SN74ACT8847
64-Bit Floating Point Unit

- Meets **IEEE Standard** for Single- and Double-Precision Formats
- Performs **Floating Point** and **Integer** Add, Subtract, Multiply, Divide, Square Root, and Compare
- 64-Bit **IEEE Divide** in **11** Cycles, 64-Bit **Square Root** in **14** Cycles
- Performs Logical Operations and Logical Shifts
- Superset of TI's SN74ACT8837
- 30-ns, 40-ns and 50-ns Pipelined Performance
- Low-Power EPIC™ CMOS

The SN74ACT8847 is a high-speed, double-precision floating point and integer processor. It performs high-accuracy, scientific computations as part of a customized host processor or as a powerful stand-alone device. Its advanced math processing capabilities allow the chip to accelerate the performance of both CISC- and RISC- based systems.

High-end computer systems, such as graphics workstations, mini-computers and 32-bit personal computers, can utilize the single-chip 'ACT8847 for both floating point and integer functions.

EPIC is a trademark of Texas Instruments Incorporated.

Contents

Contents (Continued)

Contents (Continued)

Contents (Continued)

Contents (Concluded)

List of Illustrations

List of Illustrations (Continued)

List of Illustrations (Continued)

List of Illustrations (Continued)

List of Illustrations (Concluded)

List of Tables

List of Tables (Continued)

List of Tables (Continued)

List of Tables (Continued)

List of Tables (Concluded)

Overview

Using a top-down approach, this user guide contains the following major sections:

Introduction (to Microprogrammed Architectures and the 'ACT8847)
SN74ACT8847 Architecture
Microprogramming the 'ACT8847
Easy-to-Access Reference Guide
Application Notes

The SN74ACT8847 combines a multiplier and an arithmetic-logic unit in a single microprogrammable VLSI device. The 'ACT8847 is implemented in Texas Instruments one-micron CMOS technology to offer high speed and low power consumption with exceptional flexibility and functional integration. The FPUs can be microprogrammed to operate in multiple modes to support a variety of floating point applications.

The 'ACT8847 is fully compatible with the IEEE standard for binary floating point arithmetic, STD 754-1985. This FPU performs both single- and double-precision operations, integer operations, logical operations, and division and square root operations (as single microinstructions).

Understanding the 'ACT8847 Floating Point Unit

To support floating point processing in IEEE format, the 'ACT8847 may be configured for either single- or double-precision operation. Instruction inputs can be used to select three modes of operation, including independent ALU operations, independent multiplier operations, or simultaneous ALU and multiplier operations.

Three levels of internal data registers are available. The device can be used in flowthrough mode (all registers disabled), pipelined mode (all registers enabled), or in other available register configurations. An instruction register, a 64-bit constant register, and a status register are also provided.

Each FPU can handle three types of data input formats. The ALU accepts data operands in integer format or IEEE floating point format. A third type of operand, denormalized numbers, can also be processed after the ALU has converted them to "wrapped" numbers, which are explained in detail in a later section. The 'ACT8847 multiplier operates on normalized floating point numbers, wrapped numbers, and integer operands.

Microprogramming the 'ACT8847

The 'ACT8847 is a fully microprogrammable device. Each FPU operation is specified by a microinstruction or sequence of microinstructions which set up the control inputs of the FPU so that the desired operation is performed.

Support Tools

Texas Instruments has developed functional evaluation models of the 'ACT8847 in software which permit designers to simulate operation of the FPU. To evaluate the functions of an FPU, a designer can create a microprogram with sample data inputs, and the simulator will emulate FPU operation to produce sample data output files, as well as several diagnostic displays to show specific aspects of device operation. Sample microprogram sequences are included in this section.

Design Support

Texas Instruments Regional Technology Centers, staffed with systems-oriented engineers, offer a training course to assist users of TI LSI products and their application to digital processor systems. Specific attention is given to the understanding and generation of design techniques which implement efficient algorithms designed to match high-performance hardware capabilities with desired performance levels.

Information on VLSI devices and product support can be obtained from the following Regional Technology Centers:

Atlanta
Texas Instruments Incorporated
3300 N.E. Expressway, Building 8
Atlanta, GA 30341
404/662-7945

Chicago
Texas Instruments Incorporated
515 Algonquin
Arlington Heights, IL 60005
312/640-2909

Boston
Texas Instruments Incorporated
950 Winter Street, Suite 2800
Waltham, MA 02154
617/895-9100

Dallas
Texas Instruments Incorporated
10001 E. Campbell Road
Richardson, TX 75081
214/680-5066

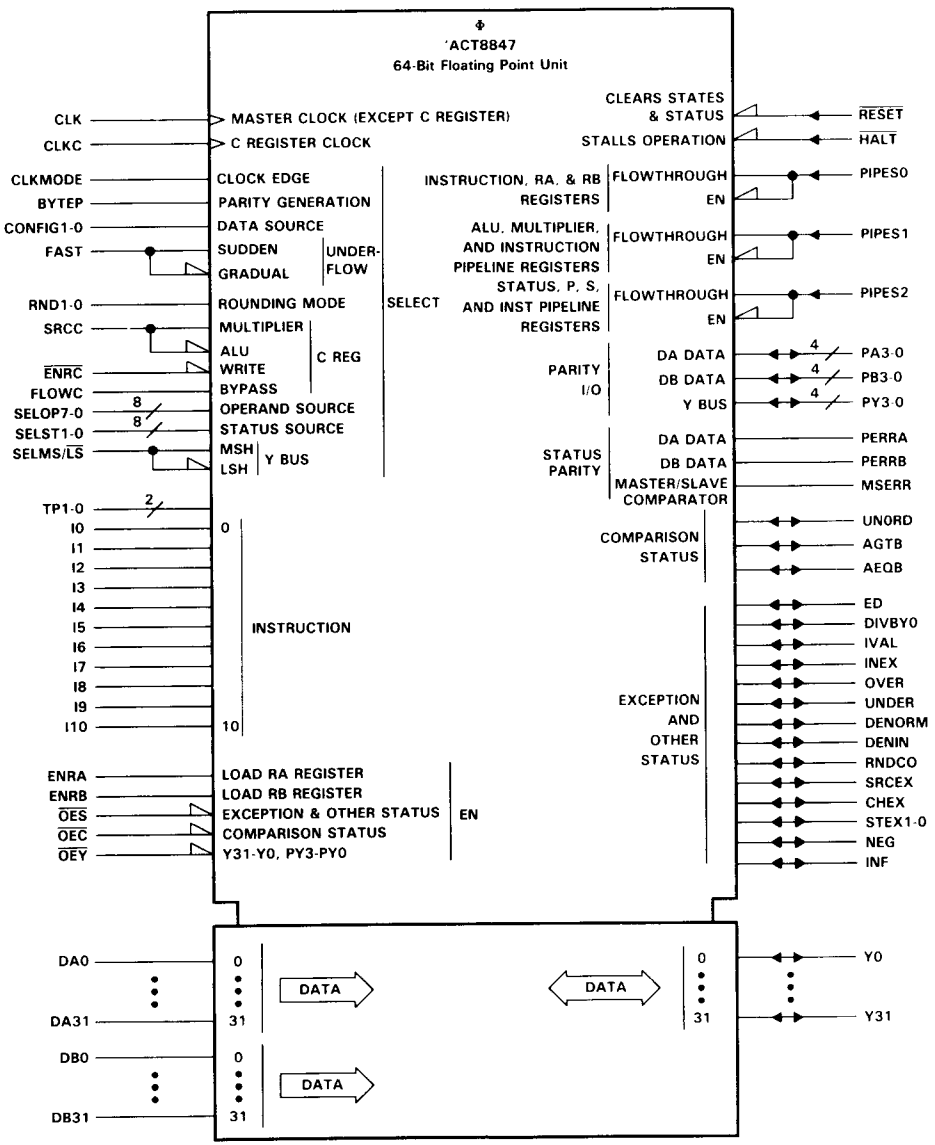
Northern California
Texas Instruments Incorporated
5353 Betsy Ross Drive
Santa Clara, CA 95054
408/748-2220

Southern California
Texas Instruments Incorporated
17891 Cartwright Drive
Irvine, CA 92714
714/660-8140

Design Expertise

Texas Instruments can provide in-depth technical design assistance through consultations with contract design services. Contact your local Field Sales Engineer for current information or contact VLSI Systems Engineering at 214/997-3970.

'ACT8847 Logic Symbol



7

SN74ACT8847

'ACT8847 Pin Descriptions

Pin descriptions and grid allocation for the 'ACT8847 are given on the following pages. The pin at location A1 has been omitted for indexing purposes.

208 PIN . . . GB PACKAGE
(TOP VIEW)

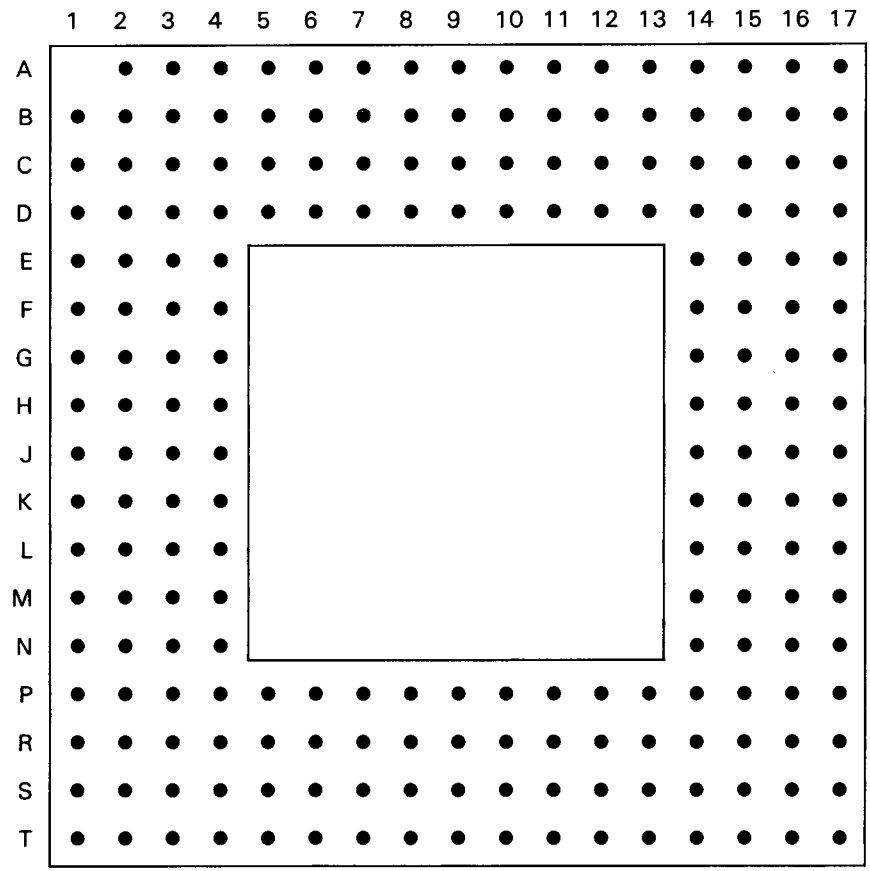


Table 1. 'ACT8847 Pin Grid Allocation

PIN NO.	NAME	PIN NO.	NAME	PIN NO.	NAME	PIN NO.	NAME	PIN NO.	NAME	PIN NO.	NAME
A1	missing	C2	Y0	E3	FAST	J15	FLOWC	P1	$\overline{ENRC}$	S1	NC
A2	INF	C3	Y3	E4	GND	J16	SRCC	P2	PIPES0	S2	PB0
A3	Y5	C4	Y6	E14	GND	J17	BYTEP	P3	$\overline{RESET}$	S3	DB0
A4	Y8	C5	Y9	E15	AGTB	K1	SELOP3	P4	PB1	S4	DB4
A5	Y11	C6	Y12	E16	AEQB	K2	SELOP4	P5	DB1	S5	DB11
A6	Y14	C7	Y15	E17	MSERR	K3	SELOP5	P6	DB5	S6	DB12
A7	Y17	C8	Y18	F1	I5	K4	GND	P7	DB9	S7	DB15
A8	Y20	C9	Y23	F2	I3	K14	GND	P8	DB16	S8	DB19
A9	Y21	C10	Y26	F3	RND0	K15	PA1	P9	DB21	S9	DB23
A10	Y24	C11	Y30	F4	GND	K16	PA2	P10	DB28	S10	DB26
A11	Y27	C12	PY1	F14	GND	K17	PA3	P11	DA0	S11	DB30
A12	Y29	C13	UNDER	F15	PERRA	L1	SELOP6	P12	DA4	S12	DA2
A13	PY0	C14	INEX	F16	$\overline{OEY}$	L2	SELOP7	P13	DA8	S13	DA6
A14	PY3	C15	DENIN	F17	$\overline{OES}$	L3	CLK	P14	DA12	S14	DA10
A15	IVAL	C16	SRCEX	G1	I7	L4	V_{CC}	P15	DA19	S15	DA14
A16	NEG	C17	CHEX	G2	I6	L14	GND	P16	DA22	S16	DA15
A17	NC	D1	I1	G3	I4	L15	DA30	P17	DA23	S17	DA17
B1	ED	D2	RND1	G4	V_{CC}	L16	DA31	R1	PIPES1	T1	NC
B2	Y2	D3	Y1	G14	V_{CC}	L17	PA0	R2	$\overline{HALT}$	T2	PB3
B3	Y4	D4	GND	G15	$\overline{OEC}$	M1	ENRB	R3	PB2	T3	DB3
B4	Y7	D5	V_{CC}	G16	SELMS/$\overline{LS}$	M2	ENRA	R4	DB2	T4	DB7
B5	Y10	D6	GND	G17	TEST1	M3	CLKC	R5	DB6	T5	DB8
B6	Y13	D7	GND	H1	I10	M4	GND	R6	DB10	T6	DB13
B7	Y16	D8	V_{CC}	H2	I9	M14	V_{CC}	R7	DB14	T7	DB17
B8	Y19	D9	GND	H3	I8	M15	DA27	R8	DB18	T8	DB20
B9	Y22	D10	GND	H4	GND	M16	DA28	R9	DB22	T9	DB24
B10	Y25	D11	V_{CC}	H14	GND	M17	DA29	R10	DB27	T10	DB25
B11	Y28	D12	GND	H15	TEST0	N1	CONFIG0	R11	DB31	T11	DB29
B12	Y31	D13	GND	H16	SELST1	N2	CONFIG1	R12	DA3	T12	DA1
B13	PY2	D14	V_{CC}	H17	SELST0	N3	CLKMODE	R13	DA7	T13	DA5
B14	OVER	D15	STEX1	J1	SELOP2	N4	PIPES2	R14	DA11	T14	DA9
B15	RNDCO	D16	STEX0	J2	SELOP1	N14	DA18	R15	DA16	T15	DA13
B16	DENORM	D17	UNORD	J3	SELOP0	N15	DA24	R16	DA20	T16	NC
B17	DIVBY0	E1	I2	J4	V_{CC}	N16	DA25	R17	DA21	T17	NC
C1	PERRB	E2	I0	J14	V_{CC}	N17	DA26				

Table 2. 'ACT8847 Pin Functional Description

PIN NAME	PIN NO.	I/O/Z†	DESCRIPTION
DATA BUS SIGNALS (96 PINS)			
DA0	P11	I	DA 32-bit input data bus. Data can be latched in a 64-bit temporary register or loaded directly into an input register
DA1	T12		
DA2	S12		
DA3	R12		
DA4	P12		
DA5	T13		
DA6	S13		
DA7	R13		
DA8	P13		
DA9	T14		
DA10	S14		
DA11	R14		
DA12	P14		
DA13	T15		
DA14	S15		
DA15	S16		
DA16	R15		
DA17	S17		
DA18	N14		
DA19	P15		
DA20	R16		
DA21	R17		
DA22	P16		
DA23	P17		
DA24	N15		
DA25	N16		
DA26	N17		
DA27	M15		
DA28	M16		
DA29	M17		
DA30	L15		
DA31	L16		
DB0	S3	I	DB 32-bit input data bus. Data can be latched in a 64-bit temporary register or loaded directly into an input register.
DB1	P5		
DB2	R4		
DB3	T3		
DB4	S4		
DB5	P6		
DB6	R5		
DB7	T4		
DB8	T5		
DB9	P7		
DB10	R6		

†Input, output, and high-impedance state.

Table 2. 'ACT8847 Pin Functional Description (Continued)

PIN NAME	PIN NO.	I/O/Z†	DESCRIPTION
DATA BUS SIGNALS (96 PINS)			
DB11	S5	I	DB32-bit input data bus. Data can be latched in a 64-bit temporary register or loaded directly into an input register.
DB12	S6		
DB13	T6		
DB14	R7		
DB15	S7		
DB16	P8		
DB17	T7		
DB18	R8		
DB19	S8		
DB20	T8		
DB21	P9		
DB22	R9		
DB23	S9		
DB24	T9		
DB25	T10		
DB26	S10		
DB27	R10		
DB28	P10		
DB29	T11		
DB30	S11		
DB31	R11		
Y0	C2	I/O/Z	32-bit Y output data bus
Y1	D3		
Y2	B2		
Y3	C3		
Y4	B3		
Y5	A3		
Y6	C4		
Y7	B4		
Y8	A4		
Y9	C5		
Y10	B5		
Y11	A5		
Y12	C6		
Y13	B6		
Y14	A6		
Y15	C7		
Y16	B7		
Y17	A7		
Y18	C8		
Y19	B8		
Y20	A8		

†Input, output, and high-impedance state.

Table 2. 'ACT8847 Pin Functional Description (Continued)

PIN NAME	PIN NO.	I/O/Z†	DESCRIPTION
DATA BUS SIGNALS (96 PINS)			
Y21 Y22 Y23 Y24 Y25 Y26 Y27 Y28 Y29 Y30 Y31	A9 B9 C9 A10 B10 C10 A11 B11 A12 C11 B12	I/O	32-bit Y output data bus
PARITY AND MASTER/SLAVE SIGNALS (16 PINS)			
BYTEP	J17	I	When high, selects parity generation for each byte of input (four parity bits for each bus). When low, selects parity generation for whole 32-bit input (one parity bit for each bus). Even parity is used.
MSERR	E17	O	Master/Slave error output pin
PA0 PA1 PA2 PA3	L17 K15 K16 K17	I	Parity inputs for DA data
PB0 PB1 PB2 PB3	S2 P4 R3 T2	I	Parity inputs for DB data
PERRA	F15	O	DA data parity error output. When high, signals a byte or word has failed an even parity check.
PERRB	C1	O	DB data parity error output. When high, signals a byte or word has failed an even parity check.
PY0 PY1 PY2 PY3	A13 C12 B13 A14	I/O/Z	Y port parity data
CLOCK, CONTROL, AND INSTRUCTION SIGNALS (46 PINS)			
CLK	L3	I	Master clock for all registers except C register
CLKC	M3	I	C register clock
CLKMODE	N3	I	Selects whether temporary register loads only on rising clock edge (CLKMODE = L) or on falling edge (CLKMODE = H).

†Input, output, and high-impedance state.

Table 2. 'ACT8847 Pin Functional Description (Continued)

PIN NAME	PIN NO.	I/O/Z†	DESCRIPTION
CLOCK, CONTROL, AND INSTRUCTION SIGNALS (46 PINS)			
CONFIG0 CONFIG1	N1 N2	I	Select data sources for RA and RB registers from DA bus, DB bus and temporary register
ENRA	M2	I	When high, enables loading of RA register on a rising clock edge if the RA register is not disabled (see PIPESO below).
ENRB	M1	I	When high, enables loading of RB register on a rising clock edge if the RB register is not disabled (see PIPESO below).
$\overline{\text{ENRC}}$	P1	I	When low, enables write to C register when CLKC goes high.
FAST	E3	I	When low, selects gradual underflow (IEEE model). When high, selects sudden underflow, forcing all denormalized inputs and outputs to zero.
FLOWC	J15	I	When high, causes product or sum to bypass C register, so that product or sum appears on the C register output bus. Timing is similar to P register or S register feedback operands. C register remains unchanged. Product or sum may also be simultaneously fed back in usual manner (not through C register).
$\overline{\text{HALT}}$	R2	I	Stalls operation without altering contents of instruction or data registers (except the CREG, which has a separate write enable). Active low.
I0 I1 I2 I3 I4 I5 I6 I7 I8 I9 I10	E2 D1 E1 F2 G3 F1 G2 G1 H3 H2 H1	I	Instruction inputs
$\overline{\text{OEC}}$	G15	I	Comparison status output enable. Active low.
$\overline{\text{OES}}$	F17	I	Exception status and other status output enable. Active low.
$\overline{\text{OEY}}$	F16	I	Y bus output enable. Active low.
PIPESO	P2	I	When low, enables instruction register and, depending on setting of ENRA and ENRB, the RA and RB input registers. When high, puts instruction, RA and RB registers in flowthrough mode.

†Input, output, and high-impedance state.

Table 2. 'ACT8847 Pin Functional Description (Continued)

PIN NAME	PIN NO.	I/O/Z†	DESCRIPTION
CLOCK, CONTROL, AND INSTRUCTION SIGNALS (46 PINS)			
PIPES1	R1	I	When low, enables pipeline registers in ALU and multiplier. When high, puts pipeline registers in flowthrough mode.
PIPES2	N4	I	When low, enables status register, product (P) and sum (S) registers. When high, puts status register, P and S registers in flowthrough mode.
$\overline{\text{RESET}}$	P3	I	Clears internal states, status, and exception disable register. Contents of internal pipeline registers are lost. Does not affect other data registers. Active low.
RND0 RND1	F3 D2	I	Rounding mode control pins. Select four IEEE rounding modes.
RNDC0	B15	I/O/Z	When high, indicates the mantissa of a number has been increased in magnitude by rounding.
SELOP0 SELOP1 SELOP2 SELOP3 SELOP4 SELOP5 SELOP6 SELOP7	J3 J2 J1 K1 K2 K3 L1 L2	I	Select operand sources for multiplier and ALU
SELST0 SELST1	H17 H16	I	Select status source during chained operation
SELMS/$\overline{\text{LS}}$	G16	I	When low, selects LSH of 64-bit result to be output on the Y bus. When high, selects MSH of 64-bit result. (No effect on single-precision operations.)
SRCC	J16	I	When low, selects ALU as data source for C register. When high, selects multiplier as data source for C register.
TEST0 TEST1	H15 G17	I	Test pins
STATUS SIGNALS (17 PINS)			
AEQB	E16	I/O/Z	Comparison status or zero detect pin. When high, indicates that A and B operands are equal during a compare operation in the ALU. If not a compare, a high signal indicates a zero result on the Y bus.

†Input, output, and high-impedance state.

Table 2. 'ACT8847 Pin Functional Description (Continued)

PIN NAME	PIN NO.	I/O/Z†	DESCRIPTION
STATUS SIGNALS (17 PINS)			
AGTB	E15	I/O/Z	Comparison status pin. When high, indicates that A operand is greater than B operand.
CHEX	C17	I/O/Z	Status pin indicating an exception during a chained function. If I6 is low, indicates the multiplier is the source of an exception. If I6 is high, indicates the ALU is the source of an exception.
DENIN	C15	I/O/Z	Status pin indicating a denormal input to the multiplier. When DENIN goes high, the STEX pins indicate which port had the denormal input.
DENORM	B16	I/O/Z	Status pin indicating a denormal output from the ALU or a wrapped output from the multiplier. In FAST mode, causes the result to go to zero when DENORM is high.
DIVBY0	B17	I/O/Z	Status pin indicating an attempted operation involved dividing by zero
ED	B1	I/O/Z	Exception detect status signal representing logical OR of all enabled exceptions in the exception disable register
INEX	C14	I/O/Z	Status pin indicating an inexact output
INF	A2	I/O/Z	Status pin. When high, indicates output value is infinity.
IVAL	A15	I/O/Z	Status pin indicating that an invalid operation or a nonnumber (NaN) has been input to the multiplier or ALU.
NEG	A16	I/O/Z	Status pin. When high, indicates result has negative sign.
OVER	B14	I/O/Z	Status pin indicating that the result is greater the largest allowable value for specified format (exponent overflow).
SRCEX	C16	I/O/Z	Status pin indicating source of exception, either ALU (SRCEX = L) or multiplier (SRCEX = H).
STEX0 STEX1	D16 D15	I/O/Z	Status pins indicating that a nonnumber (NaN) or denormal number has been input on A port (STEX1) or B port (STEX0).
UNDER	C13	I/O/Z	Status pin indicating that a result is inexact and less than minimum allowable value for format (exponent underflow).
UNORD	D17	I/O/Z	Comparison status pin indicating that the two inputs are unordered because at least one of them is a nonnumber (NaN).

†Input, output, and high-impedance state.

Table 2. 'ACT8847 Pin Functional Description (Concluded)

PIN NAME	PIN NO.	I/O/Z†	DESCRIPTION
SUPPLY AND N/C SIGNALS (33 PINS)			
V_{CC}	D5	I	5-V supply voltage pins
V_{CC}	D8		
V_{CC}	D11		
V_{CC}	D14		
V_{CC}	G4		
V_{CC}	G14		
V_{CC}	J4		
V_{CC}	J14		
V_{CC}	L4		
V_{CC}	M14		
GND	D4	I	Ground pins. NOTE: All ground pins should be used and connected.
GND	D6		
GND	D7		
GND	D9		
GND	D10		
GND	D12		
GND	D13		
GND	E4		
GND	E14		
GND	F4		
GND	F14		
GND	H4		
GND	H14		
GND	K4		
GND	K14		
GND	L14		
GND	M4		
NC	A17		No internal connection. Pins should be left floating.
NC	S1		
NC	T1		
NC	T16		
NC	T17		

†Input, output, and high-impedance state.

'ACT8847 Specifications

absolute maximum ratings over operating free-air temperature range (unless otherwise noted)†

Supply voltage, V_{CC} . . . . . −0.5 V to 6 V
Input clamp current, I_{IK} ($V_I < 0$ or $V_I > V_{CC}$) . . . . . ±20 mA
Output clamp current, I_{OK} ($V_O < 0$ or $V_O > V_{CC}$) . . . . . ±50 mA
Continuous output current, I_O ($V_O = V_{CC}$) . . . . . ±50 mA
Continuous current through V_{CC} or GND pins . . . . . ±100 mA
Operating free-air temperature range . . . . . 0°C to 70°C
Storage temperature range . . . . . −65°C to 150°C

†Stresses beyond those listed under "absolute maximum ratings" may cause permanent damage to the device. These are stress ratings only and functional operation of the device at these or any other conditions beyond those indicated under "recommended operating conditions" is not implied. Exposure to absolute-maximum-rated conditions for extended periods may affect device reliability.

recommended operating conditions

PARAMETER		SN74ACT8847			UNIT
		MIN	NOM	MAX	
V_{CC}	Supply voltage	4.75	5.0	5.25	V
V_{IH}	High-level input voltage	2		V_{CC}	V
V_{IL}	Low-level input voltage	0		0.8	V
I_{OH}	High-level output current			−8	mA
I_{OL}	Low-level output current			8	mA
V_I	Input voltage	0		V_{CC}	V
V_O	Output voltage	0		V_{CC}	V
dt/dv	Input transition rise or fall rate	0		15	ns/V
T_A	Operating free-air temperature	0		70	°C

electrical characteristics over recommended operating free-air temperature range (unless otherwise noted)

PARAMETER	TEST CONDITIONS	V_{CC}	T_A = 25°C			SN74ACT8847			UNIT
			MIN	TYP	MAX	MIN	TYP	MAX	
V_{OH}	I_{OH} = −20 μA	4.75 V		4.74		4.55			V
		5.25 V		5.24		5.05			
	I_{OH} = −8 mA	4.75 V				3.7			
		5.25 V				4.7			
V_{OL}	I_{OL} = 20 μA	4.75 V		0.01				0.10	V
		5.25 V		0.01				0.10	
	I_{OL} = 8 mA	4.75 V						0.45	
		5.25 V						0.45	
I_I	V_I = V_{CC} or 0	5.25 V						±5	μA
I_{OZ}	V_I = V_{CC} or 0, I_O	5.25 V						±10	μA
I_{CCQ}	V_I = V_{CC} or 0, I_O	5.25 V						200	μA
C_i	V_i = V_{CC} or 0	5 V					10		pF

switching characteristics

NO.	PARAMETER	FROM (INPUT)	TO (OUTPUT)	PIPELINE CONTROLS PIPES2-PIPES0	SN74ACT8847-30 MIN	SN74ACT8847-30 MAX	UNIT
1	t_{pd1}	DA/DB/Inst	Y OUTPUT	111		†	ns
2	t_{pd2}	INPUT REG	Y OUTPUT	110		70	ns
		INPUT REG	STATUS	110		70	
3	t_{pd3}	PIPELN REG	Y OUTPUT	10X		48	ns
		PIPELN REG	STATUS	10X		48	
4	t_{pd4}	OUTPUT REG	Y OUTPUT	0XX		20	ns
		OUTPUT REG	STATUS	0XX		20	
5	t_{pd5}	SELMS/$\overline{LS}$	Y OUTPUT	XXX		18	ns
6	t_{pd6}	CLK↑	Y OUTPUT INVALID	all but 111	3.0		ns
7	t_{pd7}	CLK↑	STATUS INVALID	all but 111	3.0		ns
8	t_{pd8}	SELMS/$\overline{LS}$	Y OUTPUT INVALID	XXX	1.5		ns
9	t_{d1}‡	CLK↑	CLK↑	010	56		ns
10	t_{d2}‡	CLK↑	CLK↑	000	30		
11	t_{d3}	Delay time, CLKC after CLK to insure data captured in C register is data clocked into sum or product register by that clock. (PIPES2-PIPES0 = 0XX)			12	t_d-0§	
12	t_{en1}	$\overline{OEY}$	Y OUTPUT	XXX		12	ns
13	t_{en2}	$\overline{OEC}$, $\overline{OES}$	STATUS	XXX		12	
14	t_{dis1}	$\overline{OEY}$	Y OUTPUT	XXX		12	
15	t_{dis2}	$\overline{OEC}$, $\overline{OES}$	STATUS	XXX		12	

†This parameter no longer tested and will be deleted on next Data Manual revision.
‡Minimum clock cycle period not guaranteed when operands are fed back using FLOWC to bypass the C register and operands are used on the same clock cycle.
§t_d is the clock cycle period.

setup and hold times

NO.	PARAMETER		PIPELINE CONTROLS PIPES2-PIPES0	SN74ACT8847-30 MIN	SN74ACT8847-30 MAX	UNIT
16	t_{su1}	Inst/control before CLK↑	XX0	12		ns
17	t_{su2}	DA/DB before CLK↑	XX0	11		
18	t_{su3}	DA/DB before 2nd CLK↑ (DP)	XX1	40		
19	t_{su4}	CONFIG1-0 before CLK↑	XX0	12		
20	t_{su5}	SRCC before CLKC↑	XXX	10		
21	t_{su6}	$\overline{\text{RESET}}$ before CLK↑	XX0	12		
22	t_{h1}	Inst/control after CLK↑	XXX	1		ns
23	t_{h2}	DA/DB after CLK↑	XXX	1		
24	t_{h3}	SRCC after CLKC↑	XXX	1		
25	t_{h4}	$\overline{\text{RESET}}$ after CLK↑	XX0	6		

CLK/RESET requirements

PARAMETER			SN74ACT8847-30 MIN	SN74ACT8847-30 MAX	UNIT
t_w	Pulse duration	CLK high	10		ns
		CLK low	10		
		$\overline{\text{RESET}}$	10		

switching characteristics

NO.	PARAMETER	FROM (INPUT)	TO (OUTPUT)	PIPELINE CONTROLS PIPES2-PIPES0	SN74ACT8847-40 MIN	SN74ACT8847-40 MAX	UNIT
1	t_{pd1}	DA/DB/Inst	Y OUTPUT	111		†	ns
2	t_{pd2}	INPUT REG	Y OUTPUT	110		90	ns
		INPUT REG	STATUS	110		90	
3	t_{pd3}	PIPELN REG	Y OUTPUT	10X		60	ns
		PIPELN REG	STATUS	10X		60	
4	t_{pd4}	OUTPUT REG	Y OUTPUT	0XX		24	ns
		OUTPUT REG	STATUS	0XX		24	
5	t_{pd5}	SELMS/$\overline{LS}$	Y OUTPUT	XXX		20	ns
6	t_{pd6}	CLK↑	Y OUTPUT INVALID	all but 111	3.0		ns
7	t_{pd7}	CLK↑	STATUS INVALID	all but 111	3.0		ns
8	t_{pd8}	SELMS/$\overline{LS}$	Y OUTPUT INVALID	XXX	1.5		ns
9	t_{d1}‡	CLK↑	CLK↑	010	72		ns
10	t_{d2}‡	CLK↑	CLK↑	000	40		
11	t_{d3}	Delay time, CLKC after CLK to insure data captured in C register is data clocked into sum or product register by that clock. (PIPES2-PIPES0 = 0XX)			16	t_d-0§	ns
12	t_{en1}	$\overline{OEY}$	Y OUTPUT	XXX		16	ns
13	t_{en2}	$\overline{OEC}$, $\overline{OES}$	STATUS	XXX		16	
14	t_{dis1}	$\overline{OEY}$	Y OUTPUT	XXX		16	
15	t_{dis2}	$\overline{OEC}$, $\overline{OES}$	STATUS	XXX		16	

†This parameter no longer tested and will be deleted on next Data Manual revision.
‡Minimum clock cycle period not guaranteed when operands are fed back using FLOWC to bypass the C register and operands are used on the same cycle.
§t_d is the clock cycle period.

setup and hold times

NO.	PARAMETER		PIPELINE CONTROLS PIPES2-PIPES0	SN74ACT8847-40 MIN	SN74ACT8847-40 MAX	UNIT
16	t_{su1}	Inst/control before CLK↑	XX0	14		ns
17	t_{su2}	DA/DB before CLK↑	XX0	13		
18	t_{su3}	DA/DB before 2nd CLK↑ (DP)	XX1	52		
19	t_{su4}	CONFIG1-0 before CLK↑	XX0	14		
20	t_{su5}	SRCC before CLKC↑	XXX	14		
21	t_{su6}	$\overline{\text{RESET}}$ before CLK↑	XX0	14		
22	t_{h1}	Inst/control after CLK↑	XXX	3		ns
23	t_{h2}	DA/DB after CLK↑	XXX	3		
24	t_{h3}	SRCC after CLKC↑	XXX	3		
25	t_{h4}	$\overline{\text{RESET}}$ after CLK↑	XX0	6		

CLK/RESET requirements

PARAMETER			SN74ACT8847-40 MIN	SN74ACT8847-40 MAX	UNIT
t_w	Pulse duration	CLK high	15		ns
		CLK low	15		
		$\overline{\text{RESET}}$	12		

switching characteristics

NO.	PARAMETER	FROM (INPUT)	TO (OUTPUT)	PIPELINE CONTROLS PIPES2-PIPES0	SN74ACT8847-50 MIN	SN74ACT8847-50 MAX	UNIT
1	t_{pd1}	DA/DB/Inst	Y OUTPUT	111		†	ns
2	t_{pd2}	INPUT REG	Y OUTPUT	110		120	ns
		INPUT REG	STATUS	110		120	
3	t_{pd3}	PIPELN REG	Y OUTPUT	10X		75	ns
		PIPELN REG	STATUS	10X		75	
4	t_{pd4}	OUTPUT REG	Y OUTPUT	0XX		36	ns
		OUTPUT REG	STATUS	0XX		36	
5	t_{pd5}	SELMS/$\overline{LS}$	Y OUTPUT	XXX		24	ns
6	t_{pd6}	CLK↑	Y OUTPUT INVALID	all but 111	3.0		ns
7	t_{pd7}	CLK↑	STATUS INVALID	all but 111	3.0		ns
8	t_{pd8}	SELMS/$\overline{LS}$	Y OUTPUT INVALID	XXX	1.5		ns
9	t_{d1}‡	CLK↑	CLK↑	010	100		ns
10	t_{d2}‡	CLK↑	CLK↑	000	50		
11	t_{d3}	Delay time, CLKC after CLK to insure data captured in C register is data clocked into sum or product register by that clock. (PIPES2-PIPES0 = 0XX)			16	t_d-0§	ns
12	t_{en1}	$\overline{OEY}$	Y OUTPUT	XXX		20	ns
13	t_{en2}	$\overline{OEC}$, $\overline{OES}$	STATUS	XXX		20	
14	t_{dis1}	$\overline{OEY}$	Y OUTPUT	XXX		20	
15	t_{dis2}	$\overline{OEC}$, $\overline{OES}$	STATUS	XXX		20	

†This parameter no longer tested and will be deleted on next Data Manual revision.
‡Minimum clock cycle period not guaranteed when operands are fed back using FLOWC to bypass the C register and operands are used on the same cycle.
§t_d is the clock cycle period.

setup and hold times

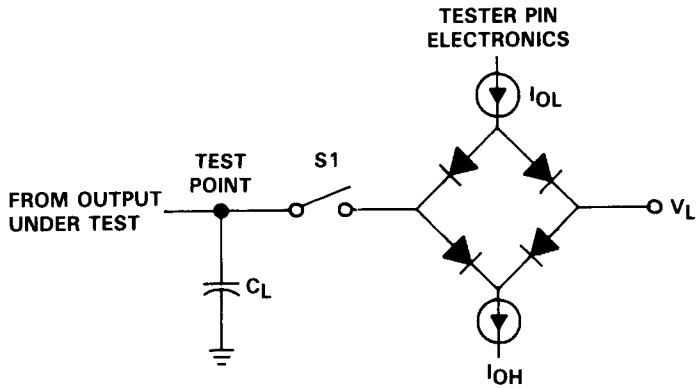
NO.	PARAMETER		PIPELINE CONTROLS PIPES2-PIPES0	SN74ACT8847-50 MIN	SN74ACT8847-50 MAX	UNIT
16	t_{su1}	Inst/control before CLK↑	XX0	16		ns
17	t_{su2}	DA/DB before CLK↑	XX0	16		
18	t_{su3}	DA/DB before 2nd CLK↑ (DP)	XX1	75		
19	t_{su4}	CONFIG1-0 before CLK↑	XX0	18		
20	t_{su5}	SRCC before CLKC↑	XXX	16		
21	t_{su6}	$\overline{\text{RESET}}$ before CLK↑	XX0	16		
22	t_{h1}	Inst/control after CLK↑	XXX	3		ns
23	t_{h2}	DA/DB after CLK↑	XXX	3		
24	t_{h3}	SRCC after CLKC↑	XXX	3		
25	t_{h4}	$\overline{\text{RESET}}$ after CLK↑	XX0	6		

CLK/RESET requirements

PARAMETER			SN74ACT8847-50 MIN	SN74ACT8847-50 MAX	UNIT
t_w	Pulse duration	CLK high	15		ns
		CLK low	15		
		$\overline{\text{RESET}}$	15		

'ACT8847 Load Circuit

The load circuit for the 'ACT8847 is shown in Figure 1.

