
b_1 : 0x7f9dea0970b8
b_2 : 0x7f9dea0970bc
b_3 : 0x7f9dea0970c0
```

fx-generated binary



Approach: encode the desired behavior (control flow + exception) into an SMT query

Let's check the *first warning* which resulted from spoofing an Invalid exception in the first execution of the multiply instruction.

$$A = \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix}$$
$$B = \begin{bmatrix} 4 & 2 \\ 3 & 1 \end{bmatrix}$$

```
... -> load acc
movss xmm0, dword ptr [rbp-0x98]
...
... -> load A(i,k)
movss xmm1, dword ptr [rcx+rdx*4]
...
... -> multiply with B(k,j)
mulss xmm1, dword ptr [rax+rcx*4]
addss xmm0, xmm1
movss dword ptr [rbp-0x98], xmm0
... -> accumulate and save acc
```

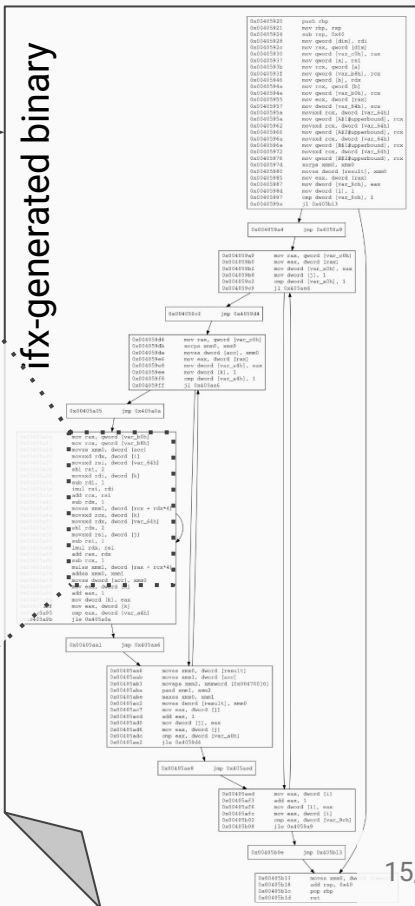
Active Symbols

```
a_0 : 0x7f9dea0970a4
a_1 : 0x7f9dea0970a8
a_2 : 0x7f9dea0970ac
a_3 : 0x7f9dea0970b0
b_0 : 0x7f9dea0970b4
b_1 : 0x7f9dea0970b8
b_2 : 0x7f9dea0970bc
b_3 : 0x7f9dea0970c0
```

Read Operands

0x7ffef9945e28

ifx-generated binary



Approach: encode the desired behavior (control flow + exception) into an SMT query

Let's check the *first warning* which resulted from spoofing an Invalid exception in the first execution of the multiply instruction.

$$A = \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix}$$
$$B = \begin{bmatrix} 4 & 2 \\ 3 & 1 \end{bmatrix}$$

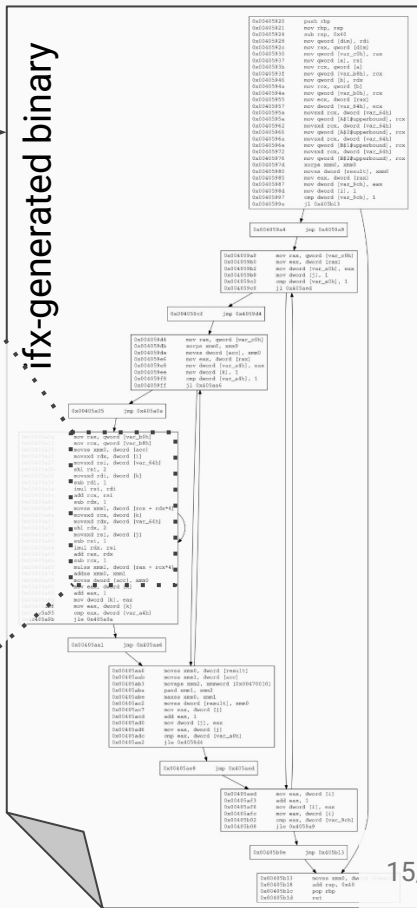
```
... -> load acc
movss xmm0, dword ptr [rbp-0x98]
... -> load A(i,k)
movss xmm1, dword ptr [rcx+rdx*4]
... -> multiply with B(k,j)
mulss xmm1, dword ptr [rax+rcx*4]
addss xmm0, xmm1
movss dword ptr [rbp-0x98], xmm0
... -> accumulate and save acc
```

Active Symbols

```
a_0 : 0x7f9dea0970a4
a_1 : 0x7f9dea0970a8
a_2 : 0x7f9dea0970ac
a_3 : 0x7f9dea0970b0
b_0 : 0x7f9dea0970b4
b_1 : 0x7f9dea0970b8
b_2 : 0x7f9dea0970bc
b_3 : 0x7f9dea0970c0
```

Read Operands

NOT IN ACTIVE SYMBOLS



Approach: encode the desired behavior (control flow + exception) into an SMT query

Let's check the *first warning* which resulted from spoofing an Invalid exception in the first execution of the multiply instruction.

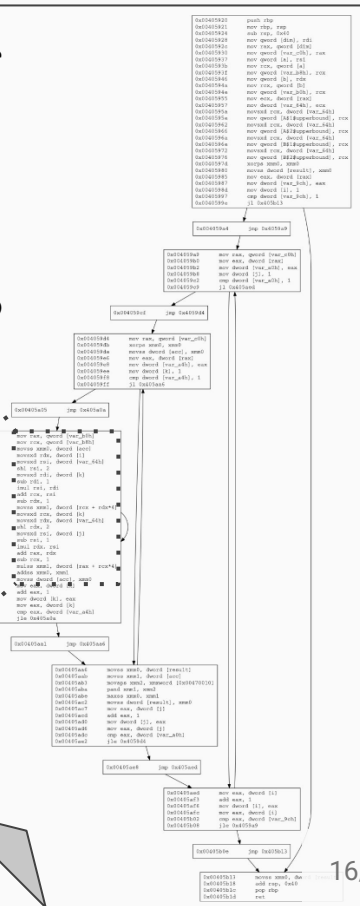
$$A = \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix}$$
$$B = \begin{bmatrix} 4 & 2 \\ 3 & 1 \end{bmatrix}$$

```
... -> load acc
movss xmm0, dword ptr [rbp-0x98]
... -> load A(i,k)
movss xmm1, dword ptr [rcx+rdx*4]
... -> multiply with B(k,j)
mulss xmm1, dword ptr [rax+rcx*4]
addss xmm0, xmm1
movss dword ptr [rbp-0x98], xmm0
... -> accumulate and save acc
```

Active Symbols

```
a_0 : 0x7f9dea0970a4
a_1 : 0x7f9dea0970a8
a_2 : 0x7f9dea0970ac
a_3 : 0x7f9dea0970b0
b_0 : 0x7f9dea0970b4
b_1 : 0x7f9dea0970b8
b_2 : 0x7f9dea0970bc
b_3 : 0x7f9dea0970c0
```

ifx-generated binary



Approach: encode the desired behavior (control flow + exception) into an SMT query

Let's check the *first warning* which resulted from spoofing an Invalid exception in the first execution of the multiply instruction.

$$A = \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix}$$
$$B = \begin{bmatrix} 4 & 2 \\ 3 & 1 \end{bmatrix}$$

```
... -> load acc
movss xmm0, dword ptr [rbp-0x98]
... -> load A(i,k)
movss xmm1, dword ptr [rcx+rdx*4]
... -> multiply with B(k,j)
mulss xmm1, dword ptr [rax+rcx*4]
addss xmm0, xmm1
movss dword ptr [rbp-0x98], xmm0
... -> accumulate and save acc
```

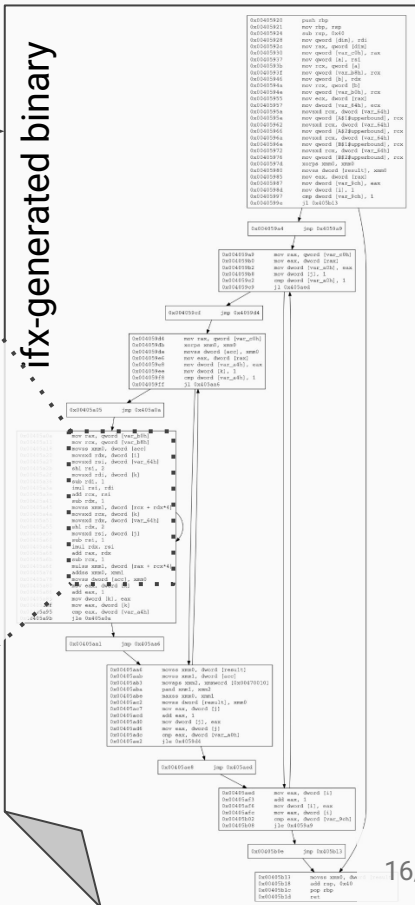
Active Symbols

```
a_0 : 0x7f9dea0970a4
a_1 : 0x7f9dea0970a8
a_2 : 0x7f9dea0970ac
a_3 : 0x7f9dea0970b0
b_0 : 0x7f9dea0970b4
b_1 : 0x7f9dea0970b8
b_2 : 0x7f9dea0970bc
b_3 : 0x7f9dea0970c0
```

Read Operands

0x7f9dea0970a4

ifx-generated binary



Approach: encode the desired behavior (control flow + exception) into an SMT query

Let's check the *first warning* which resulted from spoofing an Invalid exception in the first execution of the multiply instruction.

$$A = \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix}$$
$$B = \begin{bmatrix} 4 & 2 \\ 3 & 1 \end{bmatrix}$$

```

:      -> load acc
movss xmm0, dword ptr [rbp-0x98]

:      -> load A(i,k)
movss xmm1, dword ptr [rcx+rdx*4]

