Computer System Support for Scientific and Engineering Computation

Lecture 22 - July 14, 1988 (notes revised June 14, 1990)

1 Floating Point Exceptions

This lecture continues reports on the exception handling of various machines. One of the problems of exception handling is to figure out when an exception should be raised. For example, is $\sin(10^{38})$ exceptional? What about 0^0?

1.1 Weitek

Weitek makes several different floating point chips. They all use the IEEE format (except the 2364, which uses IBM floating point format), but vary in their compliance with the rest of the IEEE standard. They all have array multipliers, which has two consequences. First, none of the Weitek chips support denormalized numbers. Rather, they provide the hooks for software to emulate gradual underflow. Secondly they don't support 80 bit arithmetic. Recall that the Intel 8087 and Motorola 68881 support 80 bit arithmetic, but don't have an array multipler and do transcendental functions with CORDIC rather than rational approximation.

The Weitek 3164 is a pipelined chip that can be used to build a system providing allegedly complete IEEE conformance (the first such chip, according to Weitek literature). Like HP machines, it provides multiple registers for storing exception information. There are two kinds of exceptions. Source exceptions are detected before starting an operation (such as ∞/∞). Destination exceptions can only be detected after the operation is complete (such as underflow). There are three non-IEEE exceptions, Integer Overflow, NaN and Denormalized. The first is a destination exception, the last two are source exceptions.

As an example of how to simulate IEEE arithmetic, consider gradual underflow. Turning on the denormalized exception causes any operation that involves denormalized operands to trap. The trap handler can then scale the operands to be normalized, perform the operation, and then rescale the result. When the operands are normalized but the result is subnormal, an unrounded result is delivered. The trap handler can then consult the inexact bit, and either produce a correctly rounded wrapped result or a correctly rounded denormalized number.

1.2 IBM/370 FORTRAN

The 370 uses hexadecimal floating point formats; all precisions have seven bits of exponent. Unlike DEC Vaxes and IEEE arithmetic which round, the 370 truncates. For multiplication this is unambiguous: the exact result is computed and then chopped. For addition and subtraction, there is a subtlety. The expression $1 - 16^{-60}$ will not compute $1 - 16^{-60}$ exactly and then truncate (resulting in a number slightly less than 1). Rather the smaller operand (16^{-60} in this case) will be shifted into place, chopped to one additional digit beyond larger operand (the guard digit), the operation performed exactly, and then the final result chopped. Thus 16^{-60} will become 0, and $1 - 16^{-60}$ will be exactly 1.

Operations can generate zero-divide, overflow, and underflow exceptions, but there is no inexact exception. Library routines can generate many other exceptions, mostly indicating domain errors. Because there are no NaNs or ∞, the default values for mathematical functions must be numbers. Library functions valid only for nonegative arguments typically use `F(ABS(X))` as the default result. `SIN(X)` and `COS(X)` are exceptional for arguments greater than $\pi 2^{p-3}$. Minus zero and denormals are tolerated as operands but are not otherwise supported. Exponent underflow flushes to zero.

A program can specify user-written handlers. A handler is passed the exception type and operand or result values. The handler may specify a desired result value and return; execution continues from the point the exception was recognized, using the handler's result in lieu of the exceptional operations' result.

1.3 IBM RT with 4.3 BSD UNIX

To permit binaries to run on any configuration, the RT compiles arithmetic expressions into operations for a "generic" floating-point unit; at run time, the first time a generic operation is encountered it is overlaid by code generated for whatever hardware unit is present. A user option can force generated code to avoid 68881 extended precision, permitting near-identical results regardless of hardware. One drawback to the "generic" strategy is that performance is less than what a good compiler with instruction scheduling could achieve if the hardware were known at compile time.

Compile-time evaluations and conversions use default rounding and exception handling. Run-time floating point supports all rounding modes and exceptions, and includes standard-conforming binary-decimal conversion. `Printf()` and `scanf()` support `INF` and `NAN()`. There are two flavors of compare: a `CMP` operation compares for equality without trapping; a `CMPT` operation compares for ordering and traps on a NaN operand.

Exception handling uses standard BSD mechanisms and, because so few people have exploited it, is surely buggy. An enabled IEEE exception produces a SIGFPE signal. The operation and operands causing the exception are not available to the handler.

1.4 Vax

The VAX F and G formats have the same precision and essentially the same exponent range as the IEEE single and double formats.[1] The VAX also has a double precison format with an 8 bit exponent field (D format) and a quadruple precision format (H format). All

[1] IEEE single normalized numbers range from 2^{-126} to $(1-\epsilon)2^{128}$ while VAX ranges from 2^{-128} to $(1-\epsilon)2^{127}$. In double precision, IEEE ranges from 2^{-1022} to $(1-\epsilon)2^{1024}$ while VAX ranges from 2^{-1024} to $(1-\epsilon)2^{1023}$.

arithmetic operations are computed exactly and then rounded, however the VAX doesn't round $\frac{1}{2}$ to even, but instead always rounds it up. It has no rounding modes.

The VAX uses numbers with the biased exponent of 0 to represent 0 if the sign bit is 0, and to represent a reserved operand if the sign bit is 1. A reserved operand is very similar to a signaling NaN, in that it raises an exception whenever it is touched. The VAX does not support denormals, or $\pm\infty$. Evaluating $\sin(10^{38})$ returns a valid number, but 0^0 produces a reserved operand.

Users can install their own trap handlers. These trap handlers will be called with the exception that caused the trap and the value of the PC at the time of the trap. On all newer VAX models, the state of the machine when the trap handler is called is as if the faulting instruction had not yet executed (in VAX terminology, they will fault rather than trap).[2] The trap handler can fix up the operands and return, causing the faulting instruction to re-execute, or else it can force the function that caused the trap to return with a value specified by the trap handler.

VAX VMS Fortran provides exception handling options similar to IBM/370 VS Fortran.

1.5 Mips

The Mips machines automatically emulate floating point in software if the hardware is not available. In particular, this means that if the floating point hardware breaks, users will notice slower performance, but their programs will still work correctly.[3] Exceptions are precise, even though the floating point processor can be performing multiple floating point operations simultaneously. To the trap handler, it appears that all instructions before the faulting instruction have been executed, and that none of the instructions following it have been executed. The Mips machines have two sets of flags, a "sticky" set as in the IEEE standard, and an additional non-sticky set. Mips returns NaN for $\sin(10^{38})$.

Exception traps are disabled by default; otherwise a user trap handler can examine the instruction and its operands, substituting its own value. Mips also supports counting the number of exceptions, as recommended by the commentary in the draft IEEE standard.

[2] Early models of the 780 will instead produce a reserved operand as the result and the trap handler will see the state of the machine after the faulting instruction has executed.

[3] In fact, erroneous benchmark results for the HP have been published by people who didn't realize that they weren't using floating point hardware.