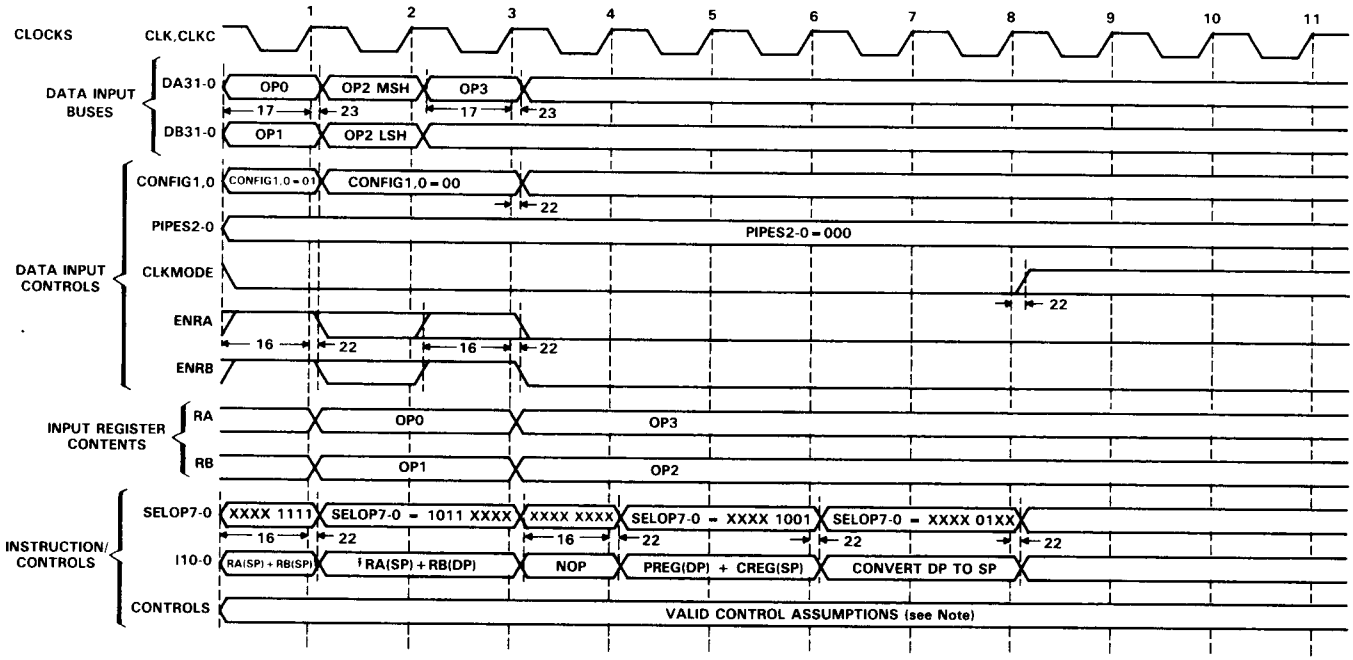


TIMING PARAMETER		C_L†	I_{OL}	I_{OH}	V_L	S1
t_{en}	t_{PZH}	50 pF	1 mA	−1 mA	1.5 V	CLOSED
	t_{PLH}					
t_{dis}	t_{PHZ}	50 pF	16 mA	−16 mA	1.5 V	CLOSED
	t_{PLZ}					
t_{pd}		50 pF	–	–	–	OPEN

†C_L includes probe and test fixture capacitance.

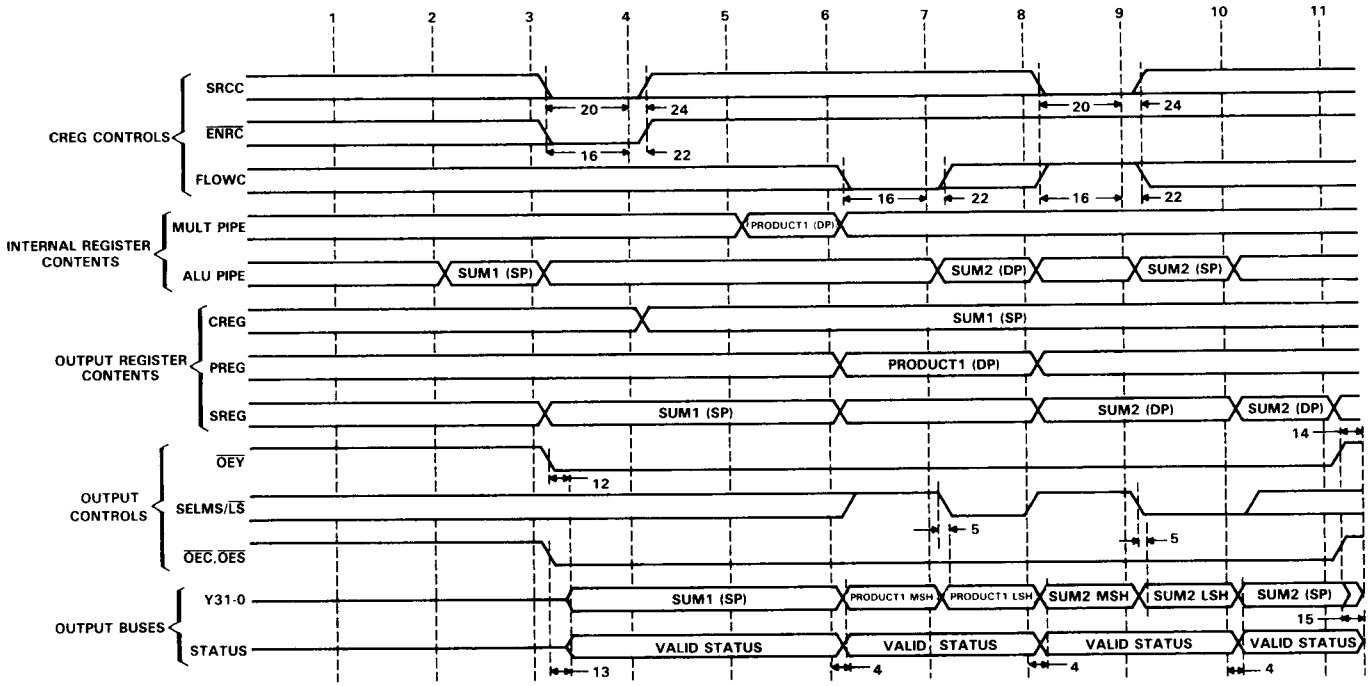
NOTE: All input pulses are supplied by generators having the following characteristics: PRR ≤ 1 MHz, Z_O = 50 Ω, t_r ≤ 6 ns, t_f ≤ 6 ns.

Figure 1. Load Circuit



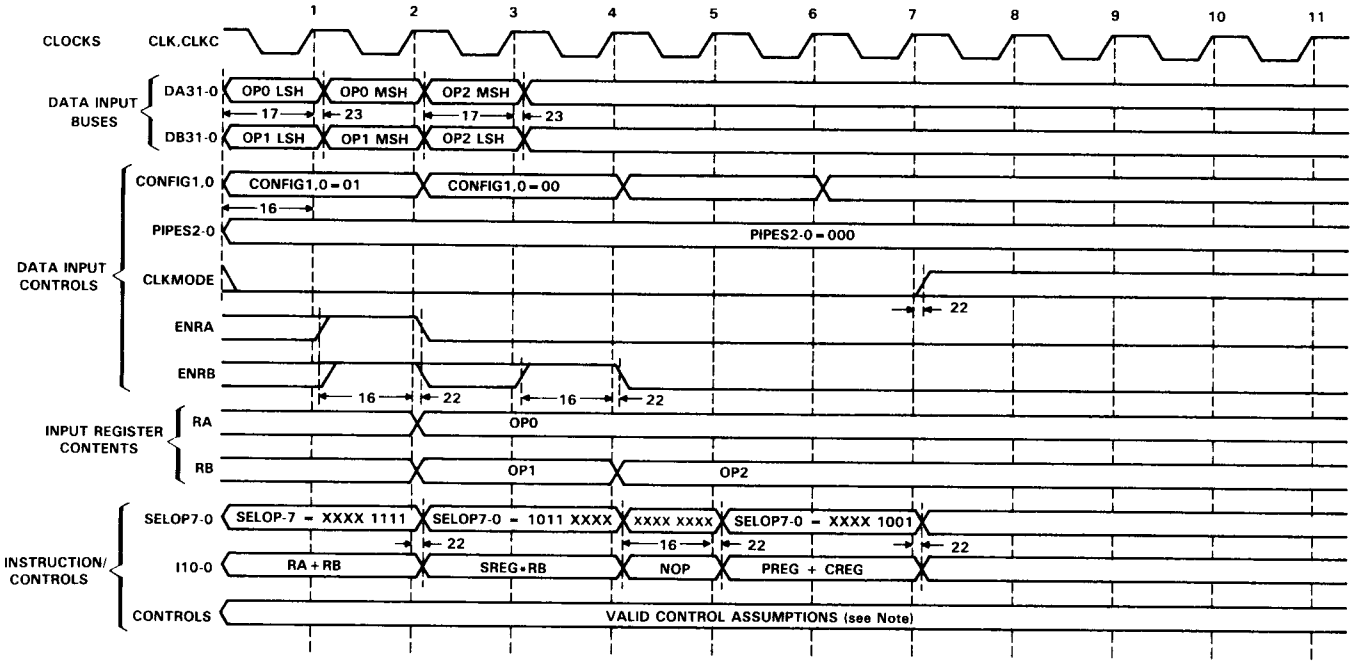
NOTES: Assume the following mixed precision operation.
Single precision OP0 + OP1 = RA + RB → SUM1 → CREG, where OP0 is SP and OP1 is SP.
Mixed precision OP2 * OP2 = RA * RB → PRODUCT1, where OP3 is SP and OP2 is DP.
NOP (must be inserted).
Mixed precision (OP3 * OP2) + (OP0 + OP1) = PREG + CREG → SUM2 (DP), and then convert to SP.
Assume valid control signals for FAST, $\overline{\text{HALT}}$ = 1, PIPES2-0 = 000 (fully pipelined mode), $\overline{\text{RESET}}$ = 1, RND1-0, SELST1-0 = 11, TP1-0 = 11.

Figure 2a. Timing Diagram for: SP ALU → DP MULT → DP ALU → Convert DP to SP



NOTES: Assume the following mixed precision operation.
Single precision OP0 + OP1 = RA + RB → SUM1 → CREG, where OP0 is SP and OP1 is SP.
Mixed precision OP2 * OP2 = RA * RB → PRODUCT1, where OP3 is SP and OP2 is DP.
NOP (must be inserted).
Mixed precision (OP3 * OP2) + (OP0 + OP1) = PREG + CREG → SUM2 (DP), and then convert to SP.
Assume valid control signals for FAST, $\overline{\text{HALT}}$ = 1, PIPES2-0 = 000 (fully pipelined mode), $\overline{\text{RESET}}$ = 1, RND1-0, SELST1-0 = 11, TP1-0 = 11.

Figure 2b. Timing Diagram for: SP ALU → DP MULT → DP ALU → Convert DP to SP



NOTES: Assume the following double precision operation.

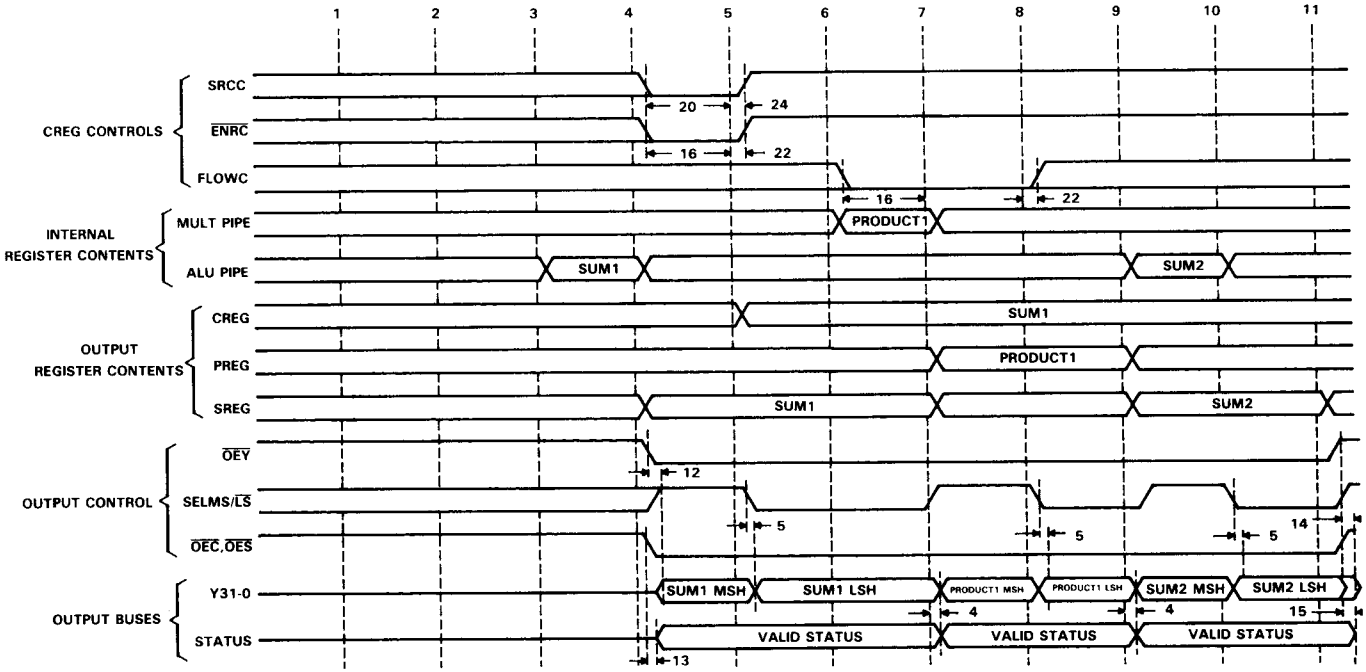
OP0 + OP1 = RA + RB → SUM1 → CREG

(OP0 + OP1) * OP2 = SREG * RB → PRODUCT1

[(OP0 + OP1) * OP2)] + (OP0 + $\overline{OP1}$) = PREG + CREG → SUM2

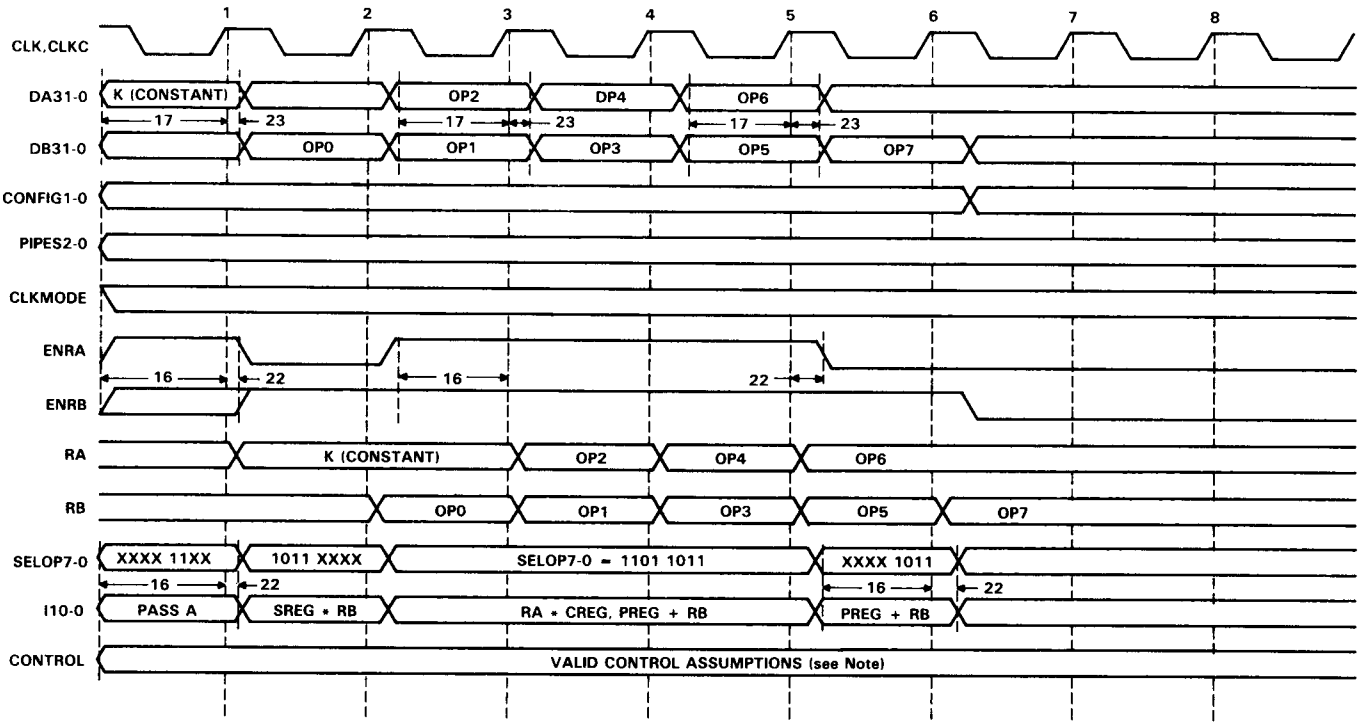
Assume valid control signals for FAST, $\overline{HALT}$ = 1, PIPES2-0 = 000 (fully pipelined mode), $\overline{RESET}$ = 1, RND1-0, SELST1-0 = 11, TP1-0 = 11.

Figure 3a. Timing Diagram for: DP ALU → DP MULT → DP ALU



NOTES: Assume the following double precision operation.
OP0 + OP1 = RA + RB → SUM1 → CREG
(OP0 + OP1) * OP2 = SREG * RB → PRODUCT1
[(OP0 + OP1) * OP2)] + (OP0 + $\overline{OP1}$) = PREG + CREG → SUM2
Assume valid control signals for FAST, $\overline{HALT}$ = 1, PIPES2-0 = 000 (fully pipelined mode), $\overline{RESET}$ = 1, RND1-0, SELST1-0 = 11, TP1-0 = 11.

Figure 3b. Timing Diagram for: DP ALU → DP MULT → DP ALU

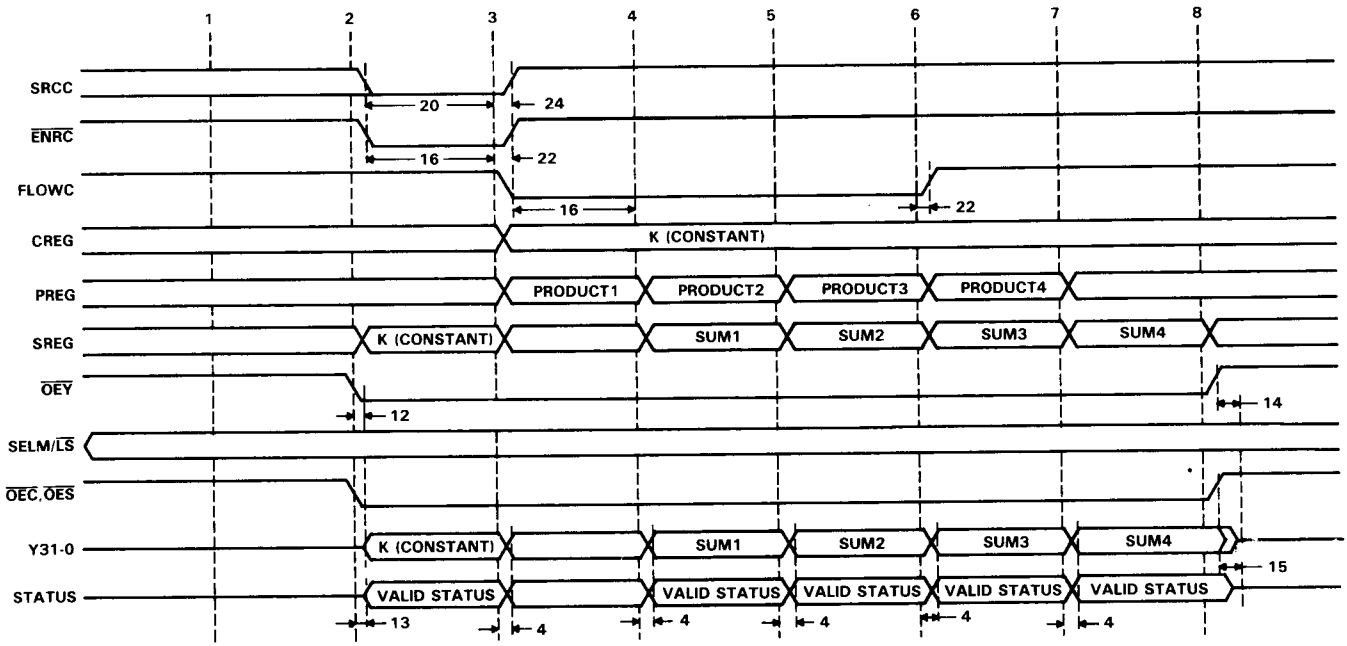


NOTES: Assume the following single precision operations.

(K * OP0) + OP1 = PRODUCT1 + OP1 → SUM1
(K * OP2) + OP3 = PRODUCT2 + OP3 → SUM2
(K * OP4) + OP5 = PRODUCT3 + OP5 → SUM3
(K * OP6) + OP7 = PRODUCT4 + OP7 → SUM4

Assume valid control signals for FAST, $\overline{\text{HALT}}$ = 1, PIPES2-0 = 010, $\overline{\text{RESET}}$ = 1, RND1-0, SELST1-0 = 11, TP1-0 = 11.

Figure 4a. Timing Diagram for: SP [(Scalar * Vector) + Vector]



NOTES: Assume the following single precision operations.

(K * OP0) + OP1 = PRODUCT1 + OP1 → SUM1

(K * OP2) + OP3 = PRODUCT2 + OP3 → SUM2

(K * OP4) + OP5 = PRODUCT3 + OP5 → SUM3

(K * OP6) + OP7 = PRODUCT4 + OP7 → SUM4

Assume valid control signals for FAST, $\overline{HALT}$ = 1, PIPES2-0 = 010, $\overline{RESET}$ = 1, RND1-0, SELST1-0 = 11, TP1-0 = 11.

Figure 4b. Timing Diagram for: SP [(Scalar * Vector) + Vector]

SN74ACT8847 64-Bit Floating Point Unit

Introduction

Designing with the SN74ACT8847 floating point unit (FPU) requires a thorough understanding of computer architectures, microprogramming, and IEEE floating point arithmetic, as well as a detailed knowledge of the 'ACT8847 itself. This introduction presents a brief overview of the 'ACT8847 and discusses a number of issues when designing and programming with this FPU.

Major Architectural Features

The overall architecture for a floating point system is determined by a combination of design factors. The principal consideration is the set of performance targets that the floating point processor has to achieve, usually expressed in terms of clock cycle period, operating mode (vector or scalar), and operand precision (32 bit, 64 bit, or other). Of almost equal importance are design constraints of cost, complexity, chip count, power consumption, and requirements for interfacing to other processors.

The architecture of the 'ACT8847 is optimized to satisfy several processing and interface requirements. The FPU has two 32-bit input buses, the DA and DB data buses, and one 32-bit output bus, the Y bus. This three-port design provides much greater I/O bus bandwidth than can be achieved by a single-port device (one 32-bit I/O bus). Two single-precision inputs can be simultaneously loaded on the input buses while a result is being output on the Y bus.

Internally, the 'ACT8847 FPU consists of two main functional blocks: the multiplier and the ALU (see Figure 5). Either the multiplier or the ALU can operate independently, or the two functional units can be used simultaneously in "chained" mode. When operating independently, each block of the FPU performs a separate set of arithmetic or logical functions. The multiplier supports multiplication, division and square roots. The ALU supports addition, subtraction, format conversions, logical operations, and shifts. Integer division and integer square root require both the multiplier and the ALU; the final result comes from the ALU.

In chained mode, a multiplier operation executes in parallel with an ALU operation. Possible examples include calculations of a sum of products (multiply and accumulate) or a product of sums (add and then multiply). The sum of products computation requires a total of four operands: two new inputs to be multiplied, the sum of previous products, and the current product to be added to the sum, as shown in Table 3.

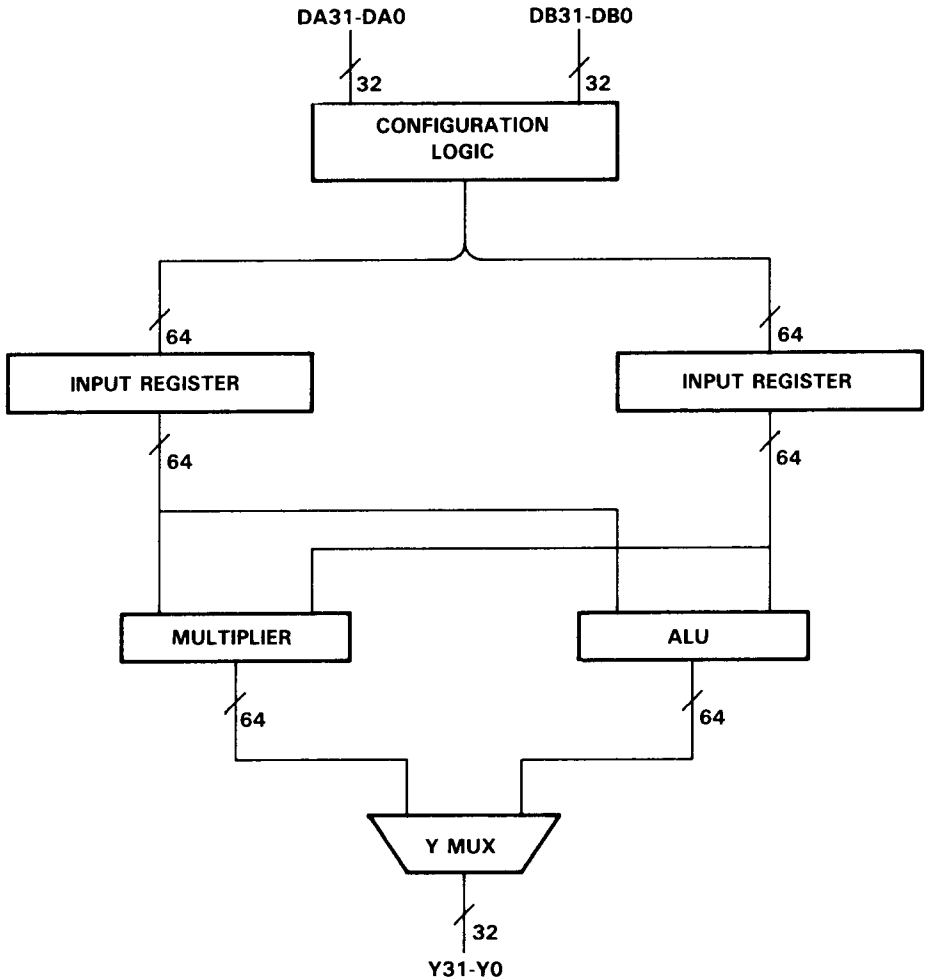


Figure 5. High Level Block Diagram

Table 3. Sum of Products Calculation

MULTIPLIER OPERATION	ALU OPERATION
A * B	—
C * D	(A * B) + 0
E * F	(C * D) + (A * B)
• • •	• • •

Because the 'ACT8847 has multiple internal data paths and data registers, this sum of products can be generated by simultaneous operations on new bus data and internal feedback, without the necessity of storing either the previous accumulation or the current product off chip. Data flow for the sum of products calculation is shown in Figure 6.

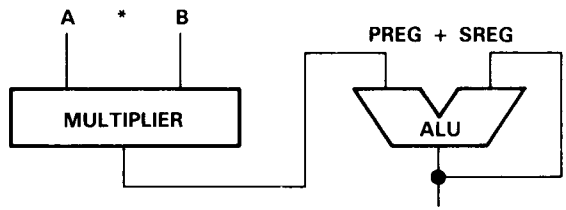


Figure 6. Multiply/Accumulate Operation

Data Flow in Pipelined Architectures

Several levels of internal data registers are available to segment the internal data paths of the 'ACT8847. The most basic choice is whether to use the device in flowthrough mode (with no internal registers enabled) or whether to enable one or more registers. When none of the internal registers are enabled, the paths through the multiplier and the ALU are not segmented. In this case, the delay from data input to result output is the longest.

Enabling one or more registers divides the data paths so that data can be clocked into internal registers, instead of from an external source to an external destination. Enabling the input registers permits data and instruction inputs to be registered on chip. Also, the hardware division and square root operations which the 'ACT8847 performs require that the input registers be enabled.

In the main data paths, three sets of internal registers are available in the ACT8847: input registers, pipeline registers in the multiplier and ALU logic blocks, and output registers to capture results from the multiplier and the ALU. When all three levels of data registers are enabled, the register-to-register delay inside the device is minimized. This is the fastest operating mode, and in this configuration the 'ACT8847 is said to be "fully pipelined." While one instruction is executing, the next instruction along with its associated operands may be input to device so that overlapped operations occur (see Figure 7).

The selection of operating mode, from flowthrough to fully pipelined, determines the latency from input to output, the number of clock cycles required for inputs to be processed and results to appear. For each register level enabled in the data path, one clock cycle is added to the latency from input to output.

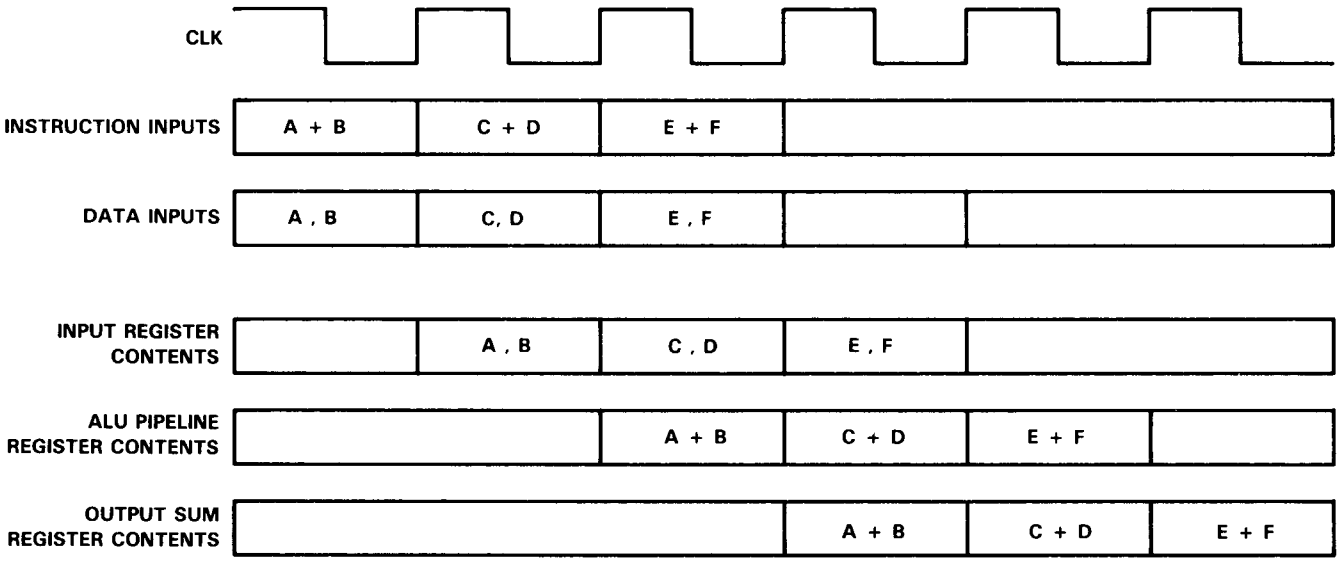


Figure 7. Example of Fully Pipelined Operation

Control Architectures for High-Speed Microprogrammed Architectures

A separate control circuit is required to sequence the operation of the 'ACT8847. A sequencer function within the control circuit controls both the sequencer and FPU as determined by FPU status outputs. Either a standard microsequencer such as the SN74ACT8818, or a custom controller such as a PLA or gate array can be used to control the FPU. Figure 8 shows an example block diagram for a PLA control circuit.

If a standard microsequencer is used, execution addresses for routines stored in the microprogram memory are generated by the microsequencer. As its name implies, microprogram memory stores the sequences of microinstructions which control FPU execution. The 'ACT8847 can be programmed by generating all control bits in a given microinstruction to select an FPU operation.

One possible control circuit for the 'ACT8847 consists of a microsequencer, microprogram memory, and one or more microinstruction registers, together with status logic as required to support a specific floating point implementation. A control circuit without an instruction register is typically too slow for use with the 'ACT8847. At least one microinstruction register is used to hold the current instruction being executed by the FPU and sequencer (see Figure 9).

Inclusion of the microinstruction register divides the critical path from the sequencer through the program memory to the FPU control inputs, permitting much faster execution times. However, when all the internal registers of the FPU are enabled, FPU operation may be fast enough to require a second register in the control circuit. In this case, a register on the output bus of the sequencer captures each microprogram address, and the microinstruction register captures each microinstruction (see Figure 10).

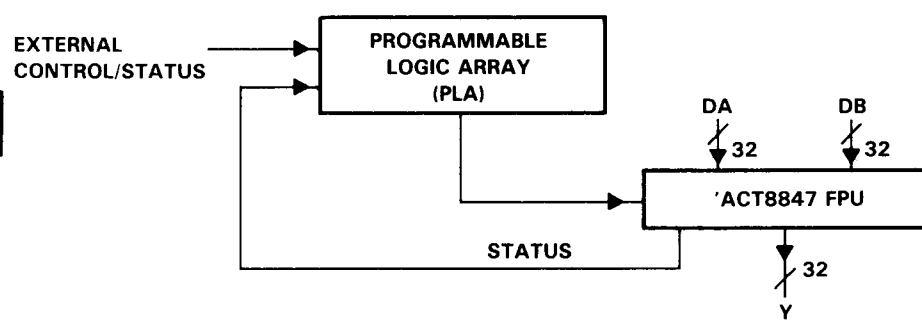


Figure 8. PLA Control Circuit Example

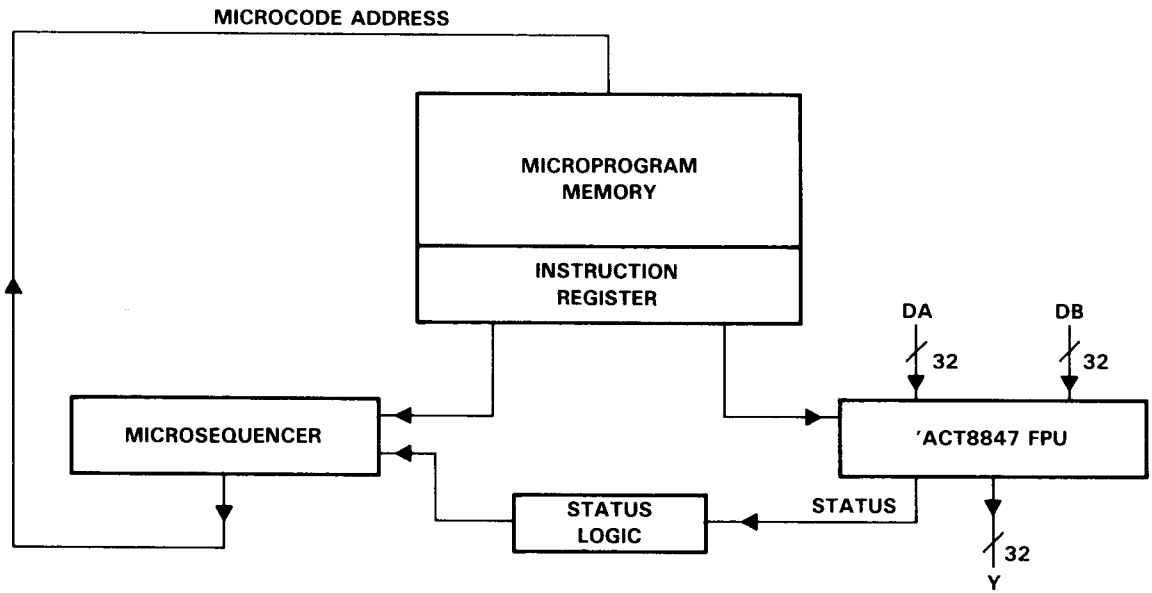


Figure 9. Microprogrammed Architecture

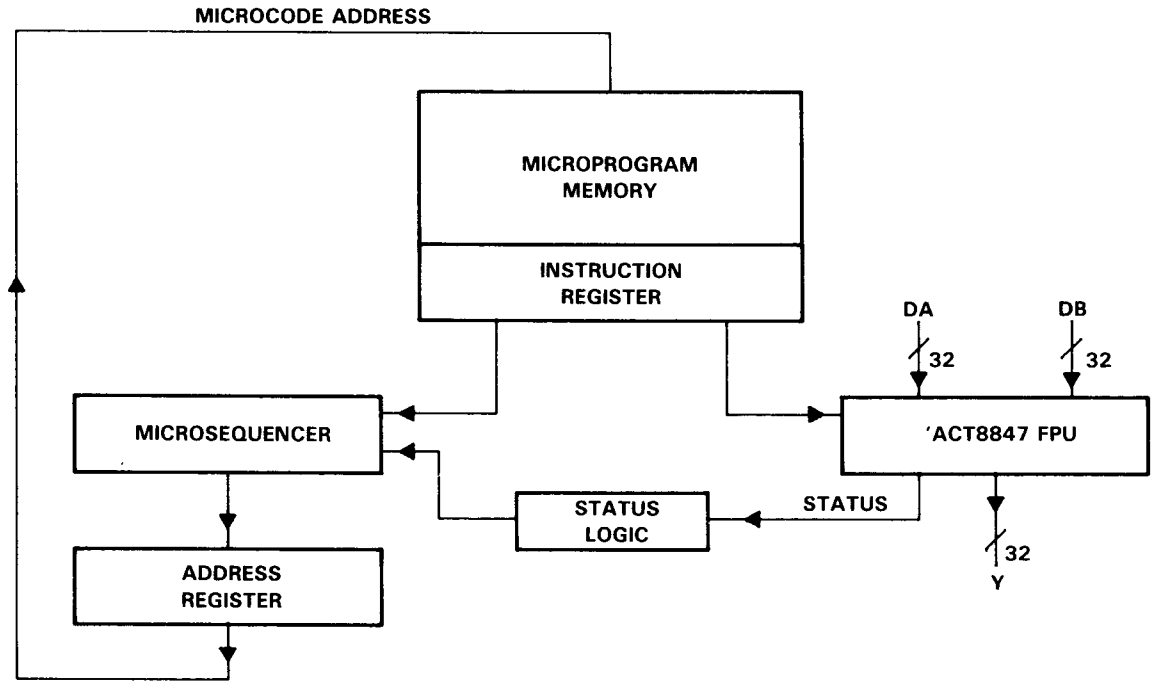


Figure 10. Microprogrammed Architecture with Address Register

Introducing registers in the FPU data paths and the control circuit complicates I/O timing, status output timing, the status logic and the microprogram for the FPU and the sequencer. These timing relationships affect branches, jumps to subroutine, and other operations depending on FPU status. Some of these programming issues are discussed below.