:      -> multiply with B(k,j)
mulss xmm1, dword ptr [rax+rcx*4]
addss xmm0, xmm1
movss dword ptr [rbp-0x98], xmm0
:      -> accumulate and save acc

```

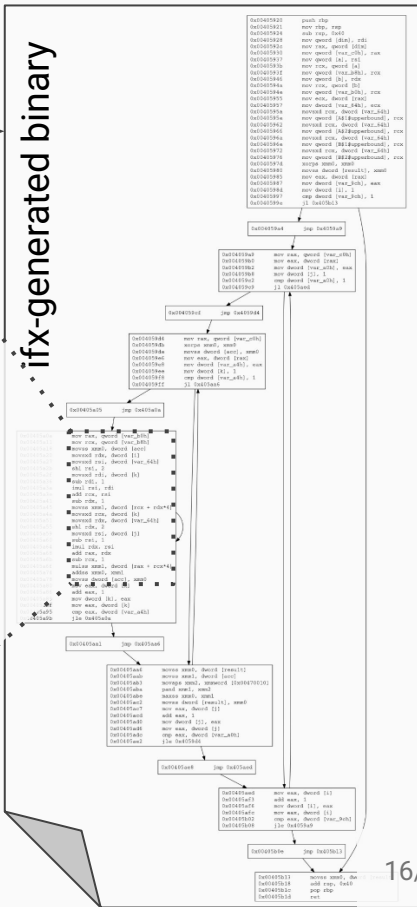
Active Symbols

```
a_0 : 0x7f9dea0970a4
a_1 : 0x7f9dea0970a8
a_2 : 0x7f9dea0970ac
a_3 : 0x7f9dea0970b0
b_0 : 0x7f9dea0970b4
b_1 : 0x7f9dea0970b8
b_2 : 0x7f9dea0970bc
b_3 : 0x7f9dea0970c0
```

Read Operands

0x7f9dea0970a4

ifx-generated binary



Approach: encode the desired behavior (control flow + exception) into an SMT query

Let's check the *first warning* which resulted from spoofing an Invalid exception in the first execution of the multiply instruction.

```
assert ( = xmm1_0 0x7f9dea0970a4_0 )
```

SMT
query

Active Symbols

```
a_0 : 0x7f9dea0970a4
a_1 : 0x7f9dea0970a8
a_2 : 0x7f9dea0970ac
a_3 : 0x7f9dea0970b0
b_0 : 0x7f9dea0970b4
b_1 : 0x7f9dea0970b8
b_2 : 0x7f9dea0970bc
b_3 : 0x7f9dea0970c0
```

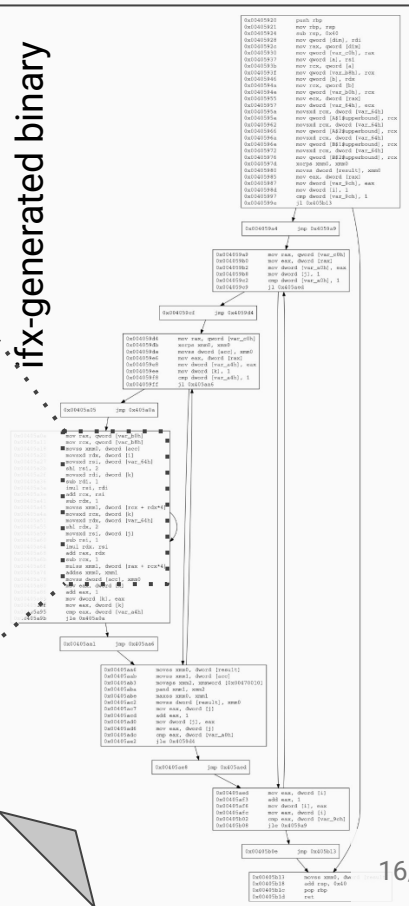
$$A = \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix}$$
$$B = \begin{bmatrix} 4 & 2 \\ 3 & 1 \end{bmatrix}$$

```
... -> load acc
movss xmm0, dword ptr [rbp-0x98]
... -> load A(i,k)
movss xmm1, dword ptr [rcx+rdx*4]
... -> multiply with B(k,j)
mulss xmm1, dword ptr [rax+rcx*4]
addss xmm0, xmm1
movss dword ptr [rbp-0x98], xmm0
... -> accumulate and save acc
```

Read Operands

0x7f9dea0970a4

ifx-generated binary



Approach: encode the desired behavior (control flow + exception) into an SMT query

Let's check the *first warning* which resulted from spoofing an Invalid exception in the first execution of the multiply instruction.

```
assert ( = xmm1_0 0x7f9dea0970a4_0 )
```

SMT
query

Active Symbols

```
a_0 : xmm1
a_0 : 0x7f9dea0970a4
a_1 : 0x7f9dea0970a8
a_2 : 0x7f9dea0970ac
a_3 : 0x7f9dea0970b0
b_0 : 0x7f9dea0970b4
b_1 : 0x7f9dea0970b8
b_2 : 0x7f9dea0970bc
b_3 : 0x7f9dea0970c0
```

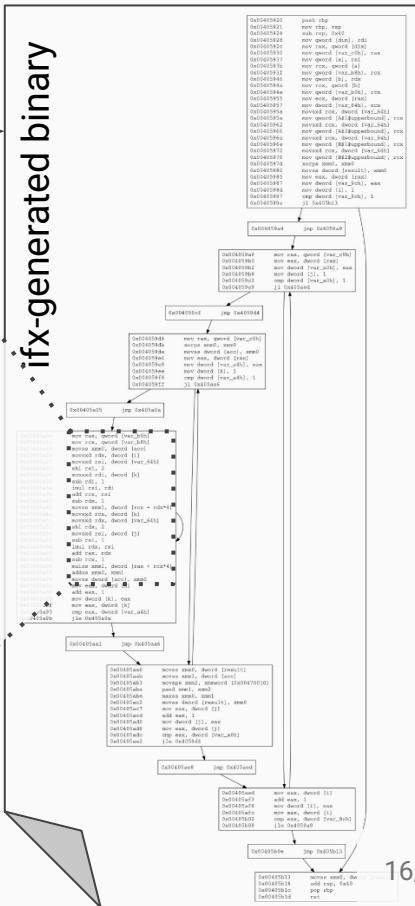
$$A = \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix}$$
$$B = \begin{bmatrix} 4 & 2 \\ 3 & 1 \end{bmatrix}$$

```
... -> load acc
movss xmm0, dword ptr [rbp-0x98]
... -> load A(i,k)
movss xmm1, dword ptr [rcx+rdx*4]
... -> multiply with B(k,j)
mulss xmm1, dword ptr [rax+rcx*4]
addss xmm0, xmm1
movss dword ptr [rbp-0x98], xmm0
... -> accumulate and save acc
```

Read Operands

0x7f9dea0970a4

ifx-generated binary



Approach: encode the desired behavior (control flow + exception) into an SMT query

Let's check the *first warning* which resulted from spoofing an Invalid exception in the first execution of the multiply instruction.

$$A = \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix}$$
$$B = \begin{bmatrix} 4 & 2 \\ 3 & 1 \end{bmatrix}$$

```

:      -> load acc
movss xmm0, dword ptr [rbp-0x98]

:      -> load A(i,k)
movss xmm1, dword ptr [rcx+rdx*4]

:      -> multiply with B(k,j)
mulss xmm1, dword ptr [rax+rcx*4]
addss xmm0, xmm1
movss dword ptr [rbp-0x98], xmm0
:      -> accumulate and save acc

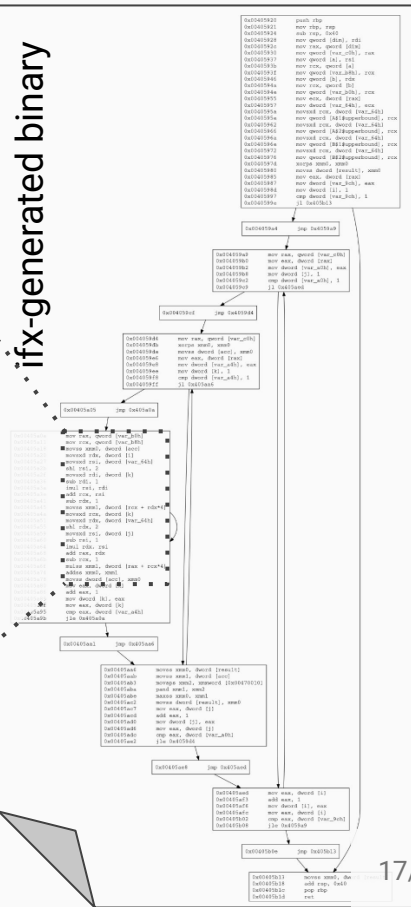
```

SMT
query

Active Symbols

```
a_0 : xmm1
a_0 : 0x7f9dea0970a4
a_1 : 0x7f9dea0970a8
a_2 : 0x7f9dea0970ac
a_3 : 0x7f9dea0970b0
b_0 : 0x7f9dea0970b4
b_1 : 0x7f9dea0970b8
b_2 : 0x7f9dea0970bc
b_3 : 0x7f9dea0970c0
```

ifx-generated binary



Approach: encode the desired behavior (control flow + exception) into an SMT query

Let's check the *first warning* which resulted from spoofing an Invalid exception in the first execution of the multiply instruction.

$$A = \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix}$$
$$B = \begin{bmatrix} 4 & 2 \\ 3 & 1 \end{bmatrix}$$