WTL 3164 SINGLE-PORT AND WTL 3364 THREE-PORT 64-BIT FLOATING-POINT DATA PATH UNITS

WEITEK

3X64 PRODUCT UPDATE

OUTLINE

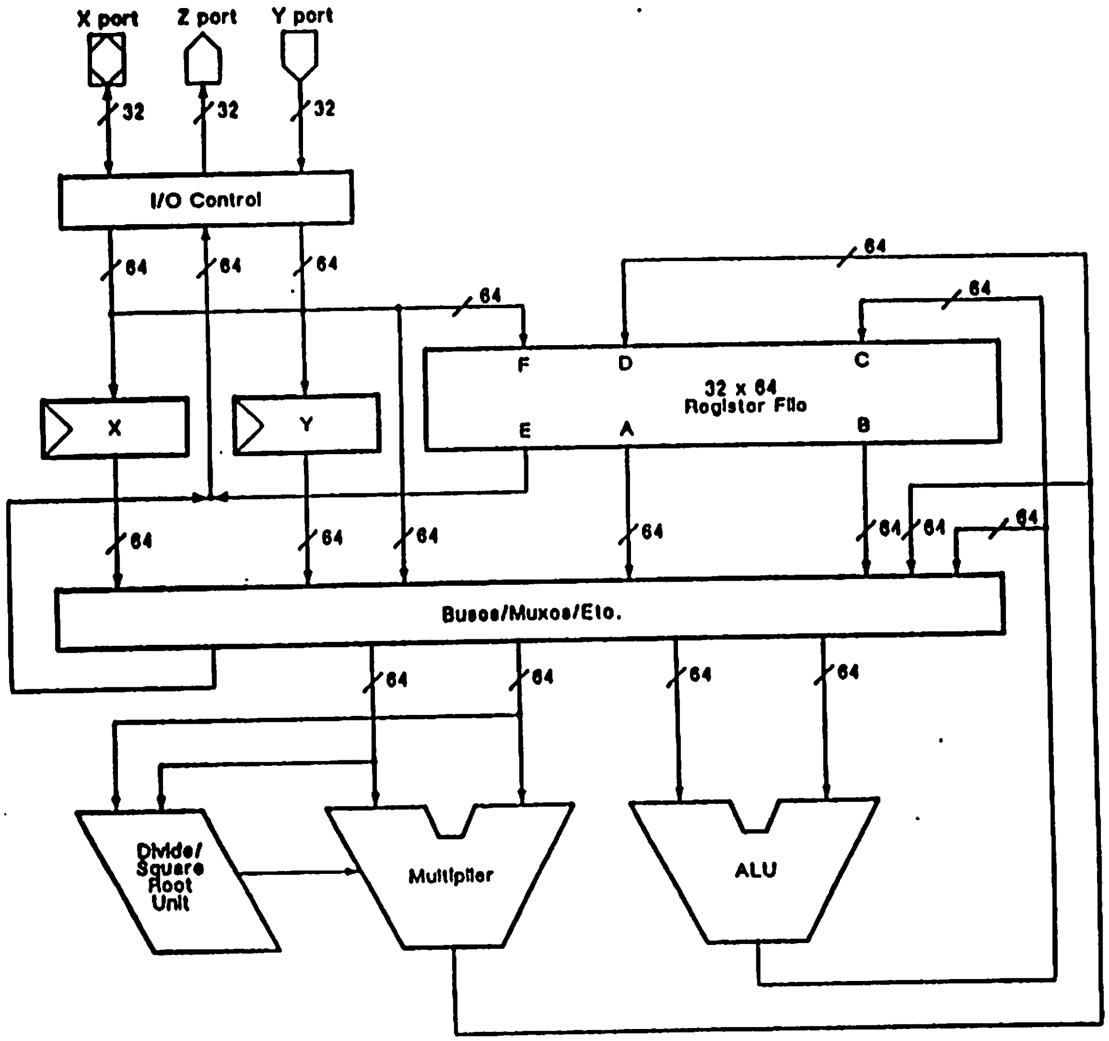
ARCHITECTURE AND FEATURES

IEEE COMPLIANCE

PERFORMANCE

STATUS AND SCHEDULE

FUTURES

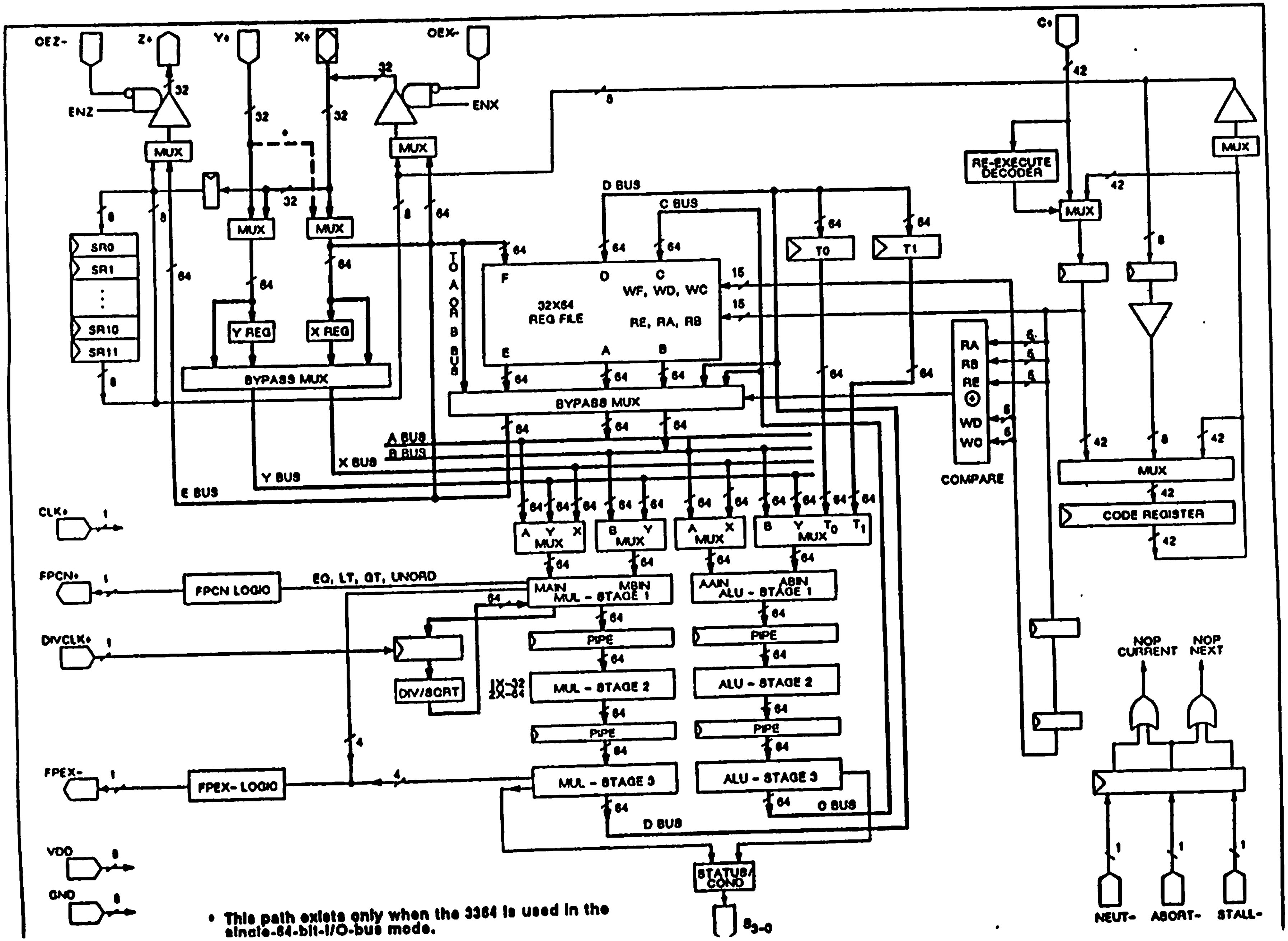


WEITEK

3X64 PRODUCT UPDATE

407

WTL 3164 / 3364 BLOCK DIAGRAM



• This path exists only when the 3364 is used in the single-64-bit-I/O-bus mode.