Microprogram Control of an 'ACT8847 FPU Subsystem

A microprogram to control the 'ACT8847 must take into account not only the FPU operation but also the sequencer operation, especially when the system is performing a branch on status or handling an exception.

Several options are available for dealing with such exceptions. The 'ACT8847 can be programmed to discard operands in invalid formats, and some exceptions caused by illegal operations. In general, though, the microprogram should be designed to handle a range of status results or exceptions. Hardware timing considerations such as pipeline delays in both control and data paths must be studied to minimize the difficulty of performing branches to status exception handlers.

Later sections of the 'ACT8847 user guide present detailed examples of microinstructions and timing waveforms, along with interpretations of status outputs and the choices involved in handling IEEE status exceptions.

'ACT8847 Data Formats

The 'ACT8847 accepts either operands as normalized IEEE floating point numbers, (ANSI/IEEE standard 754-1985), unsigned 32-bit integers, or 2's complement integers. Floating point operands may be either single precision (32 bits) or double precision (64 bits).

IEEE formats for floating point operands, both single and double precision, consist of three fields: sign, exponent, and fraction, in that order. The leftmost (most significant) bit is the sign bit. The exponent field is 8 bits long in single-precision operands and 11 bits long in double-precision operands. The fraction field is 23 bits in single precision and 52 bits in double precision. The value of the fraction contains a hidden bit, an implicit leading "1", as shown below:

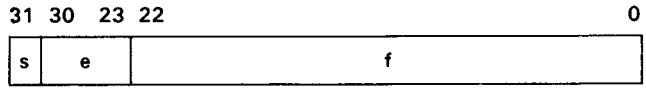
$$1.\text{fraction}$$

The representation of a normalized floating point number is:

$$(-1)^{s} * 1.f * 2^{(e\text{-bias})}$$

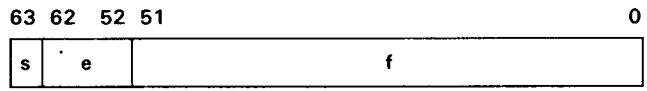
where the bias is either 127 for single-precision operands or 1023 for double-precision operands.

The formats for single-precision and double-precision numbers are shown in Figure 11 and Figure 12, respectively. Further details of IEEE formats and exceptions are provided in the IEEE Standard for Binary Floating Point Arithmetic, ANSI/IEEE Std 754-1985.



s: sign of fraction
e: 8-bit exponent biased by 127
f: 23-bit fraction

Figure 11. IEEE Single-Precision Format



s: sign of fraction
e: 11-bit exponent biased by 1023
f: 52-bit fraction

Figure 12. IEEE Double-Precision Format

The 'ACT8847 also handles two other operand formats which permit operations with very small floating point numbers. The ALU accepts denormalized floating point numbers, that is, floating point numbers so small that they could not be normalized. If these denormal operands are input to the multiplier, they will cause status exceptions. Denormals can be passed through the ALU to be "wrapped," and the wrapped operands can then be input to the multiplier.

A denormalized input has the form of a floating point number with a zero exponent, a nonzero mantissa, and a zero in the leftmost bit of the mantissa (hidden or implicit bit). Using single precision, a denorm is equal to:

$$(-1)^{s} * (2)^{-126} * \text{fraction}$$

For double precision, a denorm is equal to:

$$(-1)^{s} * (2)^{-1022} * \text{fraction}$$

A denormalized number results from decrementing the biased exponent field to zero before normalization is complete. Since a denormalized number cannot be input to the multiplier, it must first be converted to a wrapped number by the ALU. A wrapped number is a number created by normalizing a denormalized number's fraction field and subtracting from the exponent the number of shift positions (minus one) required to do so. The exponent is encoded as a two's complement negative number. When the mantissa of the denormal is normalized by shifting it left, the exponent field decrements from all zeros (wraps past zero) to a negative two's complement number (except in the case of 0.1XXX..., where the exponent is not decremented).

Floating point formats handled by the 'ACT8847 are presented in Table 4.

Table 4. IEEE Floating Point Representations

TYPE OF OPERAND	EXPONENT (e) SP (HEX)	EXPONENT (e) DP (HEX)	FRACTION (f) (BINARY)	HIDDEN BIT	VALUE OF NUMBER REPRESENTED SP (DECIMAL)†	VALUE OF NUMBER REPRESENTED DP (DECIMAL)†
Normalized Number (max)	FE	7FE	All 1's	1	$(-1)^S (2^{127}) (2-2^{-23})$	$(-1)^S (2^{1023}) (2-2^{-52})$
Normalized Number (min)	01	001	All 0's	1	$(-1)^S (2^{-126}) (1)$	$(-1)^S (2^{-1022}) (1)$
Denormalized Number (max)	00	000	All 1's	0	$(1-)^S (2^{-126}) (1-2^{-23})$	$(-1)^S (2^{-1022}) (1-2^{-52})$
Denormalized Number (min)	00	000	000...001	0	$(-1)^S (2^{-126}) (2^{-23})$	$(-1)^S (2^{-1022}) (2^{-52})$
Wrapped Number (max)	00	000	All 1's	1	$(-1)^S (2^{-127}) (2-2^{-23})$	$(-1)^S (2^{-1023}) (2-2^{-52})$
Wrapped Number (min)	EA	7CD	All 0's	1	$(-1)^S (2^{-(22+127)}) (1)$	$(-1)^S (2^{-(51+1023)}) (1)$
Zero	00	000	Zero	0	$(-1)^S (0.0)$	$(-1)^S (0.0)$
Infinity	FF	7FF	Zero	1	$(-1)^S$ (infinity)	$(-1)^S$ (infinity)
NaN (Not a Number)	FF	7FF	Nonzero	N/A	None	None

†s = sign bit.

Status Outputs

Status flags are provided to signal both floating point and integer results. Integer status is provided using AEQB for zero, NEG for sign, and OVER for overflow/carryout.

Status exceptions can result from one or more error conditions such as overflow, underflow, operands in illegal formats, invalid operations, or rounding. Exceptions may be grouped into two classes: input exceptions resulting from invalid operations or denormal inputs to the multiplier, and output exceptions resulting from illegal formats, rounding errors, or both.

SN74ACT8847 Architecture

Overview

The SN74ACT8847 is a high-speed floating point unit implemented in TI's advanced 1-μm CMOS technology. The device is fully compatible with IEEE Standard 754-1985 for addition, subtraction, multiplication, division, square root, and comparison.

The 'ACT8847 FPU also performs integer arithmetic, logical operations, and logical shifts. Absolute value conversions, floating point to integer conversions, and integer to floating point conversions are also available. The ALU and multiplier are both included in the same device and can be operated in parallel to perform sums of products and products of sums (see Figure 13).

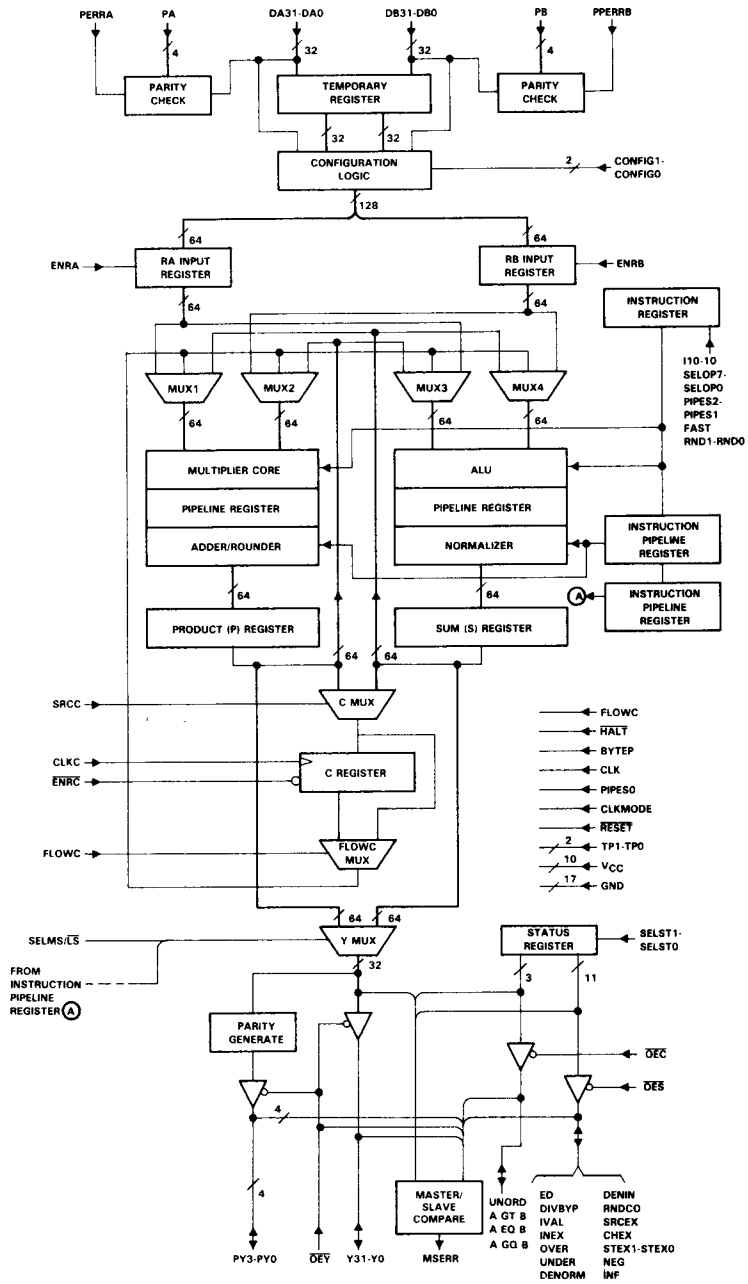


Figure 13. 'ACT8847 Detailed Block Diagram

IEEE formatted denormal numbers are directly handled by the ALU. Denormal numbers must be wrapped by the ALU before being used in multiplication, division, or square root operations. A fast mode in which all denormals are forced to zero is provided for applications not requiring gradual underflow.

The 'ACT8847 input buses can be configured to operate as two 32-bit data buses or as a single 64-bit bus, providing a number of system interface options. Registers are provided at the inputs, outputs, and inside the ALU and multiplier to support multilevel pipelining. These registers can be bypassed for nonpipelined operation.

A clock mode control allows the temporary input register to be clocked on the rising edge or the falling edge of the clock to support double-precision ALU operations at the same rate as single-precision operations. A feedback register (C register) with a separate clock is provided for temporary internal storage of a multiplier result, ALU result or constant.

Four multiplexers select the multiplier and ALU operands from the input registers, C register or previous multiplier or ALU result. Results are output on the 32-bit Y bus; a Y output multiplexer selects the most significant or least significant half of the result if a double-precision number is being output.

To ensure data integrity, parity checking is performed on input data, and parity is generated for output data. A master/slave comparator supports fault-tolerant system design. Two test pin control inputs allow all I/Os and outputs to be forced high, low, or placed in a high-impedance state to facilitate system testing.

Pipeline Controls

Six data registers in the 'ACT8847 are arranged in three levels along the data paths through the multiplier and the ALU. Each level of registers can be enabled or disabled independently of the other two levels by setting the appropriate PIPES2-PIPES0 inputs. When enabled, data is latched into the register on the rising edge of the system clock (CLK). A separate instruction pipeline register stores the instruction bits corresponding to the operation being executed at each stage.

The levels of pipelining are shown in Figure 14. The first set of registers, the RA and RB input registers, are controlled by PIPES0. These registers may be used as inputs to the ALU, multiplier, or both.

The pipeline registers are the second register set. When enabled by PIPES1, these registers latch intermediate values in the multiplier or ALU.

The results of the ALU and multiplier operations may optionally be latched into two output registers by setting PIPES2 low. The P (product) register holds the result of the multiplier operation; the S (sum) register holds the ALU result.

Table 5 shows the settings of the registers controlled by PIPES2-PIPES0. Operating modes range from fully pipelined (PIPES2-PIPES0 = 000) to flowthrough (PIPES2-PIPES0 = 111). The instruction pipeline registers are also set accordingly.

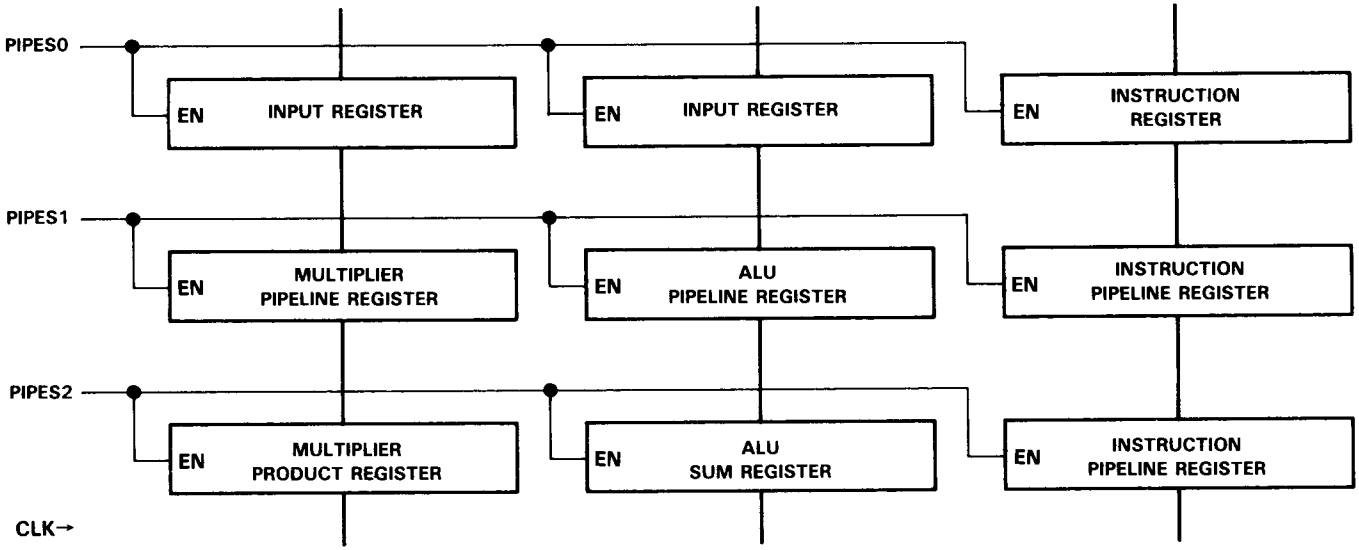


Figure 14. Pipeline Controls

Table 5. Pipeline Controls (PIPES2-PIPES0)

PIPES2-PIPES0			REGISTER OPERATION SELECTED
X	X	0	Enables input registers (RA, RB)
X	X	1	Makes input registers (RA, RB) transparent
X	0	X	Enables pipeline registers
X	1	X	Makes pipeline registers transparent
0	X	X	Enables output registers (PREG, SREG, Status)
1	X	X	Makes output registers (PREG, SREG, Status) transparent

In flowthrough mode all three levels of registers are transparent, a circumstance which may affect some double-precision operations. Since double-precision operands require two steps to input, at least half of the data must be clocked into the temporary register before the remaining data is placed on the DA and DB buses.

When all registers (except the C register) are enabled, timing constraints can become critical for many double-precision operations. In clock mode 1, the ALU can perform a double-precision operation and output a result during every clock cycle, and both halves of the result must be read out before the end of the next cycle. Status outputs are valid only for the period during which the Y output data is valid.

Similarly, double-precision multiplication is affected by pipelining, clock mode, and sequence of operations. A double-precise multiply may require two cycles to execute and two cycles to output the result, depending on the settings of PIPES2-PIPES0.

Duration of valid outputs at the Y multiplexer depends on settings of PIPES2-PIPES0 and CLKMODE, as well as whether all operations and operands are of the same type. For example, when a double-precision multiply is followed by a single-precision operation, one clock cycle must intervene between the dissimilar operations. The instruction inputs are ignored during this clock cycle.

Temporary Input Register

A temporary input register is provided to enable loading of two double-precision numbers on two 32-bit input buses in one clock cycle. The contents of the DA bus are loaded into the upper 32 bits of the temporary register; the contents of DB are loaded into the lower 32 bits.

A clock mode signal (CLKMODE) determines the clock edge on which the data will be stored in the temporary register. When CLKMODE is low, data is loaded on the rising edge of the clock. With CLKMODE set high, the temporary register loads on a falling edge and the RA and RB registers can then be loaded on the next rising edge. The temporary register loads during every clock cycle.

RA and RB Input Registers

Two 64-bit registers, RA and RB, are provided to hold input data for the multiplier and ALU. Data is taken from the DA bus, DB bus and the temporary input register. The registers are loaded on the rising edge of clock CLK if the enables ENRA and ENRB are set high. PIPES0 must be low.

Data input combinations to the 'ACT8847 vary depending on the precision of the operands and whether they are being input as A or B operands. Loading of external data operands is controlled by the settings of CLKMODE and CONFIG1-CONFIG0, which determine the clock timing for loading and the registers that are used. (See Figure 15).

Configuration Controls

Three input registers are provided to handle input of data operands, either single precision or double precision. The RA, RB, and temporary registers are each 64 bits wide. The temporary register is (ordinarily) used only during input of double-precision operands.

Double-precision operands are loaded by using the temporary register to store half of the operands prior to inputting the other half of the operands on the DA and DB buses. As shown in Table 6, four configuration modes for selecting input sources are available for loading data operands into the RA and RB registers.

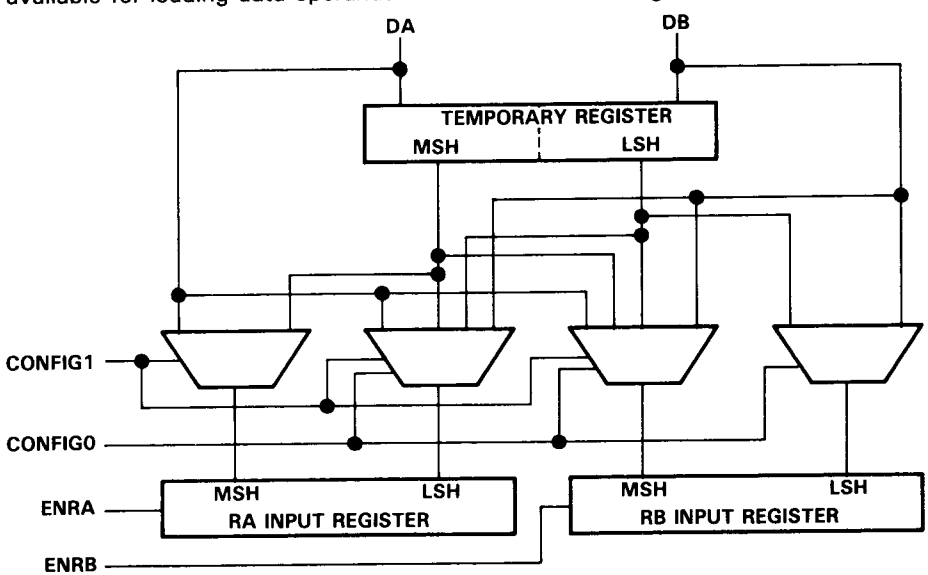


Figure 15. Input Register Control

Table 6. Double Precision Input Data Configuration Modes

		LOADING SEQUENCE			
		DATA LOADED INTO TEMP REGISTER ON FIRST CLOCK AND RA/RB REGISTERS ON SECOND CLOCK†		DATA LOADED INTO RA/RB REGISTERS ON SECOND CLOCK	
CONFIG1	CONFIG0	DA	DB	DA	DB
0	0	B operand (MSH)	B operand (LSH)	A operand (MSH)	A operand (LSH)
0	1	A operand (LSH)	B operand (LSH)	A operand (MSH)	B operand (MSH)
1	0	A operand (MSH)	B operand (MSH)	A operand (LSH)	B operand (LSH)
1	1	A operand (MSH)	A operand (LSH)	B operand (MSH)	B operand (LSH))

†On the first active clock edge (see Clock Mode Settings), data in this column is loaded into the temporary register. On the next rising edge, operands in the temporary register and the DA/DB buses are loaded into the RA and RB registers.

When single-precision or integer operands are loaded, the ordinary setting of CONFIG1-CONFIG0 is 01, as shown in Table 7. This setting loads each 32-bit operand in the most significant half (MSH) of its respective register. Single-precision operands are loaded into the MSHs and adjusted to double precision because the data paths internal to the device are all double precision. It is also possible to load single-precision operands with other CONFIG settings but two clock edges are required to load both the A and B operands on the DA bus. The operands are input as the MSHs of the A and B operands (see Table 6). For example, to load single-precision operands using CONFIG1-CONFIG0 = 10, the A and B operands are input one active clock edge before the instruction.

Table 7. Single-Precision Input Data Configuration Mode

		DATA LOADED INTO RA/RB REGISTERS ON FIRST CLOCK		
CONFIG1	CONFIG0	DA	DB	NOTE
0	1	A operand	B operand	This mode is ordinarily used for single-precision operations.

Clock Mode Settings

Timing of double-precision data inputs is determined by the clock mode setting, which allows the temporary register to be loaded on either the rising edge (CLKMODE = 0) or the falling edge of the clock (CLKMODE = 1). Since the temporary register is not used when single-precision operands are input, clock modes 0 and 1 are functionally equivalent for single-precision operations using CONFIG1-CONFIG0 = 01.

The setting of CLKMODE can be used to speed up the loading of double-precision operands. When the CLKMODE input is set high, data on the DA and DB buses are loaded on the falling edge of the clock into the MSH and LSH, respectively, of the temporary register. On the next rising edge, contents of the DA bus, DB bus, and temporary register are loaded into the RA and RB registers, and execution of the current instruction begins. The setting of CONFIG1-CONFIG0 determines the exact pattern in which operands are loaded, whether as MSH or LSH in RA or RB.

Double-precision operation in clock mode 0 is similar except that the temporary register loads only on a rising edge. For this reason, the RA and RB registers do not load until the next rising edge, when all operands are available and execution can begin.

A considerable advantage in speed can be realized by performing double-precision operations with CLKMODE set high. In this clock mode, both double-precision operands can be loaded on successive clock edges, one falling and one rising. If the instruction is an ALU operation, then the operation can be executed in the time from one rising edge of the clock to the next rising edge. Both halves of a double-precision ALU result must be read out on the Y bus within one clock cycle when the 'ACT8847 is operated in clock mode 1.

The discussion above assumes that the system is able to furnish two sets of operands in one cycle (one set on the falling edge of the clock and the other set on the next rising edge). This assumption may not be valid, since the system is required to "double pump" the input data buses.

Even for a system that is not able to double pump the input data buses, using clock mode 1 can reduce microcode size substantially resulting in increased system throughput. To illustrate, take the case of an operation where the operand(s) are furnished by one or more of the feedback registers (refer to Table 8). Since the input data buses are not being used to furnish the operands, the data on the buses at the time of the instruction is unimportant. By setting CLKMODE high, the instruction begins after the first cycle, resulting in a savings of one cycle.

Table 8a. Double-Precision CREG + PREG Using CLKMODE = 0, PIPES2-0 = 010

CYCLE	CLKMODE	DA BUS	DB BUS	TEMP REG	INSTR BUS	RA REG	RB REG	S REG
1	0	X	X	X	C + P	X	X	X
2	0	X	X	X	C + P	X	X	X
3	X	X	X	X	X	X	X	C + P

Table 8b. Double-Precision CREG + PREG Using CLKMODE = 0, PIPES2-0 = 010

CYCLE	CLKMODE	DA BUS	DB BUS	TEMP REG	INSTR BUS	RA REG	RB REG	S REG
1	1	X	X	X	C + P	X	X	X
2	X	X	X	X	X	X	X	C + P

Going one step further, take the case of an operation where only one operand needs to be furnished by the input data buses (refer to Table 9). To take advantage of clock mode 1, set the CONFIG lines so that the external operand comes directly from the DA and DB bus, as opposed to coming from the temporary register. Since the temporary register is not used to provide an operand, the data latched into it is inconsequential. It naturally follows then that the clock edge used to load the temporary register is unimportant. So by setting CLKMODE high, a double-precision instruction will begin after one cycle, instead of two cycles.

Table 9a. Double-Precision PREG + RB Using CLKMODE = 0, PIPES2-0 = 010

CYCLE	CLKMODE	DA BUS	DB BUS	TEMP REG	INSTR BUS	RA REG	RB REG	S REG
1	0	X	X	X	P + RB	X	X	X
2	0	RB(M)	RB(L)	RB	P + RB	X	RB	X
3	X	X	X	X	X	X	X	P + RB

Table 9b. Double-Precision PREG + RB Using CLKMODE = 1, PIPES2-0 = 010

CYCLE	CLKMODE	DA BUS	DB BUS	TEMP REG	INSTR BUS	RA REG	RB REG	S REG
1	1	RB(M)	RB(L)	RB	P + RB	X	RB	X
2	X	X	X	X	X	X	X	P + RB

Operand Selection

Four multiplexers select the multiplier and ALU operands from the RA and RB registers, the previous multiplier or ALU result, or the C register (see Figure 16). The multiplexers are controlled by input signals SELOP7-SELOP0 as shown in Tables 10 and 11. For division and square root operations, operands must be sourced from the input registers RA and RB.

Table 10. Multiplier Input Selection

A1 (MUX1) INPUT			B1 (MUX2) INPUT		
SELOP7	SELOP6	OPERAND SOURCE†	SELOP5	SELOP4	OPERAND SOURCE†
0	0	Reserved	0	0	Reserved
0	1	C register	0	1	C register
1	0	ALU feedback	1	0	Multiplier feedback
1	1	RA input register	1	1	RB input register

† For division or square root operations, only RA and RB registers can be selected as sources.

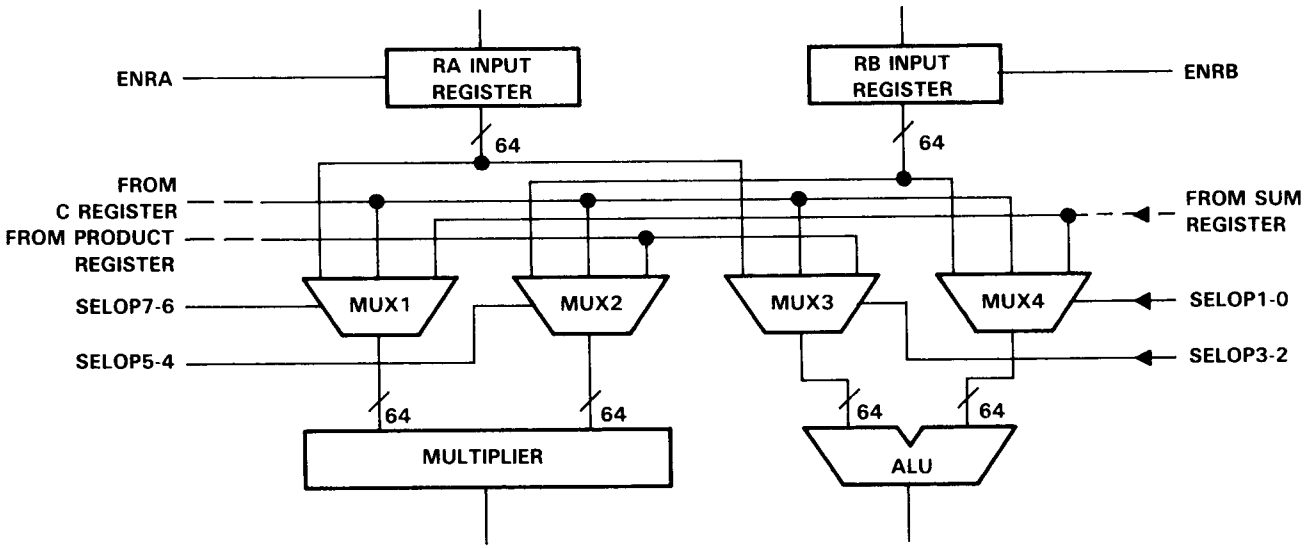


Figure 16. Operand Selection Multiplexer

Table 11. ALU Input Selection

A2 (MUX3) INPUT			B2 (MUX4) INPUT		
SELOP3	SELOP2	OPERAND SOURCE†	SELOP1	SELOP0	OPERAND SOURCE†
0	0	Reserved	0	0	Reserved
0	1	C register	0	1	C register
1	0	Multiplier feedback	1	0	ALU feedback
1	1	RA input register	1	1	RB input register

† For division or square root operations, only RA and RB registers can be selected as sources.

As shown in Tables 10 and 11, data operands can be selected from five possible sources, including external inputs from the RA and RB registers, feedback from the P (Product) and S (Sum) registers, and a stored value in the C register. Contents of the C register may be selected as either the A or the B operand in the ALU, the multiplier, or both. When an external input is selected, the RA input always becomes the A operand, and the RB input is the B operand.

Feedback from the ALU can be selected as the A operand to the multiplier or as the B operand to the ALU. Similarly, multiplier feedback may be used as the A operand to the ALU or the B operand to the multiplier. During division or square root operations, operands may not be selected except from the RA and RB input registers (SELOP7-SELOP0 = 11111111).

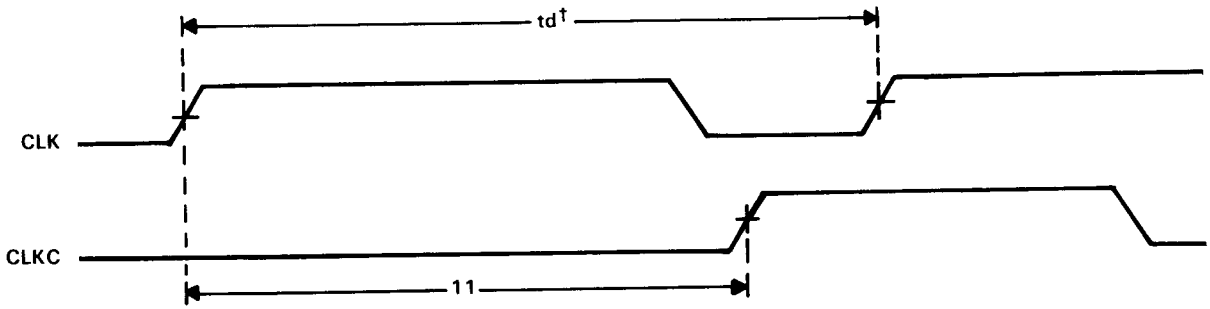
Selection of operands also interacts with the selected operation in the ALU or the multiplier. ALU operations with one operand are performed only on the A operand (with the exception of the Pass B operation). Also, depending on the instruction selected, the B operand may optionally be forced to zero in the ALU or to one in the multiplier.

If an operation uses one or more feedback registers as operands, the unused bus(es) can be used to preload operand(s) for a later operation. The data is loaded into the RA or RB input register(s); when the data is needed as an operand, the SELOPS pins are set to select the RA or RB register(s), but the register input enables (ENRA, ENRB) are not enabled. The one restriction on preloading data is that the operation being performed during the preload MUST use the same data type (single-precision, double-precision, or integer) as the data being loaded. Operands cannot be preloaded within square root or divide instructions.

C Register

The 64-bit constant (C) register is available for storing the result of an ALU or multiplier operation before feedback to the multiplier or ALU. The C register has a separate clock input (CLKC), input source select (SRCC), and write enable ($\overline{\text{ENRC}}$, active low).

The C register loads from the P or the S register output, depending on the setting of SRCC. SRCC = 1 selects the multiplier as the input source. Otherwise, the ALU is selected when SRCC = 0. The SRCC input is not registered with the instruction inputs. Depending on the operation selected and the settings of PIPES2-PIPES0, an offset of one or more cycles may be necessary to load the desired result into the C register. The register only loads on a rising edge of CLCK when $\overline{\text{ENRC}}$ is low. (See Figure 17).



†td is the clock cycle period.

Figure 17. C Register Timing

A separate control (FLOWC) is available to bypass the C register when feeding an operand back on the C register feedback bus. When FLOWC is high, the output of the P or S register (as selected by SRCC) bypasses the C register without affecting the C register's contents. Direct P or S feedback is unaffected by the FLOWC setting.

Pipelined ALU

The pipelined ALU contains a circuit for floating point addition and/or subtraction of aligned operands, a pipeline register, an exponent adjuster and a normalizer/rounder as shown in Figure 18. An exception circuit is provided to detect denormal inputs; these can be flushed to zero if the FAST input is set high. If the FAST input is low, the ALU accepts a denormal as input. A denorm exception flag (DENORM) goes high when the ALU output is a denormal.

Integer processing in the ALU includes both arithmetic and logical operations on either two's complement numbers or unsigned integers. The ALU performs addition, subtraction, comparison, logical shifts, logical AND, logical OR, and logical XOR.

The ALU may be operated independently or in parallel with the multiplier. Possible ALU functions during independent operation are given in Table 12.

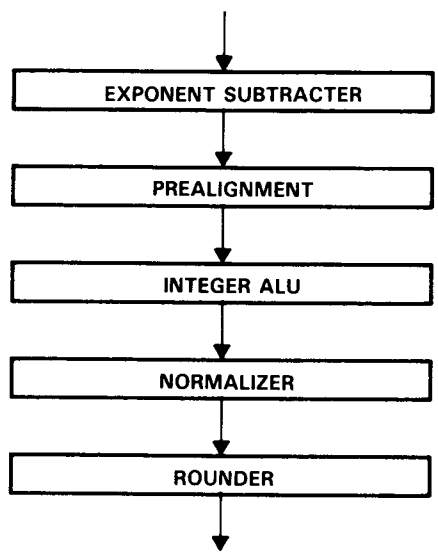


Figure 18. Functional Diagram for ALU

Table 12. Independent ALU Operations

SINGLE OPERAND	TWO OPERANDS
Pass	Add
Move	Subtract
Format Conversions	Compare
Wrap Denormalized Number	AND
Unwrap	OR
Shift	XOR

Pipelined Multiplier

The pipelined multiplier (see Figure 19) performs a basic multiply function, division and square root. The operands can be single-precision or double-precision floating point numbers and can be converted to absolute values before multiplication takes place. Integer operands may also be used. Independent multiplier operations are summarized in Table 13.

If the operands to the multiplier are double precision or mixed precision (ie. one single precision and one double precision), then one extra clock cycle is required to get the product through the multiplier pipeline. This means that for PIPES1 = 1, one clock cycle is required for the multiplier pipeline; for PIPES1 = 0, two clock cycles are required for the multiplier pipeline.

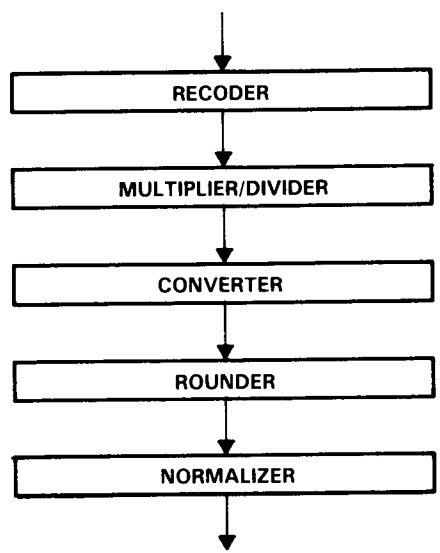


Figure 19. Functional Diagram for Multiplier

Table 13. Independent Multiplier Operations

SINGLE OPERAND	TWO OPERANDS
Square Root	Multiply Divide

An exception circuit is provided to detect denormalized inputs; these are indicated by a high on the DENIN signal. Denormalized inputs must be wrapped by the ALU before multiplication, division, or square root. If results are wrapped (signaled by a high on the DENORM status pin), they must be unwrapped by the ALU.

The multiplier and ALU can be operated simultaneously by setting the I10 instruction input high. Division and square root are performed as independent multiplier operations, even though both multiplier and ALU are active during divide and SQRT operations.

Data Output Controls

Selection and duration of results from the Y output multiplexer may be affected by several factors, including the operation selected, precision of the operands, registers enabled, and the next operation to be performed. The data output controls are not registered with the data and instruction inputs. When the device is microprogrammed, the effects of pipelining and sequencing of operations should be taken into account.

Two particular conditions need to be considered. Depending on which registers are enabled, an offset of one or more cycles must be allowed before a valid result is available at the Y output multiplexer. Also, certain sequences of operations may require both halves of a double-precision result to be read out within a single clock cycle. This is done by toggling the SELMS/$\overline{\text{LS}}$ signal in the middle of the clock period.

When a single-precision result is output, the SELMS/$\overline{\text{LS}}$ signal has no effect. The SELMS/$\overline{\text{LS}}$ signal is set low only to read out the LSH of a double-precision result (see Figure 20). To read out a result on the Y bus, the output enable $\overline{\text{OEY}}$ must be low. $\overline{\text{OEY}}$ is an asynchronous signal.