```
... -> load acc
movss xmm0, dword ptr [rbp-0x98]
... -> load A(i,k)
movss xmm1, dword ptr [rcx+rdx*4]
... -> multiply with B(k,j)
mulss xmm1, dword ptr [rax+rcx*4]
addss xmm0, xmm1
movss dword ptr [rbp-0x98], xmm0
... -> accumulate and save acc
```

SMT
query

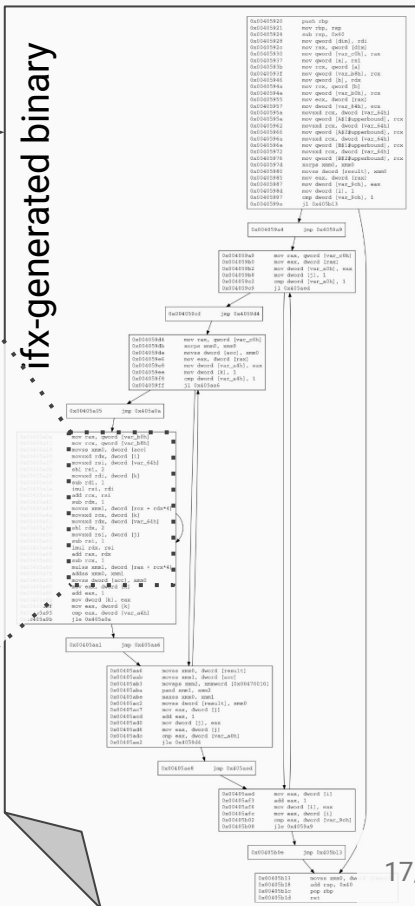
Active Symbols

```
a_0 : xmm1
a_0 : 0x7f9dea0970a4
a_1 : 0x7f9dea0970a8
a_2 : 0x7f9dea0970ac
a_3 : 0x7f9dea0970b0
b_0 : 0x7f9dea0970b4
b_1 : 0x7f9dea0970b8
b_2 : 0x7f9dea0970bc
b_3 : 0x7f9dea0970c0
```

Read Operands

xmm1
0x7f9dea0970b4

ifx-generated binary



Approach: encode the desired behavior (control flow + exception) into an SMT query

Let's check the *first warning* which resulted from spoofing an Invalid exception in the first execution of the multiply instruction.

$$A = \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix}$$
$$B = \begin{bmatrix} 4 & 2 \\ 3 & 1 \end{bmatrix}$$

```
... -> load acc
movss xmm0, dword ptr [rbp-0x98]
... -> load A(i,k)
movss xmm1, dword ptr [rcx+rdx*4]
... -> multiply with B(k,j)
mulss xmm1, dword ptr [rax+rcx*4]
addss xmm0, xmm1
movss dword ptr [rbp-0x98], xmm0
... -> accumulate and save acc
```

SMT
query

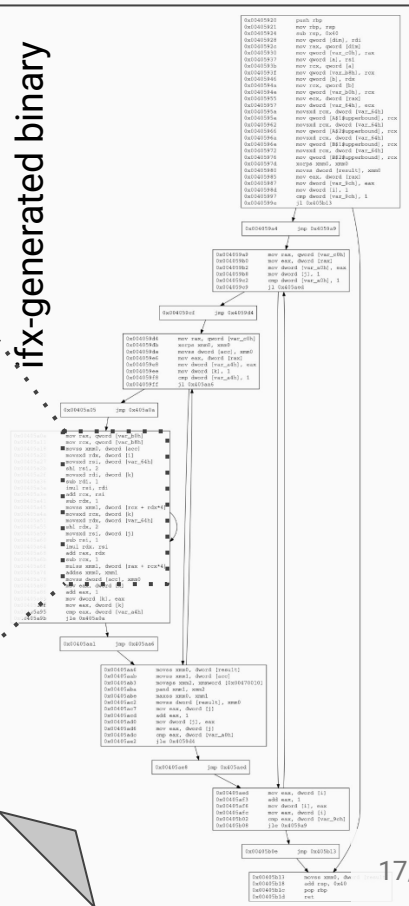
Active Symbols

```
a_0 : xmm1
a_1 : 0x7f9dea0970a4
a_2 : 0x7f9dea0970a8
a_3 : 0x7f9dea0970ac
b_0 : 0x7f9dea0970b4
b_1 : 0x7f9dea0970b8
b_2 : 0x7f9dea0970bc
b_3 : 0x7f9dea0970c0
```

Read Operands

xmm1
0x7f9dea0970b4

ifx-generated binary



Approach: encode the desired behavior (control flow + exception) into an SMT query

Let's check the *first warning* which resulted from spoofing an Invalid exception in the first execution of the multiply instruction.

```
assert ( =  
    xmm1_1  
    ( fp.mul rm  
        xmm1_0  
        0x7f9dea0970b4_0  
    )  
)
```

SMT
query

Active Symbols

```
a_0 : xmm1  
a_1 : 0x7f9dea0970a4  
a_2 : 0x7f9dea0970a8  
a_3 : 0x7f9dea0970ac  
a_4 : 0x7f9dea0970b0  
b_0 : 0x7f9dea0970b4  
b_1 : 0x7f9dea0970b8  
b_2 : 0x7f9dea0970bc  
b_3 : 0x7f9dea0970c0
```

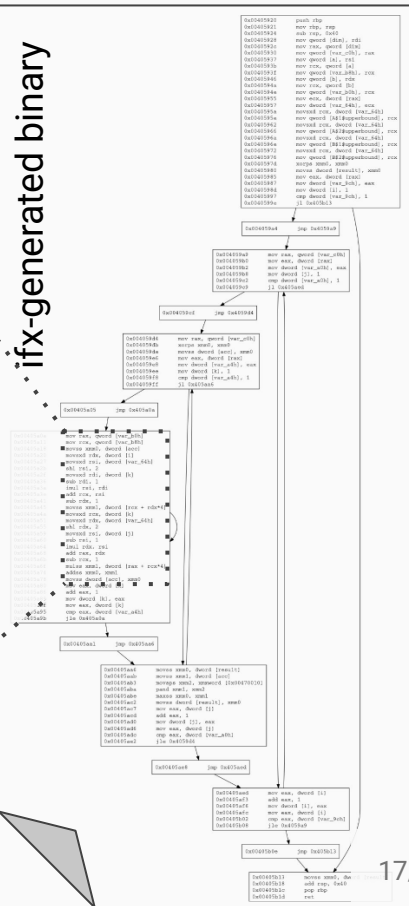
$$A = \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix}$$
$$B = \begin{bmatrix} 4 & 2 \\ 3 & 1 \end{bmatrix}$$

```
...  
-> load acc  
movss xmm0, dword ptr [rbp-0x98]  
...  
-> load A(i,k)  
movss xmm1, dword ptr [rcx+rdx*4]  
...  
-> multiply with B(k,j)  
mulss xmm1, dword ptr [rax+rcx*4]  
addss xmm0, xmm1  
movss dword ptr [rbp-0x98], xmm0  
...  
-> accumulate and save acc
```

Read Operands

xmm1
0x7f9dea0970b4

ifx-generated binary



Approach: encode the desired behavior (control flow + exception) into an SMT query

Let's check the *first warning* which resulted from spoofing an Invalid exception in the first execution of the multiply instruction.

```
assert ( =  
    xmm1_1  
    ( fp.mul rm  
        xmm1_0  
        0x7f9dea0970b4_0  
    )  
    ,  
    )
```

SMT
query

Active Symbols

```
a 0 * b 0 : xmm1  
a_0 : 0x7f9dea0970a4  
a_1 : 0x7f9dea0970a8  
a_2 : 0x7f9dea0970ac  
a_3 : 0x7f9dea0970b0  
b_0 : 0x7f9dea0970b4  
b_1 : 0x7f9dea0970b8  
b_2 : 0x7f9dea0970bc  
b_3 : 0x7f9dea0970c0
```

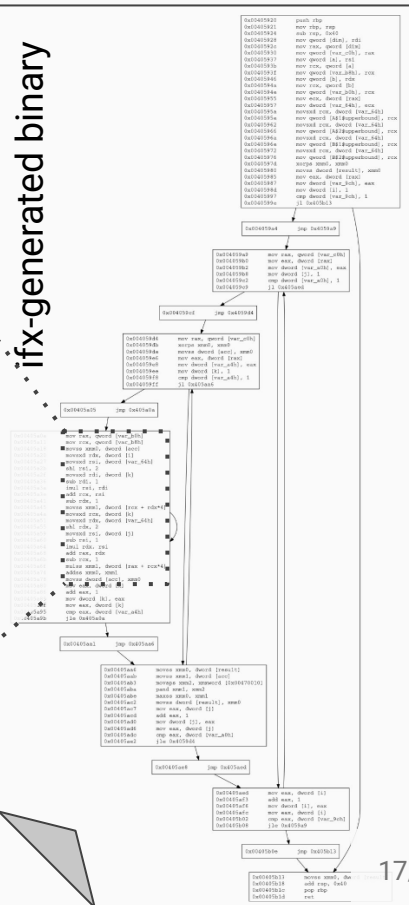
$$A = \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix}$$
$$B = \begin{bmatrix} 4 & 2 \\ 3 & 1 \end{bmatrix}$$

```
... -> load acc  
movss xmm0, dword ptr [rbp-0x98]  
...  
... -> load A(i,k)  
movss xmm1, dword ptr [rcx+rdx*4]  
...  
... -> multiply with B(k,j)  
mulss xmm1, dword ptr [rax+rcx*4]  
addss xmm0, xmm1  
movss dword ptr [rbp-0x98], xmm0  
...  
... -> accumulate and save acc
```

Read Operands

xmm1
0x7f9dea0970b4

ifx-generated binary



Approach: encode the desired behavior (control flow + exception) into an SMT query

Let's check the *first warning* which resulted from spoofing an Invalid exception in the first execution of the multiply instruction.