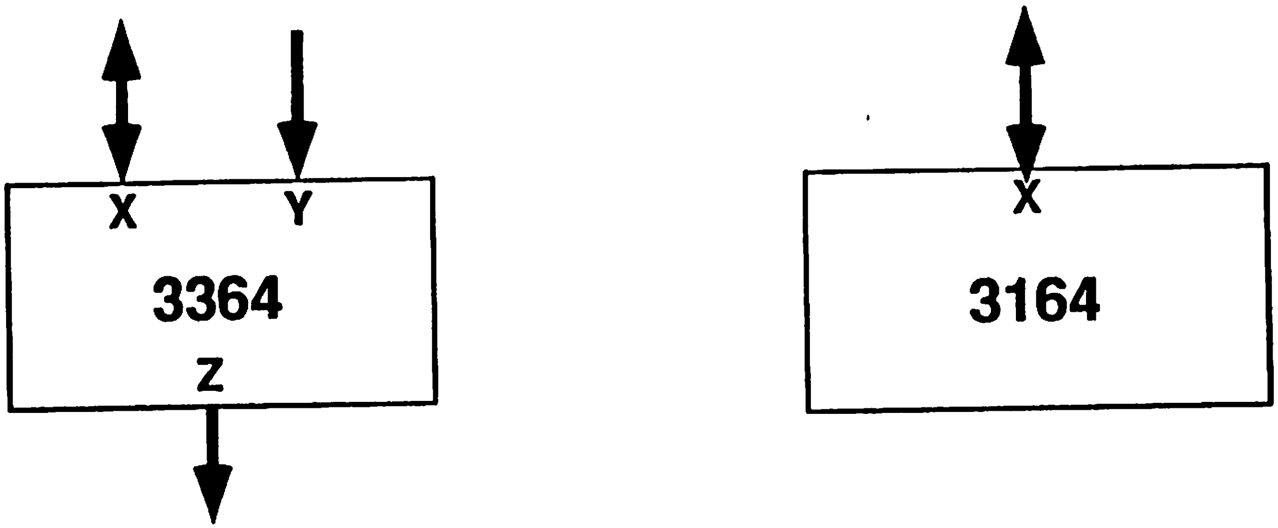
410

ARCHITECTURE

- 64-BIT IEEE FLOATING-POINT DATA PATH
- THREE INDEPENDENT ARITHMETIC UNITS CAN OPERATE IN PARALLEL
  - 64-BIT ALU
  - 64-BIT MULTIPLIER
  - 64-BIT DIVIDE/SQRT UNIT
- REGISTER FILE: SIX-PORT, 32-DEEP BY 64-BIT-WIDE
  - THREE READ PORTS
  - THREE WRITE PORTS
  - CAN BE BYPASSED ON LOADS, STORES, AND REGISTER-TO-REGISTER OPERATIONS
- INDEPENDENT LOAD/STORE
- SIX MAJOR INTERNAL 64-BIT BUSES
- EXTENSIVE STATUS AND CONTROL LOGIC

ARCHITECTURE CONT'D

- FLEXIBLE I/O STRUCTURE
  - X PORT: I/O
  - Y PORT: INPUT ONLY
  - Z PORT: OUTPUT ONLY



- THREE 32-BIT PORTS
- SINGLE 64-BIT I/O PORT

- SINGLE 32-BIT I/O PORT

ARCHITECTURE CONT'D

- TWO-CYCLE REGISTER-TO-REGISTER LATENCY
- SINGLE-CYCLE THROUGHPUT FOR
  - Xi * Yi -->Zi and / or Ra + Rb --> Rc
  - Ra * Xi + Yi --> Zi/Rc
  - ∑ Xi * Yi (requires 3364)
  - ∑ (Xi + Yi) (requires 3364)
- DIVIDE/SQRT LATENCY

	DIVIDE	SQRT
64-BIT	17	30
32-BIT	10	16

- DIV/SQRT CAN OVERLAP WITH MULTIPLIER AND/OR ALU OPERATION

FULL FUNCTION

- 64- AND 32-BIT FLOATING-POINT AND 32-BIT INTEGER INSTRUCTIONS
    - MULTIPLY, ADD, MULTIPLY-ADD, DIV, SQRT, ETC.
- 64-BIT LOGICAL
- ABSOLUTE VALUE
- COMPARE
- MIN/MAX
- FORMAT CONVERSION

FULLY INTERRUPTIBLE

- NEUT, STALL, ABORT

IEEE COMPLIANCE

- FULL IEEE SUPPORT IN PIPELINED ENVIRONMENT
  - INCLUDING DIVIDE AND SQUARE ROOT
- ALL FOUR ROUNDING MODES
- DENORMALIZED NUMBER SUPPORT
  - FAST MODE IF DNRM SUPPORT NOT REQUIRED
- FULL STATUS AND CONDITION SUPPORT
  - FPEX PIN SIGNALS OCCURENCE OF ENABLED EXCEPTION
  - EXCEPTIONS
    - SOURCE: NAN, DNRM, DVZ, INV
    - RESULT: OVF, IOVF, UNF, INX
  - FLOATING-POINT CONDITION PIN REPORTS RESULTS OF COMPARE OPERATIONS: EQ, LT, GT, UNORDERED

4/5

SIMPLE PROGRAMMING MODEL

- REGISTER-BASED PROGRAMMING MODEL
- MATCHED LATENCY FOR ALL OPERATIONS
    - EXCEPT DIVIDE AND SQRT
- BOTH SOURCES AND DESTINATIONS SPECIFIED ONLY ONCE -- WHEN INSTRUCTION IS ISSUED
    - DESTINATION ADDRESSES ARE DELAYED AUTOMATICALLY
- ALL INFORMATION NECESSARY FOR EXCEPTION HANDLING INCLUDED ON-CHIP
    - DEDICATED STATUS REGISTERS STORE THE STATUS AND REG FILE DESTINATION ADDRESSES

HIGH-LEVEL LANGUAGE SUPPORT

- AVAILABLE FOR BOTH 3164 AND 3364 WHEN USED IN XL SYSTEM
  - HLL COMPILERS (C, FORTRAN)
  - DEVELOPMENT SYSTEM
  - OTHER SUPPORT: DEBUGGERS, SIMULATORS
- PERFORMANCE OF XL-3164 IN XL8164 SYSTEM

	-1ØØ	-Ø8Ø
• LINPACK (MFLOPS) (HAND-CODED BLAS)		
SINGLE OR DOUBLE PRECISION	4.7	5.8
• WHETSTONES (MWHETS)		
SINGLE OR DOUBLE PRECISION	8.7	10.9
• DHRYSTONES	11,834	14,793

WEITEK

WEITEK 3164/3364

Pipe is 2 deep
and can do mult & add
in parallel

- 32 + 64-Bit IEEE Formats
- Denorms Handled In Software
- User Controlled Traps On All Five IEEE Exceptions
- Status Regs Allow Recovery From Exceptions
- ADD, SUB, MUL, DIV, SQRT On Chip
- DREM In Software

3164/3364 EXCEPTION HANDLING

- Source Exceptions:
  (Invalid, Divide-By-Zero) ∞ / ∞
  - Stop Current Operation
  - Denorm has to be handled in software
- Destination Exceptions
  (Underflow, Overflow, Inexact)
  - Operation Completes Before Trapping
  - Correctly Rounded Result Supplied To User Trap Handler

mult
(1) wrap denorm
[illegible]
[illegible] other input
(2) multiply
[illegible] underflows