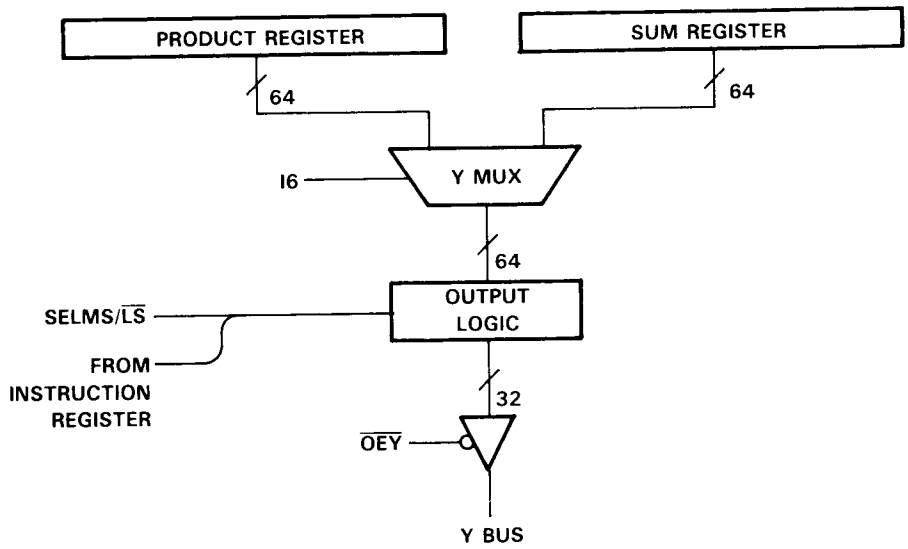


Figure 20. Y Output Control

Parity Checker/Generator

When BYTEP is high, internal even parity is generated for each byte of input data at the DA and DB ports and compared to the PA and PB parity inputs respectively. If an odd number of bits is set high in a data byte, a parity check can also be performed on the entire input data word by setting BYTEP low. In this mode, PA0 is the parity input for DA data and PB0 is the parity input for DB data.

Even parity is generated for the Y multiplexer output, either for each byte or for each word of output, depending on the setting of BYTEP. When BYTEP is high, the parity generator computes four parity bits, one for each byte of the Y multiplexer output. Parity bits are output on the PY3-PY0 pins; PY0 represents parity for the least significant byte. A single parity bit can also be generated for the entire output data word by setting BYTEP low. In this mode, PY0 is the parity output.

Master/Slave Comparator

A master/slave comparator is provided to compare data bytes from the Y output multiplexer and the status outputs with data bytes on the external Y and status ports when $\overline{OEY}$, $\overline{OES}$ and $\overline{OEC}$ are high. If the data bytes are not equal, a high signal is generated on the master/slave error output pin (MSERR).

Figure 21 shows an example master/slave circuit. Two 'ACT8847 slave devices verify the data/status integrity of the 'ACT8847 master.

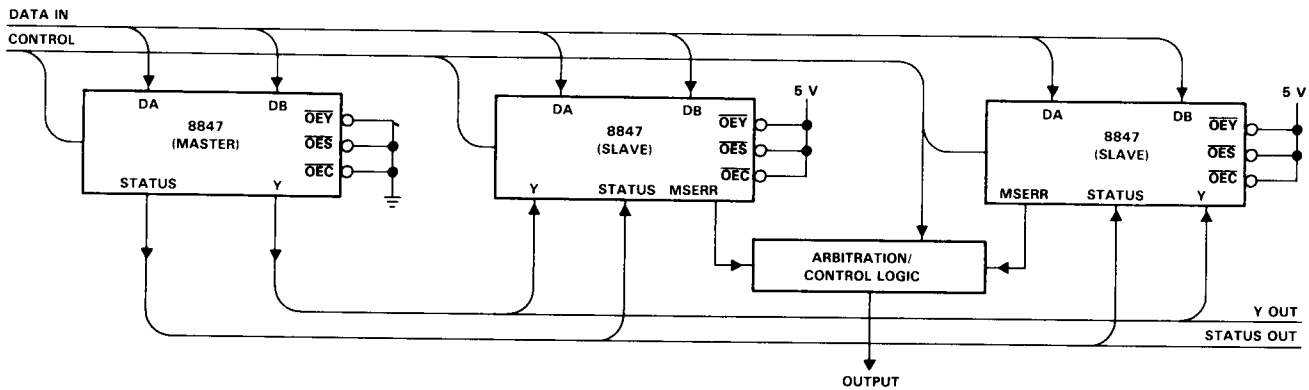


Figure 21. Example of Master/Slave Operation

Status and Exception Generation

A status and exception generator produces several output signals to indicate invalid operations as well as overflow, underflow, non-numerical and inexact results, in conformance with IEEE Standard 754-1985. If output registers are enabled (PIPES2 = 0), status and exception results are latched in the status register on the rising edge of the clock. Status results are valid at the same time as associated data results are valid.

Duration and availability of status results are affected by the same timing constraints that apply to data results on the Y bus. Status outputs are enabled by two signals, $\overline{OEC}$ for comparison status and $\overline{OES}$ for other status and exception outputs. Status outputs are summarized in Tables 14 and 15.

Table 14. Comparison Status Outputs

SIGNAL	RESULT OF COMPARISON (ACTIVE HIGH)
AEQB	The A and B operands are equal. A high signal on the AEQB output indicates a zero result from the selected source except during a compare operation in the ALU. During integer operations, indicates zero status output.
AGTB	The A operand is greater than the B operand.
UNORD	The two inputs of a comparison operation are unordered, i.e., one or both of the inputs is a NaN.

During a compare operation in the ALU, the AEQB output goes high when the A and B operands are equal. When any operation other than a compare is performed, either by the ALU or the multiplier, the AEQB signal is used as a zero detect.

Table 15. Status Outputs

SIGNAL	STATUS RESULT
CHEX	If I6 is low, indicates the multiplier is the source of an exception during a chained function. If I6 is high, indicates the ALU is the source of an exception during a chained function.
DENIN	Input to the multiplier is a denorm. When DENIN goes high, the STEX pins indicate which port had the denormal input.
DENORM	The multiplier output is a wrapped number or the ALU output is a denorm. In the FAST mode, this condition causes the result to go to zero. It also indicates an invalid integer operaion, i.e., PASS (-A) with unsigned integer operand.
DIVBY0	An invalid operation involving a zero divisor has been detected by the multiplier.
ED	Exception detect status signal representing logical OR of all enabled exceptions in the exception disable register.
INEX	The result of an operation is not exact.
INF	The output is the IEEE representation of infinity.
IVAL	A NaN has been input to the multiplier or the ALU, or an invalid operation [(0 * ∞) or (+∞ − ∞) or (−∞ + ∞)] has been requested. This signal also goes high if an operation involves the square root of a negative number. When IVAL goes high, the STEX pins indicate which port had the NaN.
NEG	Output value has negative sign.
OVER	The result is greater than the largest allowable value for the specified format.
RNDCO	The mantissa of a number has been increased in magnitude by rounding. If the number generated was wrapped, then the unwrap round instruction must be used to properly unwrap the wrapped number (see Table 8).
SRCEX	The status was generated by the multiplier. (When SRCEX is low, the status was generated by the ALU.)
STEX0	A NaN or a denorm has been input on the B port.
STEX1	A NaN or a denorm has been input on the A port.
UNDER	The result is inexact and less than the minimum allowable value for the specified format. In the FAST mode, this condition causes the result to go to zero.

In chained mode, results to be output are selected based on the state of the I6 (source output) pin (if I6 is low, ALU status will be selected; if I6 is high, multiplier status will be selected). If the nonselected output source generates an exception, CHEX is set high. Status of the nonselected output source can be forced using the SELST pins, as shown in Table 16.

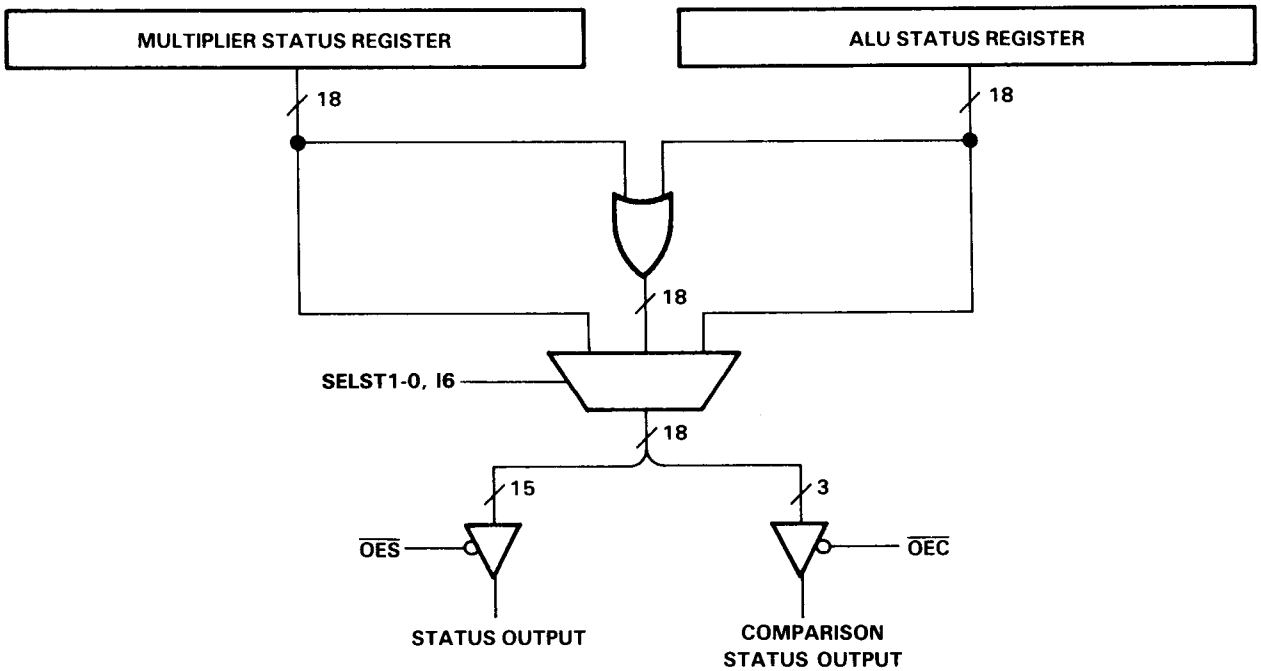


Figure 22. Status Output Control

Table 16. Status Output Selection (Chained Mode)

SELST1-SELST0	STATUS SELECTED
00	Logical OR of ALU and multiplier exceptions (bit by bit)
01	Selects multiplier status
10	Selects ALU status
11	Normal operation (selection based on result source specified by I6 input)

An exception detect mask register is available to mask out selected exceptions from the multiplier, ALU, or both. Multiply status is disabled during an independent ALU instruction, and ALU status is disabled during multiplier instructions. During chained operation, both status outputs are enabled.

When the exception mask register has been loaded with a mask, the mask is applied to the contents of the status register to disable unnecessary exceptions. Status results for enabled exceptions are then ORed together and, if true, the exception detect (ED) status output pin is set high (see Figure 23). Individual status outputs remain active and can be read independently from mask register operations.

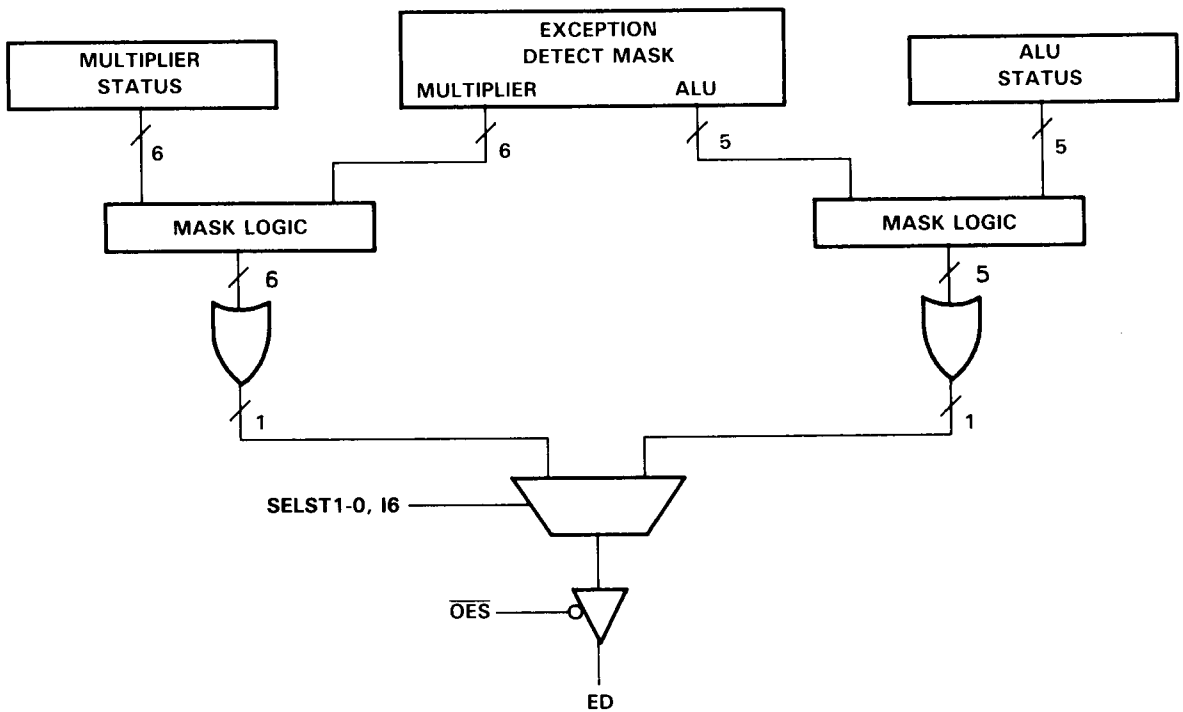


Figure 23. Exception Detect Mask Logic

Microprogramming the 'ACT8847

Because the 'ACT8847 is microprogrammable, it can be configured to operate on either integer or single- or double-precision data operands, and the operations of the registers, ALU, and multiplier can be programmed to support a variety of applications. The following sections present not only control settings but the timings of the specific operations required to execute the sample instructions.

Control Inputs

Control inputs to the 'ACT8847 are summarized in Table 17 below. Several of the inputs have already been discussed; refer to the page listed in the table for detailed information.

The remaining inputs are discussed in the following sections. All control signals and their associated tables are also listed in the 'ACT8847 Reference Guide to provide a complete, easy-to-access reference for the programmer already familiar with 'ACT8847 operation.

Table 17. Control Inputs

SIGNAL	HIGH	LOW	PAGE NO.
BYTEP	Selects byte parity generation and test	Selects single bit parity generation and test	7-75
CLK	Clocks all registers (except C) on rising edge	No effect	7-62
CLKC	Clocks C register on rising edge	No effect	7-70
CLKMODE	Enables temporary input register load on falling clock edge	Enables temporary input register load on rising clock edge	7-66
CONFIG1-CONFIG0	See Table 6 (RA and RB register data source selects)	See Table 42 (RA and RB register data source selects)	7-65
$\overline{\text{ENRC}}$	No effect	Enables C register load when CLKC goes high.	7-70
ENRA	If register is not in flowthrough, enables clocking of RA register	If register is not in flowthrough, holds contents of RA register	7-65
ENRB	If register is not in flowthrough, enables clocking of RB register	If register is not in flowthrough, holds contents of RB register	7-65
FAST	Places device in FAST mode	Places device in IEEE mode	7-84
FLOW_C	Causes output value to bypass C register and appear on C register output bus.	No effect	7-72
$\overline{\text{HALT}}$	No effect	Stalls device operation but does not affect registers, internal states, or status. C register loading is not disabled	7-85
$\overline{\text{OEC}}$	Disables compare pins	Enables compare pins	7-77
$\overline{\text{OES}}$	Disables status outputs	Enables status outputs	7-77
$\overline{\text{OEY}}$	Disables Y bus	Enables Y bus	7-74
PIPES2-PIPES0	See Table 5 (Pipeline Mode Control)	See Table 5 (Pipeline Mode Control)	7-62
$\overline{\text{RESET}}$	No effect	Clears internal states, status, internal pipeline registers, and exception disable register. Does not affect other data registers.	7-86
RND1-RND0	See Table 18 (Rounding Mode Control)	See Table 18 (Rounding Mode Control)	7-84
SELOP7-SELOP0	See Tables 10 and 11 (Multiplier/ALU operand selection)	See Tables 10 and 11 (Multiplier/ALU operand selection	7-68
SELMS/$\overline{\text{LS}}$	Selects MSH of 64-bit result for output on the Y bus (no effect on single-precision operands)	Selects LSH of 64-bit result for output on the Y bus (no effect on single-precision operands)	7-74
SELST1-SELST0	See Table 16 (Status Output Selection)	See Table 16 (Status Output Selection)	7-78
SRCC	Selects multiplier result for input to C register	Selects ALU result for input to C register	7-70
TP1-TP0	See Table 22 (Test Pin Control Inputs)	See Table 22 (Test Pin Control Inputs)	7-86

Rounding Modes

The 'ACT8847 supports the four IEEE standard rounding modes: round to nearest, round towards zero (truncate), round towards infinity (round up), and round towards minus infinity (round down). The rounding function is selected by control pins RND1 and RND0, as shown in Table 18.

Table 18. Rounding Modes

RND1-RND0	ROUNDING MODE SELECTED
0 0	Round towards nearest
0 1	Round towards zero (truncate)
1 0	Round towards infinity (round up)
1 1	Round towards negative infinity (round down)

Rounding mode should be selected to minimize procedural errors which may otherwise accumulate and affect the accuracy of results. Rounding to nearest introduces a procedural error not exceeding half of the least significant bit for each rounding operation. Since rounding to nearest may involve rounding either upward or downward in successive steps, rounding errors tend to cancel each other.

In contrast, directed rounding modes may introduce errors approaching one bit for each rounding operation. Since successive rounding operations in a procedure may all be similarly directed, each introducing up to a one-bit error, rounding errors may accumulate rapidly, especially in single-precision operations.

FAST and IEEE Modes

The device can be programmed to operate in FAST mode by asserting the FAST pin. In the FAST mode, all denormalized inputs and outputs are forced to zero.

Placing a zero on the FAST pin causes the chip to operate in IEEE mode. In this mode, the ALU can operate on denormalized inputs and return denormals. If a denorm is input to the multiplier, the DENIN flag will be asserted, and the result will be invalid. Denormal numbers must be wrapped before being input to the multiplier. If the multiplier result underflows, a wrapped number will be output.

Handling of Denormalized Numbers (FAST)

The FAST input selects the mode for handling denormalized inputs and outputs. When the FAST input is set low, the ALU accepts denormalized inputs but the multiplier generates an exception when a denormal is input. When FAST is set high, the DENIN status exception is disabled and all denormalized numbers, both inputs and results, are forced to zero.

A denormalized input has the form of a floating point number with a zero exponent, a nonzero mantissa, and a zero in the leftmost bit of the mantissa (hidden or implicit bit). A denormalized number results from decrementing the biased exponent field to

zero before normalization is complete. Since a denormalized number cannot be input to the multiplier, it must first be converted to a wrapped number by the ALU. When the mantissa of the denormal is normalized by shifting it left, the exponent field decrements from all zeros (wraps past zero) to a negative two's complement number (except in the case of 0.1XXX...), where the exponent is not decremented.

Exponent underflow is possible during multiplication of small operands even when the operands are not wrapped numbers. Setting FAST = 0 selects gradual underflow so that denormal inputs can be wrapped and wrapped results are not automatically discarded. When FAST is set high, denormal inputs and wrapped results are forced to zero immediately.

When the multiplier is in IEEE mode and produces a wrapped number as its result, the result may be passed to the ALU and unwrapped. If the wrapped number can be unwrapped to an exact denormal, it can be output without causing the underflow status flag (UNDER) to be set. UNDER goes high when a result is an inexact denormal, and a zero is output from the FPU if the wrapped result is too small to represent as a denormal (smaller than the minimum denorm). Table 10 describes the handling of wrapped multiplier results and the status flags that are set when wrapped numbers are output from the multiplier.

Table 19. Handling Wrapped Multiplier Outputs

TYPE OF RESULT	STATUS FLAGS SET			NOTES
	DENORM	INEX	RNDCO	
Wrapped, exact	1	0	0	Unwrap with 'Wrapped exact' ALU instruction
Wrapped, inexact	1	1	0	Unwrap with 'Wrapped inexact' ALU instruction
Wrapped, increased in magnitude	1	1	1	Unwrap with 'Wrapped rounded' ALU instruction

When operating in chained mode, the multiplier may output a wrapped result to the ALU during the same clock cycle that the multiplier status is output. In such a case the ALU cannot unwrap the operand prior to using it, for example, when accumulating the results of previous multiplications. To avoid this situation, the FPU can be operated in FAST mode to simplify exception handling during chained operations. Otherwise, wrapped outputs from the multiplier may adversely affect the accuracy of the chained operation, because a wrapped number may appear to be a large normalized number instead of a very small denormalized number.

Because of the latency associated with interpreting the FPU status outputs and determining how to process the wrapped output, it is necessary that a wrapped operand be stored external to the FPU (for example, in an external register file) and reloaded to the A port of the ALU for unwrapping and further processing.

Stalling the Device

Operation of the 'ACT8847 can be stalled nondestructively by means of the $\overline{\text{HALT}}$ signal. Bringing the $\overline{\text{HALT}}$ input low causes the device to inhibit the next rising clock edge. Register contents are unaltered when the device is stalled, and normal operation resumes at the next low clock period after the $\overline{\text{HALT}}$ signal is set high.

Stalling the device does not stall the C register. If $\overline{\text{ENRC}}$ is low, CLKC will clock in data from the source selected by SRCC.

For some operations, such as a double-precision multiply with CLKMODE = 1, setting the $\overline{\text{HALT}}$ input low may interrupt loading of the RA, RB, and instruction registers, as well as stalling operation. In clock mode 1, the temporary register loads on the falling edge of the clock, but the $\overline{\text{HALT}}$ signal going low would prevent the RA, RB, and instruction registers from loading on the next rising clock edge. It is therefore necessary to have the instruction and data inputs on the pins when the $\overline{\text{HALT}}$ signal is set high again and normal operation resumes.

$\overline{\text{RESET}}$

The $\overline{\text{RESET}}$ input is an active-low signal that asynchronously clears the internal states, status, and exception disable mask. Internal pipeline registers are cleared, but the RA, RB, and C registers are not. Operation resumes when $\overline{\text{RESET}}$ goes high again.

Test Pins

Two pins, TP1-TP0, support system testing. These may be used, for example, to place all outputs in a high-impedance state, isolating the chip from the rest of the system (see Table 20).

Table 20. Test Pin Control Inputs

TP1	TP0	OPERATION
0	0	All outputs and I/Os are forced low
0	1	All outputs and I/Os are forced high
1	0	All outputs are placed in a high impedance state
1	1	Normal operation

Independent ALU Operations

Configuration and operation of the 'ACT8847 can be selected to perform single- or double-precision floating point and integer calculations in operating modes ranging from flowthrough to fully pipelined. Timing and sequences of operations are affected by settings of clock mode, data and status registers, input data configurations, and rounding mode, as well as the instruction inputs controlling the ALU and the multiplier.

Three modes of operation can be selected with inputs I10-I0, including independent ALU operation, independent multiplier operation, or simultaneous (chained) operation of ALU and multiplier. Each of these operating modes is treated separately in the following sections.

The ALU executes single- and double-precision operations which can be divided according to the number of operands involved, one or two. Tables 21 and 22 show independent ALU operations with one operand, along with the inputs I10-I0 which select each operation. Conversions from one format to another are handled in this mode, with the exception of adjustments to precision during two-operand ALU operations. The wrapping and unwrapping of operands is also done in this mode.

Most format conversions involve double-precision timing. Conversions between single- and double-precision floating point format are treated as mixed-precision operations requiring two cycles to load the operands. A single-precision number is loaded in the upper half (MSH) of its input register. During integer to floating point conversions, the integer input should be loaded into the upper half of the RA register. If converting from integer to double precision, then two cycles are required.

Logical shifts can be performed on integer operands using the instructions shown in Table 22. The data operand to be shifted is input from any valid operand source and the number of bit positions the operand is to be shifted is input only from the DB bus. The shift number on the DB bus should be in positive 32-bit integer format, although only the lowest eight bits are used. The shift number cannot be selected from sources other than the RB register, and the shift number must be loaded on the same cycle as the instruction.

Table 21. Independent ALU Operations, Single Floating Point Operand (I10 = 0, I9 = 0, I6 = 0, I5 = 1)

CHAINED OPERATION I10	OPERAND FORMAT I9	PRECISION RA I8	PRECISON RB I7	OUTPUT SOURCE I6	OPERAND TYPE I5	ABSOLUTE VALUE A I4	ALU OPERATION	
							I3-I0	RESULT
0 = Not Chained	0 = Floating point	0 = A(SP) 1 = A(DP)	0 = B(SP) 1 = B(DP) must equal I8	0 = ALU result	1 = Single Operand	0 = A 1 = |A|	0000	Pass A operand
							0001	Pass −A operand
							0010	2's complement integer to floating point conversion†
							0011	Floating point to 2's complement integer conversion†
							0100	Move A operand (pass without NaN detect or exception flags active)
							0101	Pass B operand
							0110	Floating point to floating point conversion‡
							0111	Floating point to unsigned integer conversion†
							1000	Wrap (denormal) input operand
							1010	Unsigned integer to floating point conversion†
							1100	Unwrap exact number
							1101	Unwrap inexact number
							1110	Unwrap rounded input

†The precision of the integer to floating point conversion is set by I8. If I8 = 1, the operation is timed like a double-precision operation, requiring clock edges to load.
‡This converts single-precision floating point to double-precision floating point and vice versa. If the I8 pin is low to indicate a single-precision input, the result of the conversion will be double precision. If the I8 pin is high, indicating a double-precision input, the result of the conversion will be single precision. This operation is timed like a double-precision operation, requiring 2 clock edges to load.

Table 22. Independent ALU Operations, Single Integer Operand (I10 = 0, I9 = 1, I6 = 0 I5 = 1)

CHAINED OPERATION	OPERAND FORMAT/PRECISION			OUTPUT SOURCE	OPERAND TYPE	ALU OPERATION	
I10	I9	I8	I7	I6	I5	I4-10	RESULT
0 = Not Chained	1 = Integer	0	0 = SP 2's complement	0 = ALU result	1 = Single Operands	00000	Pass A operand
						00001	Pass (−A) operand
		1	1 = SP unsigned integer			00010	Negate A operand (1's complement)
						00101	Pass B operand
						01000	Shift A operand left logical†
						01001	Shift A operand right logical†
						01101	Shift A operand right arithmetic†

†B operand is number of bit positions A is to be shifted (See instruction description for "Independent ALU Operations".) The B operand must be input on the same cycle that shift is to be performed.

Tables 23 and 24 present independent ALU operations with two operands. When the operands are different in precision, one single and the other double, the settings of the precision selects I8-I7 will identify the single-precision operand so that it can automatically be reformatted to double-precision before the selected operation is executed, and the result of the operation will be double precision.

Precision of each data operand is indicated by the setting of instruction input I8 for single-operand ALU instructions, or the settings of I8-I7 for two-operand instructions. For single-operand instructions, I7 must be set equal to I8. When the ALU receives mixed-precision operands (one operand in single precision and the other in double precision), the single-precision data input is converted to double and the operation is executed in double precision. It is unnecessary to use the 'convert float-to-float' instruction to convert the single-precision operand prior to performing the desired operation on the mixed-precision operands. Setting I8 and I7 properly achieves the same effect without wasting an instruction cycle.

Timing for operations with mixed-precision operands is the same as for a corresponding double-precision operation. In a mixed-precision operation, the single-precision operand must be loaded into the upper half of its input register. If both operands are single precision, a single-precision result is output by the ALU. Operations on mixed-precision data inputs produce double-precision results.

Table 23. Independent ALU Operations, Two Floating-Point Operands (I10 = 0, I9 = 0, I5 = 0)

CHAINED OPERATION I10	OPERAND FORMAT I9	PRECISION RA I8	PRECISION RB I7	OUTPUT SOURCE I6	OPERAND TYPE I5	ABSOLUTE VALUE A I4	ABSOLUTE VALUE B I3	ABSOLUTE VALUE Y I2	ALU OPERATION	
									I1-I0	RESULT
0 = Not chained	0 = Floating point	0 = A(SP) 1 = A(DP)	0 = B(SP) 1 = B(DP)	0 = ALU result	0 = Two operands	0 = A 1 = |A|	0 = B 1 = |B|	0 = Y 1 = |Y|	00 01 10 11	A + B A − B Compare A, B B − A

Table 24. Independent ALU Operations, Two Integer Operands (I10 = 0, I9 = 1, I6 = 0, I5 = 0)

CHAINED OPERATION I10	OPERAND FORMAT/PRECISION			OUTPUT SOURCE I6	OPERAND TYPE I5	ALU OPERATION	
	I9	I8	I7			I4-10	RESULT
0 = Not Chained	1 = Integer	0 0	0 = SP 2's complement 1 = SP unsigned integer	0 = ALU result	0 = Two Operands	00000 00001 00010 00011 01000 01001 01010 01011 01100 01101	A + B A − B Compare A, B B − A Logical AND (A, B) Logical AND (A, NOT B) Logical AND (NOT A, B) Logical AND (NOT A, NOT B) Logical OR (A, B) Logical XOR (A, B)

Two additional independent ALU operations may also be coded. The first of these is for loading the exception detect mask register.

The exception detect mask register can be loaded with a mask to enable or disable selected status exceptions. Status bits for enabled exceptions are logically ORed, and when the result is true, the ED pin goes high. During chained operations, both multiplier and ALU results are ORed. During independent operation, the nonselected status results are forced to zero.

If the FPU is reset ($\overline{\text{RESET}}$ = 0), the exception detect mask register is cleared. Table 25 describes the settings for the mask register load instruction and the status exceptions which can be enabled or disabled with the mask.

Table 25. Loading the Exception Disable Mask Register

INSTRUCTION INPUTS	RESULTS
I10-I7 = 0111	Exception mask load instruction
I6	0 = Load ALU exception disable register 1 = Load multiplier exception disable register
I5†	0 = IVAL exception enabled 1 = IVAL exception disabled
I4	0 = OVER exception enabled 1 = OVER exception disabled
I3	0 = UNDER exception enabled 1 = UNDER exception disabled
I2	0 = INEX exception enabled 1 = INEX exception disabled
I1	0 = DIVBY0 exception enabled 1 = DIVBY0 exception disabled‡
I0	0 = DENORM exception enabled 1 = DENORM exception disabled

† Disabling IVAL in multiplier exception mask register also disables DENIN exception
‡ Only significant when I6 = 1

The second additional independent ALU operation is the NOP (no operation). The table below shows the coding for the NOP instruction.

Table 26. NOP Instruction

I10-I0	Operation
01100000000	NOP

Because NOP, in effect, just prevents loading of the P or S registers, these registers must be enabled (PIPES2 = 0) for the NOP to work correctly.

Timing of a NOP instruction is the same as any single-precision ALU operation, taking one clock cycle per pipeline stage that is enabled. For example, when the 'ACT8847 is fully pipelined (PIPES2-PIPES0 = 000), a NOP's effect (preventing the overwriting of the P and S registers) will be seen on the third cycle. To hold the results of an operation on the Y bus for an extra cycle, the NOP instruction is inserted directly after the instruction whose results are to be held.

The NOP freezes the output register's contents until new results are to be loaded into these registers.

Independent Multiplier Operations

In this mode, the multiplier operates on two of five input sources which can be either single precision, double precision, or mixed. Multiplication, division and square root may be coded as independent multiplier operations.

Operand precision is selected by I8 and I7, as for ALU operations. The multiplier can multiply the A and B operands, either operand with the absolute value of the other, or the absolute values of both operands. The result can also be negated when it is output. Operations involving absolute value or negated results are valid only when floating point format is selected. If both operands are single precision, a single-precision result is output. Operations on mixed-precision data inputs produce double-precision results.

Floating point operands may be normalized or wrapped numbers, as indicated by the settings for instruction inputs I1-I0. As shown in Table 27, the multiplier can be set to operate on the absolute value of either or both floating point operands, and the result of any operation can be negated when it is output from the multiplier. Converting a single-precision denormal number to double precision does not normalize or wrap the denormal, so it is still an invalid input to the multiplier. Independent multiplier operations are summarized in Tables 27 thru 29.

Table 27. Independent Multiplier Operations (I10 = 0, I6 = 1)

CHAINED OPERATION	OPERAND FORMAT/PRECISION			OUTPUT SOURCE	MULTIPLY/ DIVIDE	ABSOLUTE VALUE A	ABSOLUTE VALUE B/ DIV/SQRT	NEGATE RESULT	WRAP A	WRAP B
I10	I9	I8	I7‡	I6	I5	I4†	I3†	I2†	I1†	I0†
0 = Not chained	0 = floating point	0 = A(SP) 1 = A(DP)	0 = B(SP) 1 = B(DP)	1 = Multiplier result	0 = multiply	0 = A 1 = |A|	0 = B 1 = |B|	0 = Y 1 = −Y	0 = Normal format 1 = A is a wrapped number	0 = Normal format 1 = B is a wrapped number
	1 = integer	0	0 = SP 2's complement		1 = Div/SQRT	0 = A 1 = |A|	0 = Div 1 = SQRT			
		0	1 = SP unsigned integer							

†See also Tables 13 and 14. Operations involving absolute values, negated results or wrapped numbers are valid only when floating point format is selected (I9 = 0).

‡For square root operations, I7 must be equal to I8.

Table 28. Independent Multiply Operations Selected by I4-I2 (I10 = 0, I6 = 1, I5 = 0)

ABSOLUTE VALUE A I4	ABSOLUTE VALUE B I3	NEGATE RESULT I2	OPERATION SELECTED	
			I4-I2	RESULTS‡
0 = A	0 = B	0 = Y	000	A * B
1 = |A|	1 = |B|	1 = −Y	001	−(A * B)
			010	A * |B|
			011	−(A * |B|)
			100	|A| * B
			101	−(|A| * B)
			110	|A| * |B|
			111	−(|A| * |B|)

‡Operations involving absolute values or negated results are valid only when floating point format is selected (I9 = 0).

Table 29. Independent Divide/Square Root Operations Selected by I4-I2 (I10 = 0, I6 = 1, I5 = 1)

ABSOLUTE VALUE A I4	DIVIDE/ SQRT I3	NEGATE RESULT I2	OPERATION SELECTED	
			I4-I2	RESULTS†
0 = A	0 = Divide	0 = Y	000	A / B
1 = A	1 = SQRT	1 = −Y	001	−(A / B)
			010	SQRT A
			011	−(SQRT A)
			100	|A| / B
			101	−(|A| / B)
			110	SQRT |A|
			111	−(SQRT |A|)

†Operations involving absolute values or negated results are valid only when floating point format is selected (I9 = 0).

Chained Multiplier/ALU Operations

In chained mode, the 'ACT8847 performs simultaneous operations in the multiplier and the ALU. Operations not only include addition, subtraction, and multiplication, but also several optional operations which increase the flexibility of the device (see Table 30). Division and square root operations are not available in chained mode. Format conversions, absolute values, and wrapping or unwrapping of denormal numbers are also not available.

The B operand to the ALU can be set to zero so that the ALU passes the A operand unaltered. The B operand to the multiplier can be forced to the value 1 so that the A operand to the multiplier is passed unaltered.

Since in chained mode there are four operands but only two bits (I8 and I7) to select the operand precision, care must be taken with mixed-precision operations. The A input to the ALU and to the multiplier must be of the same precision; just as the B input to the ALU and to the multiplier must be of the same precision.

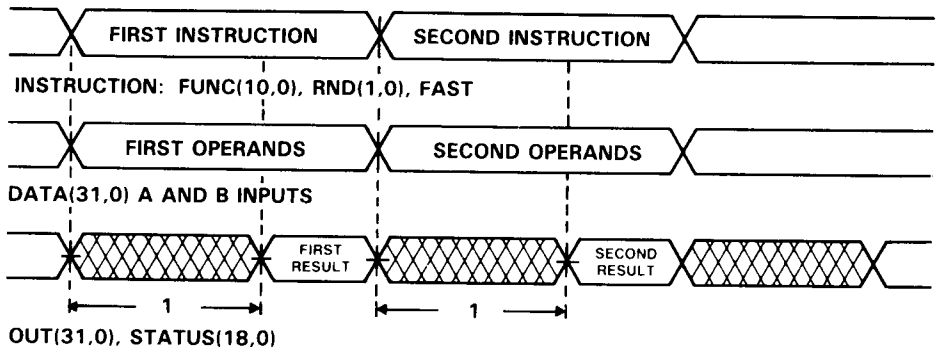
Table 30. Chained Multiplier/ALU Operations (I10 = 1)

CHAINED OPERATION	OPERAND FORMAT/PRECISION			OUTPUT SOURCE	ADD ZERO	MULTIPLY BY ONE	NEGATE ALU RESULT	NEGATE MULTIPLIER RESULT	ALU OPERATIONS	
I10	I9	I8	I7	I6	I5	I4	I3†	I2†	I1-I0	RESULT
1 = Chained	0 = floating point	0 = A(SP) 1 = A(DP)	0 = B(SP) 1 = B(DP)	0 = ALU result	0 = Normal operation	0 = Normal operation	0 = Normal operation	0 = Normal operation	00	A + B
									01	A − B
									10	2 − A
									11	B − A
	1 = integer	0	0 = SP 2's complement	1 = Multiplier result	1 = Forces B2 input of ALU to zero	1 = Forces B1 input of multiplier to one	1 = Negate ALU result	1 = Negate multiplier result		
		0	1 = SP unsigned integer							

†Operations involving negated results are valid only when floating point format is selected (I9 = 0).

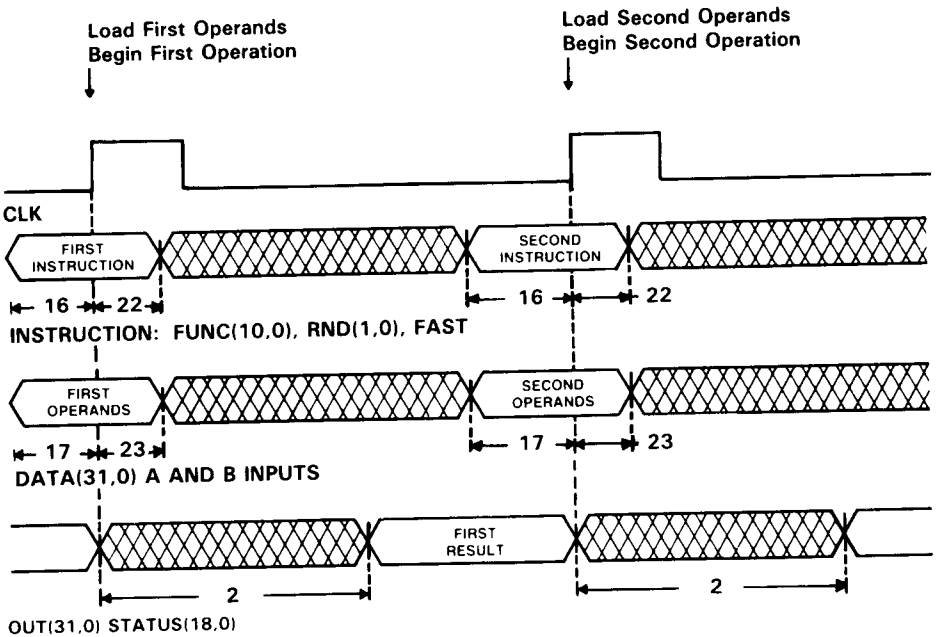
Sample Independent ALU Microinstructions

The following independent ALU timing diagram examples show four register settings, ranging from fully flowthrough to fully pipelined. X = don't care.