```
assert ( =  
    xmm1_1  
    ( fp.mul rm  
        xmm1_0  
        0x7f9dea0970b4_0  
    )  
)
```

```
assert ( fp.isNaN xmm1_1 )
```

SMT
query

Active Symbols

a 0 * b 0 : xmm1

a_0 : 0x7f9dea0970a4
a_1 : 0x7f9dea0970a8
a_2 : 0x7f9dea0970ac
a_3 : 0x7f9dea0970b0
b_0 : 0x7f9dea0970b4
b_1 : 0x7f9dea0970b8
b_2 : 0x7f9dea0970bc
b_3 : 0x7f9dea0970c0

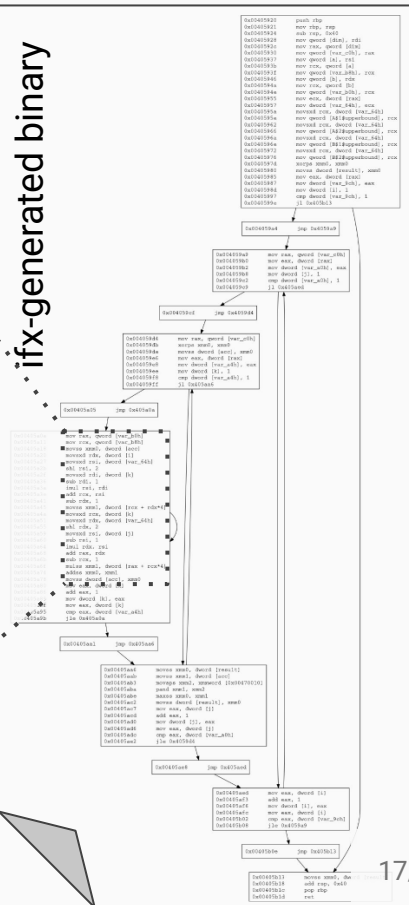
$$A = \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix}$$
$$B = \begin{bmatrix} 4 & 2 \\ 3 & 1 \end{bmatrix}$$

```
... -> load acc  
movss xmm0, dword ptr [rbp-0x98]  
...  
... -> load A(i,k)  
movss xmm1, dword ptr [rcx+rdx*4]  
...  
... -> multiply with B(k,j)  
mulss xmm1, dword ptr [rax+rcx*4]  
addss xmm0, xmm1  
movss dword ptr [rbp-0x98], xmm0  
...  
... -> accumulate and save acc
```

Read Operands

xmm1
0x7f9dea0970b4

ifx-generated binary



Approach: encode the desired behavior (control flow + exception) into an SMT query

Let's check the *first warning* which resulted from spoofing an Invalid exception in the first execution of the multiply instruction.

$$A = \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix}$$
$$B = \begin{bmatrix} 4 & 2 \\ 3 & 1 \end{bmatrix}$$

```
... -> load acc
movss xmm0, dword ptr [rbp-0x98]
... -> load A(i,k)
movss xmm1, dword ptr [rcx+rdx*4]
... -> multiply with B(k,j)
mulss xmm1, dword ptr [rax+rcx*4]
addss xmm0, xmm1
movss dword ptr [rbp-0x98], xmm0
... -> accumulate and save acc
```

SMT
query

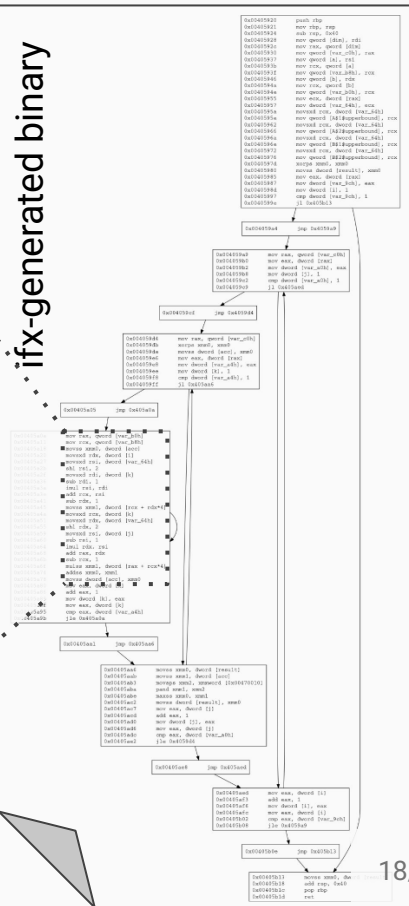
Active Symbols

```
a_0 * b_0 : xmm1
a_0 : 0x7f9dea0970a4
a_1 : 0x7f9dea0970a8
a_2 : 0x7f9dea0970ac
a_3 : 0x7f9dea0970b0
b_0 : 0x7f9dea0970b4
b_1 : 0x7f9dea0970b8
b_2 : 0x7f9dea0970bc
b_3 : 0x7f9dea0970c0
```

Read Operands

xmm1
xmm0

ifx-generated binary



Approach: encode the desired behavior (control flow + exception) into an SMT query

Let's check the *first warning* which resulted from spoofing an Invalid exception in the first execution of the multiply instruction.

```
assert ( =  
    xmm0_0  
    ( fp.add rm  
        #b00000000000000000000000000000000  
        xmm1_1  
    )  
    )
```

SMT
query

Active Symbols

```
a_0 * b_0 : xmm1  
a_0 : 0x7f9dea970a4  
a_1 : 0x7f9dea970a8  
a_2 : 0x7f9dea970ac  
a_3 : 0x7f9dea970b0  
b_0 : 0x7f9dea970b4  
b_1 : 0x7f9dea970b8  
b_2 : 0x7f9dea970bc  
b_3 : 0x7f9dea970c0
```

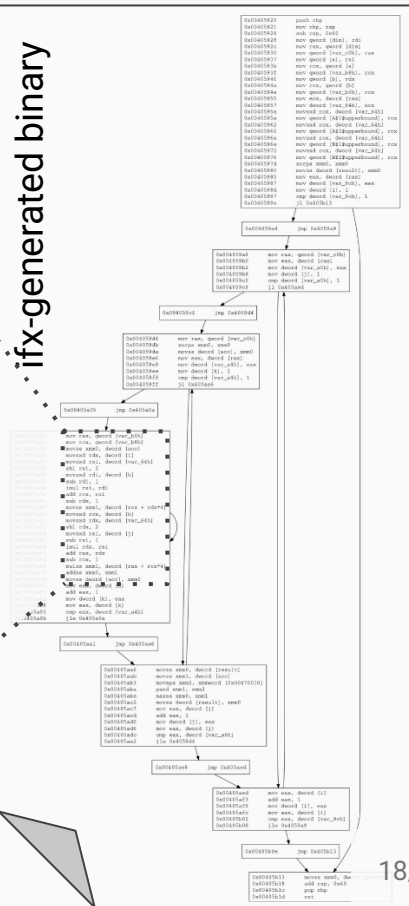
$$A = \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix}$$
$$B = \begin{bmatrix} 4 & 2 \\ 3 & 1 \end{bmatrix}$$

```
... -> load acc  
movss xmm0, dword ptr [rbp-0x98]  
...  
... -> load A(i,k)  
movss xmm1, dword ptr [rcx+rdx*4]  
...  
... -> multiply with B(k,j)  
mulss xmm1, dword ptr [rax+rcx*4]  
addss xmm0, xmm1  
movss dword ptr [rbp-0x98], xmm0  
... -> accumulate and save acc
```

Read Operands

xmm1
xmm0

ifx-generated binary



Approach: encode the desired behavior (control flow + exception) into an SMT query

Let's check the *first warning* which resulted from spoofing an Invalid exception in the first execution of the multiply instruction.

```
assert ( =
    xmm0_0 }
    ( fp.add rm
      #b00000000000000000000000000000000
      xmm1_1
    )
  )
```

SMT
query

Active Symbols

```
0 + a 0 * b 0 : xmm0
  a_0 * b_0 : xmm1
a_0 : 0x7f9dea0970a4
a_1 : 0x7f9dea0970a8
a_2 : 0x7f9dea0970ac
a_3 : 0x7f9dea0970b0
b_0 : 0x7f9dea0970b4
b_1 : 0x7f9dea0970b8
b_2 : 0x7f9dea0970bc
b_3 : 0x7f9dea0970c0
```

$$A = \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix}$$
$$B = \begin{bmatrix} 4 & 2 \\ 3 & 1 \end{bmatrix}$$

```

:      -> load acc
movss xmm0, dword ptr [rbp-0x98]

:      -> load A(i,k)
movss xmm1, dword ptr [rcx+rdx*4]

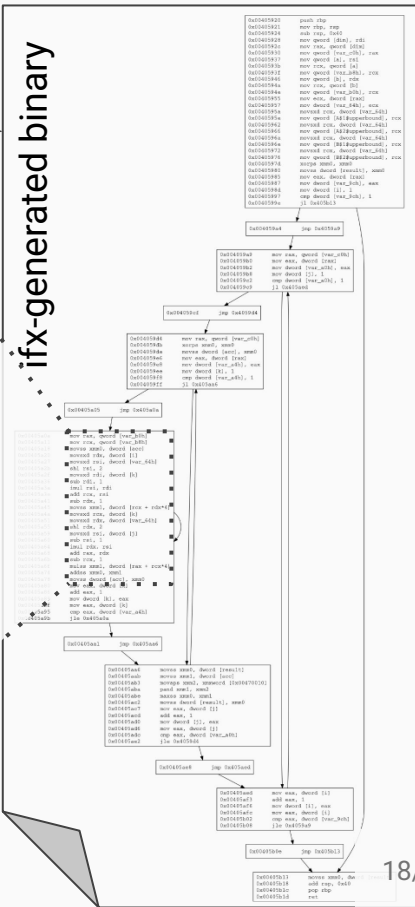
:      -> multiply with B(k,j)
mulss xmm1, dword ptr [rax+rcx*4]
addss xmm0, xmm1
movss dword ptr [rbp-0x98], xmm0
:      -> accumulate and save acc

```

Read Operands

xmm1
xmm0

fx-generated binary



Approach: encode the desired behavior (control flow + exception) into an SMT query

Let's check the *first warning* which resulted from spoofing an Invalid exception in the first execution of the multiply instruction.

$$A = \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix}$$
$$B = \begin{bmatrix} 4 & 2 \\ 3 & 1 \end{bmatrix}$$