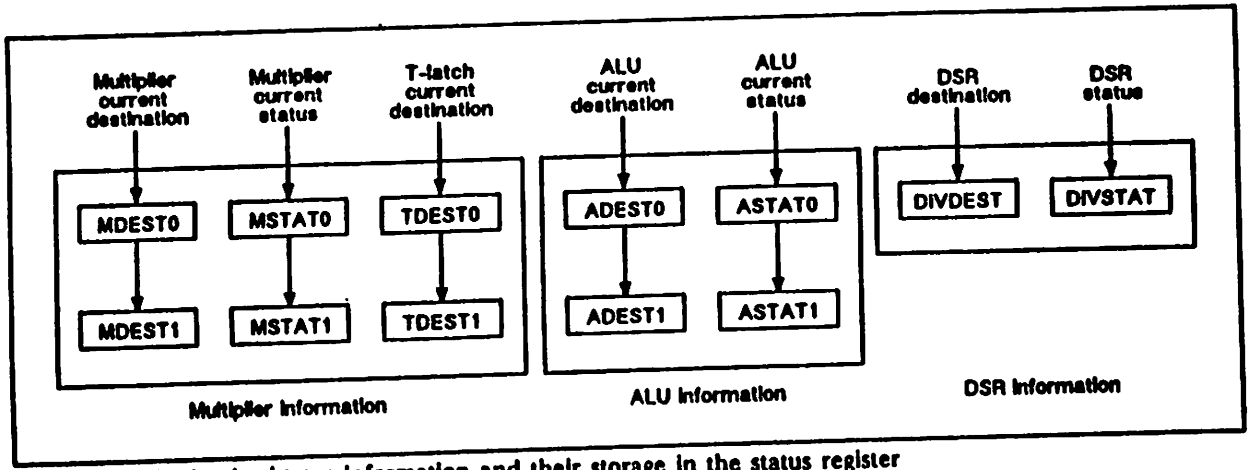
product of 2 barely denormed numbers is 0.

add/sub
wrap the denorm
[illegible] input
[illegible] other
Do the add
[illegible]

Kahan says that denorm could be done in hardware

No hardware [illegible]
ADD R0, R1, R2.
software → MUL R2
must insert trap here

WEITEK



Operations' destination/status information and their storage in the status register

SR#	BIT#								COMMENTS
	7	6	5	4	3	2	1	0	
SR0	Multiplier Latency	FPEX-Sticky	0	0	Internal NEUT- on	Rounding Mode		Fast Mode	Modes
SR1	0 XL Software Underflow	Bypass on	FPEX-Delay	I/O Mode					Modes
SR2	NaN EN	INV EN	DVZ EN	DNRM EN	OVF EN	UNF EN	INX EN	IOVF EN	Trap Enables
SR3	NaN	INV	DVZ	DNRM	OVF	UNF	INX	IOVF	Sticky Bits
SR4	0	TDEST0		MDEST0					Destination
SR5	0	0	0	ADEST0					Destination
SR6	ASTAT0				MSTAT0				Status
SR7	0	0	0	DIVDEST					Destination
SR8	0	TDEST1		MDEST1					Destination
SR9	0	0	0	ADEST1					Destination
SR10	ASTAT1				MSTAT1				Status
SR11	FPEX-Taken	DSR in progress	FPCN	Carry	DIVSTAT				Status

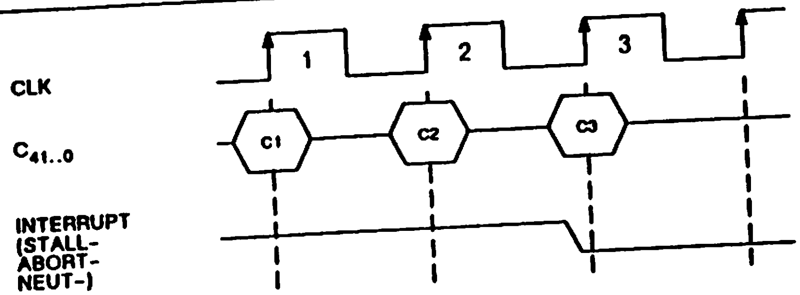
Status register structure

Non IEEE
flags for
software sim

Before you
get underflow
must pick
denorm

Two [illegible]
can't go to denorm
or other
1-bit denorm!
with chopping

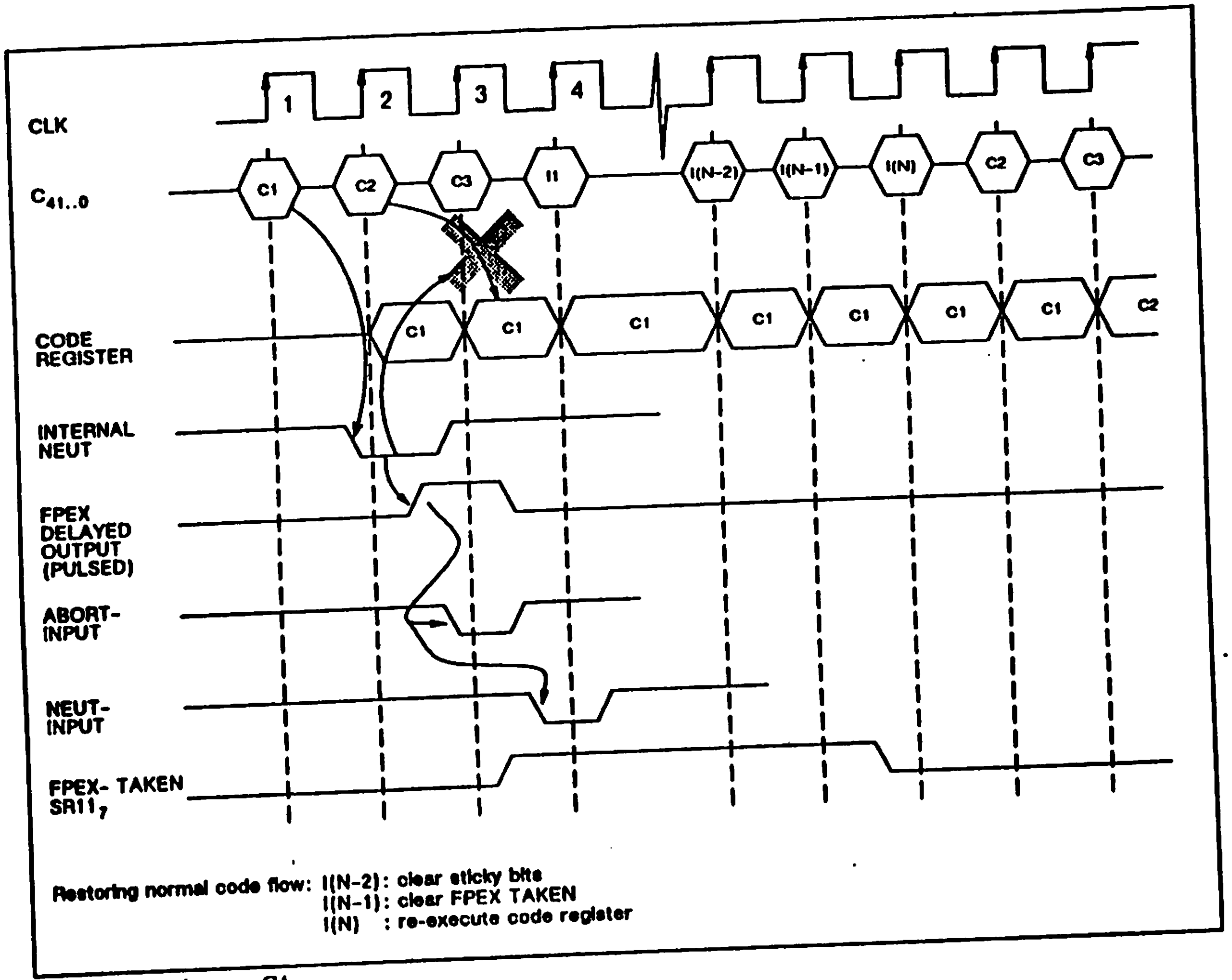
System trap handler
always traps
on underflow



Systems that can back up: both C2 and C3 can be seamlessly re-executed. Delayed FPEX.