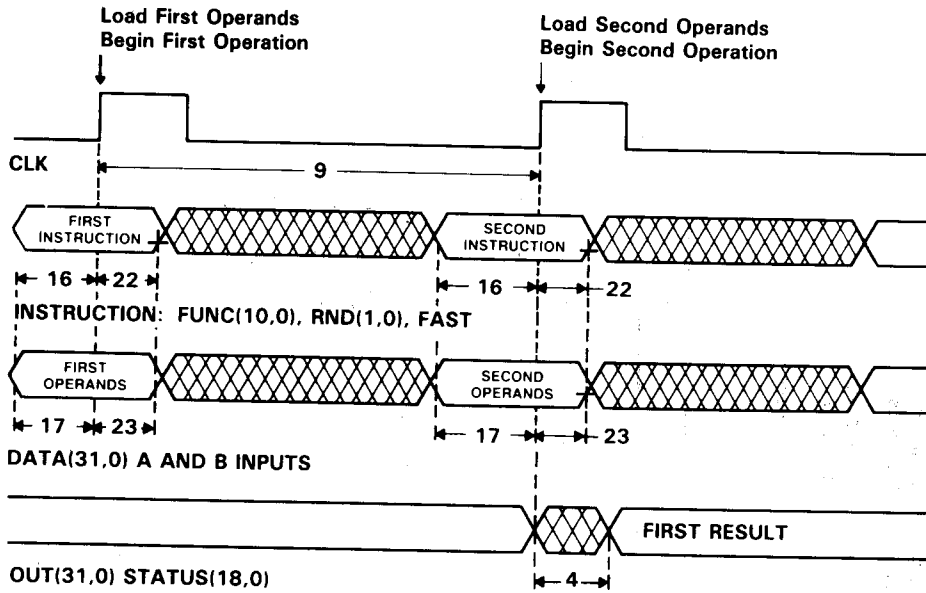
NOTE: Assume PIPES2-0 = 111, CONFIG1-0 = 01, ENRA = X, ENRB = X, SELMS/$\overline{LS}$ = X, $\overline{OEY}$ = 0, $\overline{OEC}$ = $\overline{OES}$ = 0, $\overline{RESET}$ = $\overline{HALT}$ = 1, TP1-0 = 11

Figure 24. Single-Precision Independent ALU Operation, All Registers Disabled (PIPES2-PIPES0 = 111, CLKMODE = X)



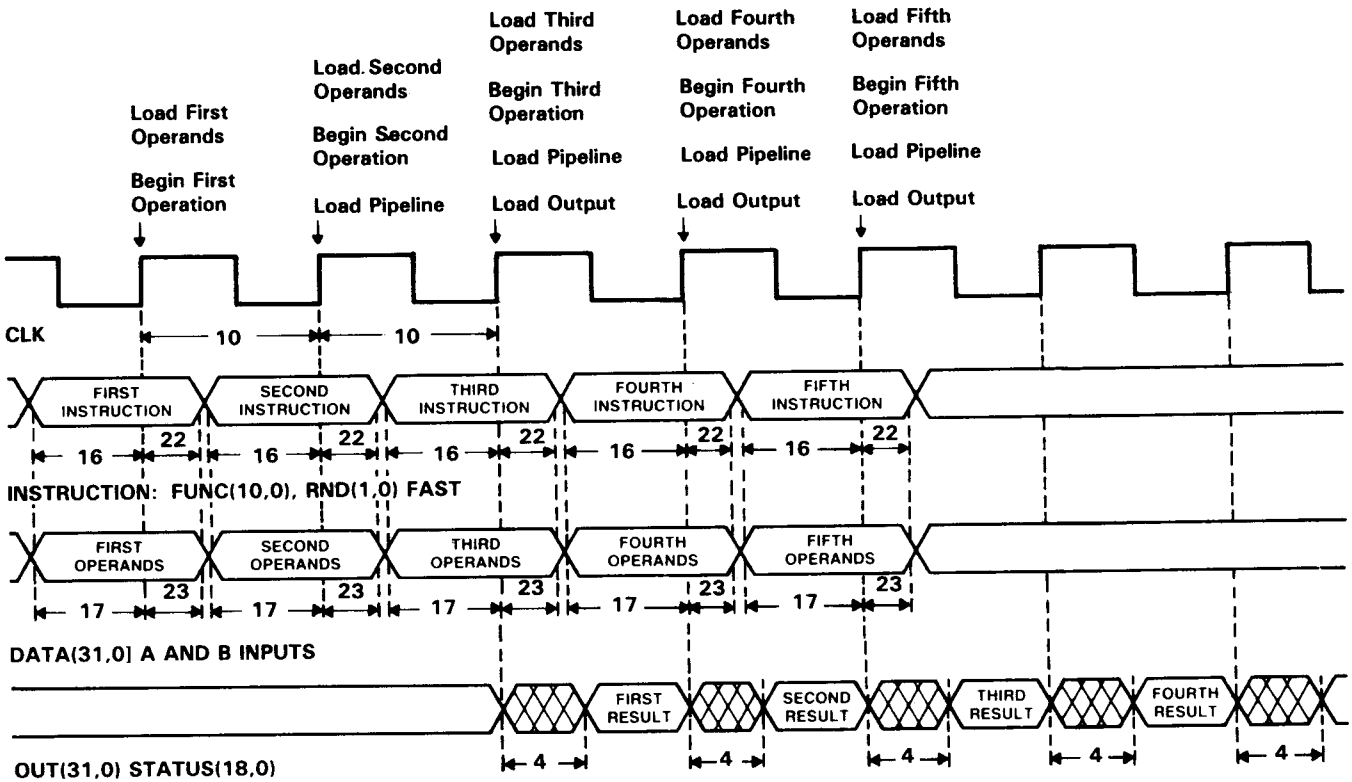
NOTE: Assume PIPES2-0 = 110, CONFIG1-0 = 01, ENRA = 1, ENRB = 1, SELMS/$\overline{LS}$ = X, $\overline{OEY}$ = 0, $\overline{OEC}$ = $\overline{OES}$ = 0, $\overline{RESET}$ = $\overline{HALT}$ = 1, TP1-0 = 11

Figure 25. Single-Precision Independent ALU Operation, Input Registers Enabled (PIPES2-PIPES0 = 110, CLKMODE = X)



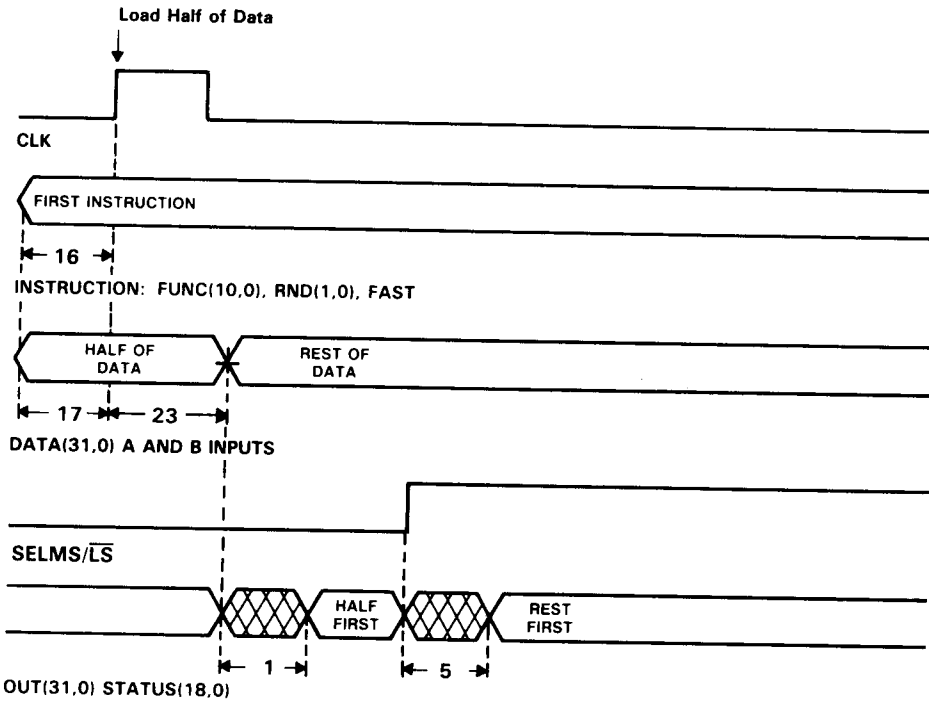
NOTE: Assume PIPES2-0 = 010, CONFIG1-0 = 01, ENRA = 1, ENRB = 1, SELMS/$\overline{LS}$ = X, $\overline{OEY}$ = 0, $\overline{OEC}$ = $\overline{OES}$ = 0, $\overline{RESET}$ = $\overline{HALT}$ = 1, TP1-0 = 11

Figure 26. Single-Precision Independent ALU Operation, Input and Output Registers Enabled (PIPES2-PIPES0 = 010, CLKMODE = X)



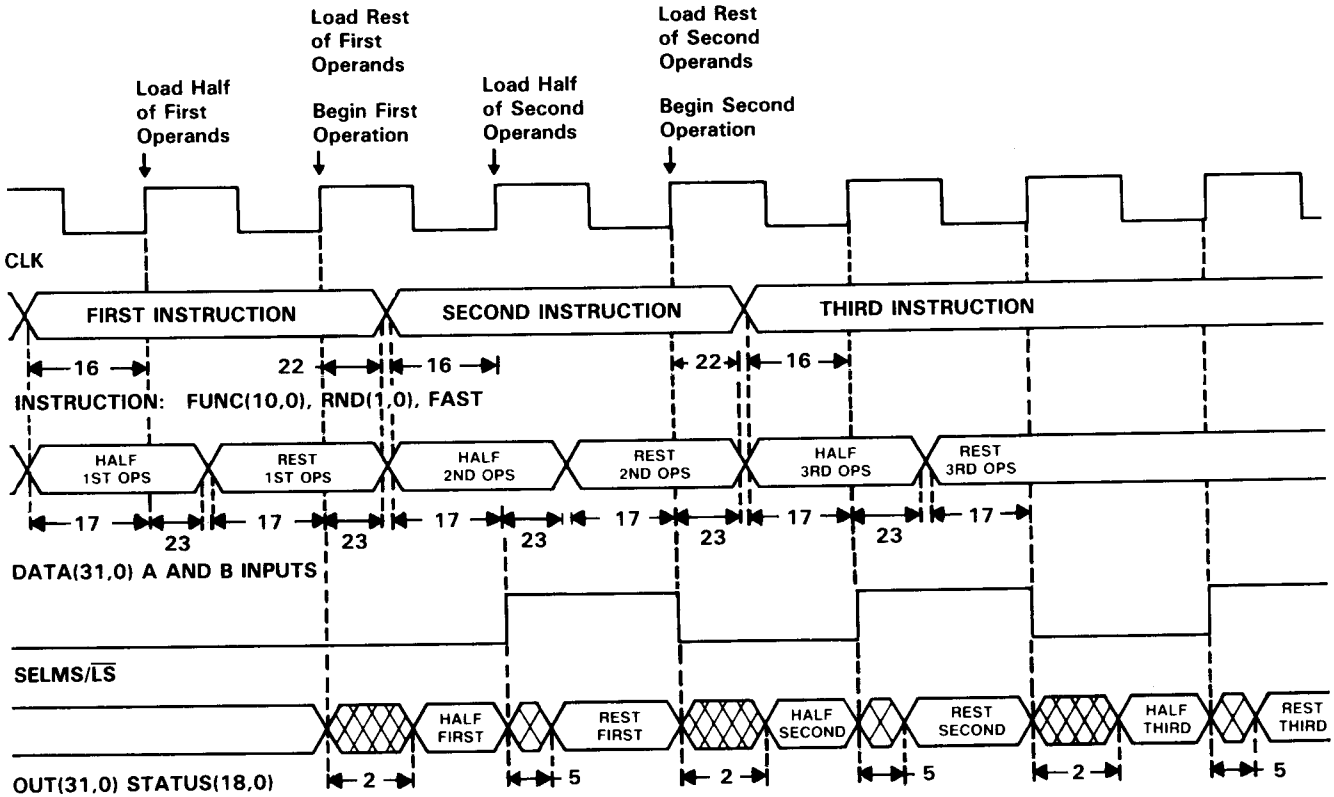
NOTE: Assume PIPES2-0 = 000, CONFIG1-0 = 01, ENRA = 1, ENRB = 1, SELMS/$\overline{LS}$ = X, $\overline{OEY}$ = 0, $\overline{OEC}$ = $\overline{OES}$ = 0, $\overline{RESET}$ = $\overline{HALT}$ = 1, TP1-0 = 11

Figure 27. Single-Precision Independent ALU Operation, All Registers Enabled (PIPES2-PIPES0 = 000, CLKMODE = X)



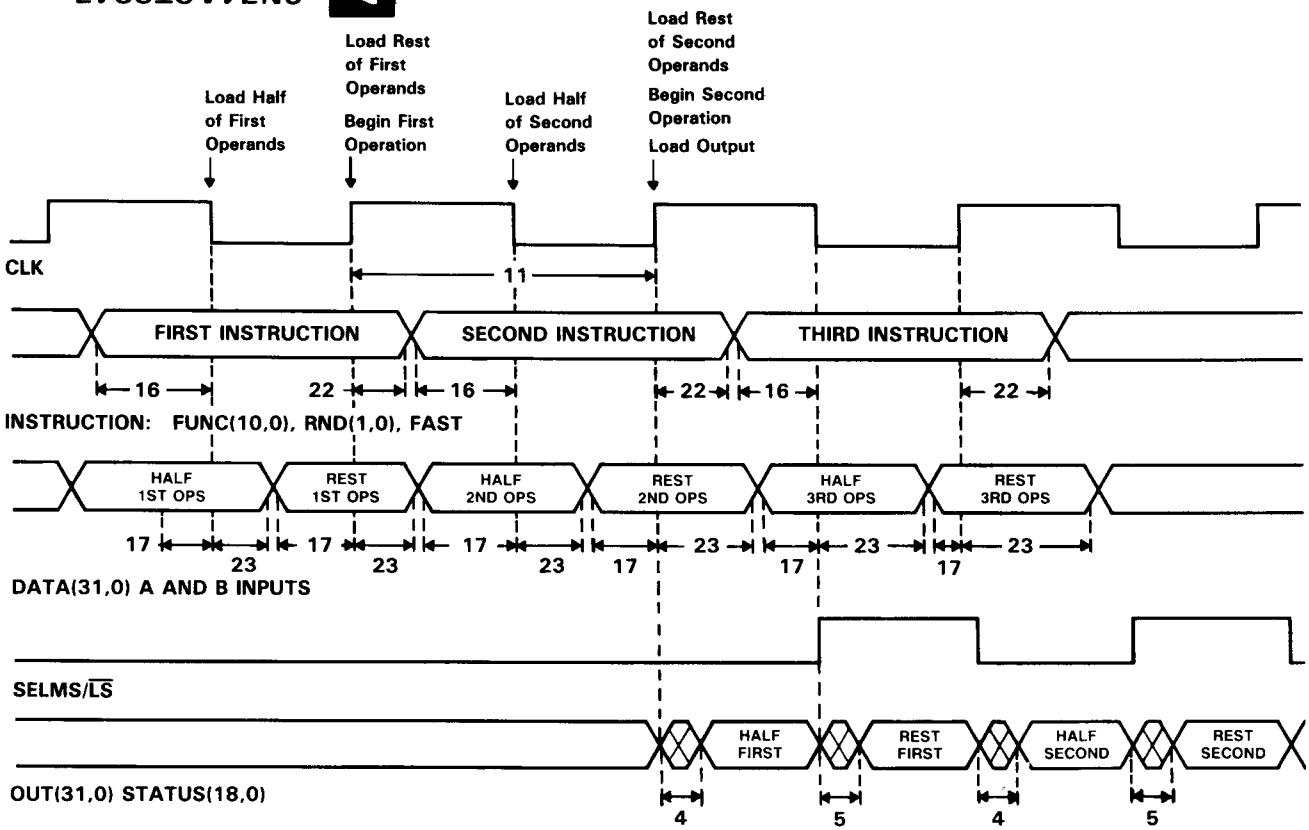
NOTE: Assume PIPES2-0 = 111, CLKMODE = 0, CONFIG1-0 = 11, ENRA = X, ENRB = X, $\overline{OEY}$ = 0, $\overline{OEC}$ = $\overline{OES}$ = 0, $\overline{RESET}$ = HALT = 1, TP1-0 = 11

Figure 28. Double-Precision Independent ALU Operation, All Registers Disabled (PIPES2-PIPES0 = 111, CLKMODE = 0)



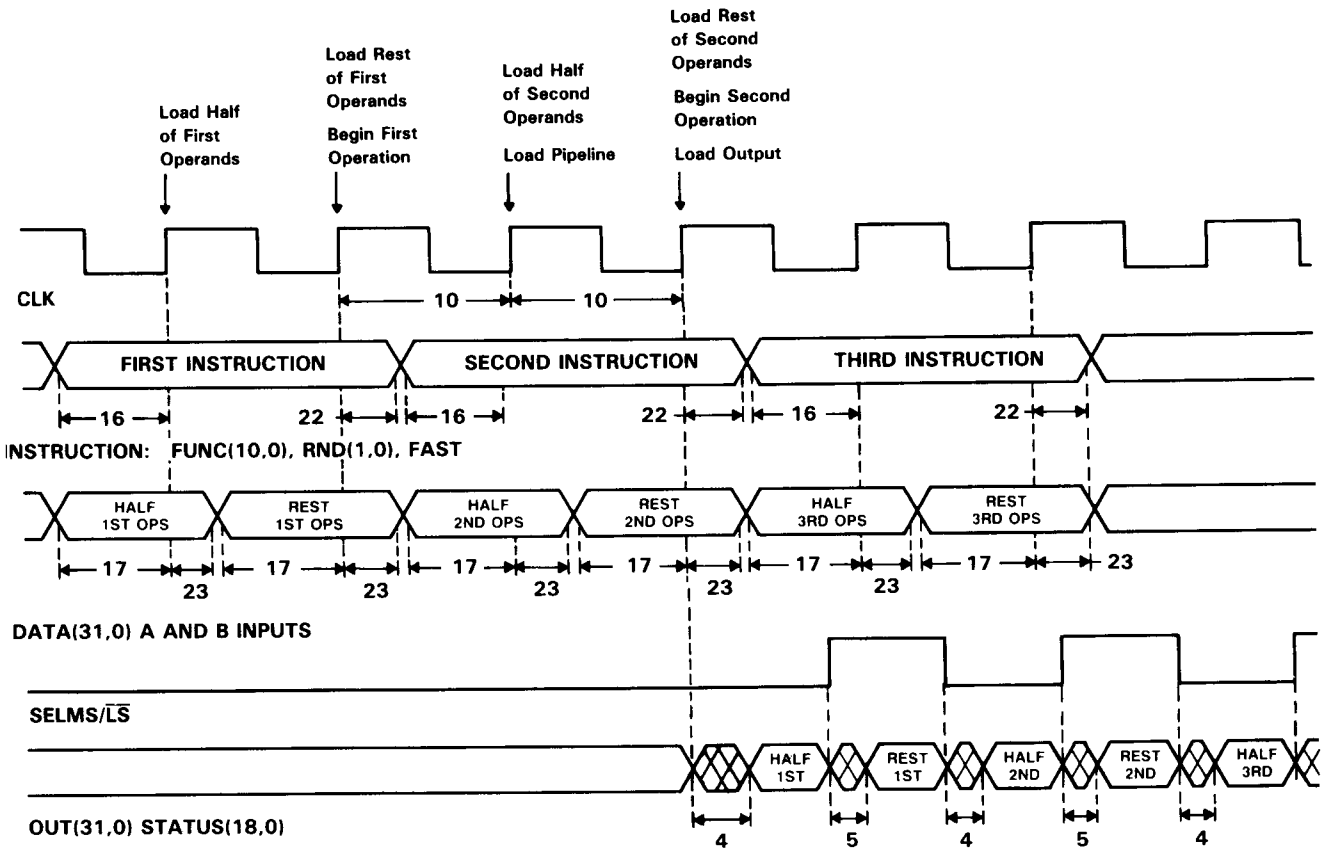
NOTE: Assume PIPES2-0 = 110, CLKMODE = 0, CONFIG1-0 = 00, ENRA = 1, ENRB = 1, $\overline{OEY}$ = 0, $\overline{OEC}$ = $\overline{OES}$ = 0, $\overline{RESET}$ = $\overline{HALT}$ = 1, TP1-0 = 11

Figure 29. Double-Precision Independent ALU Operation, Input Registers Enabled (PIPES2-PIPES0 = 110, CLKMODE = 0)



NOTE: Assume PIPES2-0 = 010, CLKMODE = 1, CONFIG1-0 = 11, ENRA = 1, ENRB = 1, $\overline{OEY}$ = 0, $\overline{OEC}$ = $\overline{OES}$ = 0, $\overline{RESET}$ = $\overline{HALT}$ = 1, TP1-0 = 11

Figure 30. Double-Precision Independent ALU Operation, Input and Output Registers Enabled (PIPES2-PIPES0 = 010, CLKMODE = 1)

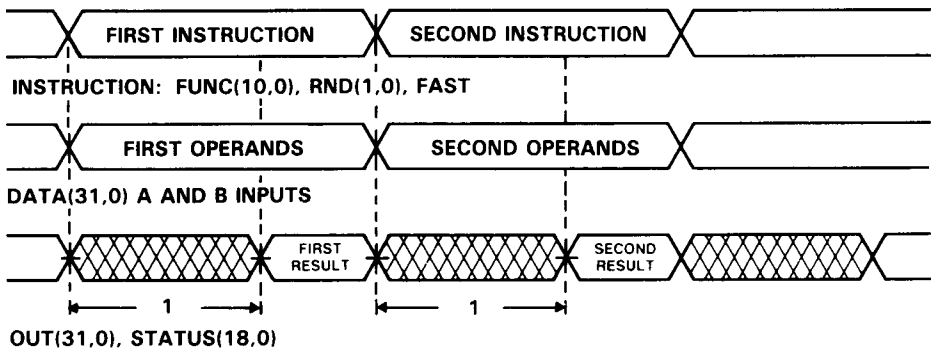


NOTE: Assume PIPES2-0 = 000, CLKMODE = 0, CONFIG1-0 = 11, ENRA = 1, ENRB = 1, $\overline{OEY}$ = 0, $\overline{OEC}$ = $\overline{OES}$ = 0, $\overline{RESET}$ = $\overline{HALT}$ = 1, TP1-0 = 11

Figure 31. Double-Precision Independent ALU Operation, All Registers Enabled (PIPES2-PIPES0 = 000, CLKMODE = 0)

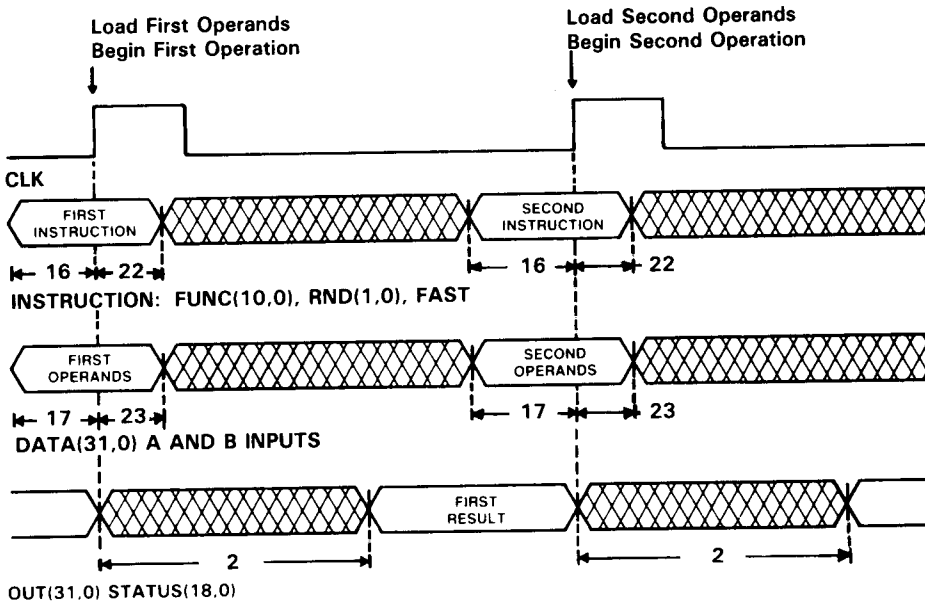
Sample Independent Multiplier Microinstructions

The following independent multiplier timing diagram examples show five register settings, ranging through fully pipelined. Examples for divide and square root are included in this section. X = don't care.



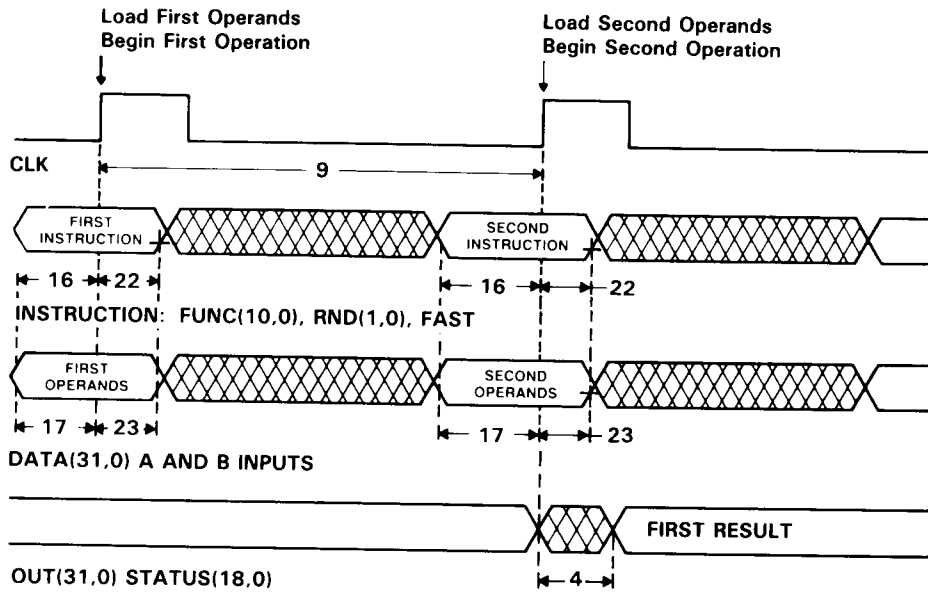
NOTE: Assume PIPES2-0 = 111, CONFIG1-0 = 01, ENRA = X, ENRB = X, SELMS/$\overline{\text{LSX}}$, $\overline{\text{OEY}}$ = 0, $\overline{\text{OEC}}$ = $\overline{\text{OES}}$ = 0, $\overline{\text{RESET}}$ = $\overline{\text{HALT}}$ = 1 TP1-0 = 11

Figure 32. Single-Precision Independent Multiplier Operation, All Registers Disabled (PIPES2-PIPES0 = 111, CLKMODE = X)



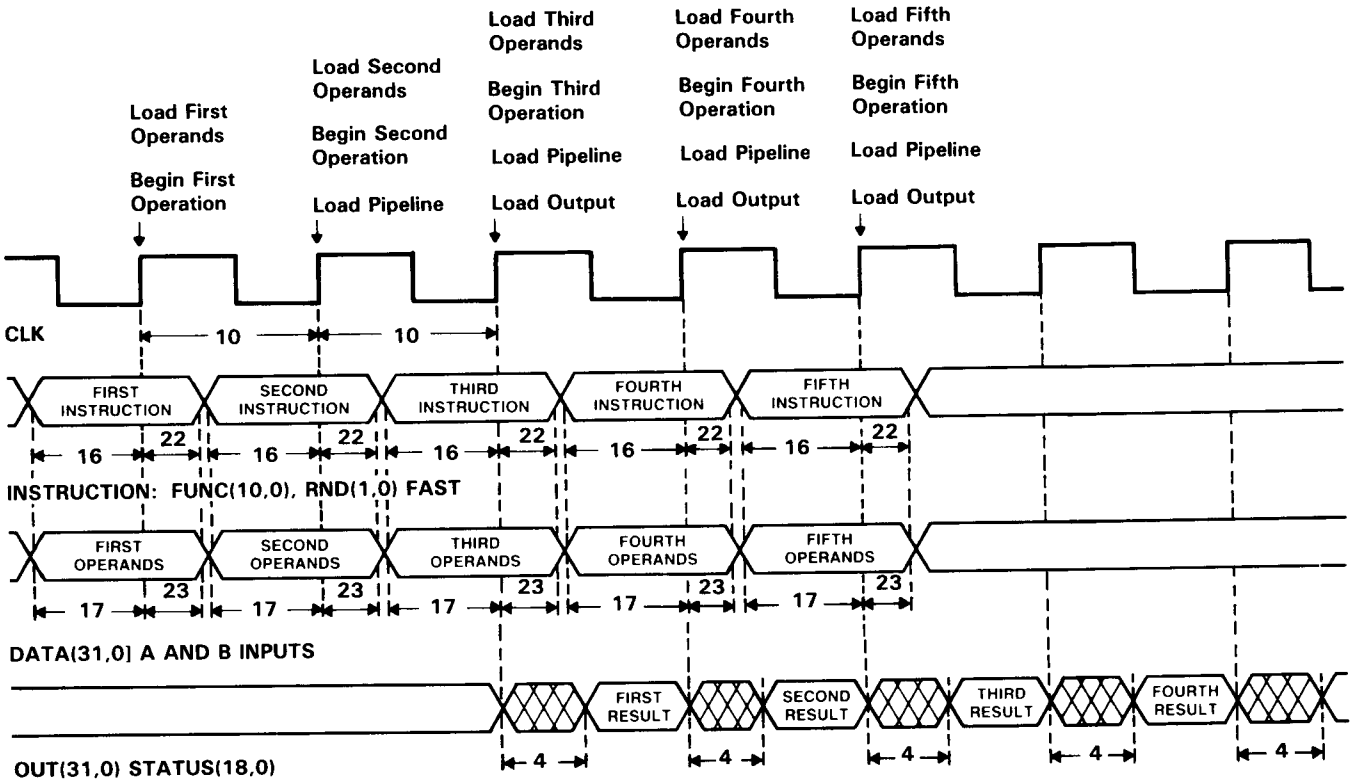
NOTE: Assume PIPES2-0 = 110, CONFIG1-0 = 01, ENRA = 1, ENRB = 1, SELMS/$\overline{LS}$ = X, $\overline{OEY}$ = 0, $\overline{OEC}$ = $\overline{OES}$ = 0, $\overline{RESET}$ = $\overline{HALT}$ = 1 TP1-0 = 11

Figure 33. Single-Precision Independent Multiplier Operation, Input Registers Enabled (PIPES2-PIPES0 = 010, CLKMODE = X)



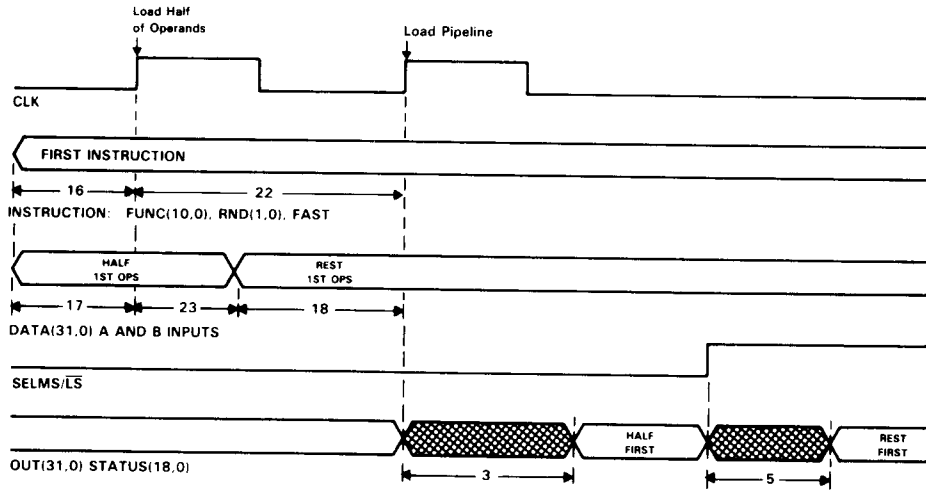
NOTE: Assume PIPES2-0 = 010, CONFIG1-0 = 01, ENRA = 1, ENRB = 1, SELMS/$\overline{LS}$ = X, $\overline{OEY}$ = 0, $\overline{OEC}$ = $\overline{OES}$ = 0, $\overline{RESET}$ = $\overline{HALT}$ = 1 TP1-0 = 11

Figure 34. Single-Precision Independent Multiplier Operation, Input and Output Registers Enabled (PIPES2-PIPES0 = 010, CLKMODE = X)



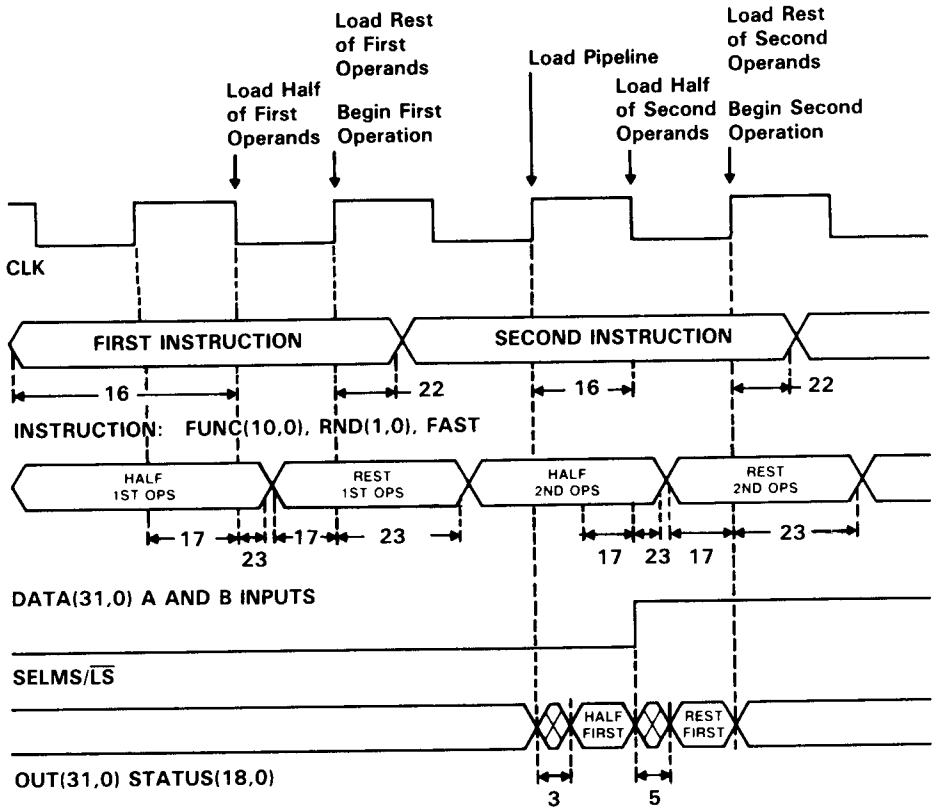
NOTE: Assume PIPES2-0 = 000, CONFIG1-0 = 01, ENRA = 1, ENRB = 1, SELMS/$\overline{LS}$ = X, $\overline{OEY}$ = 0, $\overline{OEC}$ = $\overline{OES}$ = 0, $\overline{RESET}$ = $\overline{HALT}$ = 1, TP1-0 = 11

Figure 35. Single-Precision Independent Multiplier Operation, All Registers Enabled (PIPES2-PIPES0 = 000, CLKMODE = X)



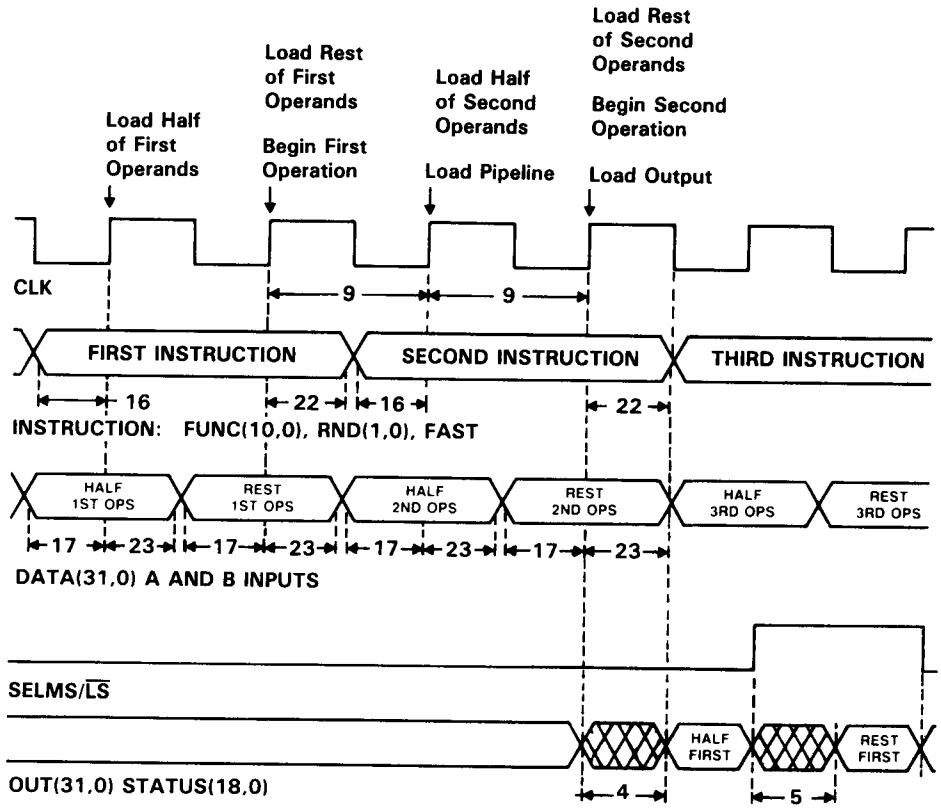
NOTE: Assume PIPES2-0 = 111, CLKMODE = 0, CONFIG1-0 = 11, ENRA = X, ENRB = X, $\overline{OEY}$ = 0, $\overline{OEC}$ = $\overline{OES}$ = 0, $\overline{RESET}$ = $\overline{HALT}$ = 1, TP1-0 = 11