```
... -> load acc
movss xmm0, dword ptr [rbp-0x98]
... -> load A(i,k)
movss xmm1, dword ptr [rcx+rdx*4]
... -> multiply with B(k,j)
mulss xmm1, dword ptr [rax+rcx*4]
addss xmm0, xmm1
movss dword ptr [rbp-0x98], xmm0
... -> accumulate and save acc
```

SMT
query

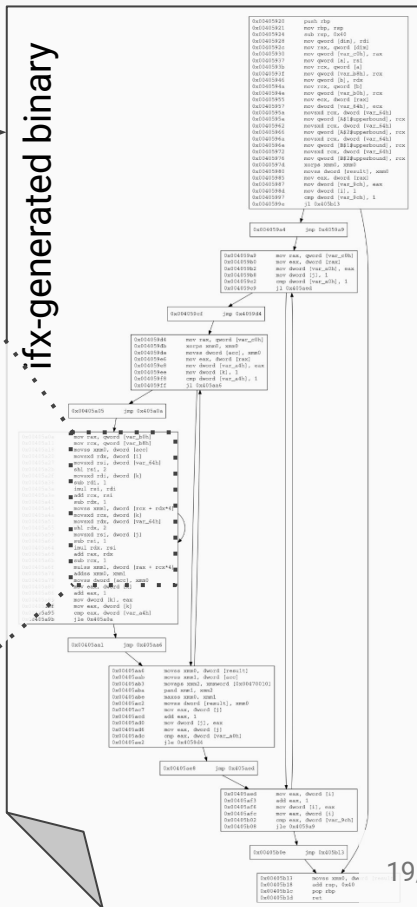
Active Symbols

```
0 + a_0 * b_0 : xmm0
a_0 * b_0 : xmm1
a_0 : 0x7f9dea0970a4
a_1 : 0x7f9dea0970a8
a_2 : 0x7f9dea0970ac
a_3 : 0x7f9dea0970b0
b_0 : 0x7f9dea0970b4
b_1 : 0x7f9dea0970b8
b_2 : 0x7f9dea0970bc
b_3 : 0x7f9dea0970c0
```

Read Operands

xmm0

ifx-generated binary



Approach: encode the desired behavior (control flow + exception) into an SMT query

Let's check the *first warning* which resulted from spoofing an Invalid exception in the first execution of the multiply instruction.

$$A = \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix}$$
$$B = \begin{bmatrix} 4 & 2 \\ 3 & 1 \end{bmatrix}$$

```
assert ( = 0x7ffef9945e28 1 xmm0 0 )
```

SMT
query

Active Symbols

```

:      -> load acc
movss xmm0, dword ptr [rbp-0x98]

:      -> load A(i,k)
movss xmm1, dword ptr [rcx+rdx*4]

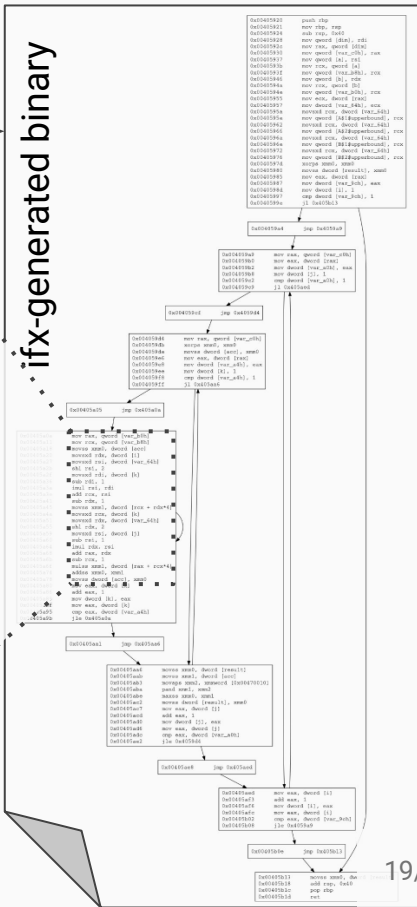
:      -> multiply with B(k,j)
mulss xmm1, dword ptr [rax+rcx*4]
addss xmm0, xmm1
movss dword ptr [rbp-0x98], xmm0
:      -> accumulate and save acc

```

Read Operands

xmm0

ifx-generated binary



Approach: encode the desired behavior (control flow + exception) into an SMT query

Let's check the *first warning* which resulted from spoofing an Invalid exception in the first execution of the multiply instruction.

$$A = \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix}$$
$$B = \begin{bmatrix} 4 & 2 \\ 3 & 1 \end{bmatrix}$$

```
assert ( = 0x7ffef9945e28_1 xmm0_0 )
```

SMT
query

Active Symbols

```

0x7ffef9945e28
0 + a_0 * b_0 : xmm0
  a_0 * b_0 : xmm1
a_0 : 0x7f9dea0970a4
a_1 : 0x7f9dea0970a8
a_2 : 0x7f9dea0970ac
a_3 : 0x7f9dea0970b0
b_0 : 0x7f9dea0970b4
b_1 : 0x7f9dea0970b8
b_2 : 0x7f9dea0970bc
b_3 : 0x7f9dea0970c0

```

```

:      -> load acc
movss xmm0, dword ptr [rbp-0x98]

:      -> load A(i,k)
movss xmm1, dword ptr [rcx+rdx*4]

:      -> multiply with B(k,j)
mulss xmm1, dword ptr [rax+rcx*4]
addss xmm0, xmm1
movss dword ptr [rbp-0x98], xmm0

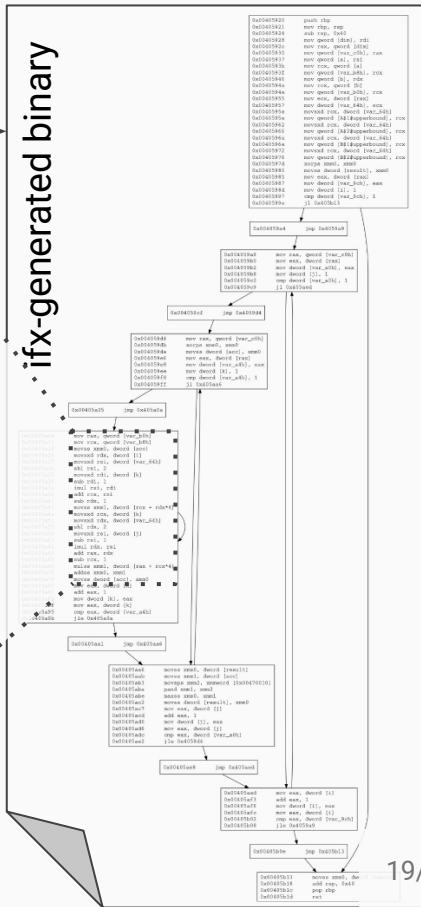
:      -> accumulate and save acc

```

Read Operands

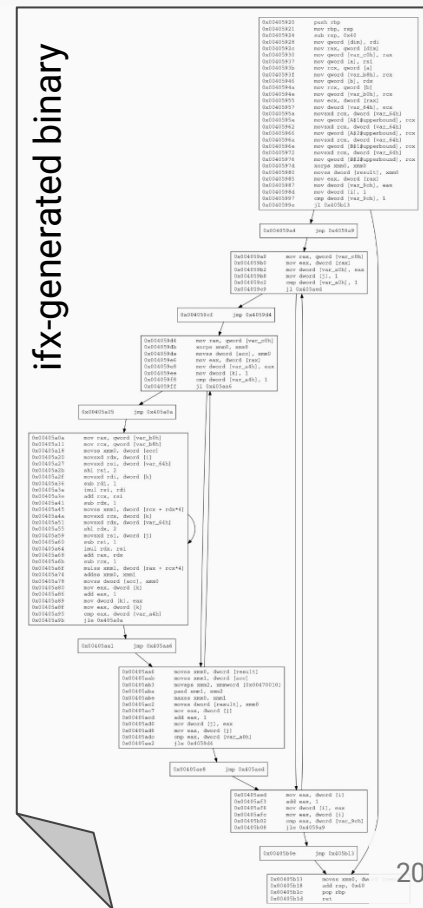
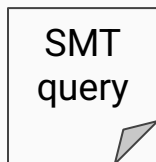
xmm0

fx-generated binary



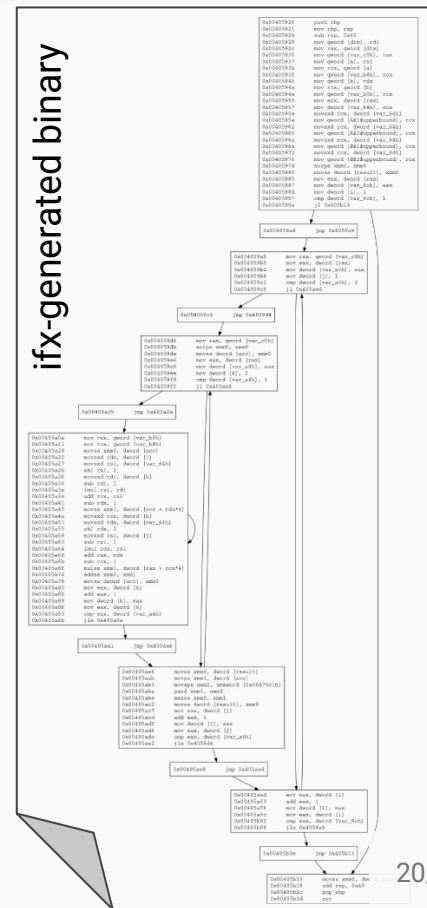
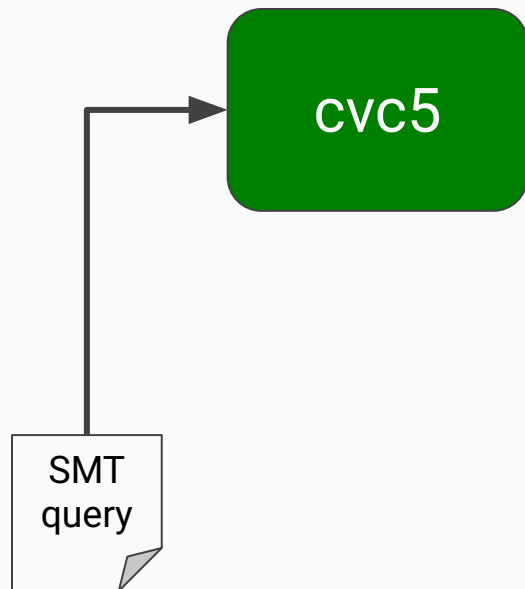
Approach: encode the desired behavior (control flow + exception) into an SMT query

Let's check the *first warning* which resulted from spoofing an Invalid exception in the first execution of the multiply instruction.



Approach: encode the desired behavior (control flow + exception) into an SMT query

Let's check the *first warning* which resulted from spoofing an Invalid exception in the first execution of the multiply instruction.



Approach: encode the desired behavior (control flow + exception) into an SMT query

Let's check the *first warning* which resulted from spoofing an Invalid exception in the first execution of the multiply instruction.

$$A = \begin{bmatrix} NaN & 0 \\ 0 & 0 \end{bmatrix}$$

$$B = \begin{bmatrix} -1.17 \times 10^{-38} & 0 \\ 0 & 0 \end{bmatrix}$$

cvc5

SMT
query

ifx-generated binary

[illegible]