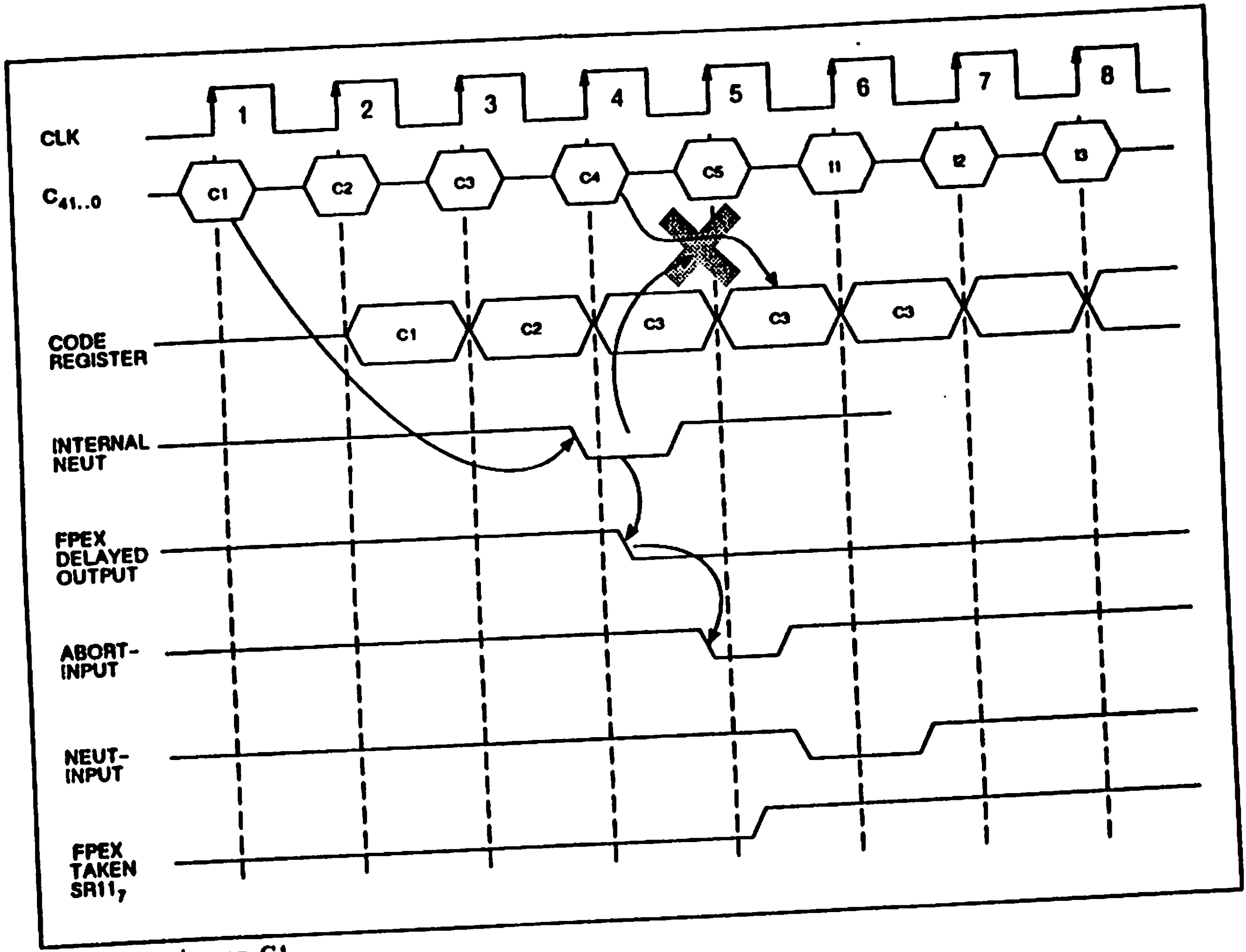
Systems that cannot back up: only C3 can be re-executed. Undelayed FPEX.

System types

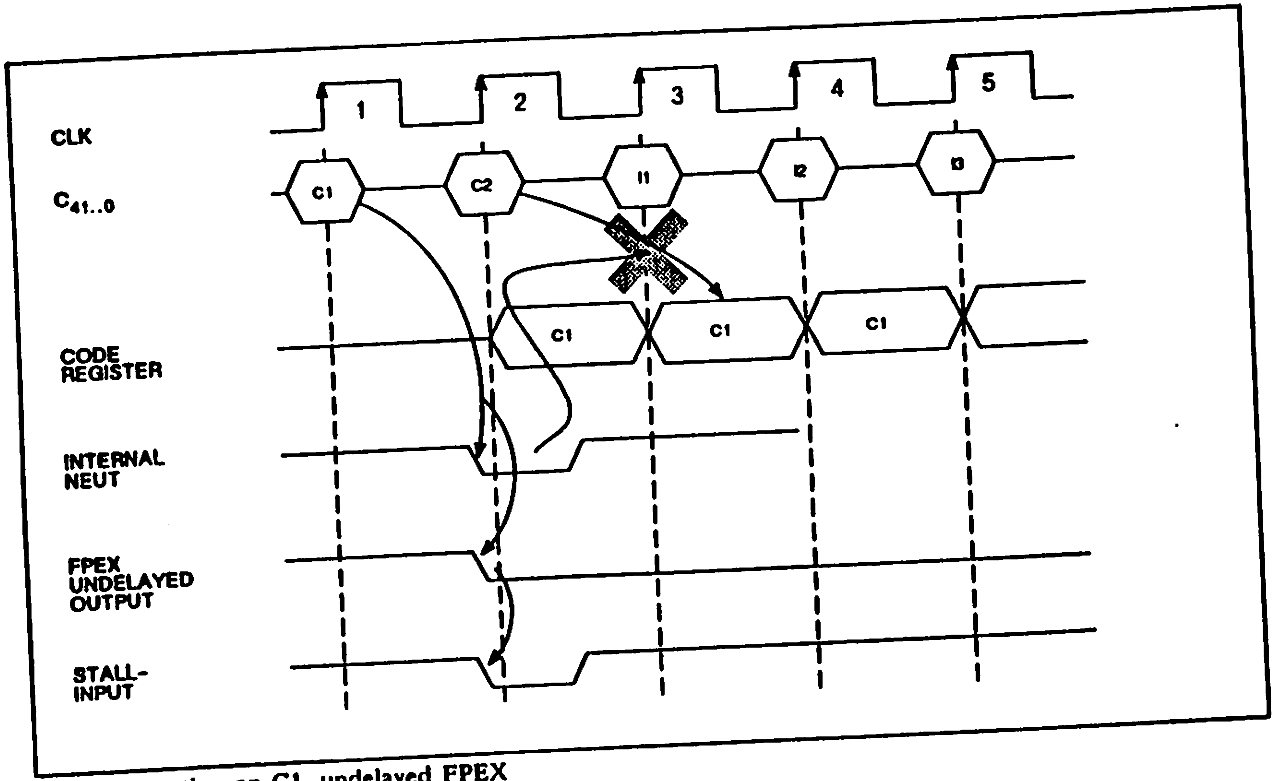


Restoring normal code flow: I(N-2): clear sticky bits
I(N-1): clear FPEX TAKEN
I(N) : re-execute code register