Figure 36. Double-Precision Independent Multiplier Operation, All Registers Disabled (PIPES2-PIPES0 = 111, CLKMODE = 0)



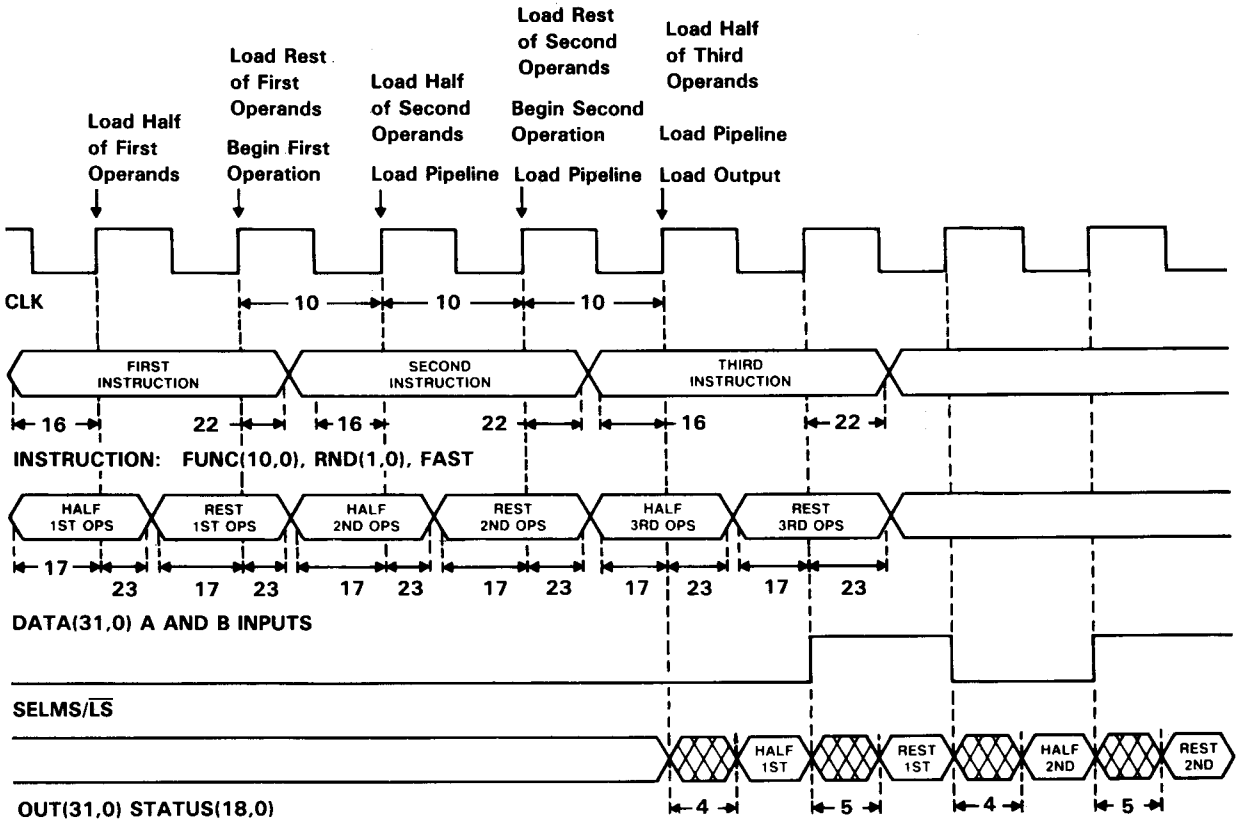
NOTE: Assume PIPES2-0 = 110, CONFIG1-0 = 11, ENRA = 1, ENRB = 1, $\overline{OEY}$ = 0, $\overline{OEC}$ = $\overline{OES}$ = 0, $\overline{RESET}$ = $\overline{HALT}$ = 1, TP1-0 = 11

Figure 37. Double-Precision Independent Multiplier Operation, Input Registers Enabled (PIPES2-PIPES0 = 110, CLKMODE = 1)



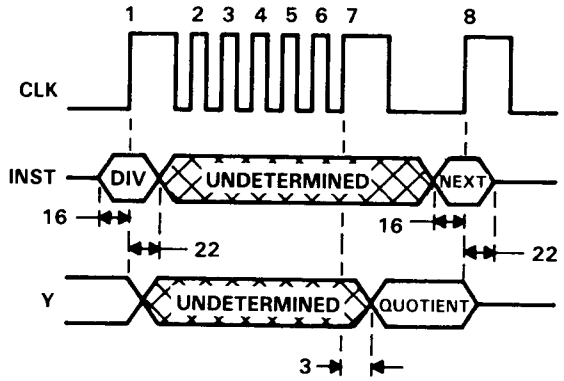
NOTE: Assume PIPES2-0 = 010, CONFIG1-0 = 10, ENRA = 1, ENRB = 1, $\overline{OEY}$ = 0, $\overline{OEC}$ = $\overline{OES}$ = 0, $\overline{RESET}$ = $\overline{HALT}$ = 1, TP1-0 = 11

Figure 38. Double-Precision Independent Multiplier Operation, Input and Output Registers Enabled (PIPES2-PIPES0 = 010, CLKMODE = 0)



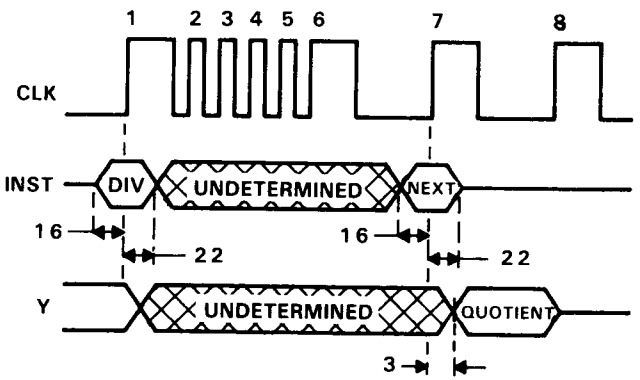
NOTE: Assume PIPES2-0 = 000, CONFIG1-0 = 01, ENRA = 1, ENRB = 1, $\overline{OEY}$ = 0, $\overline{OEC}$ = 0, $\overline{OES}$ = 0, $\overline{RESET}$ = $\overline{HALT}$ = 1, TP1-0 = 11

Figure 39. Double-Precision Independent Multiplier Operation, All Registers Enabled (PIPES2-PIPES0 = 000, CLKMODE = 0)



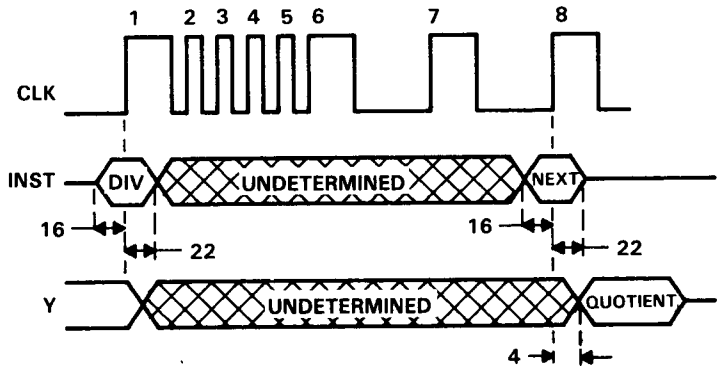
NOTE: Assume PIPES2-0 = 110, CONFIG1-0 = 01, ENRA = 1, ENRB = 1, SELMS/$\overline{LS}$ = X, $\overline{OEY}$ = 0, $\overline{OEC}$ = $\overline{OES}$ = 0, $\overline{RESET}$ = $\overline{HALT}$ = 1, TP1-0 = 11

Figure 40. Single-Precision Floating Point Division (PIPES2-PIPES0 = 110, CLKMODE = X)



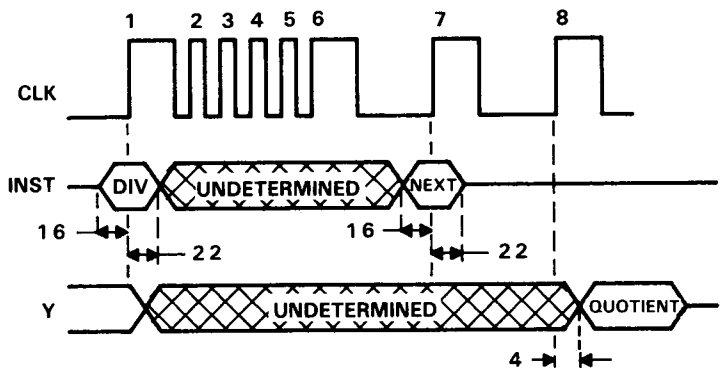
NOTE: Assume PIPES2-0 = 100, CONFIG1-0 = 01, ENRA = 1, ENRB = 1, SELMS/$\overline{LS}$ = X, $\overline{OEY}$ = 0, $\overline{OEC}$ = $\overline{OES}$ = 0, $\overline{RESET}$ = $\overline{HALT}$ = 1 TP1-0 = 11

Figure 41. Single-Precision Floating Point Division (PIPES2-PIPES0 = 100, CLKMODE = X)



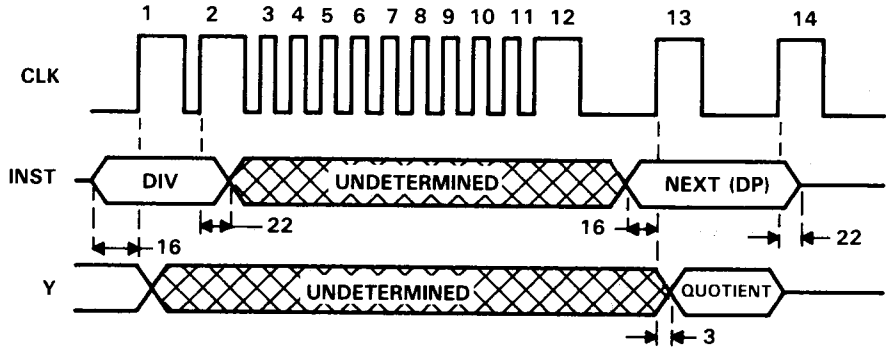
NOTE: Assume PIPES2-0 = 010, CONFIG1-0 = 01, ENRA = 1, ENRB = 1, SELMS/$\overline{LS}$ = X, $\overline{OEY}$ = 0, $\overline{OEC}$ = $\overline{OES}$ = 0, $\overline{RESET}$ = $\overline{HALT}$ = 1, TP1-0 = 11

Figure 42. Single-Precision Floating Point Division (PIPES2-PIPES0 = 010, CLKMODE = X)



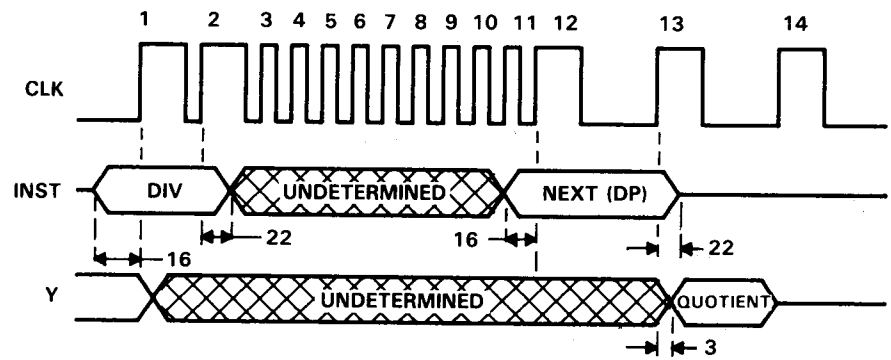
NOTE: Assume PIPES2-0 = 000, CONFIG1-0 = 01, ENRA = 1, ENRB = 1, SELMS/$\overline{LS}$ = X, $\overline{OEY}$ = 0, $\overline{OEC}$ = $\overline{OES}$ = 0, $\overline{RESET}$ = $\overline{HALT}$ = 1, TP1-0 = 11

Figure 43. Single-Precision Floating Point Division (PIPES2-PIPES0 = 000, CLKMODE = X)



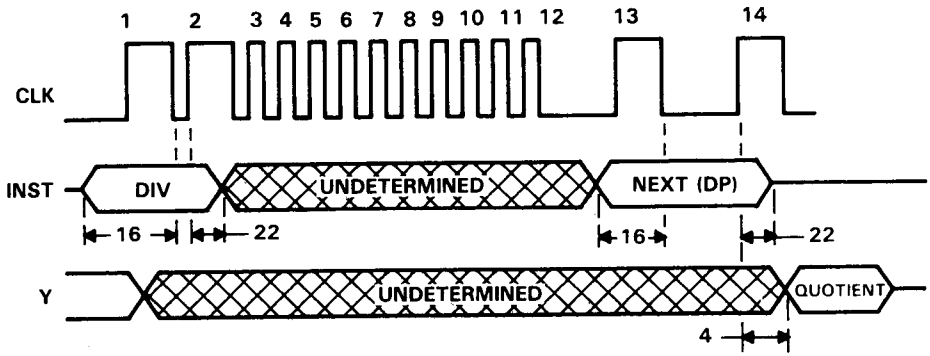
NOTE: Assume PIPES2-0 = 110, CONFIG1-0 = 01, ENRA = 1, ENRB = 1, $\overline{OEY}$ = 0, $\overline{OEC}$ = $\overline{OES}$ = 0, $\overline{RESET}$ = $\overline{HALT}$ = 1, TP1-0 = 11

Figure 44. Double-Precision Floating Point Division (PIPES2-PIPES0 = 110, CLKMODE = 0)



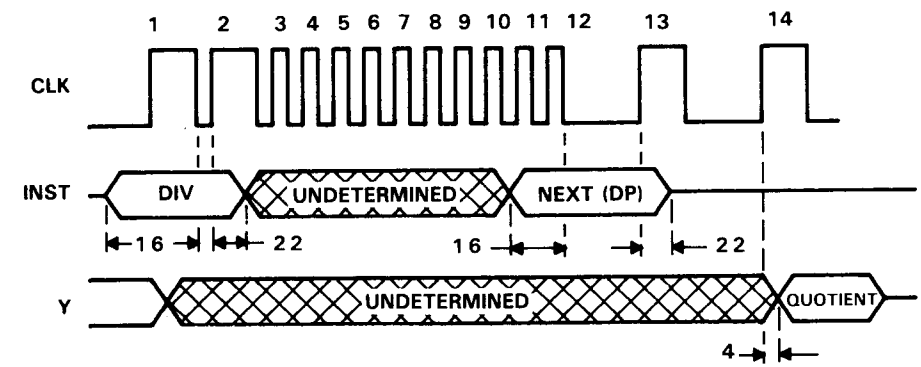
NOTE: Assume PIPES2-0 = 100, CONFIG1-0 = 01, ENRA = 1, ENRB = 1, $\overline{OEY}$ = 0, $\overline{OEC}$ = $\overline{OES}$ = 0, $\overline{RESET}$ = $\overline{HALT}$ = 1, TP1-0 = 11

Figure 45. Double-Precision Floating Point Division (PIPES2-PIPES0 = 100, CLKMODE = 0)



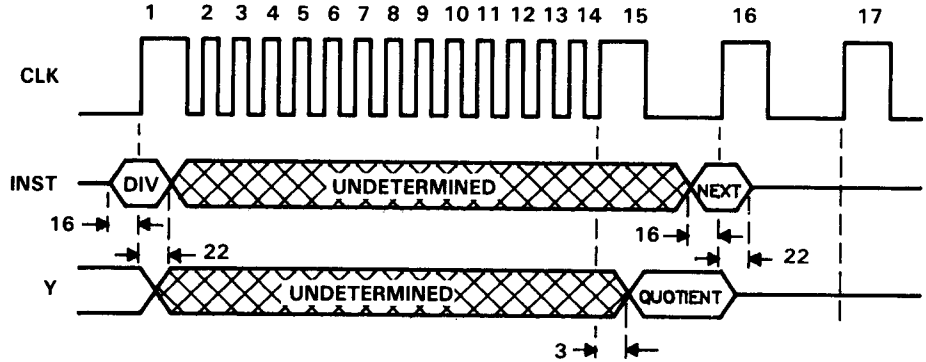
NOTE: Assume PIPES2-0 = 010, CONFIG1-0 = 01, ENRA = 1, ENRB = 1, SELMS/$\overline{LS}$ = X, $\overline{OEY}$ = 0, $\overline{OEC}$ = $\overline{OES}$ = 0, $\overline{RESET}$ = $\overline{HALT}$ = 1, TP1-0 = 11

Figure 46. Double-Precision Floating Point Division (PIPES2-PIPES0 = 010, CLKMODE = 1)



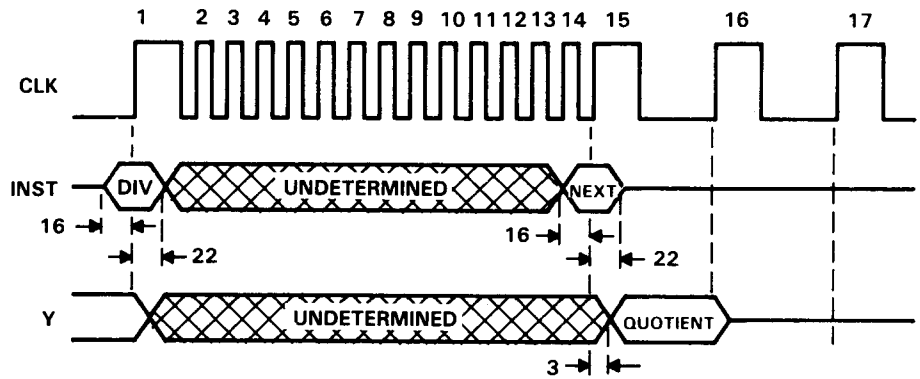
NOTE: Assume PIPES2-0 = 000, CONFIG1-0 = 00, ENRA = 1, ENRB = 1, $\overline{OEY}$ = 0, $\overline{OEC}$ = $\overline{OES}$ = 0, $\overline{RESET}$ = $\overline{HALT}$ = 1, TP1-0 = 11

Figure 47. Double-Precision Floating-Point Division, All Registers Enabled (PIPES2-PIPES0 = 000, CLKMODE = 1)



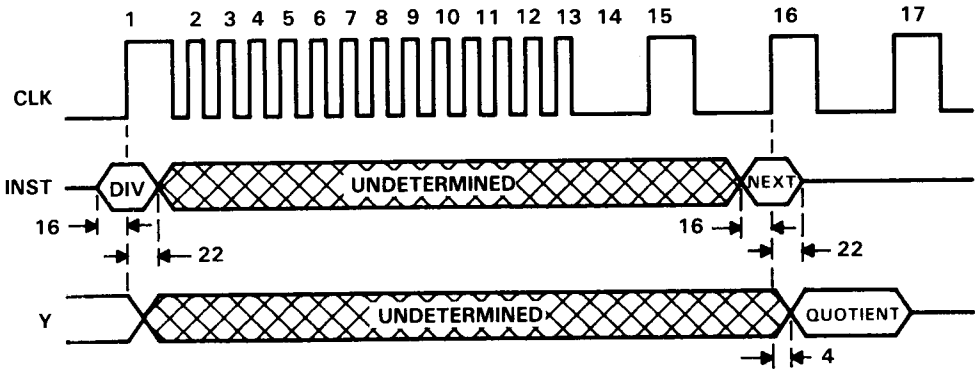
NOTE: Assume PIPES2-0 = 110, CONFIG1-0 = 01, ENRA = 1, ENRB = 1, SELMS/$\overline{LS}$ = X, $\overline{OEY}$ = 0, $\overline{OEC}$ = $\overline{OES}$ = 0, $\overline{RESET}$ = $\overline{HALT}$ = 1, TP1-0 = 11. The result appears in the SREG.

Figure 48. Integer Division, Input Registers Enabled (PIPES2-PIPES0 = 110, CLKMODE = X)



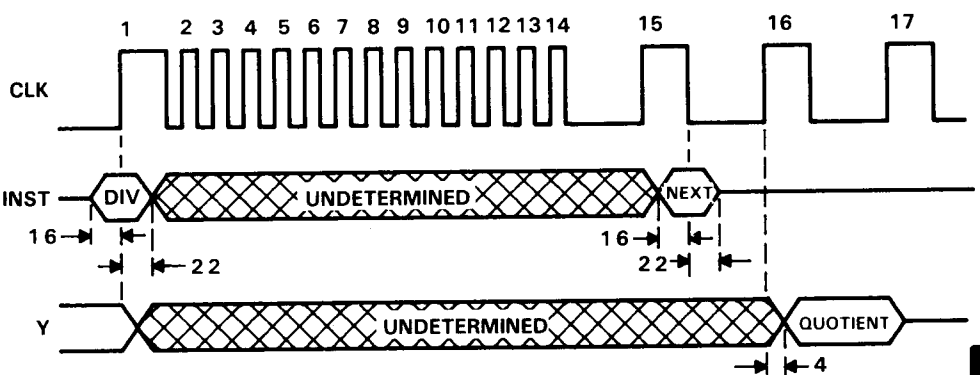
NOTE: Assume PIPES2-0 = 100, CONFIG1-0 = 01, ENRA = 1, ENRB = 1, SELMS/$\overline{LS}$ = X, $\overline{OEY}$ = 0, $\overline{OEC}$ = $\overline{OES}$ = 0, $\overline{RESET}$ = $\overline{HALT}$ = 1, TP1-0 = 11. The result appears in the SREG.

Figure 49. Integer Division, Input and Pipeline Registers Enabled (PIPES2-PIPES0 = 100, CLKMODE = X)



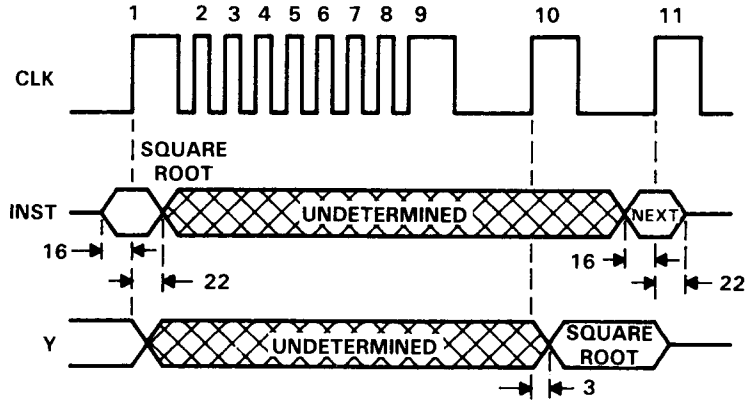
NOTE: Assume PIPES2-0 = 010, CONFIG1-0 = 01, ENRA = 1, ENRB = 1, SELMS/$\overline{LS}$ = X, $\overline{OEY}$ = 0, $\overline{OEC}$ = $\overline{OES}$ = 0, $\overline{RESET}$ = $\overline{HALT}$ = 1, TP1-0 = 11. The result appears in the SREG.

Figure 50. Integer Division, Input and Output Registers Enabled (PIPES2-PIPES0 = 010, CLKMODE = X)



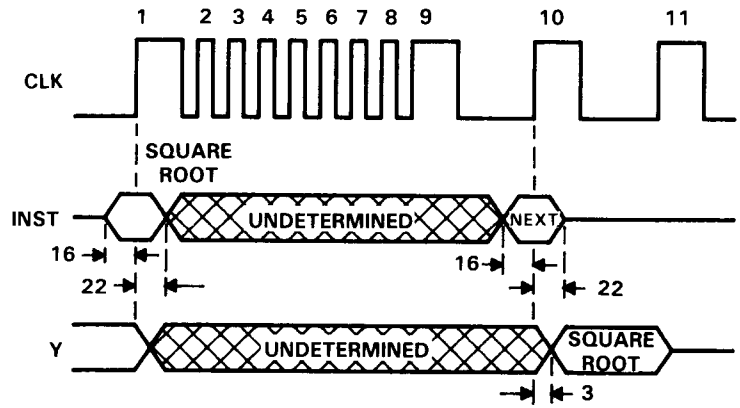
NOTE: Assume PIPES2-0 = 000, CONFIG1-0 = 01, ENRA = 1, ENRB = 1, SELMS/$\overline{LS}$ = X, $\overline{OEY}$ = 0, $\overline{OEC}$ = $\overline{OES}$ = 0, $\overline{RESET}$ = $\overline{HALT}$ = 1, TP1-0 = 11. The result appears in the SREG.

Figure 51. Integer Division, All Registers Enabled (PIPES2-PIPES0 = 000, CLKMODE = X)



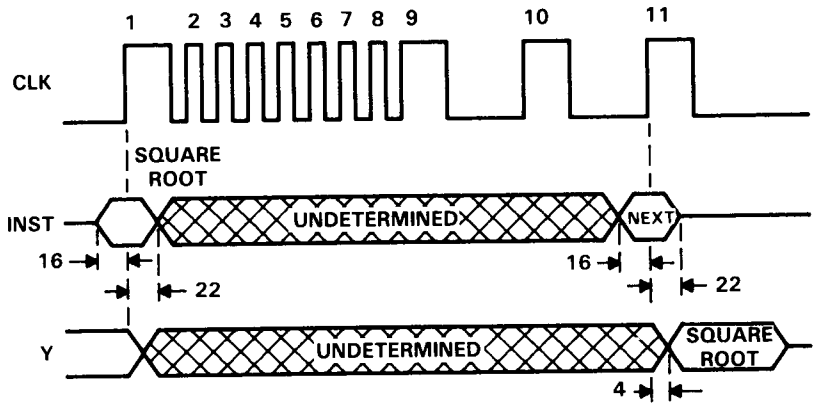
NOTE: Assume PIPES2-0 = 110, CONFIG1-0 = 01, ENRA = 1, ENRB = 1, SELMS/$\overline{LS}$ = X, $\overline{OEY}$ = 0, $\overline{OEC}$ = $\overline{OES}$ = 0, $\overline{RESET}$ = HALT = 1, TP1-0 = 11

Figure 52. Single-Precision Floating Point Square Root, Input Registers Enabled (PIPES2-PIPES0 = 110, CLKMODE = X)



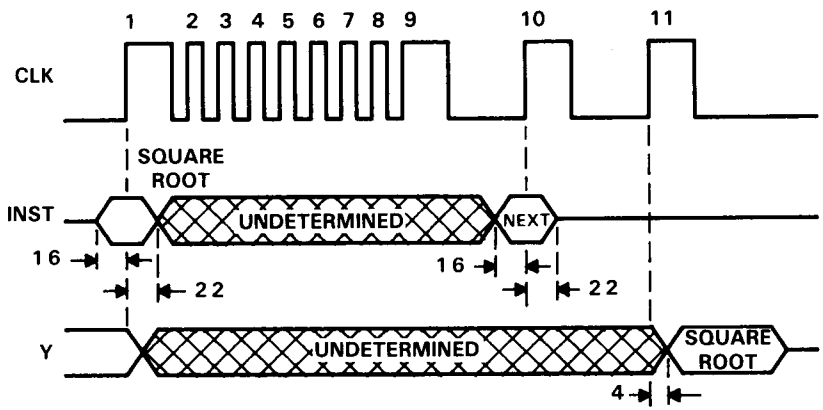
NOTE: Assume PIPES2-0 = 110, CONFIG1-0 = 01, ENRA = 1, ENRB = 1, SELMS/$\overline{LS}$ = X, $\overline{OEY}$ = 0, $\overline{OEC}$ = $\overline{OES}$ = 0, $\overline{RESET}$ = HALT = 1, TP1-0 = 11

Figure 53. Single-Precision Floating Point Square Root, Input and Pipeline Registers Enabled (PIPES2-PIPES0 = 100, CLKMODE = X)



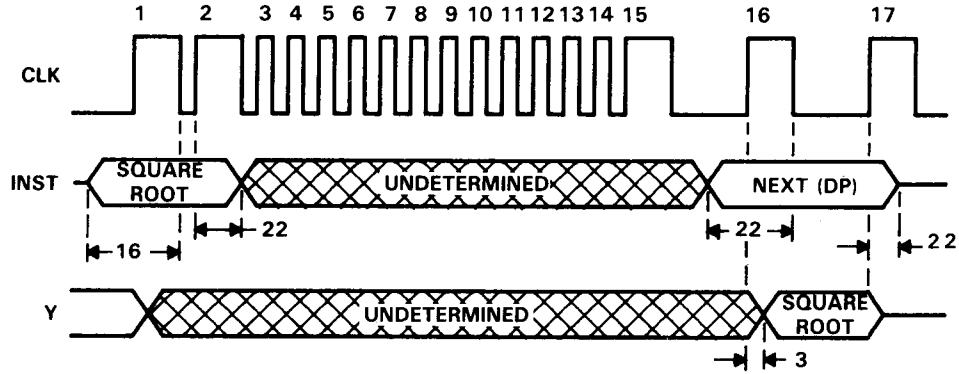
NOTE: Assume PIPES2-0 = 010, CONFIG1-0 = 01, ENRA = 1, SELMS/$\overline{LS}$ = X, $\overline{OEY}$ = 0, $\overline{OEC}$ = $\overline{OES}$ = 0, $\overline{RESET}$ = $\overline{HALT}$ = 1, TP1-0 = 11

Figure 54. Single-Precision Floating Point Square Root, Input and Output Registers Enabled (PIPES2-PIPES0 = 010, CLKMODE = X)



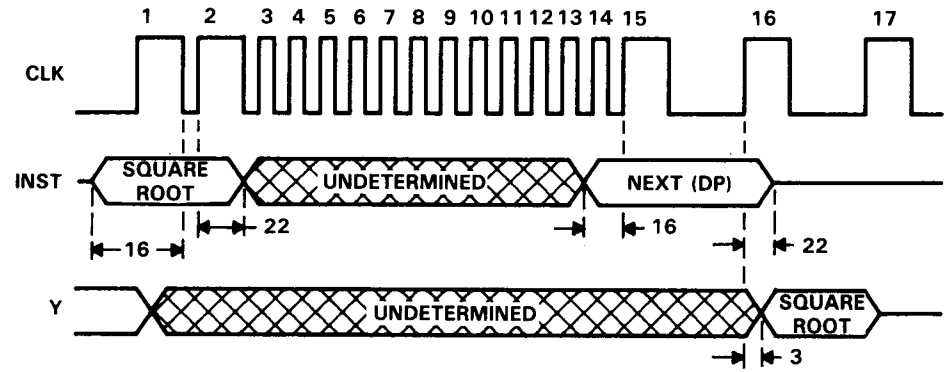
NOTE: Assume PIPES2-0 = 000, CONFIG1-0 = 00, ENRA = 1, SELMS/$\overline{LS}$ = X, $\overline{OEY}$ = 0, $\overline{OEC}$ = $\overline{OES}$ = 0, $\overline{RESET}$ = $\overline{HALT}$ = 1, TP1-0 = 11

Figure 55. Single-Precision Floating Point Square Root, All Registers Enabled (PIPES2-PIPES0 = 000, CLKMODE = X)



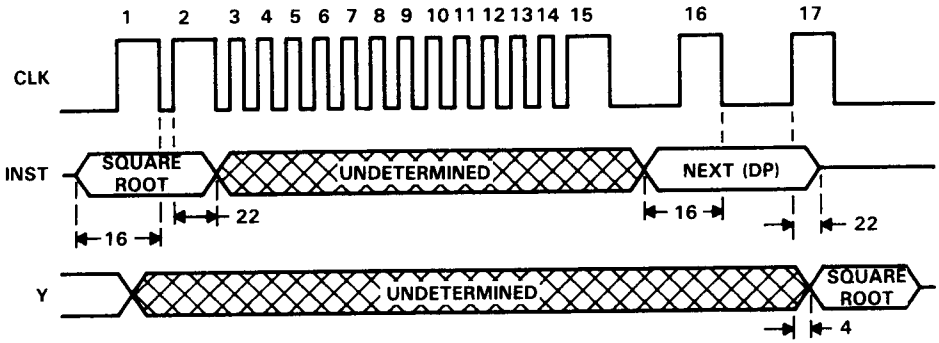
NOTE: Assume PIPES2-0 = 110, CONFIG1-0 = 11, ENRA = 1, $\overline{OEY}$ = 0, $\overline{OEC}$ = $\overline{OES}$ = 0
$\overline{RESET}$ = $\overline{HALT}$ = 1, TP1-0 = 11

Figure 56. Double-Precision Floating Point Square Root, Input Registers Enabled (PIPES2-PIPES0 = 110, CLKMODE = 1)



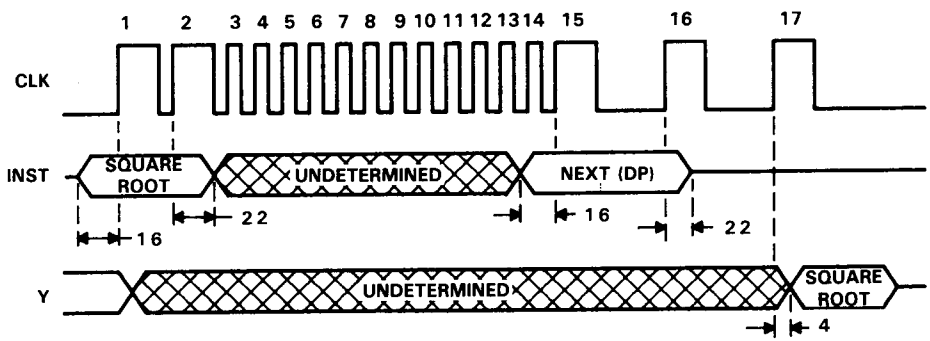
NOTE: Assume PIPES2-0 = 100, CONFIG1-0 = 01, ENRA = 1, $\overline{OEY}$ = 0, $\overline{OEC}$ = $\overline{OES}$ = 0,
$\overline{RESET}$ = $\overline{HALT}$ = 1, TP1-0 = 11

Figure 57. Double-Precision Floating Point Square Root, Input and Pipeline Registers Enabled (PIPES2-PIPES0 = 100, CLKMODE = 0)



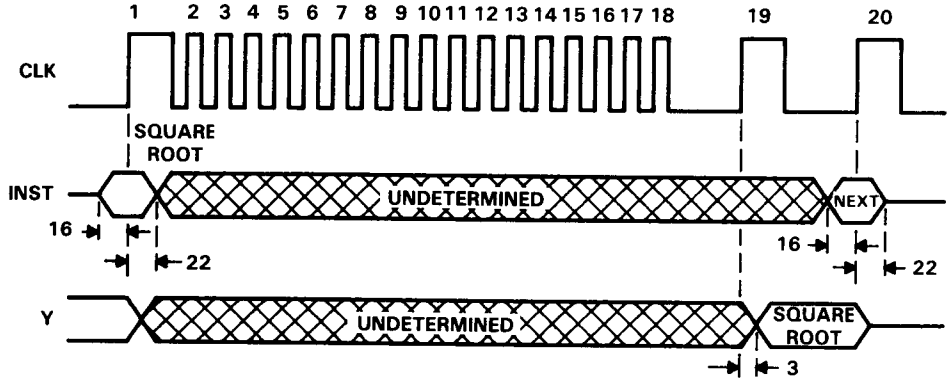
NOTE: Assume PIPES2-0 = 010, CONFIG1-0 = 10, ENRA = 1, $\overline{OEY}$ = 0, $\overline{OEC}$ = $\overline{OES}$ = 0, $\overline{RESET}$ = $\overline{HALT}$ = 1, TP1-0 = 11

Figure 58. Double-Precision Floating Point Square Root, Input and Output Registers Enabled (PIPES2-PIPES0 = 010, CLKMODE = 1)



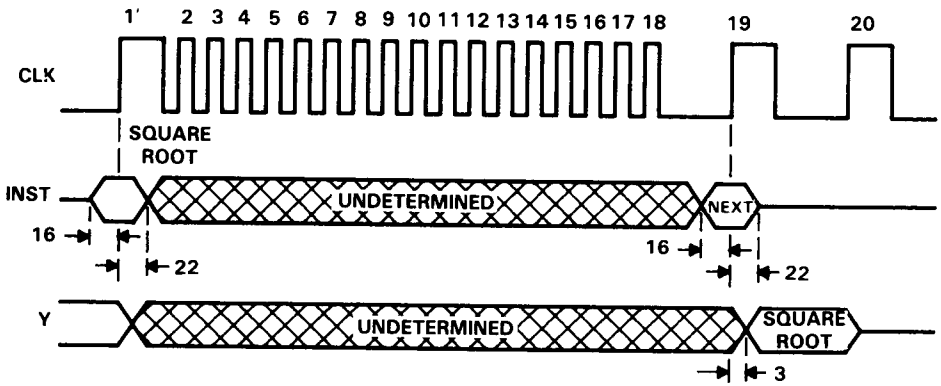
NOTE: Assume PIPES2-0 = 000, CONFIG1-0 = 00, ENRA = 1, $\overline{OEY}$ = 0, $\overline{OEC}$ = $\overline{OES}$ = 0, $\overline{RESET}$ = $\overline{HALT}$ = 1, TP1-0 = 11

Figure 59. Double-Precision Floating Point Square Root, All Registers Enabled (PIPES2-PIPES0 = 000, CLKMODE = 0)



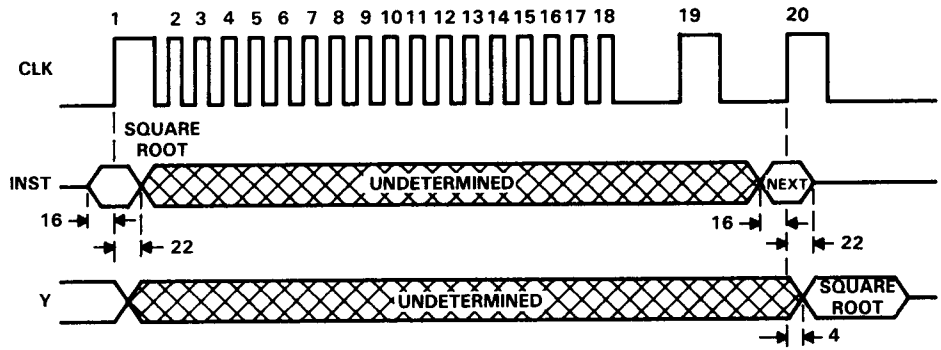
NOTE: Assume PIPES2-0 = 110, CONFIG1-0 = 01, ENRA = 1, SELM/$\overline{LS}$ = X, $\overline{OEY}$ = 0, $\overline{OEC}$ = $\overline{OES}$ = 0, $\overline{RESET}$ = $\overline{HALT}$ = 1 TP1-0 = 11. The result appears in the SREG.