```

00000000  mov     rax, qword [var_40h]
00000001  mov     eax, dword [rax]
00000002  mov     dword [var_30h], eax
00000003  mov     dword [i], 1
00000004  cmp     dword [var_30h], 1
00000005  jz      0040000d

```

```

00000000: jmp 0x00000004
00000004: mov rax, qword [rax_rbx]
00000005: scasd word, word
00000006: movs dword [edi], word
00000007: mov ax, dword [rax]
00000008: movs dword [rax_rbx], ax
00000009: movs dword [di], 1
0000000a: cmp dword [rax_rbx], 1
0000000b: jz 0x0000000c

```

[illegible][illegible]

```
0x04075a6 jmp 0x075a6d
0x04075a6d mov max, dword [i]
0x04075a6f add max, 1
0x04075a70 mov dword [i], max
0x04075a72 mov max, dword [i]
0x04075a74 cmp max, dword [var_9ch]
0x04075a76 jbe 0x075b49
```

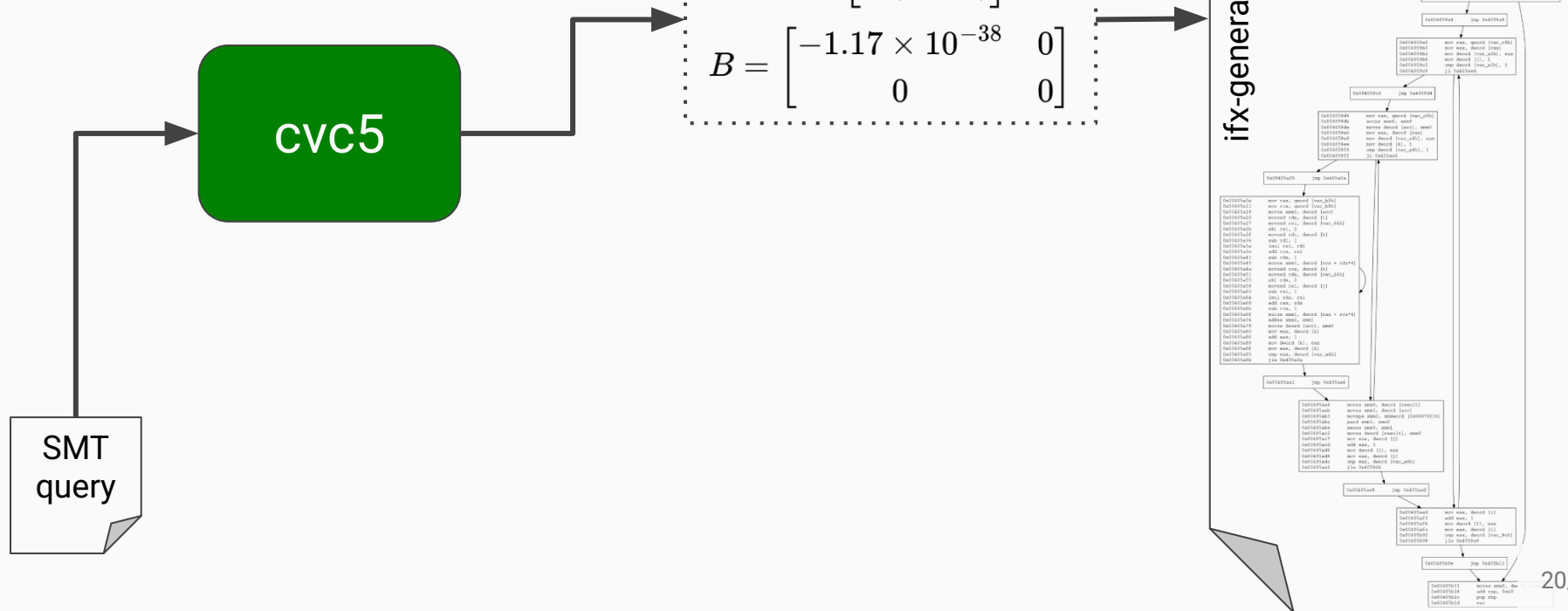
```

0x00405b1e      jmp 0x005b13
0x00405b13      movss xmm0, dw
0x00405b18      add     esp, 0x10
0x00405b1c      pop     ebp
0x00405b1d      ret

```

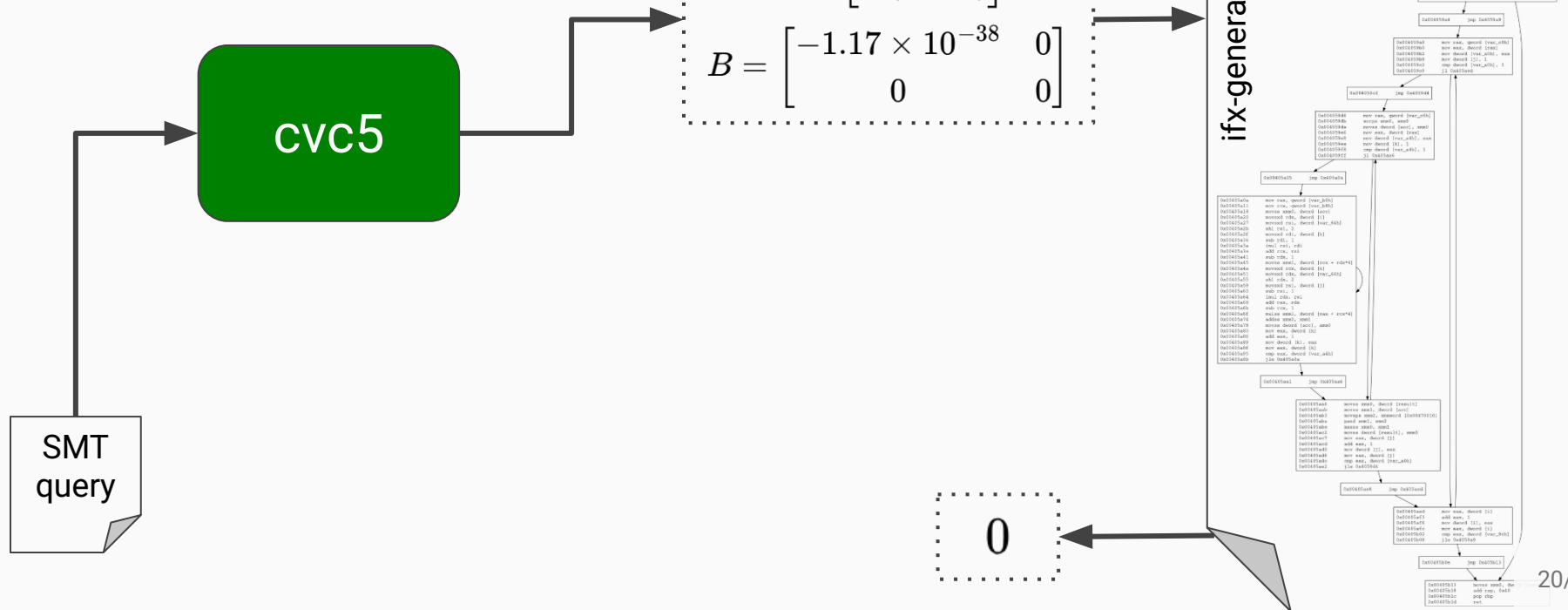
Approach: encode the desired behavior (control flow + exception) into an SMT query

Let's check the *first warning* which resulted from spoofing an Invalid exception in the first execution of the multiply instruction.



Approach: encode the desired behavior (control flow + exception) into an SMT query

Let's check the *first warning* which resulted from spoofing an Invalid exception in the first execution of the multiply instruction.



Approach: encode the desired behavior (control flow + exception) into an SMT query

Let's check the *first warning* which resulted from spoofing an Invalid exception in the first execution of the multiply instruction.

$$A = \begin{bmatrix} NaN & 0 \\ 0 & 0 \end{bmatrix}$$

$$B = \begin{bmatrix} -1.17 \times 10^{-38} & 0 \\ 0 & 0 \end{bmatrix}$$

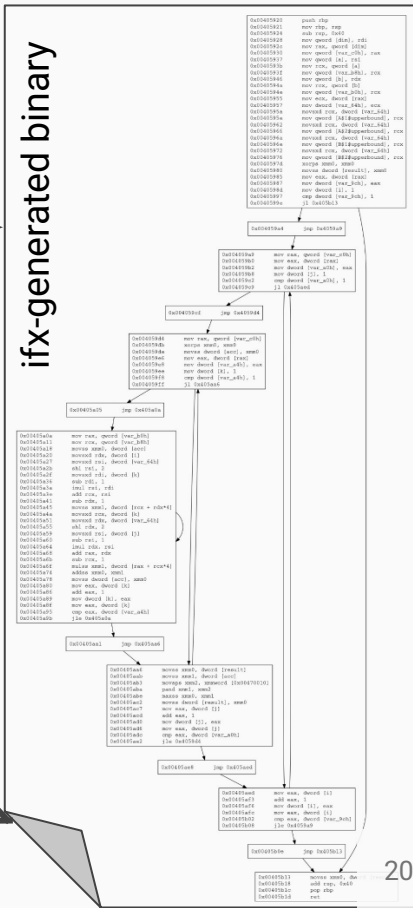
cvc5

Disassembly	Source Location	Event	Taint Count
movss xmm1, dword ptr [rcx+rdx*4]	main.f90:17	G---	1
mulss xmm1, dword ptr [rax+rcx*4]	main.f90:17	-P-r	1
addss xmm0, xmm1	main.f90:17	-P-r	2
...			
maxss xmm0, xmm1	main.f90:19	--Kr	1
movss dword ptr [rbp-0x44], xmm0	main.f90:19	--K-	0

SMT
query

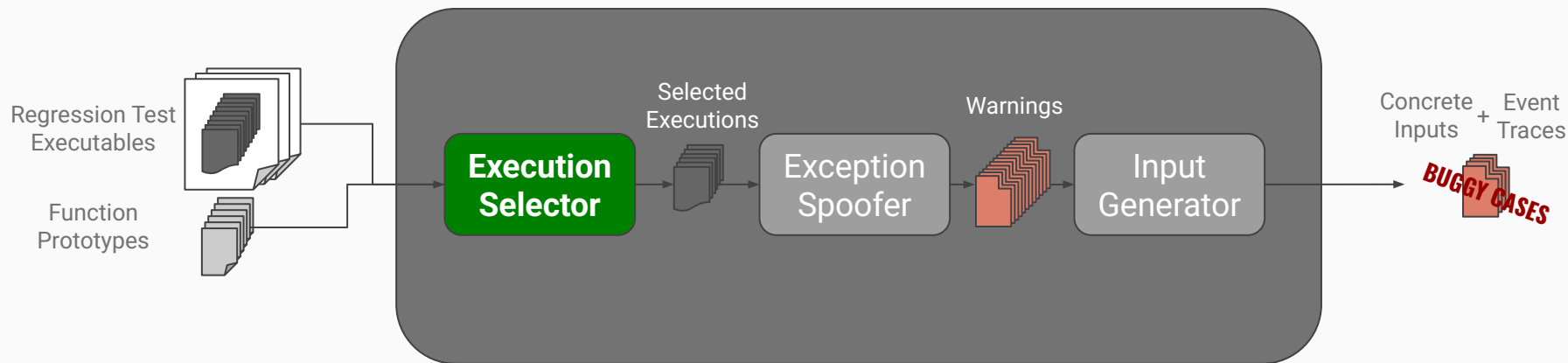
0

ifx-generated binary



Challenge 3: How do we get a representative set of function executions to test?

Approach: Take executions from the library's regression tests



Evaluation: A BLAS Case Study

- **26 functions** from Levels 1 & 2
- **Reference BLAS** (v1.12), **OpenBLAS** (v0.3.28), and **BLIS** (v1.0)
- **GNU** (gfortran, gcc) and **Intel** (ifx, icx) compilers
 - **various optimization combinations**: default, -O3, and -ffast-math/-fp-model=fast=[1|2], etc...

-> **598 (function, implementation, compiler, optimizations) tuples**

-> 12 hours of total testing time

Finding 1: Only 4.4% of spoofed exceptions resulted in warnings.

Finding 2: EXCVATE found inputs triggering exception-handling failures in 5/26 BLAS functions

sgemv / sger

$$\mathbf{y} := \alpha \mathbf{A} \mathbf{x} + \beta \mathbf{y}$$

$$\mathbf{A} := \alpha \mathbf{x} \mathbf{y}^T + \mathbf{A}$$

EXCVATE found
exception-handling failures
caused by **compiler
optimizations that change
control flow** in the face of
comparisons involving NaN

sgemv / sger

$$\mathbf{y} := \alpha \mathbf{A} \mathbf{x} + \beta \mathbf{y}$$

$$\mathbf{A} := \alpha \mathbf{x} \mathbf{y}^T + \mathbf{A}$$

EXCVATE found
exception-handling failures
caused by **compiler
optimizations that change
control flow** in the face of
comparisons involving NaN