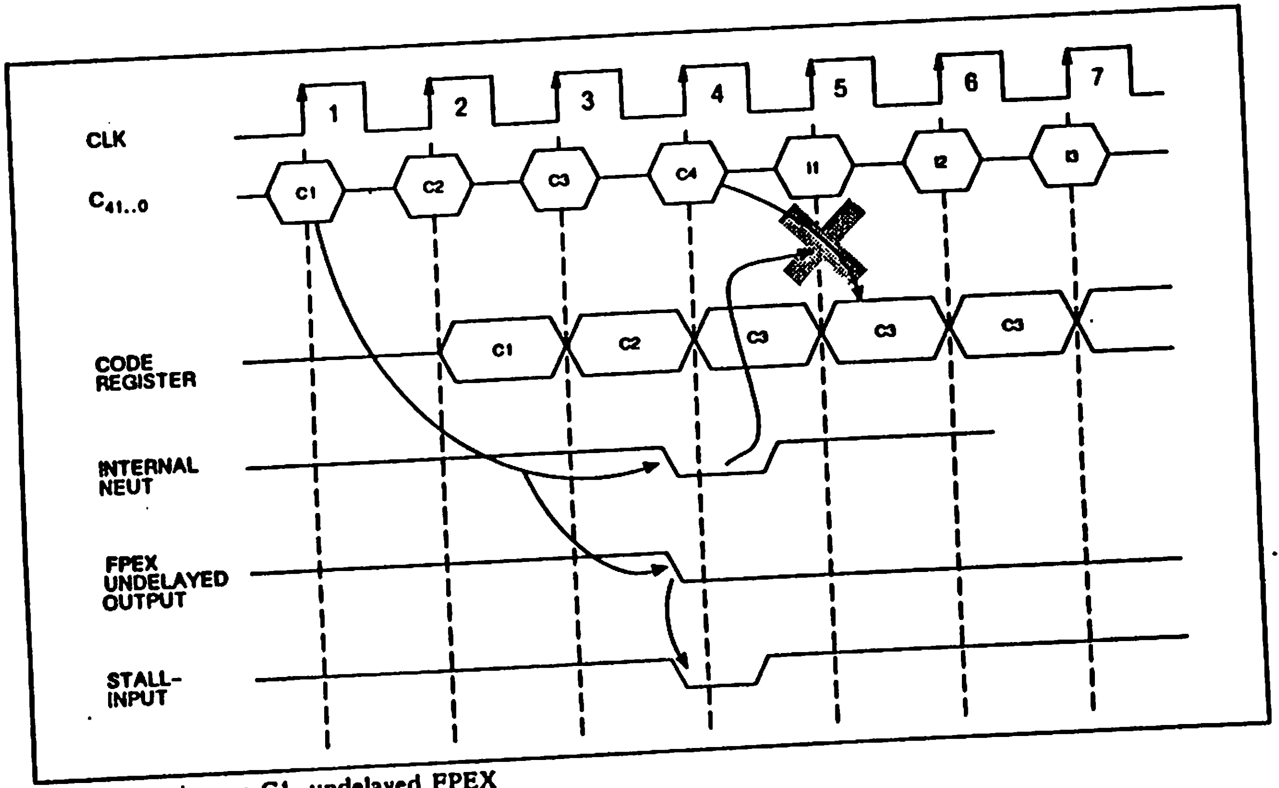
Source exception on C1



Result exception on C1



Source exception on C1, undelayed FPEX



Result exception on C1, undelayed FPEX

IBM S/370 FLOATING POINT FORMAT

Short Floating-Point Number

S	Characteristic	6-Digit Fraction	
0	1	8	31

Long Floating-Point Number

S	Characteristic	14-Digit Fraction	
0	1	8	63

Extended Floating-Point Number

High-Order Part

S	High-Order Characteristic	Leftmost 14 Digits of 28-Digit Fraction	
0	1	8	63

Low-Order Part

S	Low-Order Characteristic	Rightmost 14 Digits of 28-Digit Fraction	
64		72	127

Normalized range: 16^{-65} to $(1-\delta) \times 16^{63}$ or

$\sim 5.4 \times 10^{-79}$ to $\sim 7.2 \times 10^{75}$

No NaNs

No $\pm\infty$

-0 allowed, not generated

Denormals (usually) tolerated, not (usually) generated

No H/W gradual u'flow; user trap routine can generate

Results chopped (except LRER and LRDR)

Larry Breed
14 July 1985

IBM VS FORTRAN

Exception	*Default*	*Alternative*
x/0	Message and 0 if x=0 else signed MAXREAL	User trap; DVCHK
Overflow	Message and signed MAXREAL	User trap; OVERFL

Underflow	Message and
	0
		User trap;
		OVERFL;
		XUFLOW

Inexact	Not available	None
Invalid Op'n	Message and see next pages	User trap

User can reset max # errors before halt (up to ∞)
User can reset max # messages produced for each error
User can trap on error to user-written (FORTRAN) routine
List of errors and count of each produced at pgm end

IBM VS FORTRAN

User trap example

```
external DIVIDE_FIX,OVER_AND_UNDERFLOW_FIX
...
call ERRSET(207, 10, 5,0,DIVIDE_FIX,207)
call ERRSET(208,256,-1,0,OVER_AND_UNDERFLOW_FIX,209)
...
```

User-written error handler for divide-by-zero (error 207) is named DIVIDE_FIX. Up to 10 errors can occur before program halt but standard error messages are printed for only the first 5. The handler for underflow and overflow (errors 208-209) is named OVER_AND_UNDERFLOW_FIX. Unlimited numbers of each may occur, but no messages are generated.

```
subroutine over_and_underflow_fix(icode,ierr,qval,iexponent)
real*16 qval
data huge/Z65100000/
if(ierrno.eq.209)go to 209        !fix overflows down below
if(qval.lt.huge)then
qval=0   !Number too small.  Generate true zero.
else     !Generate denormal result.
...
```