Figure 60. Integer Square Root, Input Registers Enabled (PIPES2-PIPES0 = 110, CLKMODE = X)



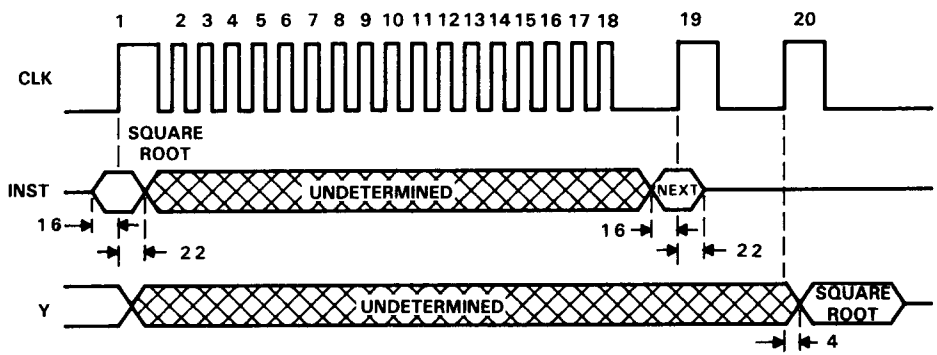
NOTE: Assume PIPES2-0 = 100, CONFIG1-0 = 00, ENRA = 1, SELMS/$\overline{LS}$ = X, $\overline{OEY}$ = 0, $\overline{OEC}$ = $\overline{OES}$ = 0, $\overline{RESET}$ = $\overline{HALT}$ = 1, TP1-0 = 11. The result appears in the SREG.

Figure 61. Integer Square Root, Input and Pipeline Registers Enabled (PIPES2-PIPES0 = 100, CLKMODE = X)



NOTE: Assume PIPES2-0 = 010, CONFIG1-0 = 01, ENRA = 1, SELMS/$\overline{LS}$ = X, $\overline{OEY}$ = 0, $\overline{OEC}$ = $\overline{OES}$ = 0, $\overline{RESET}$ = $\overline{HALT}$ = 1, TP1-0 = 11. The result appears in the SREG.

Figure 62. Integer Square Root, Input and Output Registers Enabled (PIPES2-PIPES0 = 010, CLKMODE = X)

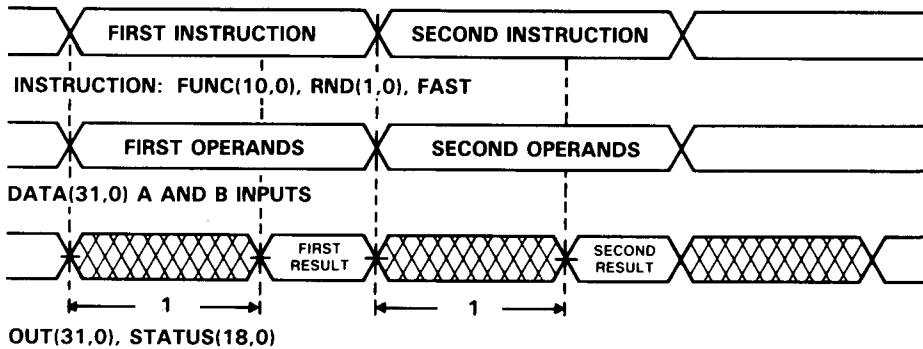


NOTE: Assume PIPES2-0 = 000, CONFIG1-0 = 00, ENRA = 1, SELMS/$\overline{LS}$ = X, $\overline{OEY}$ = 0, $\overline{OEC}$ = $\overline{OES}$ = 0, $\overline{RESET}$ = $\overline{HALT}$ = 1, TP1-0 = 11. The result appears in the SREG.

Figure 63. Integer Square Root, All Registers Enabled (PIPES2-PIPES0 = 000, CLKMODE = X)

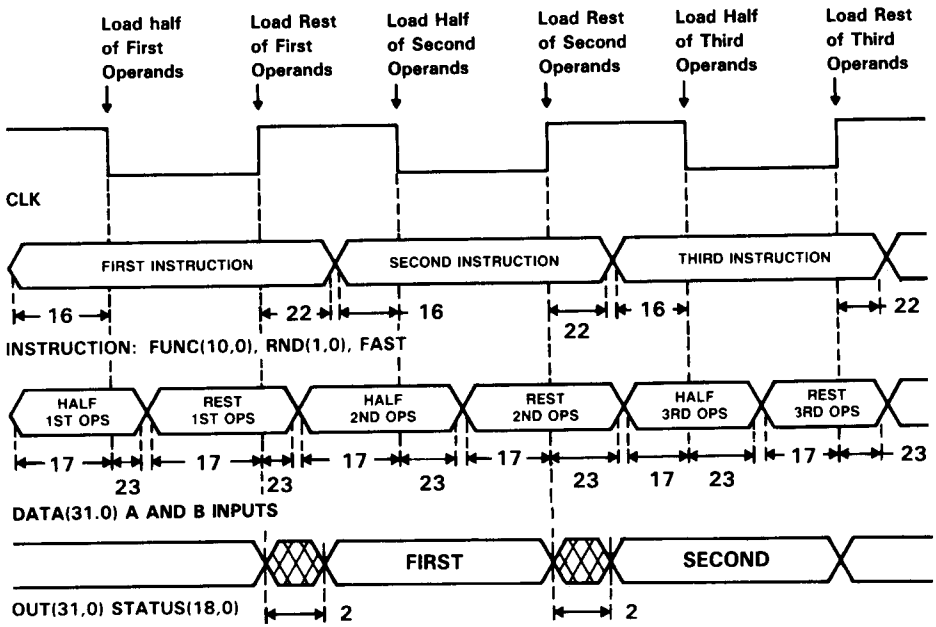
Sample Chained Mode Microinstructions

The following chained mode timing diagram examples show four register settings, ranging from fully flowthrough to fully pipelined.



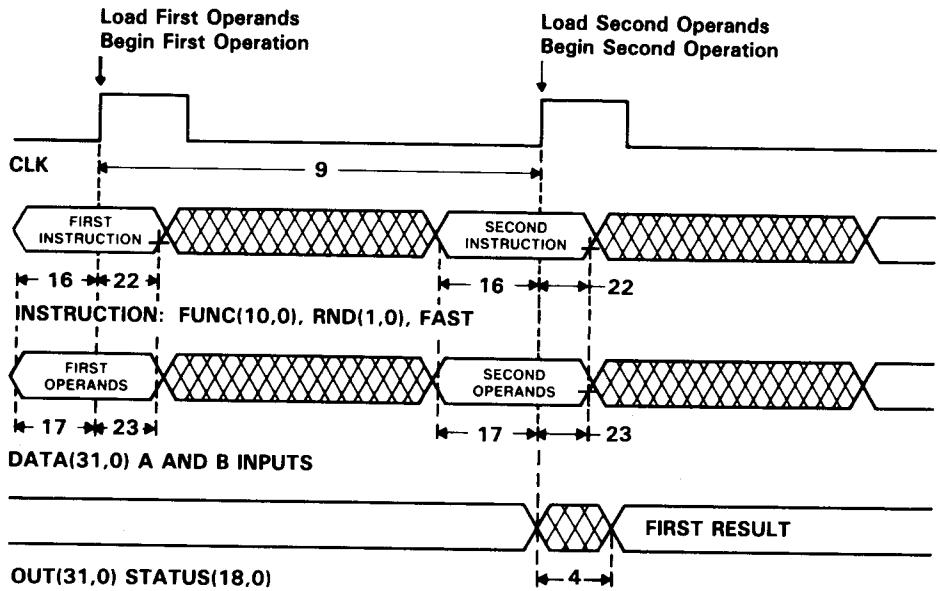
NOTE: Assume PIPES2-0 = 111, CONFIG1-0 = 01, ENRA = X, ENRB = X, SELMS/$\overline{LS}$, $\overline{OEY}$ = 0, $\overline{OEC}$ = $\overline{OES}$ = 0, $\overline{RESET}$ = $\overline{HALT}$ = 1, TP1-0 = 11

Figure 64. Single-Precision Chained Mode Operation, All Registers Disabled (PIPES2-PIPES0 = 111, CLKMODE = X)



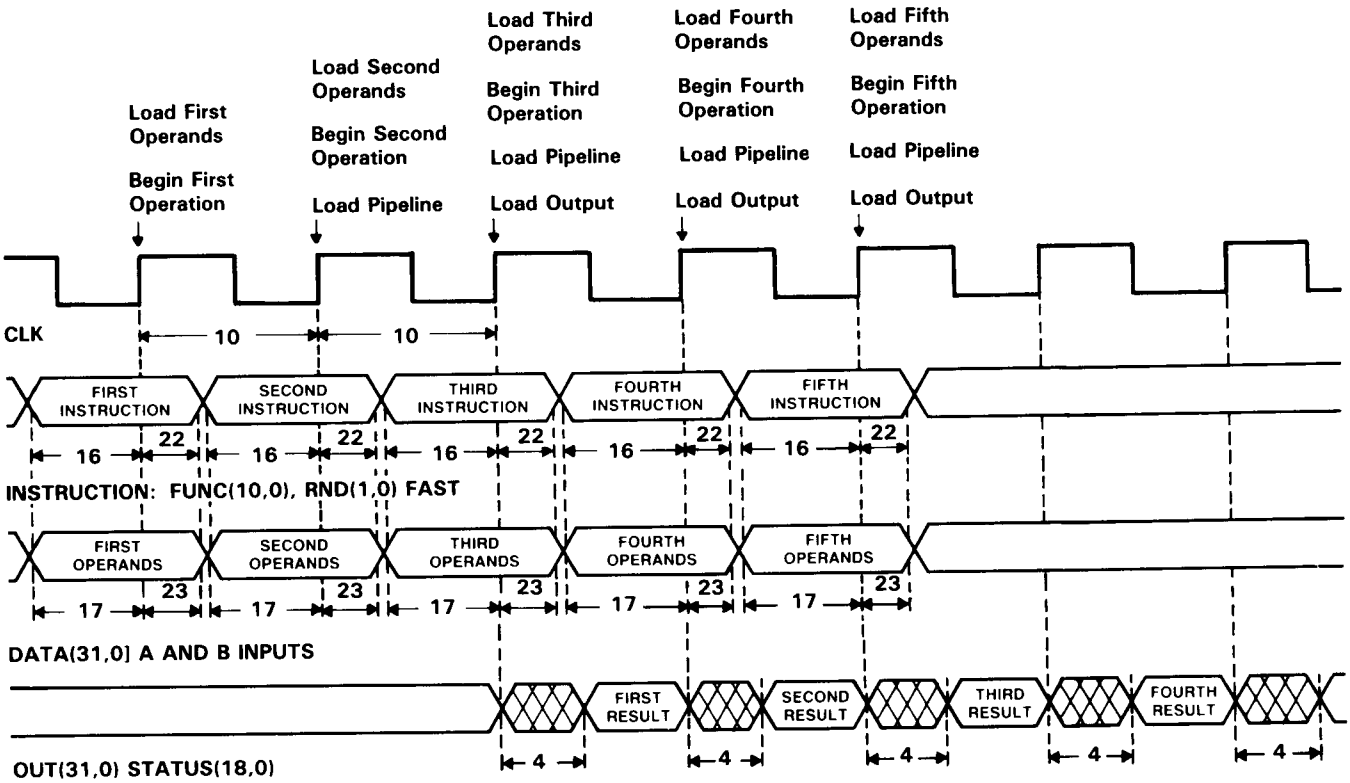
NOTE: Assume PIPES2-0 = 110, CONFIG1-0 = 11, ENRA = 1, ENRB = 1, SELMS/$\overline{LS}$ = X, $\overline{OEY}$ = 0, $\overline{OEC}$ = $\overline{OES}$ = 0, $\overline{RESET}$ = $\overline{HALT}$ = 1, TP1-0 = 11

Figure 65. Single-Precision Chained Mode Operation, Input Registers Enabled (PIPES2-PIPES0 = 110, CLKMODE = 1)



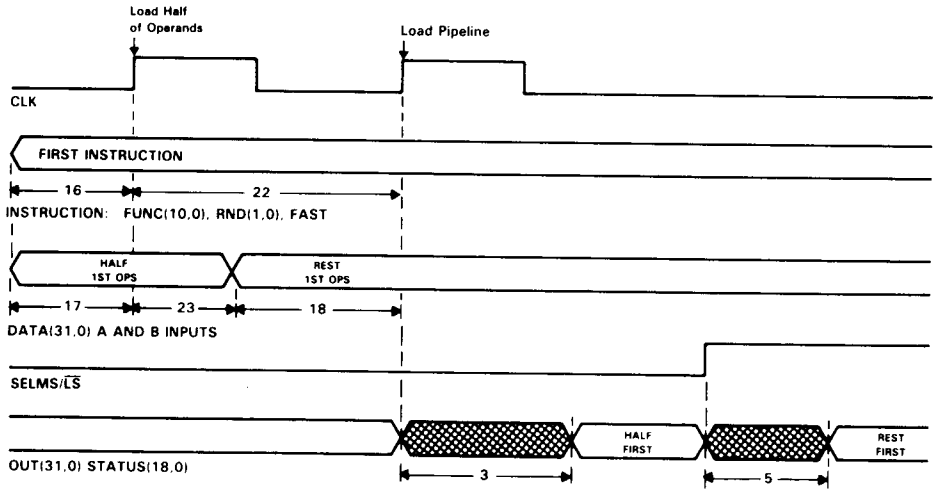
NOTE: Assume PIPES2-0 = 010, CONFIG1-0 = 01, ENRA = 1, SELMS/$\overline{LS}$ = X, $\overline{OEY}$ = 0, $\overline{OEC}$ = $\overline{OES}$ = 0, $\overline{RESET}$ = $\overline{HALT}$ = 1, TP1-0 = 11

Figure 66. Single-Precision Chained Mode Operation, Input and Output Registers Enabled (PIPES2-PIPES0 = 010, CLKMODE = X)



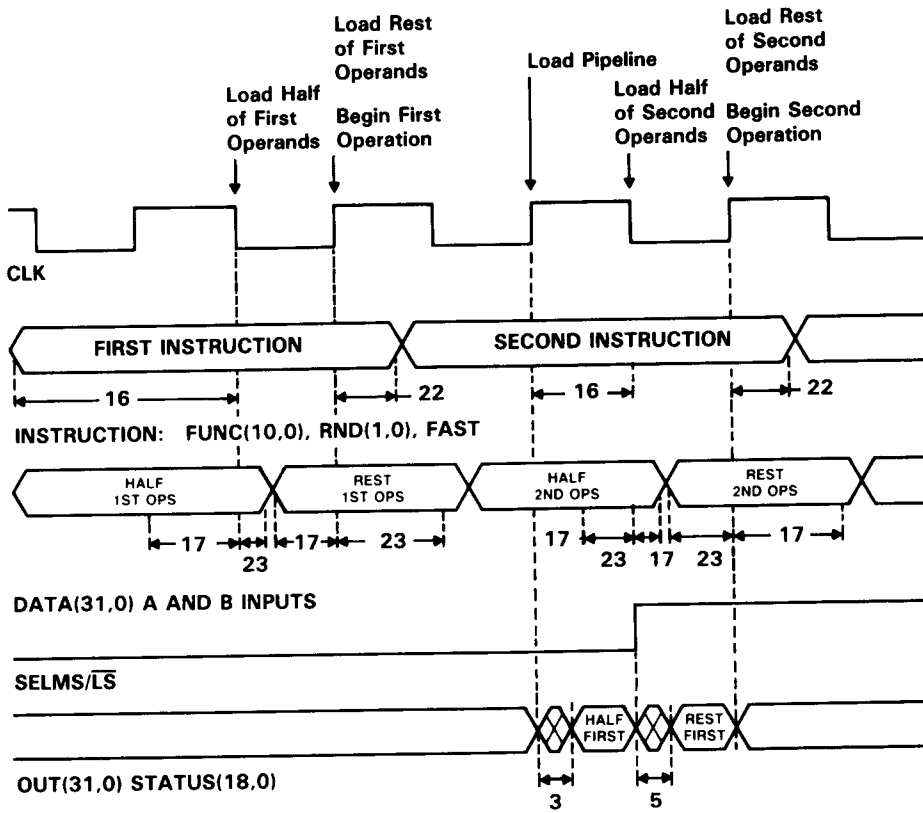
NOTE: Assume PIPES2-0 = 000, CONFIG1-0 = 01, ENRA = 1, ENRB = 1, SELMS/$\overline{LS}$ = X, $\overline{OEY}$ = 0, $\overline{OEC}$ = $\overline{OES}$ = 0, $\overline{RESET}$ = $\overline{HALT}$ = 1, TP1-0 = 11

Figure 67. Single-Precision Chained Mode Operation, All Registers Enabled (PIPES2-PIPES0 = 000, CLKMODE = X)



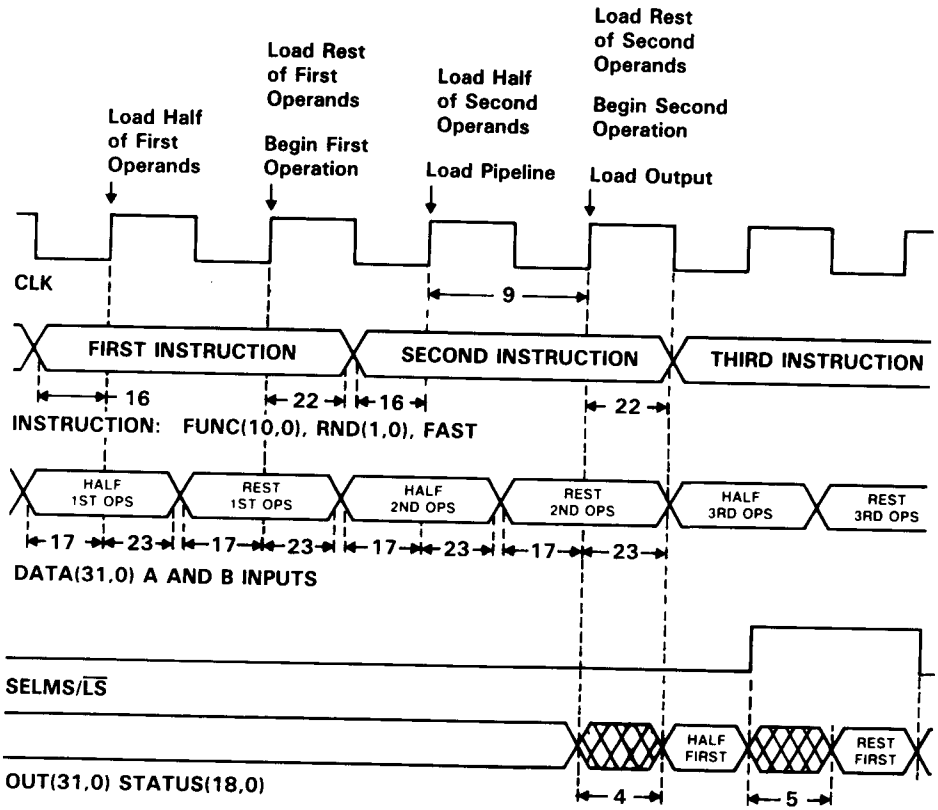
NOTE: Assume $\overline{PIPES2}$-0 = 111, CONFIG1-0 = 11, ENRA = 1, ENRB = 1, $\overline{OEY}$ = 0, $\overline{OEC}$ = $\overline{OES}$ = 0, $\overline{RESET}$ = $\overline{HALT}$ = 1, TP1-0 = 11

Figure 68. Double-Precision Chained Mode Operation, All Registers Disabled (PIPES2-PIPES0 = 111, CLKMODE = 0)



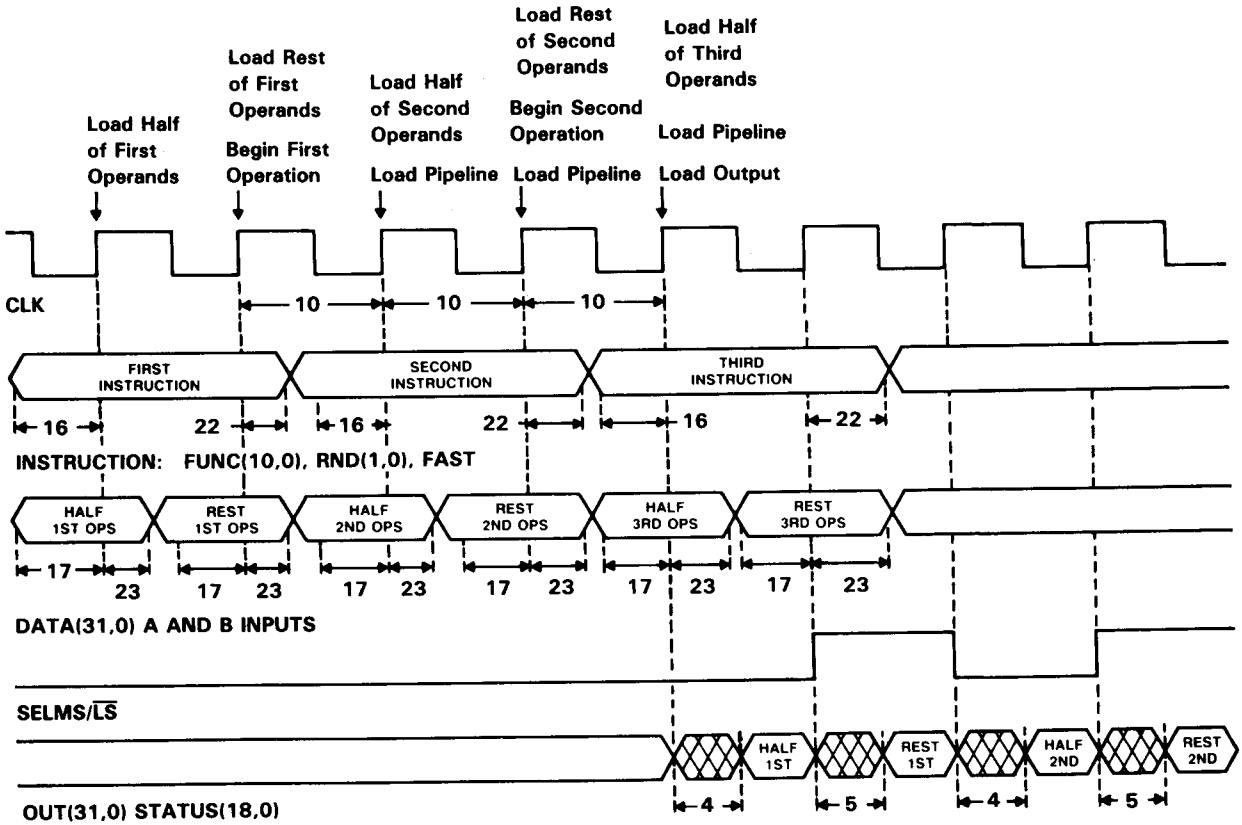
NOTE: Assume PIPES2-0 = 110, CONFIG1-0 = 11, ENRA = 1, $\overline{OEY}$ = 0, $\overline{OEC}$ = $\overline{OES}$ = 0, $\overline{RESET}$ = $\overline{HALT}$ = 1, TP1-0 = 11

Figure 69. Double-Precision Chained Mode Operation, Input Registers Enabled (PIPES2-PIPES0 = 110, CLKMODE = 1)



NOTE: Assume PIPES2-0 = 010, CONFIG1-0 = 10, ENRA = 1, ENRB = 1, $\overline{OEY}$ = 0, $\overline{OEC}$ = $\overline{OES}$ = 0, $\overline{RESET}$ = $\overline{HALT}$ = 1, TP1-0 = 11

Figure 70. Double-Precision Chained Mode Operation, Input and Output Registers Enabled (PIPES2-PIPES0 = 010, CLKMODE = 0)



NOTE: Assume PIPES2-0 = 000, CONFIG1-0 = 01, ENRA = 1, ENRB = 1, $\overline{OEY}$ = 0, $\overline{OEC}$ = $\overline{OES}$ = 0, $\overline{RESET}$ = $\overline{HALT}$ = 1, TP1-0 = 11

Figure 71. Double-Precision Chained Mode Operation, All Registers Enabled (PIPES2-PIPES0 = 000, CLKMODE = 0)

Instruction Timing

The following table details the number of clock cycles required to complete an operation in different pipelined modes. For more detail, see the sample microinstructions shown in the previous section.

Clock duration and output delay depend on the pipeline mode selected. See the note in the table and timing parameters listed at the beginning of this document.

Table 31. Number of Clocks Required to Complete an Operation

OPERATION	PIPES2-0 = 000 (t_{pd4})	PIPES2-0 = 100 (t_{pd3})	PIPES2-0 = 110 (t_{pd2})	PIPES2-0 = 111 (t_{pd1})	PIPES2-0 =010 (t_{pd4})
Single-Precision Floating Point					
ALU Operation or Multiply‡	3	2	1	0	2
Divide	8	7	7	X	8
Square Root	11	10	10	X	11
Double-Precision Floating Point					
ALU Operation†	4	3	2	1	3
Multiply‡	5	4	3	2	4
Divide	14	13	13	X	14
Square Root	17	16	16	X	17
Integer					
ALU Operation or Multiply‡	3	2	1	0	2
Divide	16	15	15	X	16
Square Root	20	19	19	X	20

Y output and status valid following this t_{pd} delay after the designated number of clocks
†Includes every conversion involving double-precision (DP ⟷ SP or DP ⟷ Integer)
‡Includes all chained mode operations
X = invalid

When using fast cycle times and double-precision operations, two cycles may be required to output and capture both halves of a double-precision result. To insure the result remains valid for two cycles, a NOP instruction may need to be inserted between the operations. Table 32 shows the number of NOPs necessary to insert into the instruction stream for fully pipelined operation (PIPES2-PIPES0 = 000).

Table 32. NOPs Inserted to Guarantee That Double-Precision Results Remain Valid for Two Clock Cycles (PIPES2-PIPES0 = 000)

1ST OPERATION	FOLLOWED BY 2ND OPERATION	# NOPs INSERTED BETWEEN OPERATIONS	# CYCLES RESULT IS VALID
DP → 32 BIT	DP → 32 BIT	0	2
	32 BIT → DP	0	2
	32 BIT OP	0	1
	DP ALU	0	2
	DP Multiply	0	2
	DP Sqrt	0	2
	DP Divide	0	2
32 BIT → DP	DP → 32 BIT	0	2
	32 BIT → DP	0	2
	32 BIT OP	1	2
	DP ALU	0	2
	DP Multiply	0	2
	DP Sqrt	0	2
	DP Divide	0	2
32 BIT OP	DP → 32 BIT	0	2
	32 BIT → DP	0	2
	32 BIT OP	0	1
	DP ALU	0	2
	DP Multiply	0	2
	DP Sqrt	0	2
	DP Divide	0	2
DP ALU	DP → 32 BIT	0	2
	32 BIT → DP	0	2
	32 BIT OP	1	2
	DP ALU	0	2
	DP Multiply	0	2
	DP Sqrt	0	2
	DP Divide	0	2
DP Multiply	DP → 32 BIT	1	2
	32 BIT → DP	1	2
	32 BIT OP	2†	2
	DP ALU	1	2
	DP Multiply	0	2
	DP Sqrt	1	2
	DP Divide	1	2

NOTE: 32-bit operation refers to a single-precision floating point or integer ALU operation or multiply, except conversion to or from double-precision. This assumes the instruction following a double-precision divide may begin loading on the 12th clock cycle, following a double-precision square root on the 15th cycle.

†The device will not load a single-precision operation on the first clock edge following this operation, so any single-precision instruction may be used. A NOP is recommended. The second instruction must be a NOP.

Table 32. NOPs Inserted to Guarantee That Double-Precision Results Remain Valid for Two Clock Cycles (PIPES2-PIPES0 = 000) (Continued)

1ST OPERATION	FOLLOWED BY 2ND OPERATION	# NOPs INSERTED BETWEEN OPERATIONS	# CYCLES RESULT IS VALID
DP SQRT	DP → 32 BIT	1	2
	32 BIT → DP	1	2
	32 BIT OP	2†	2
	DP ALU	1	2
	DP Multiply	0	2
	DP Sqrt	0	2
	DP Divide	0	2
DP Divide	DP → 32 BIT	1	2
	32 BIT → DP	1	2
	32 BIT OP	2†	2
	DP ALU	1	2
	DP Multiply	0	2
	DP Sqrt	0	2
	DP Divide	0	2

NOTE: 32-bit operation refers to a single-precision floating point or integer ALU operation or multiply, except conversion to or from double-precision. This assumes the instruction following a double-precision divide may begin loading on the 12th clock cycle, following a double-precision square root on the 15th cycle.

†The device will not load a single-precision operation on the first clock edge following this operation, so any single-precision instruction may be used. A NOP is recommended. The second instruction must be a NOP.

Exception and Status Handling

Exception and status flags for the 'ACT8847 were listed previously in Tables 14 and 15.

Output exception signals are provided to indicate both the source and type of the exception. DENORM, INEX, OVER, UNDER, and RNDCO indicate the exception type, and CHEX and SRCEX indicate the source of an exception. SRCEX indicates the source of a result as selected by instruction bit I6, and SRCEX is active whenever a result is output, not only when an exception is being signalled. The chained-mode exception signal CHEX indicates that an exception has be generated by the source not selected for output by I6. The exception type signalled by CHEX cannot be read unless status select controls SELST1-SELST0 are used to force status output from the deselected source.

Output exceptions may be due either to a result in an illegal format or to a procedural error. Results too large or too small to be represented in the selected precision are signalled by OVER and UNDER. When INF is high, the output is the IEEE representation of infinity. Any ALU output which has been increased in magnitude by rounding causes INEX to be set high. DENORM is set when the multiplier output is wrapped or the ALU output is denormalized. DENORM is also set high when an illegal operation on an integer is performed. Wrapped outputs from the multiplier may be inexact or increased in magnitude by rounding, which may cause the INEX and RNDCO status signals to be set high. A denormal output from the ALU (DENORM = 1) may also cause INEX to be set, in which case UNDER is also signalled.

Ordinarily, SELST1-SELST0 are set high so that status selection defaults to the output source selected by instruction input I6. The ALU is selected as the output source when I6 is low, and the multiplier when I6 is high.

When the device operates in chained mode, it may be necessary to read the status results not associated with the output source. As shown in Table 16, SELST1-SELST0 can be used to read the status of either the ALU or the multiplier regardless of the I6 setting.

Status results are registered only when the output (P and S) registers are enabled (PIPES2 = 0). Otherwise, the status register is transparent. In either case, to read the status outputs, the output enables ($\overline{OES}$, $\overline{OEC}$, or both) must be low.

Status flags are provided to signal both floating point and integer results. Integer status is provided using AEQB for zero, NEG for sign, and OVER for overflow/carryout.

Several status exceptions are generated by illegal data or instruction inputs to the FPU. Input exceptions may cause the following signals to be set high: IVAL, DIVBY0, DENIN, and STEX1-STEX0. If the IVAL flag is set, either an invalid operation such as the square root of $-|X|$, has been requested or a NaN (Not a Number) has been input. When DENIN is set, a denormalized number has been input to the multiplier. DIVBY0 is set when the divisor is zero. STEX1-STEX0 indicate which port (RA, RB, or both) is the source of the exception when either a denormal is input to the multiplier (DENIN = 1) or a NaN (IVAL = 1) is input to the multiplier or the ALU.

NaN inputs are all treated as IEEE signalling NaNs, causing the IVAL flag to be set. When output from the FPU, the fraction field from a NaN is set high (all 1s) and the sign bit is 0, regardless of the original fraction and sign fields of the input NaN.

When the 'ACT8847 outputs a NaN, it is always in the form of a signalling NaN along with the IVAL (Invalid) and appropriate STEX flag set high (except for the MOVE A instruction which passes any operand as is without setting exception flags).

Certain operations involving floating point zeros and infinities are invalid, causing the 'ACT8847 to set the IVAL flag and output a NaN. Operations involving zero and infinity are detailed below.

A floating point zero is represented by an all zero exponent and fraction field. The sign bit may be 0 or 1, to represent $+0$ OR -0 respectively.

Zero divided by zero is an invalid operation. The result is a NaN with the IVAL and DIVBY0 flags set. Any other number divided by zero results in the appropriately signed infinity with the DIVBY0 flag set.

For operations with floating point zeros: ±0 multiplied by any number is the appropriately signed 0.

$$+0 + (-0) = +0$$
$$+0 + (+0) = +0$$
$$-0 + (-0) = -0$$
$$-0 + (+0) = +0$$
$$+0 - (-0) = +0$$
$$+0 - (+0) = +0$$
$$-0 - (-0) = +0$$
$$-0 - (+0) = -0$$

Floating point infinity is represented by an all 1 exponent field with an all 0 fraction field. The sign bit determines positive or negative infinity (0 or 1 respectively).

Infinity divided by infinity is an invalid operation, setting the IVAL flag and resulting in a NaN output. Division of infinity by any other number results in the appropriately signed infinity. Division of any number (except infinity or zero) by infinity results in an appropriately signed zero. Infinity divided by zero results in the appropriately signed infinity with the DIVBY0 flag set.

For invalid operations with infinity listed below, the output is a signalling NaN with the IVAL flag set.

±infinity multiplied by ±0
±infinity divided by ±0
+infinity + (−infinity)
−infinity + (+infinity)
+infinity − (+infinity)
−infinity − (−infinity)

Any other number added to or multiplied by infinity results in the appropriately signed infinity as output.

Instruction Inputs

Operations are summarized in Tables 33 thru 41.

Table 33. Independent ALU Operations, Single Floating Point Operand

ALU OPERATION ON A OPERAND	INSTRUCTION INPUTS I10-I0	NOTES
Pass A operand	00x x01x 0000	
Pass −A operand	00x x01x 0001	
Convert from 2's complement integer to floating point†	00x x010 0010	
Convert from floating point to 2's complement integer†	00x x01x 0011	x = Don't care I8 selects precision of A operand
Move A operand (pass without NaN detect or status flags active)	00x x01x 0100	0 = A (SP) 1 = A (DP)
Pass B operand	00x x01x 0101	I7 selects precision of B operand and must equal I8.
Convert from floating point to floating point (adjusts precision of input: SP → DP, DP → SP)‡	00x x01x 0110	I4 selects absolute value of a operand: 0 = A 1 = |A|
Floating point to unsigned integer conversion†	00x x01x 0111	During integer to floating point conversion, |A| is not allowed as a result.
Wrap denormal operand	00x x01x 1000	
Unsigned integer to floating point conversion†	00x x01x 1010	
Unwrap exact number	00x x01x 1100	
Unwrap inexact number	00x x01x 1101	
Unwrap rounded input	00x x01x 1110	

†During this operation, I8 selects the precision of the result. If the conversion involves double-precision, the operation requires 2 cycles to load.
‡Requires 2 cycles to load the operation, even if input is SP.

Table 34. Independent ALU Operations, Two Floating Point Operands

ALU OPERATIONS AND OPERANDS	INSTRUCTION INPUTS I10-I0	NOTES
Add A + B	00x x000 0x00	
Add |A| + B	00x x001 0x00	
Add A + |B|	00x x000 1x00	x = Don't Care
Add |A| + |B|	00x x001 1x00	I8 selects precision of A operand:
Subtract A − B	00x x000 0x01	0 = A (SP)
Subtract |A| − B	00x x001 0x01	1 = A (DP)
Subtract A − |B|	00x x000 1x01	I7 selects precision of B operand:
Subtract |A| − |B|	00x x001 1x01	0 = B (SP)
Compare A, B	00x x000 0x10	1 = B (DP)
Compare |A| , B	00x x001 0x10	I2 selects either Y or its absolute value:
Compare A, |B|	00x x000 1x10	0 = Y
Compare |A|, |B|	00x x001 1x10	1 = |Y|
Subtract B − A	00x x000 0x11	
Subtract B − |A|	00x x001 0x11	
Subtract |B| − A	00x x000 1x11	
Subtract |B| − |A|	00x x001 1x11	

Table 35. Independent ALU Operations, One Integer Operand

ALU OPERATION ON A OPERAND	INSTRUCTION INPUTS I10-I0	NOTES
Pass A operand	010 xx10 0000	x = Don't Care
Pass −A operand (2's complement)‡	010 xx10 0001	I7 selects format of A or B integer operand:
Negate A operand (1's complement)	010 xx10 0010	0 = Single-precision 2's complement
Pass B operand	010 xx10 0101	1 = Single-precision unsigned integer
Shift left logical†	010 xx10 1000	I8 must equal I7
Shift right logical†	010 xx10 1001	
Shift right arithmetic†	010 xx10 1101	

†B operand is number of bit positions A is to be shifted and must be input on the same cycle as the instruction.
‡Pass (−A) of unsigned integer takes 1's complement.