```
(in) TRANS: N  
(in) M: 1  
(in) N: 3  
(in) ALPHA: 1.14752e-41  
(in) LDA: 2  
(in) INCX: 1  
(in) BETA: 1  
(in) INCY: 1  
(in) A: 0 0 0 0 0 0  
(in) X: nan 0 0  
(in) Y: 0
```

sgemv / sger

$$\mathbf{y} := \alpha \mathbf{A} \mathbf{x} + \beta \mathbf{y}$$

$$\mathbf{A} := \alpha \mathbf{x} \mathbf{y}^T + \mathbf{A}$$

EXCVATE found
exception-handling failures
caused by **compiler
optimizations that change
control flow** in the face of
comparisons involving NaN

```
(in) TRANS: N
(in) M: 1
(in) N: 3
(in) ALPHA: 1.14752e-41
(in) LDA: 2
(in) INCX: 1
(in) BETA: 1
(in) INCY: 1
(in) A: 0 0 0 0 0 0
(in) X: nan 0 0
(in) Y: 0
```

	Reference BLAS	BLIS	OpenBLAS
GNU default			
w/ -O3			
w/ -ffast-math			
w/ -O3 -ffast-math			
Intel default			
w/ -O3			
w/ -O3 -fp-model=fast=1			
w/ -O3 -fp-model=fast=2			

sgemv / sger

$$y := \alpha Ax + \beta y$$

$$A := \alpha xy^T + A$$

EXCVATE found
exception-handling failures
caused by **compiler
optimizations that change
control flow** in the face of
comparisons involving NaN

```
(in) TRANS: N
(in) M: 1
(in) N: 3
(in) ALPHA: 1.14752e-41
(in) LDA: 2
(in) INCX: 1
(in) BETA: 1
(in) INCY: 1
(in) A: 0 0 0 0 0 0
(in) X: nan 0 0
(in) Y: 0
```

	Reference BLAS	BLIS	OpenBLAS
GNU default			
w/ -O3			
w/ -ffast-math		(out) Y: 0	
w/ -O3 -ffast-math		(out) Y: 0	
Intel default			
w/ -O3	(out) Y: nan		
w/ -O3 -fp-model=fast=1			
w/ -O3 -fp-model=fast=2		(out) Y: 0	

sgemv / sger

$$y := \alpha Ax + \beta y$$

$$A := \alpha xy^T + A$$

EXCVATE found
exception-handling failures
caused by **compiler
optimizations that change
control flow** in the face of
comparisons involving NaN

```
(in) M: 2  
(in) N: 1  
(in) ALPHA: 1.14752e-41  
(in) INCX: 1  
(in) INCY: 1  
(in) LDA: 3  
(in) X: 0 0  
(in) Y: nan  
(in) A: 0 0 0
```

	Reference BLAS	BLIS	OpenBLAS
GNU default			
w/ -O3			
w/ -ffast-math	(out) A: 0 0 0	(out) A: 0 0 0	
w/ -O3 -ffast-math	(out) A: 0 0 0	(out) A: 0 0 0	
Intel default			
w/ -O3			(out) A: nan nan 0
w/ -O3 -fp-model=fast=1			
w/ -O3 -fp-model=fast=2	(out) A: 0 0 0	(out) A: 0 0 0	

srotmg / srotm

Generate and apply a modified Givens rotation, respectively

EXCVATE found multiple different exception-handling failures **necessitating clearer documentation and even possible deprecation**

	Reference BLAS	BLIS	OpenBLAS
GNU default			
w/ -O3			
w/ -ffast-math			
w/ -O3 -ffast-math			
Intel default			
w/ -O3			
w/ -O3 -fp-model=fast=1			
w/ -O3 -fp-model=fast=2			

srotmg / srotm

Generate and apply a
modified Givens rotation,
respectively

EXCVATE found multiple
different exception-handling
failures **necessitating clearer
documentation and even
possible deprecation**

```
(in) N: 1  
(in) INCX: 1  
(in) INCY: 1  
(in) SPARAM: nan 0 0 0 0  
(in) SX: 0  
(in) SY: 0
```

	Reference BLAS	BLIS	OpenBLAS
GNU default			
w/ -O3			
w/ -ffast-math			
w/ -O3 -ffast-math			
Intel default			
w/ -O3			
w/ -O3 -fp-model=fast=1			
w/ -O3 -fp-model=fast=2			

srotmg / srotm

Generate and apply a
modified Givens rotation,
respectively

EXCVATE found multiple
different exception-handling
failures **necessitating clearer
documentation and even
possible deprecation**

```
(in) N: 1  
(in) INCX: 1  
(in) INCY: 1  
(in) SPARAM: nan 0 0 0 0  
(in) SX: 0  
(in) SY: 0
```

	Reference BLAS	BLIS	OpenBLAS
GNU default			
w/ -O3			
w/ -ffast-math			
w/ -O3 -ffast-math			
Intel default			
w/ -O3			
w/ -O3 -fp-model=fast=1			
w/ -O3 -fp-model=fast=2			

```
(out) SX: 0  
(out) SY: 0
```

srotmg / srotm

Generate and apply a
modified Givens rotation,
respectively

EXCVATE found multiple
different exception-handling
failures **necessitating clearer
documentation and even
possible deprecation**

```
(in) SD1: inf
(in) SD2: -1.99976
(in) SX1: nan
(in) SY1: -inf
(in) SPARAM: 0 0 0 0 0
```