Error Code	FORTRAN Reference[1]	Invalid Argument Range	Options Standard Corrective Action[2], [3]	Options Parameters Passed to User Exit[4]
118	XA=X**Y	X < 0, Y ≠0	XA=|X|**Y	A, B, X, Y
119	DA=D**DB	D < 0, DB ≠ 0	DA=|D|**DB	A, B, D, DB
241	K=I**J	I=0, J≤0	K=0	A, B, I, J
242 [3]	Y=X**I	X=0, I≤0	If I=0, Y=1 If I < 0, Y=•	A, B, X, I
243 [3]	DA=D**I	D=0, I≤0	If I=0, Y=1 IF I < 0, Y=•	A, B, D, I
244	XA=X**Y	X=0, Y≤0	If Y=0, XA=1 If Y<0, XA=•	A, B, X, Y
245	DA=D**DB	D=0, DB ≤0	If DB=0, DA=1 If DB<0, DA=•	A, B, D, DB
246	CA=C**I	C=0 + 0i, I≤0	If I=0, C=1 + 0i IF I < 0, C=• + 0i	A, B, C, I
247	CDA=CD**I	C=0 + 0i, I≤0	If I=0, C=1 + 0i If I < 0, C=•+ 0i	A, B, CD, I
248 [3]	Q=QA**J	QA=0, J≤0	J < 0, Q=• J=0, Q=1	A, B, QA, J
249	Q=QA**QB	QA=0, QB≤0	QB < 0, Q=• QB=0, Q=1	A, B, QA, QB
		QA < 0, QB≠0	Q=|QA|**QB	
250	Q=QA**QB	$\log_2(QA) \times QB \geq 252$	Q=•	A, B, QA, QB
251	Y=SQRT (X)	X < 0	$Y=\|X\|^{1/2}$	A, B, X
252	Y=EXP (X)	X > 174.673	Y•	A, B, X

253	Y=ALOG (X)	X=0
		X < 0
			Y=-•
			Y=log|X|
				A, B, X
				A, B, X

	Y=ALOG10 (X)	X=0 X lt 0	Y=-• $Y=\log_{10}\|X\|$	A, B, X

254	Y=COS (X)
	Y=SIN (X)
		$\|X\| \geq (2^{18})\pi$	$Y = \sqrt{2/2}$	

255	Y=ATAN2 (X,XA)	X=0, XA=0	Y=0	A, B, X, XA

256	Y=SINH (X)
	Y=COSH (X)
		|H| ≥ 175.366	Y=(SIGN of X) •
			Y=•
				A, B, X

257	Y=ASIN (X)	|X| > 1	If X > 1.0, ASIN (X)=π/2 If X < - 1.0, ASIN (X)= − π/2	
	Y=ACOS (X)		If X > 1.0, ACCOS=0 If X <- 1.0, ACOS=π	

258	Y=TAN(X)
	Y=COTAN(X)
		$\|X\| \geq (2^{18})\pi$	Y=1	

	Y=COTAN (X)	X=0	Y=•	
260	Q=2**QA	QA >252	Q=•	A, B, QA
261	DA=DSQRT (D)	D < 0	$DA=\|D\|^{1/2}$	A, B, D
262	DA + DEXP (D)	D > 174.673	D=•	A, B, D

263	DA=DLOG (D)	D=0
		D < 0
			DA= —•
			DA=log|X|	

	DA=DLOG10 (D)	D=0 D < 0	DA=-• $DA=\log_{10}\|X\|$	A, B, D

Figure 55 (Part 1 of 3). Corrective Action after Mathematical Subroutine Error

Error Code	FORTRAN Reference[1]	Invalid Argument Range	Options Standard Corrective Action[2], [3]	Options Parameters Passed to User Exit[4]

264	DA=DSIN (D)
	DA=DCOS (D)
		|D| ≥ $(2^{50})\pi$	DA=$\sqrt{2/2}$	A, B, D

265	DA=DATAN2 (D.DB)	D=0. DB=0	DA=0	A, B, D, DB

266	DA=DSINH (D)
	DA=DCOSH (D)
		|D| ≥ 175.366	DA=(SIGN of X)•
			DA=•
				A, B, D

267	DA=DASIN (D)	|D| > 1	If D > 1.0, DASIN=π/2 If D < -1.0, DASIN= − π/2	
	DA=DACOS (D)		If D > 1.0, DACOS (D)=0 If D < - 1.0, DACOS (D)=π	

268	DA=DTAN (D)
	DA=DCOTAN (D)
		|X| ≥ $(2^{50})\pi$	DA=1	A, B, D

	DA=DCOTAN (D)	D=0	DA=•	A, B, D
270[6]	CQ=CQA**J	CQA=0 + 0i J ≤ 0	J=0, CQ=1 + 0,i J < 0, CQ=•+ 0,i	A, B, CQA, J
271[7]	Z=CEXP (C)	X_1 < 174.673	Z=•(cos X_2 + iSIN X_2)	A, B, C
272	Z=CEXP (C)	|X_2| ≥ $(2^{18})\pi$	Z=e^x + 0i	A, B, C
273	Z=CLOG (C)	C=0 + 0i	Z=-• + 0i	A, B, C
274	Z=CSIN (C)	|X_1| ≥ $(2^{18})\pi$	Z=0 + SINH (X_2)π	A, B, C
	Z=CCOS (C)		Z=COSH (X_2) + 0i	A, B, C
275	Z=CSIN (C)	X_2 < 174.673	Z=$\frac{\bullet}{2}$(SIN X_1 − iCOS X_1)	A, B, C
	Z=CCOS (C)		Z=$\frac{\bullet}{2}$(COS X_1 − iSIN X_1)	A, B, C
275	Z=CSIN (C)	X_2 < - 174.673	Z= $\frac{\bullet}{2}$(SIN X_1 − iCOS X_1)	A, B, C
	Z=CCOS (C)		Z= $\frac{\bullet}{2}$(COS X_1 + iSIN X_1)	A, B, C
276[8]	Z=CQEXP (CQ)	X_1 > 174.673	Z=•(COS X_2 − iSIN X_2)	A, B, CQ
277	Z=CQEXP (CQ)	|X_2| > 2_{100}	Z=e_{x1} − 0i	A, B, CQ
278	Z=CQLOG (CQ)	CQ=0 + 0i	Z=-• + 0i	A, B, CQ

279	Z=CQCOS (CQ)
	Z=CQCOS (CQ)
		|X_1| ≥ 2^{100}	Z=0 + DSINH (X_2)i
			Z=DCOSH (X_2) + 0i
				A, B, CQ

280	Z=CQSIN (CQ)	X_2 > 174.673	Z=$\frac{\bullet}{2}$(SIN X_1 + iCOS X_1)	A, B, CQ
	Z=CQCOS (CQ)		Z=$\frac{\bullet}{2}$(COS X_1 = iSIN X_1)	A, B, CQ

	Z=CQSIN (CQ)
	Z=CQCOS (CQ)
		X_2 < - 174.673	Z=$\frac{\bullet}{2}$(COS X_1 = iSIN X_1)
			Z=$\frac{\bullet}{2}$(COS X_1 = ISIN X_1)
				A, B, C Q

281[9]	Z=CDEXP (CD)	X_1 > 174.673	Z=•(COS X_2 − iSIN X_2)	A, B, CD
282	Z=CDEXP (CD)	|X_2| ≥ $(2^{50})\pi$	Z=$e^x{}_1$ + 0i	A, B, CD
283	Z=CDLOG (CD)	CD = 0 + 0i	Z=-• + 0i	A, B, CD
284	Z=CDSIN (DC)	|X_1| ≥ $(2^{50})\pi$	Z=0 + SINH (X_2)i	A, B, CD