```
(in) SD1: 1.17549e-38
(in) SD2: -1.4013e-45
(in) SX1: nan
(in) SY1: -1.66667
(in) SPARAM: 0 0 0 0 0
```

	Reference BLAS	BLIS	OpenBLAS
GNU default			
w/ -O3			
w/ -ffast-math			
w/ -O3 -ffast-math			
Intel default			
w/ -O3			
w/ -O3 -fp-model=fast=1			
w/ -O3 -fp-model=fast=2			

srotmg / srotm

Generate and apply a
modified Givens rotation,
respectively

EXCVATE found multiple
different exception-handling
failures **necessitating clearer
documentation and even
possible deprecation**

```
(in) SD1: inf
(in) SD2: -1.99976
(in) SX1: nan
(in) SY1: -inf
(in) SPARAM: 0 0 0 0 0
```

```
(in) SD1: 1.17549e-38
(in) SD2: -1.4013e-45
(in) SX1: nan
(in) SY1: -1.66667
(in) SPARAM: 0 0 0 0 0
```

	Reference BLAS	BLIS	OpenBLAS
GNU default			
w/ -O3			
w/ -ffast-math			
w/ -O3 -ffast-math			
Intel default			
w/ -O3			
w/ -O3 -fp-model=fast=1			
w/ -O3 -fp-model=fast=2			

```
(out) SD1: 0
(out) SD2: 0
(out) SX1: 0
(out) SPARAM: -1 0 0 0 0
```

srotmg / srotm

Generate and apply a
modified Givens rotation,
respectively

EXCVATE found multiple
different exception-handling
failures **necessitating clearer
documentation and even
possible deprecation**

```
(in) SD1: inf
(in) SD2: -1.99976
(in) SX1: nan
(in) SY1: -inf
(in) SPARAM: 0 0 0 0 0
```

```
(in) SD1: 1.17549e-38
(in) SD2: -1.4013e-45
(in) SX1: nan
(in) SY1: -1.66667
(in) SPARAM: 0 0 0 0 0
```

	Reference BLAS	BLIS	OpenBLAS
GNU default			
w/ -O3			
w/ -ffast-math			
w/ -O3 -ffast-math			
Intel default			
w/ -O3			
w/ -O3 -fp-model=fast=1			
w/ -O3 -fp-model=fast=2			

(out) SD1: 0
(out) SD2: 0
(out) SX1: 0
(out) SPARAM: -1 0 0 0 0 0

**UNDOCUMENTED
ERROR CODE**

srotmg / srotm

Generate and apply a modified Givens rotation, respectively

EXCVATE found multiple different exception-handling failures **necessitating clearer documentation and even possible deprecation**

```
(in) SD1: inf
(in) SD2: -1.23382e-05
(in) SX1: -1.81899e-12
(in) SY1: -2.21834e-39
(in) SPARAM: 0 0 0 0 0
```

	Reference BLAS	BLIS	OpenBLAS
GNU default			
w/ -O3			
w/ -ffast-math			
w/ -O3 -ffast-math			
Intel default			
w/ -O3			
w/ -O3 -fp-model=fast=1			
w/ -O3 -fp-model=fast=2			

srotmg / srotm

Generate and apply a modified Givens rotation, respectively

EXCVATE found multiple different exception-handling failures **necessitating clearer documentation and even possible deprecation**

```
(in) SD1: inf
(in) SD2: -1.23382e-05
(in) SX1: -1.81899e-12
(in) SY1: -2.21834e-39
(in) SPARAM: 0 0 0 0 0
```

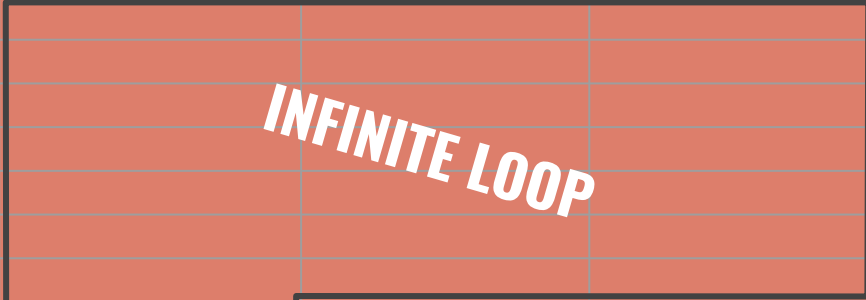
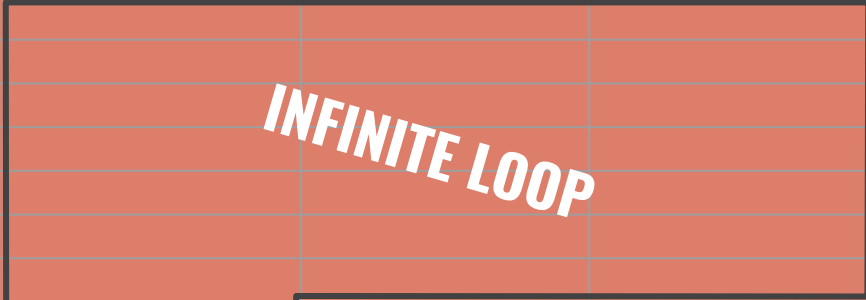
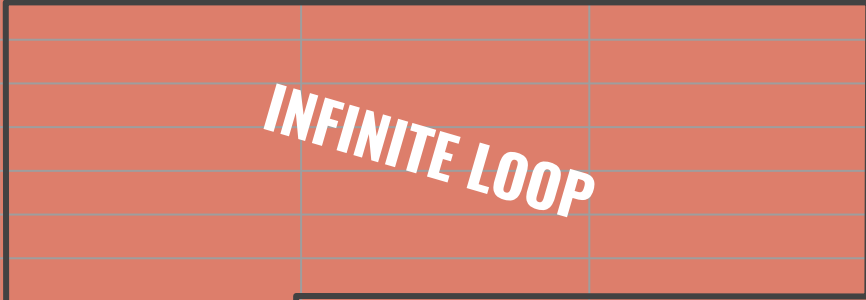
	Reference BLAS	BLIS	OpenBLAS
GNU default			
w/ -O3			
w/ -ffast-math			
w/ -O3 -ffast-math			
Intel default			
w/ -O3			
w/ -O3 -fp-model=fast=1			
w/ -O3 -fp-model=fast=2		Undocumented Error Code	NaNs in output

srotmg / srotm

Generate and apply a modified Givens rotation, respectively

EXCVATE found multiple different exception-handling failures **necessitating clearer documentation and even possible deprecation**

```
(in) SD1: inf
(in) SD2: -1.23382e-05
(in) SX1: -1.81899e-12
(in) SY1: -2.21834e-39
(in) SPARAM: 0 0 0 0 0
```

	Reference BLAS	BLIS	OpenBLAS
GNU default			
w/ -O3			
w/ -ffast-math			
w/ -O3 -ffast-math			
Intel default			
w/ -O3			
w/ -O3 -fp-model=fast=1			
w/ -O3 -fp-model=fast=2			
		Undocumented Error Code	NaNs in output

sgbmv

$$y := \alpha Ax + \beta y$$

EXCVATE found exception handling failures in all tuples due to the handling of **implicit zeros in the banded matrix representation**.

	Reference BLAS	BLIS	OpenBLAS
GNU default			
w/ -O3			
w/ -ffast-math			
w/ -O3 -ffast-math			
Intel default			
w/ -O3			
w/ -O3 -fp-model=fast=1			
w/ -O3 -fp-model=fast=2			

sgbmv

$$y := \alpha Ax + \beta y$$

EXCVATE found exception handling failures in all tuples due to the handling of **implicit zeros in the banded matrix representation.**

```
(in) TRANS: N
(in) M: 2
(in) N: 5
(in) KL: 0
(in) KU: 0
(in) ALPHA: 1
(in) LDA: 2
(in) INCX: -1
(in) BETA: 0
(in) INCY: 1
(in) A: 4.26326e-14 -1 -1
-1 -1 -1 -1 -1 -1 -1
(in) X: nan -1 -1 -1 0
(in) Y: 0 -1
```

$$\begin{bmatrix} 4.26326 \times 10^{-14} & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 \end{bmatrix}$$

$$\begin{bmatrix} 0 \\ -1 \\ -1 \\ -1 \\ NaN \end{bmatrix}$$

	Reference BLAS	BLIS	OpenBLAS
GNU default			
w/ -O3			
w/ -ffast-math			
w/ -O3 -ffast-math			
Intel default			
w/ -O3			
w/ -O3 -fp-model=fast=1			
w/ -O3 -fp-model=fast=2			

sgbmv

$$y := \alpha Ax + \beta y$$

EXCVATE found exception handling failures in all tuples due to the handling of **implicit zeros in the banded matrix representation.**

```
(in) TRANS: N
(in) M: 2
(in) N: 5
(in) KL: 0
(in) KU: 0
(in) ALPHA: 1
(in) LDA: 2
(in) INCX: -1
(in) BETA: 0
(in) INCY: 1
(in) A: 4.26326e-14 -1 -1
-1 -1 -1 -1 -1 -1 -1
(in) X: nan -1 -1 -1 0
(in) Y: 0 -1
```

$$\begin{bmatrix} 4.26326 \times 10^{-14} & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 \end{bmatrix}$$

$$\begin{bmatrix} 0 \\ -1 \\ -1 \\ -1 \\ NaN \end{bmatrix}$$

	Reference BLAS	BLIS	OpenBLAS
GNU default			
w/ -O3			
w/ -ffast-math			
w/ -O3 -ffast-math			
Intel default			
w/ -O3			
w/ -O3 -fp-model=fast=1			
w/ -O3 -fp-model=fast=2			

(out) Y: 0 1

We have...

...introduced the problem of testing exception handling

- And why current input-generation tools are a poor fit for the problem

...described a novel approach to this problem

- Targeting binary executables using exception spoofing and constraint solving
- Implemented in the prototype tool EXCVATE

...demonstrated our approach on the BLAS

- Tested across multiple implementations, compilers, compiler optimizations
- Found exception-handling failures in 5/26 functions

Source code and data available at <https://github.com/ucd-plse/EXCVATE>