Figure 55 (Part 2 of 3). Corrective Action after Mathematical Subroutine Error

Error Code	FORTRAN Reference[1]	Invalid Argument Range	Options Standard Corrective Action[2], [3]	Options Parameters Passed to User Exit[4]
	Z=CDCOS (CD)		$Z=\cosh(X_2) + 0i$	A, B, CD
285	Z=CDSIN (CD)	$X_2 > 174.673$	$Z=\frac{*}{2}(\text{SIN } X_1 + i\text{COS } X_1)$	A, B, CD
	Z=CDCOS (CD)		$Z=\frac{*}{2}(\text{COS } X_1 - i\text{SIN } X_1)$	A, B, CD
	Z=CDSIN (CD)	$X_2 < -174.673$	$Z=\frac{*}{2}(\text{SIN } X_1 - i\text{COS } X_1)$	A, B, CD
	Z=CDCOS (CD)		$Z=\frac{*}{2}(\text{COS } X_1 + i\text{SIN } X_1)$	A, B, CD
289	QA=QSQRT (Q)	$Q < 0$	$QA=\lvert Q\rvert^{1/2}$	A, B, Q
290	Y=GAMMA (X)	$X \le 2^{-252}$ or $X \ge 57.5744$	Y=*	A, B, X
291	Y=ALGAMA (X)	$X \le 0$ or $X \ge 4.2937 \times 10^{73}$	Y=*	A, B, X
292	QA=QEXP (Q)	$Q > 174.673$	QA=*	A, B, Q
293	QA=QLOG (Q)	$Q=0$ $Q < 0$	QA=-* $QA=\log\lvert X\rvert$	A, B, Q
	QA=QLOG10 (Q)	$Q=0$ $Q < 0$	QA=* $QA=\log_{10}\lvert X\rvert$	A, B, Q A, B, Q

294	QA=QSIN (Q)
	QA=QCOS (Q)
		$\lvert Q\rvert \ge 2^{100}$	$QA=\sqrt{2/2}$	A, B, Q

295	QA=QATAN2 (Q, QB)	Q=0, QB=0	QA=0	A, B, Q, QB

296	QA=QSINH (Q)
	QA=QCOSH (Q)
		$\lvert Q\rvert \ge 175.366$	QA=*(SIGN Q)
			QA=*
				A, B, Q

297	QA=QARSIN (Q)	$\lvert Q\rvert > 1$	If Q > 1.0, QARSIN=π/2 If Q < -1.0, QARSIN=π/2	A, B, Q A, B, Q
	QA=QARCOS (Q)		If Q > 1.0, QARCOS (Q)=0 If Q < -1.0, QARCOS (Q)=π	

298	QA=QTAN (Q)
	QA=QCOTAN (Q)
		$\lvert Q\rvert > 2^{100}$	QA=1	A, B, Q

299	QA=QTAN (Q)	Q is too close to an odd multiple of π/2	QA=*	A, B, Q
	QA=QCOTAN (Q)	Q is too close to a multiple of π	QA=*	A, B, Q
300	DA=DGAMMA (D)	$D \le 2^{-252}$ or $D \ge 57.5774$	DA=*	A, B, D
301	DA=DLGAMA (D)	$D \le 0$ or $D \ge 4.2937\ 10^{73}$	DA=*	

Figure 55 (Part 3 of 3). Corrective Action after Mathematical Subroutine Error

Floating Arithmetic on 4.3/RT

FP engines:
- software
- proprietary accelerators
- mc68881

Data formats: ieee 754 single and double

Evaluation precision: single, double, extended
depends on user option and hardware

Compile-time operations done in default modes

Math library: ieee version of Kahan's 4.3BSD libm
Decimal conversion conforms to 754
printf() and scanf() support "INF" and "NAN()"

Comparisons:
CMP for = and !=
CMPT for < <= > >= (NaNs trap)

Exception handling
Incomplete; surely buggy
Exception treated as SIGFPE signal
Operands and operation not available

Larry Breed
14 July 88

VAX Floating Point

* 4 formats, all with hidden bit

F 1, 8, 23+1
D 1, 8, 55+1
G 1, 11, 52+1
H 1, 15, 112+1

* Accuracy

* All operations are 0.5 ulp accurate, but no round to even

* Round to nearest always used, no rounding modes

* Special operands

* No Infinity, NaN, denorms

* -0 is a reserved operand

* Exceptions

* Overflow produces reserved operand and always takes a floating overflow trap

* Divide by zero produces reserved operand and always takes a floating divide by zero trap

* Underflow always produces zero and takes a floating underflow trap if it is enabled
Underflow traps are disable on every procedure call

* Reserved operand (-0) aborts instruction and always takes reserved operand trap

Earl Killian
14 July 86

VMS Exception handling -- General

* Machine faults are translated into software signals with specific condition numbers

* Programs can establish signal handlers on a per-procedure basis which are passed the condition number

* Handlers can pass the condition, unwind, or continue

* If program does not handle condition the a default handler prints something like

```
%SYSTEM-F-FLTOVF_F, arithmetic fault, floating overflow
                  at PC=00000711, PSL=03C0
%TRACE-F-TRACEBACK, symbolic stack dump follows
module name     routine name                  line
TEST            TEST                         30
```

and then stops

VMS Exception handling -- Floating point

1 / 0	-0	%SYSTEM-F-FLTDIV_F
0 / 0	-0	%SYSTEM-F-FLTDIV_F
Overflow	-0	%SYSTEM-F-FLTOVF_F
Underflow	0	%SYSTEM-F-FLTUND_F (if enabled
Bad input	0	%FOR-F-INPCONERR
sqrt(-1)	-0	%MTH-F-SQUROONEG, square root of negative value
sqrt(-0)	none	%SYSTEM-F-ROPRAND
log(0)	-0	%MTH-F-LOGZERNEG, logarithm of zero or negative value
log(-1)	-0	%MTH-F-LOGZERNEG
log(-0)	-0	%MTH-F-LOGZERNEG
exp(89)	-0	%MTH-F-FLOOVEMAT, floating overflow in math library
exp(-89)	0	
exp(-0)	1.0	
sin(1e38)	0.989164472	
atan(0,0)	none	%MTH-F-INVARGMAT
0**0	-0	

MIPS Floating Point Architecture

* Hardware + software fully conforms to IEEE 754 and also fully supports the standard's recommendations

* All floating point operations are fully emulated in kernel

* Programs run correctly in systems without FP hardware

* Particular hardware implementations may trap to software for any "difficult" operation

* Exceptions are recorded in sticky and non-sticky flags, and trap if enabled

* Exceptions are precise

EPC identifies faulting instruction; it and
all subsequent instructions have not been executed;
all previous instructions have been executed

Earl Killian
14 July 88

MIPS Floating Point Hardware

* IEEE 754 Single and Double formats supported in R3010 FP coprocessor

  * R3010 traps to kernel software for denorms and NaNs

  * Low-latency operations:

dp operation	cycles	ns
add, subtract	2	80
multiply	5	200
divide	19	760
move/abs/neg/compare	1	40

  * Integer, load/store, fp add, fp multiply, and fp divide units can operate in parallel

  * 4 DP, 7 SP LINPACK MFLOPS

MIPS Math Library

* Single and double versions of most functions

* 0.5 ulp SQRT done in software using Kahan algorithm

* LOG, EXP, SIN, COS, TAN, SINH, COSH, TANH based on Cody and Waite algorithms

  * rational or polynomial approximations

  * generally 100 to 130 cycles with 1.5ulp accuracy

* Other functions based on 4.3bsd math library (including DREM, POW, LOG1P, EXPM1, ATAN, ...)

* Binary <-> decimal conversion done using 64-bit integer arithmetic and fully accurate powers of 10

* Math library functions do not act as atomic operations; there is only partial attempt to obey rounding mode and signal exceptions as in hardware operations

* Math library does return NaN, Infinity etc. as appropriate
  sqrt(-1) = NaN
  log(0) = -Infinity
  log(-1) = NaN
  exp(1000) = Infinity
  sin(1e38) = NaN
  etc.