Table 36. Independent ALU Operations, Two Integer Operands

ALU OPERATIONS AND OPERANDS	INSTRUCTION INPUTS I10-I0	NOTES
Add A + B	010 x000 0000	x = Don't Care I7 selects format of A and B operands: 0 = Single-precision 2's complement 1 = Single-precision unsigned integer
Subtract A − B	010 x000 0001	
Compare A, B	010 x000 0010	
Subtract B − A	010 x000 0011	
Logical AND A, B	010 x000 1000	
Logical AND A, NOT B	010 x000 1001	
Logical AND NOT A, B	010 x000 1010	
Logical OR A, B	010 x000 1100	
Logical XOR A, B	010 x000 1101	

Table 37. Independent Floating Point Multiply Operations

MULTIPLIER OPERATION AND OPERANDS	INSTRUCTION INPUTS I10-I0	NOTES
Multiply A * B	00x x100 00xx	x = Don't Care I8 selects A operand precision (0 = SP, 1 = DP) I7 selects B operand precision (0 = SP, 1 = DP) I1 selects A operand format (0 = Normal, 1 = Wrapped) I0 selects B operand format (0 = Normal, 1 = Wrapped)
Multiply −(A * B)	00x x100 01xx	
Multiply A * |B|	00x x100 10xx	
Multiply −(A * |B|)	00x x100 11xx	
Multiply |A| * B	00x x101 00xx	
Multiply −(|A| * B)	00x x101 01xx	
Multiply |A| * |B|	00x x101 10xx	
Multiply −(|A| * |B|)	00x x101 11xx	

Table 38. Independent Floating Point Divide/Square Root Operations

MULTIPLIER OPERATION AND OPERANDS†	INSTRUCTION INPUTS I10-I0	NOTES
Divide A / B	00x x110 0xxx	x = Don't Care I8 selects A operand precision and I7 selects B operand precision (0 = SP, 1 = DP) I2 negates multiplier result (0 = Normal, 1 = Negated) I1 selects A operand format and I0 selects B operand format (0 = Normal, 1 = Wrapped)
SQRT A	00x x110 1xxx	
Divide |A| / B	00x x111 0xxx	
SQRT |A|	00x x111 1xxx	

†I7 should be equal to I8 for square root operations

Table 39. Independent Integer Multiply/Divide/Square Root Operations

MULTIPLIER OPERATION AND OPERANDS†	INSTRUCTION INPUTS I10-I0	NOTES
Multiply A * B Divide A / B SQRT A	010 x100 0000 010 x110 0000 010 x110 1000	x = Don't care I7 selects operand format: 0 = SP 2's complement 1 = SP unsigned integer

†Operations involving absolute values, wrapped operands, or negated results are valid only when floating point format is selected (I9 = 0).

Table 40. Chained Multiplier/ALU Floating Point Operations‡

CHAINED OPERATIONS		OUTPUT	INSTRUCTION	NOTES
MULTIPLIER	ALU	SOURCE	INPUTS I10-I0	
A * B	A + B	ALU	10x x000 xx00	
A * B	A + B	Multiplier	10x x100 xx00	
A * B	A − B	ALU	10x x000 xx01	
A * B	A − B	Multiplier	10x x100 xx01	
A * B	2 − A	ALU	10x x000 xx10	x = Don't Care
A * B	2 − A	Multiplier	10x x100 xx10	I8 selects precision of RA inputs:
A * B	B − A	ALU	10x x000 xx11	0 = RA (SP)
A * B	B − A	Multiplier	10x x100 xx11	1 = RA (DP)
A * B	A + 0	ALU	10x x010 xx00	I7 selects precision of RB inputs:
A * B	A + 0	Multiplier	10x x110 xx00	0 = RB (SP)
A * B	0 − A	ALU	10x x010 xx11	1 = RB (DP)
A * B	0 − A	Multiplier	10x x110 xx11	I3 negates ALU result:
A * 1	A + B	ALU	10x x001 xx00	0 = Normal
A * 1	A + B	Multiplier	10x x101 xx00	1 = Negated
A * 1	A − B	ALU	10x x001 xx01	I2 negates multiplier result:
A * 1	A − B	Multiplier	10x x101 xx01	0 = Normal
A * 1	2 − A	ALU	10x x001 xx10	1 = Negated
A * 1	2 − A	Multiplier	10x x101 xx10	
A * 1	B − A	ALU	10x x001 xx11	
A * 1	B − A	Multiplier	10x x101 xx11	
A * 1	A + 0	ALU	10x x011 xx00	
A * 1	A + 0	Multiplier	10x x111 xx00	
A * 1	0 − A	ALU	10x x011 xx11	
A * 1	0 − A	Multiplier	10x x111 xx11	

‡The I10-I0 setting 1xx xx1x xx10 is invalid, since it attempts to force the B operand of the ALU to both 0 and 2 simultaneously.

Table 41. Chained Multiplier/ALU Integer Operations

CHAINED OPERATIONS		OUTPUT	INSTRUCTION	NOTES
MULTIPLIER	ALU	SOURCE	INPUTS I10-I0	
A * B	A + B	ALU	110 x000 0000	
A * B	A + B	Multiplier	110 x100 0000	
A * B	A − B	ALU	110 x000 0001	
A * B	A − B	Multiplier	110 x100 0001	
A * B	2 − A	ALU	110 x000 0010	
A * B	2 − A	Multiplier	110 x100 0010	
A * B	B − A	ALU	110 x000 0011	
A * B	B − A	Multiplier	110 x100 0011	x = Don't Care I7 selects format of A and B operands: 0 = SP 2's complement 1 = SP unsigned integer
A * B	A + 0	ALU	110 x010 0000	
A * B	A + 0	Multiplier	110 x110 0000	
A * B	0 − A	ALU	110 x010 0011	
A * B	0 − A	Multiplier	110 x110 0011	
A * 1	A + B	ALU	110 x001 0000	
A * 1	A + B	Multiplier	110 x101 0000	
A * 1	A − B	ALU	110 x001 0001	
A * 1	A − B	Multiplier	110 x101 0001	
A * 1	2 − A	ALU	110 x001 0010	
A * 1	2 − A	Multiplier	110 x101 0010	
A * 1	B − A	ALU	110 x001 0011	
A * 1	B − A	Multiplier	110 x101 0011	
A * 1	A + 0	ALU	110 x011 0000	
A * 1	A + 0	Multiplier	110 x111 0000	
A * 1	0 − A	ALU	110 x011 0011	
A * 1	0 − A	Multiplier	110 x111 xx11	

Input Configuration

CONFIG1-CONFIG0 control the order in which double-precision operands are loaded, as shown in the Table 42.

Table 42. Double-Precision Input Data Configuration Modes

		LOADING SEQUENCE			
		DATA LOADED INTO TEMP REGISTER ON FIRST CLOCK AND RA/RB REGISTERS ON SECOND CLOCK†		DATA LOADED INTO RA/RB REGISTERS ON SECOND CLOCK	
CONFIG1	CONFIG0	DA	DB	DA	DB
0	0	B operand (MSH)	B operand (LSH)	A operand (MSH)	A operand (LSH)
0	1‡	A operand (LSH)	B operand (LSH)	A operand (MSH)	B operand (MSH)
1	0	A operand (MSH)	B operand (MSH)	A operand (LSH)	B operand (LSH)
1	1	A operand (MSH)	A operand (LSH)	B operand (MSH)	B operand (LSH))

†On the first active clock edge (see CLKMODE), data in this column is loaded into the temporary register. On the next rising edge, operands in the temporary register and the DA/DB buses are loaded into the RA and RB registers.
‡Use CONFIG1-0 = 01 as normal single-precision input configuration.

Operand Source Select

Multiplier and ALU operands are selected by SELOP7-SELOP0 as shown in Tables 43 and 44.

Table 43. Multiplier Input Selection

A1 (MUX1) INPUT			B1 (MUX2) INPUT		
SELOP7	SELOP6	OPERAND SOURCE†	SELOP5	SELOP4	OPERAND SOURCE†
0	0	Reserved	0	0	Reserved
0	1	C register	0	1	C register
1	0	ALU feedback	1	0	Multiplier feedback
1	1	RA input register	1	1	RB input register

† For division or square root operations, only RA and RB registers can be selected as sources.

Table 44. ALU Input Selection

A2 (MUX3) INPUT			B2 (MUX4) INPUT		
SELOP3	SELOP2	OPERAND SOURCE†	SELOP1	SELOP0	OPERAND SOURCE†
0	0	Reserved	0	0	Reserved
0	1	C register	0	1	C register
1	0	Multiplier feedback	1	0	ALU feedback
1	1	RA input register	1	1	RB input register

†For division or square root operations, only RA and RB registers can be selected as sources.

Pipeline Control

Pipelining levels are turned on by PIPES2-PIPES0 as shown below.

Table 45. Pipeline Controls (PIPES2-PIPES0)

PIPES2-PIPES0	REGISTER OPERATION SELECTED
X X 0	Enables input registers (RA, RB)
X X 1	Makes input registers (RA, RB) transparent
X 0 X	Enables pipeline registers
X 1 X	Makes pipeline registers transparent
0 X X	Enables output registers (PREG, SREG, Status)
1 X X	Makes output registers (PREG, SREG, Status) transparent

Round Control

RND1-RND0 select the rounding mode as shown in Table 46.

Table 46. Rounding Modes

RND1-RND0	ROUNDING MODE SELECTED
0 0	Round towards nearest
0 1	Round towards zero (truncate)
1 0	Round towards infinity (round up)
1 1	Round towards negative infinity (round down)

Status Output Selection

SELST1-SELST0 choose the status output as shown below.

Table 47. Status Output Selection (Chained Mode)

SELST1-SELST0	STATUS SELECTED
00	Logical OR of ALU and multiplier exceptions (bit by bit)
01	Selects multiplier status
10	Selects ALU status
11	Normal operation (selection based on result source specified by I6 input)

Test Pin Control

Testing is controlled by TP1-TP0 as shown below.

Table 48. Test Pin Control Inputs

TP1-TP0	OPERATION
0 0	All outputs and I/Os are forced low
0 1	All outputs and I/Os are forced high
1 0	All outputs are placed in a high impedance state
1 1	Normal operation

Miscellaneous Control Inputs

The remaining control inputs are shown in the Table 49.

Table 49. Miscellaneous Control Inputs

SIGNAL	HIGH	LOW
BYTEP	Selects byte parity generation and test	Selects single bit parity generation and test
CLKMODE	Enables temporary input register load on falling clock edge	Enables temporary input register load on rising clock edge
$\overline{\text{ENRC}}$	No effect	Enables C register load when CLKC goes high.
ENRA	If register is not in flowthrough, enables clocking of RA register	If register is not in flowthrough, through, holds contents of RA register
ENRB	If register is not in flowthrough, enables enables clocking of RB register	If register is not in flowthrough, holds contents of RB register
FAST	Places device in FAST mode	Places device in IEEE mode
FLOW_C	Causes output value to bypass C register and appear on C register output bus.	No effect
$\overline{\text{HALT}}$	No effect	Stalls device operation but does not affect registers, internal states, or status
$\overline{\text{OEC}}$	Disables compare pins	Enables compare pins
$\overline{\text{OES}}$	Disables status outputs	Enables status outputs
$\overline{\text{OEY}}$	Disables Y bus	Enables Y bus
$\overline{\text{RESET}}$	No effect	Clears internal states, status, internal pipeline registers, and exception disable register. Does not affect other data registers.
SELMS/$\overline{\text{LS}}$	Selects MSH of 64-bit result for output output on the Y bus (no effect on single-precision operands)	Selects LSH of 64-bit result for output on the Y bus (no effect on single-precision operands)
SRCC	Selects multiplier result for input to C register	Selects ALU result for input to C register

Glossary

Biased exponent — The true exponent of a floating point number plus a constant called the exponent field's excess. In IEEE data format, the excess or bias is 127 for single-precision numbers and 1023 for double-precision numbers.

Denormalized number (denorm) — A number with an exponent equal to zero and a nonzero fraction field, with the implicit leading (leftmost) bit of the fraction field being 0.

NaN (not a number) — Data that has no mathematical value. The 'ACT8847 produces a NaN whenever an invalid operation such as 0 * ∞ is executed. The output format for an NaN is an exponent field of all ones, a fraction field of all ones, and a zero sign bit. Any number with an exponent of all ones and a nonzero fraction is treated as a NaN on the input.

Normalized number — A number in which the exponent field is between 1 and 254 (single precision) or 1 and 2046 (double precision). The implicit leading bit is 1.

Wrapped number — A number created by normalizing a denormalized number's fraction field and subtracting from the exponent the number of shift positions required to do so. The exponent is encoded as a two's complement negative number.

SN74ACT8847 Application Notes

Sum of Products and Product of Sums

Performing fully pipelined double-precision operations requires a detailed understanding of timing constraints imposed by the multiplier. In particular, sum of products and product of sums operations can be executed very quickly, mostly in chained mode, assuming that timing relationships between the ALU and the multiplier are coded properly.

Pseudocode tables for these sequences are provided, (Table 38 and Table 39) showing how data and instructions are input in relation to the system clock. The overall patterns of calculations for an extended sum of products and an extended product of sums are presented. These examples assume FPU operation in CLKMODE 0, with the CONFIG setting 10 to load operands by MSH and LSH, all registers enabled (PIPES2 - PIPES0 = 000), and the C register clock tied to the system clock.

In the sum of products timing table, the two initial products are generated in independent multiplier mode. Several timing relationships should be noted in the table. The first chained instruction loads and begins to execute following the sixth rising edge of the clock, after the first product P1 has already been held in the P register for one clock. For this reason, P1 is loaded into the C register so that P1 will be stable for two clocks.

On the seventh clock, the ALU pipeline register loads with an unwanted sum, P1 + P1. However, because the ALU timing is constrained by the multiplier, the S register will not load until the rising edge of CLK9, when the ALU pipe contains the desired sum, P1 + P2. The remaining sequence of chained operations then execute in the desired manner.

Table 50. Pseudocode for Fully Pipelined Double-Precision Sum of Products†
(CLKMODE = 0, CONFIG1-CONFIG0 = 10, PIPES2-PIPES0 = 000)

CLK	DA BUS	DB BUS	TEMP REG	INS BUS	INS REG	RA REG	RB REG	MUL PIPE	P REG	C REG	ALU PIPE	S REG	Y BUS
1	A1 MSH	B1 MSH	A1,B1MSH	A1 * B1									
2	A1 LSH	B1 LSH	A1,B1LSH	A1 * B1	A1 * B1	A1	B1						
3	A2 MSH	B2 MSH	A2,B2MSH	A2 * B2	A1 * B1	A1	B1	A1 * B1					
4	A2 LSH	B2 LSH	A2,B2LSH	A2 * B2	A2 * B2	A2	B2	A1 * B1					
5	A3 MSH	B3 MSH	A3,B3MSH	PR + CR A3 * B3	A2 * B2	A2	B2	A2 * B2	P1				
6	A3 LSH	B3 LSH	A3,B3LSH	PR + CR A3 * B3	PR + CR, A3 * B3	A3	B3	A2 * B2	P1	P1			
7	A4 MSH	B4 MSH	A4,B4MSH	PR + SR A4 * B4	PR + SR, A3 * B3	A3	B3	A3 * B3	P2	P1	P1 + P1		
8	A4 LSH	B4 LSH	A4,B4LSH	PR + SR A4 * B4	PR + SR, A4 * B4	A4	B4	A3 * B3	P2	P1	P1 + P2		
9	A5 MSH	B5 MSH	A5,B5MSH	PR + SR A5 * B5	PR + SR, A4 * B4	A4	B4	A4 * B4	P3	P2	S1 + P2	S1	
10	A5 LSH	B5 LSH	A5,B5LSH	PR + SR A5 * B5	PR + SR, A5 * B5	A5	B5	A4 * B4	P3	P2	S1 + P3	S1	
11	A6 MSH	B6 MSH	A6,B6MSH	PR + SR A6 * B6	PR + SR, A5 * B5	A5	B5	A5 * B5	P4	P2	XXXXX	S2	
12									P4	P2		S2	

†PR = Product Register
SR = Sum Register
CR = Constant (C) Register

Table 51. Pseudocode for Fully Pipelined Double-Precision Product of Sums†
(CLKMODE = 0, CONFIG1-CONFIG0 = 10, PIPES2-PIPES0 = 000)

CLK	DA BUS	DB BUS	TEMP REG	INS BUS	INS REG	RA REG	RB REG	MUL PIPE	P REG	C REG	ALU PIPE	S REG	Y BUS
1	A1MSH	B1MSH	A1,B1MSH	A1 + B1									
2	A1LSH	B1LSH	A1,B1LSH	A1 + B1	A1 + B1	A1	B1						
3	A2MSH	B2MSH	A2,B2MSH	A2 + B2	A1 + B1	A1	B1				A1 + B1		
4	A2LSH	B2LSH	A2,B2LSH	A2 + B2	A2 + B2	A2	B2				A1 + B1	S1	
5	A3MSH	B3MSH	A3,B3MSH	CR * SR A3 + B3	A2 + B2	A2	B2			$\overline{ENRC} = 0$ S1	A2 + B2	S1	
6	A3LSH	B3LSH	A3,B3LSH	CR * SR A3 + B3	CR * SR A3 + B3	A3	B3			S1	A2 + B2	S2	
7	XXX	XXX	XXX	NOP	CR * SR A3 + B3	A3	B3	S1 * S2		S1	A3 + B3	S2	
8	A4MSH	B4MSH	A4,B4MSH	PR * SR A4 + B4	NOP	ENRA = 0 A3	ENRB = 0 B3	S1 * S2		S1		XXX	
9	A4LSH	B4LSH	A4,B4LSH	PR * SR A4 + B4	PR * SR A4 + B4	A4	B4	XXX	P1	S1	XXX	S3	
10	XXX	XXX	XXX	NOP	PR * SR A4 + B4	A4	B4	P1 * S3	P1	S1	A4 + B4	S3	
11	A5MSH	B5MSH	A5,B5MSH	PR * SR A5 + B5	NOP	ENRA = 0 A4	ENRB = 0 B4	P1 * S3	XXX	S1	A4 + B4	XXX	
12	A5LSH	B5LSH	A5,B5LSH	PR * SR A5 + B5	PR * SR A5 + B5	A5	B5	XXX	P2	S1	X	S4	

NOTE: NOP instruction is 011 0000 0000.
†PR = Product Register
SR = Sum Register
CR = Constant (C) Register

Matrix Operations

The 'ACT8847 floating point unit can also be used to perform matrix manipulations involved in graphics processing or digital signal processing. The FPU multiplies and adds data elements, executing sequences of microprogrammed calculations to form new matrices.

Representation of Variables

In state representations of control systems, an n-th order linear differential equation with constant coefficients can be represented as a sequence of n first-order linear differential equations expressed in terms of state variables:

$$\frac{dx1}{dt} = x2, \ldots, \quad \frac{dx(n-1)}{dt} = xn$$

For example, in vector-matrix form the equations of an nth-order system can be represented as follows:

$$\frac{d}{dt}\begin{bmatrix} x1 \\ x2 \\ \vdots \\ xn \end{bmatrix} = \begin{bmatrix} a11 & a12 & \cdots & a1n \\ \vdots & \vdots & & \vdots \\ an1 & an2 & \cdots & ann \end{bmatrix} \begin{bmatrix} x1 \\ x2 \\ \vdots \\ xn \end{bmatrix} + \begin{bmatrix} b11 & \cdots & b1n \\ \vdots & & \vdots \\ bn1 & \cdots & bnn \end{bmatrix} \begin{bmatrix} u1 \\ u2 \\ \vdots \\ un \end{bmatrix}$$

or, $\dot{X} = aX + bu$

Expanding the matrix equation for one state variable, dx_1/dt, results in the following expression:

$$\dot{X}1 = (a_{11} * x_1 + \ldots + a_{1n} * x_n) + (b_{11} * u_1 + \ldots + b_{1n} * u_n)$$

where $\dot{X}1 = dx_1/dt$.

Sequences of multiplications and additions are required when such state space transformations are performed, and the 'ACT8847 has been designed to support such sum-of-products operations. An n × n matrix A multiplied by an n × n matrix X yields an n × n matrix C whose elements cij are given by this equation:

$$c_{ij} = \sum_{k=1}^{n} a_{ik} * x_{kj} \quad \text{for } i=1, \ldots, n \quad j=1, \ldots, n \qquad (1)$$

For the c_{ij} elements to be calculated by the 'ACT8847, the corresponding elements a_{ik} and x_{kj} must be stored outside the 'ACT8847 and fed to the 'ACT8847 in the proper order required to effect a matrix multiplication such as the state space system representation just discussed.

Sample Matrix Transformation

The matrix manipulations commonly performed in graphics systems can be regarded as geometrical transformations of graphic objects. A matrix operation on another matrix representing a graphic object may result in scaling, rotating, transforming, distorting, or generating a perspective view of the image. By performing a matrix operation on the position vectors which define the vertices of an image surface, the shape and position of the surface can be manipulated.

The generalized 4 × 4 matrix for transforming a three-dimensional object with homogeneous coordinates is shown below:

$$T = \left[\begin{array}{cccc:c} a & b & c & : & d \\ e & f & g & : & h \\ i & j & k & : & l \\ \hdashline m & n & o & : & p \end{array}\right]$$

The matrix T can be partitioned into four component matrices, each of which produces a specific effect on the resultant image:

$$\left[\begin{array}{ccc:c} & & & : & \\ & & & : & 3 \\ 3 \times 3 & & & : & x \\ & & & : & 1 \\ \hdashline 1 \times 3 & & & : & 1 \times 1 \end{array}\right]$$

The 3 × 3 matrix produces linear transformation in the form of scaling, shearing and rotation. The 1 × 3 row matrix produces translation, while the 3 × 1 column matrix produces perspective transformation with multiple vanishing points. The final single element 1 × 1 produces overall scaling. Overall operation of the transformation matrix T on the position vectors of a graphic object produces a combination of shearing, rotation, reflection, translation, perspective, and overall scaling.

The rotation of an object about an arbitrary axis in a three-dimensional space can be carried out by first translating the object such that the desired axis of rotation passes through the origin of the coordinate system, then rotating the object about the axis

through the origin, and finally translating the rotated object such that the axis of rotation resumes its initial position. If the axis of rotation passes through the point P = [a b c 1], then the transformation matrix is representable in this form:

$$[x\ y\ z\ h] = [x\ y\ z\ 1] \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -a & -b & -c & 1 \end{bmatrix} \begin{bmatrix} R \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ a & b & c & 1 \end{bmatrix} \quad (2)$$

translation to origin — rotation about origin — translation back to initial position

where R may be expressed as:

$$R = \begin{bmatrix} n1^2 + (1-n)^2\cos\phi & n1n2(1-\cos\phi) + n3\sin\phi & n1n3(1-\cos\phi) - n2\sin\phi & 0 \\ n1n2(1-\cos\phi) - n3\sin\phi & n2^2 + (1-n2)^2\cos\phi & n2n3(1-\cos\phi) + n1\sin\phi & 0 \\ n1n3(1-\cos\phi) + n2\sin\phi & n2n3(1-\cos\phi) - n1\sin\phi & n3^2 + (1-n3)^2\cos\phi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

and $n1 = q1/(q1^2 + q2^2 + q3^2)^{1/2}$ = direction cosine for x-axis of rotation

$n2 = q2/(q1^2 + q2^2 + q3^2)^{1/2}$ = direction cosine for y-axis of rotation

$n3 = q3/(q1^2 + q2^2 + q3^2)^{1/2}$ = direction cosine for z-axis of rotation

$\overline{n} = (n1\ n2\ n3)$ = unit vector for $\overline{Q}$

$\overline{Q}$ = vector defining axis of rotation = [q1 q2 q3]

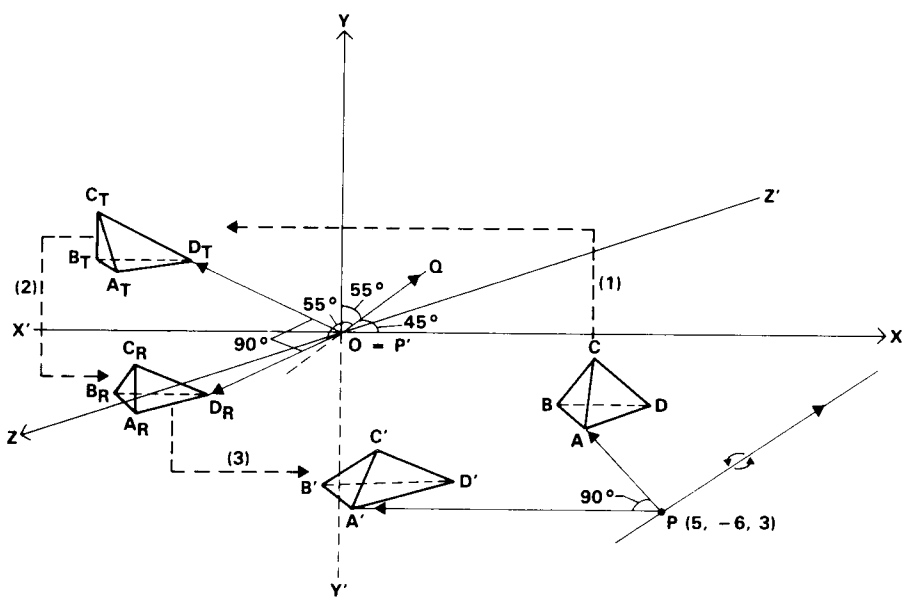
ϕ = the rotation angle about $\overline{Q}$

A general rotation using equation (2) is effected by determining the [x y z] coordinates of a point A to be rotated on the object, the direction cosines of the axis of rotation [n1, n2, n3], and the angle ϕ of rotation about the axis, all of which are needed to

define matrix [R]. Suppose, for example, that a tetrahedron ABCD, represented by the coordinate matrix below is to be rotated about an axis of rotation RX which passes through a point P = [5 −6 3 1] and whose direction cosines are given by unit vector [n1 = 0.866, n2 = 0.5, n3 = 0.707]. The angle of rotation 0 is 90 degrees (see Figure 72). The rotation matrix [R] becomes

$$\begin{bmatrix} 2 & -3 & 3 & 1 \\ 1 & -2 & 2 & 1 \\ 2 & -1 & 2 & 1 \\ 2 & -2 & 2 & 1 \end{bmatrix} \begin{matrix} A \\ B \\ C \\ D \end{matrix}$$

$$R = \begin{bmatrix} 0.750 & 1.140 & 0.112 & 0 \\ -0.274 & 0.250 & 1.220 & 0 \\ 1.112 & -0.513 & 0.500 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



(1) THIS ARROW DEPICTS THE FIRST TRANSLATION
(2) THIS AROW DEPICTS THE 90° ROTATION
(3) THIS ARROW DEPICTS THE BACK TRANSLATION

Figure 72. Sequence of Matrix Operations

The point transformation equation (2) can be expanded to include all the vertices of the tetrahedron as follows:

$$\begin{bmatrix} xa & ya & za & h1 \\ xb & yb & zb & h2 \\ xc & yc & zc & h3 \\ xd & yd & zd & h4 \end{bmatrix} = \begin{bmatrix} 2 & -3 & 3 & 1 \\ 1 & -2 & 2 & 1 \\ 2 & -1 & 2 & 1 \\ 2 & -2 & 2 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -5 & 6 & -3 & 1 \end{bmatrix} \begin{bmatrix} 0.750 & 1.140 & 0.112 & 0 \\ -0.274 & 0.250 & 1.22 & 0 \\ 1.112 & -0.513 & 0.500 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 5 & -6 & 3 & 1 \end{bmatrix} \quad (3)$$

translation to origin — rotation about origin — translation back to initial position

The 'ACT8847 floating point unit can perform matrix manipulation involving multiplications and additions such as those represented by equation (1). The matrix equation (3) can be solved by using the 'ACT8847 to compute, as a first step, the product matrix of the coordinate matrix and the first translation matrix of the right-hand side of equation (3) in that order. The second step involves postmultiplying the rotation matrix by the product matrix. The third step implements the back-translation by premultiplying the matrix result from the second step by the second translation matrix of equation (3). Details of the procedure to produce a three-dimensional rotation about an arbitrary axis are explained in the following steps:

Step 1

Translate the tetrahedron so that the axis of rotation passes through the origin. This process can be accomplished by multiplying the coordinate matrix by the translation matrix as follows:

$$\begin{bmatrix} 2 & -3 & 3 & 1 \\ 1 & -2 & 2 & 1 \\ 2 & -1 & 2 & 1 \\ 2 & -2 & 2 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -5 & 6 & -3 & 1 \end{bmatrix} = \begin{bmatrix} (2-5) & (-3+6) & (3-3) & 1 \\ (1-5) & (-2+6) & (2-3) & 1 \\ (2-5) & (-1+6) & (2-3) & 1 \\ (2-5) & (-2+6) & (2-3) & 1 \end{bmatrix}$$

translation to origin

vertices of translated tetrahedron

$$= \begin{bmatrix} -3 & +3 & 0 & 1 \\ -4 & +4 & -1 & 1 \\ -3 & +5 & -1 & 1 \\ -3 & +4 & -1 & 1 \end{bmatrix} \begin{matrix} \text{AT} \\ \text{BT} \\ \text{CT} \\ \text{DT} \end{matrix}$$

The 'ACT8847 could compute the translated coordinates AT, BT, CT, DT as indicated above. However, an alternative method resulting in a more compact solution is presented below.

Step 2

Rotate the tetrahedron about the axis of rotation which passes through the origin after the translation of Step 1. To implement the rotation of the tetrahedron, postmultiply the rotation matrix [R] by the translated coordinate matrix from Step 1. The resultant matrix represents the rotated coordinates of the tetrahedron about the origin as follows:

$$\begin{bmatrix} -3 & 3 & 0 & 1 \\ -4 & 4 & -1 & 1 \\ -3 & 5 & -1 & 1 \\ -3 & 4 & -1 & 1 \end{bmatrix} \begin{bmatrix} 0.750 & 1.140 & 0.112 & 0 \\ -0.274 & 0.250 & 1.22 & 0 \\ 1.112 & -0.513 & 0.500 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} -3.072 & -2.670 & 3.324 & 1 \\ -5.208 & -3.047 & 3.932 & 1 \\ -4.732 & -1.657 & 5.264 & 1 \\ -4.458 & -1.907 & 4.044 & 1 \end{bmatrix}$$

rotation about origin

rotated coordinates

Step 3

Translate the rotated tetrahedron back to the original coordinate space. This is done by premultiplying the resultant matrix of Step 2 by the translation matrix. The following calculations produces the final coordinate matrix of the transformed object:

$$\begin{bmatrix} -3.072 & -2.670 & 3.324 & 1 \\ -5.208 & -3.047 & 3.932 & 1 \\ -4.732 & -1.657 & 5.264 & 1 \\ -4.458 & -1.907 & 4.044 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 5 & -6 & 3 & 1 \end{bmatrix} = \begin{bmatrix} 1.928 & -8.670 & 6.324 & 1 \\ -0.208 & -9.047 & 6.932 & 1 \\ 0.268 & -7.657 & 8.264 & 1 \\ 0.542 & -7.907 & 7.044 & 1 \end{bmatrix}$$

translate back — final rotated coordinates

A more compact solution to these transformation matrices is a product matrix that combines the two translation matrices and the rotation matrix in the order shown in equation (3). Equation (3) will then take the following form:

$$\begin{bmatrix} xa & ya & za & h1 \\ xb & yb & zb & h2 \\ xc & yc & zc & h3 \\ xd & yd & zd & h4 \end{bmatrix} =$$

$$\begin{bmatrix} 2 & -3 & 3 & 1 \\ 1 & -2 & 2 & 1 \\ 2 & -1 & 2 & 1 \\ 2 & -2 & 2 & 1 \end{bmatrix} \begin{bmatrix} 0.750 & 1.140 & 0.112 & 0 \\ -0.274 & 0.250 & 1.220 & 0 \\ 1.112 & -0.513 & 0.500 & 0 \\ -3.730 & -8.661 & 8.260 & 1 \end{bmatrix}$$

transformation matrix

The newly transformed coordinates resulting from the postmultiplication of the transformation matrix by the coordinate matrix of the tetrahedron can be computed using equation (1) which was cited previously:

$$c_{ij} = \sum_{k=1}^{n} a_{ik} * x_{kj} \quad \text{for } i = 1, \ldots, n \quad j = 1, \ldots, n \tag{1}$$

For example, the coordinates may be computed as follows:

$$\begin{aligned} xa = c_{11} &= a_{11} * x_{11} + a_{12} * x_{21} + a_{13} * x_{31} + a_{14} * x_{41} \\ &= 2 * 0.750 + (-3) * (-0.274) + 3 * 1.112 + 1 * (-3.73) \\ &= 1.5 + 0.822 + 3.336 - 3.73 \\ &= 1.928 \end{aligned}$$

$$\begin{aligned} ya = c_{12} &= a_{11} * x_{12} + a_{12} * x_{22} + a_{13} * x_{32} + a_{14} * x_{42} \\ &= 2 * 1.140 + (-3) * 0.250 + 3 * (-0.513) + 1x(-8.661) \\ &= 2.28 - 0.75 - 1.539 - 8.661 \\ &= -8.67 \end{aligned}$$

$$\begin{aligned} za = c_{13} &= a_{11} * x_{13} + a_{12} * x_{23} + a_{13} * x_{33} + a_{14} * x_{43} \\ &= 2 * 0.112 + (-3) * 1.220 + 3 * 0.500 + 1 * 8.260 \\ &= 0.224 - 3.66 + 1.5 + 8.260 \\ &= 6.324 \end{aligned}$$

$$\begin{aligned} h1 = c_{14} &= a_{11} * x_{14} + a_{12} * x_{24} + a_{13} * x_{34} + a_{14} * x_{44} \\ &= 2 * 0 + (-3) * 0 + 3 * 0 + 1 * 1 \\ &= 0 + 0 + 0 + 1 \\ &= 1 \end{aligned}$$

$$A' = [1.928 \quad -8.67 \quad 6.324 \quad 1]$$

The other rotated vertices are computed in a similar manner:

$$B' = [-5.208 \quad -3.047 \quad 3.932 \quad 1]$$
$$C' = [-4.732 \quad -1.657 \quad 5.264 \quad 1)$$
$$D' = [-4.458 \quad -1.907 \quad 4.044 \quad 1]$$

Microinstructions for Sample Matrix Manipulation

The 'ACT8847 FPU can compute the coordinates for graphic objects over a broad dynamic range. Also, the homogeneous scalar factors h1, h2, h3 and h4 may be made unity due to the availability of large dynamic range. In the example presented below, some of the calculations pertaining to vertex A' are shown but the same approach can be applied to any number of points and any vector space.

The calculations below show the sequence of operations for generating two coordinates, xa and ya, of the vertex A′ after rotation. The same sequence could be continued to generate the remaining two coordinates for A′ (za and h1). The other vertices of the tetrahedron, B′, C′, and D′, can be calculated in a similar way.

Table 52 presents a pseudocode description of the operations, clock cycles, and register contents for a single-precision matrix multiplication using the sum-of-products sequence presented in an earlier section. Registers used include the RA and RB input registers and the product (P) and sum (S) registers.

Table 52. Single-Precision Matrix Multiplication (PIPES2-PIPES0 = 010)

CLOCK CYCLE	MULTIPLIER/ALU OPERATIONS	PSEUDOCODE
1	Load a11, x11 SP Multiply	a11 → RA, x11 →RB p1 = a11 * x11
2	Load a12, x21 SP Multiply Pass P to S	a12 →RA, x21 →RB p2 = a12 * x21 p1 → P(p1)
3	Load a13, x31 SP Multiply Add P to S	a13 → RA, x31 →RB p3 = a13 * x31, p2 →P(p2) P(p1) + 0 → S(p1)
4	Load a14, x41 SP Multiply Add P to S	a14 → RA, x41 → RB p4 = a14 * x41, p3 → P(p3) P(p2) + S(p1) → S(p1 + p2)
5	Load a11, x12 SP Multiply Add P to S	a11 → RA, x12 → RB p5 = a11 * x12, p4 → P(p4) P(p3) + S(p1 + p2) → S(p1 + p2 + p3)
6	Load a12, x22 SP Multiply Pass P to S Output S	a12 → RA, x22 → RB p6 = a12 * x22, p5 → P(p5) P(p4) + S(p1 + p2 + p3) → S(p1 + p2 + p3 + p4)
7	Load a13, x32 SP Multiply Add P to S	a13 →RA, x32→ RB p7 = a13 * x32, p6 → P(p6) P(p5) + 0 → S(p5)
8	Load a14, x42 SP Multiply Add P to S	a14→RA, x42 →RB p8 = a14 * x42, p7 → P(p7) P(p6) + S(p5)→ S(p5 + p6)
9	Next operands Next instruction Add P to S	A → RA, B → RB pi = A * B, p8 → P(p8) P(p7) + S(p5 + p6) → S(p5 + p6 + p7)
10	Next operands Next instruction Output S	C → RA, D → RB pj = C * D, pi → P(pi) P(p8) + S(p5 + p6 + p7) → S(p5 + p6 + p7 + p8)

A microcode sequence to generate this matrix multiplication is shown in Table 53.

Table 53. Microinstructions for Sample Matrix Multiplication

I 10–0	CLKMODE	CONFIG 1–0	PIPES 2–0	SELOP 7–0	RND 1–0	FAST	ENRA	ENRB	SRCC	SELMS/$\overline{LS}$	$\overline{OEY}$	$\overline{OEC}$	$\overline{OES}$	BYTEP	SELST 1–0	$\overline{RESET}$	$\overline{HALT}$	TP 1–0
000 0100 0000	0	01	010	1111 xxxx	00	0	1	1	x	x	x	x	x	x	xx	1	1	11
100 0110 0000	0	01	010	1111 xxxx	00	0	1	1	x	x	x	x	x	x	xx	1	1	11
100 0000 0000	0	01	010	1111 1010	00	0	1	1	x	x	x	x	x	x	xx	1	1	11
100 0000 0000	0	01	010	1111 1010	00	0	1	1	x	x	x	x	x	x	xx	1	1	11
100 0000 0000	0	01	010	1111 1010	00	0	1	1	x	x	x	x	x	x	xx	1	1	11
100 0110 0000	0	01	010	1111 xxxx	00	0	1	1	x	x	x	x	x	x	xx	1	1	11
100 0000 0000	0	01	010	1111 1010	00	0	1	1	x	x	x	x	x	x	xx	1	1	11
100 0000 0000	0	01	010	1111 1010	00	0	1	1	x	x	x	x	x	x	xx	1	1	11
100 0000 0000	0	01	010	1111 1010	00	0	1	1	x	x	x	x	x	x	xx	1	1	11
100 0110 0000	0	01	010	1111 xxxx	00	0	1	1	x	x	x	x	x	x	xx	1	1	11

Six cycles are required to complete calculation of xa, the first coordinate, and after four more cycles the second coordinate ya is output. Each subsequent coordinate can be calculated in four cycles so the 4-tuple for vertex A′ requires a total of 18 cycles to complete.

Calculations for vertices B′, C′, and D′, can be executed in 48 cycles, 16 cycles for each vertex. Processing time improves when the transformation matrix is reduced, i.e., when the last column has the form shown below:

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

The h-scalars h1, h2, h3, and h4 are equal to 1. The number of clock cycles to generate each 4-tuple can then be decreased from 16 to 13 cycles. Total number of clock cycles to calculate all four vertices is reduced from 66 to 54 clocks. Figure 73 summarizes the overall matrix transformation.

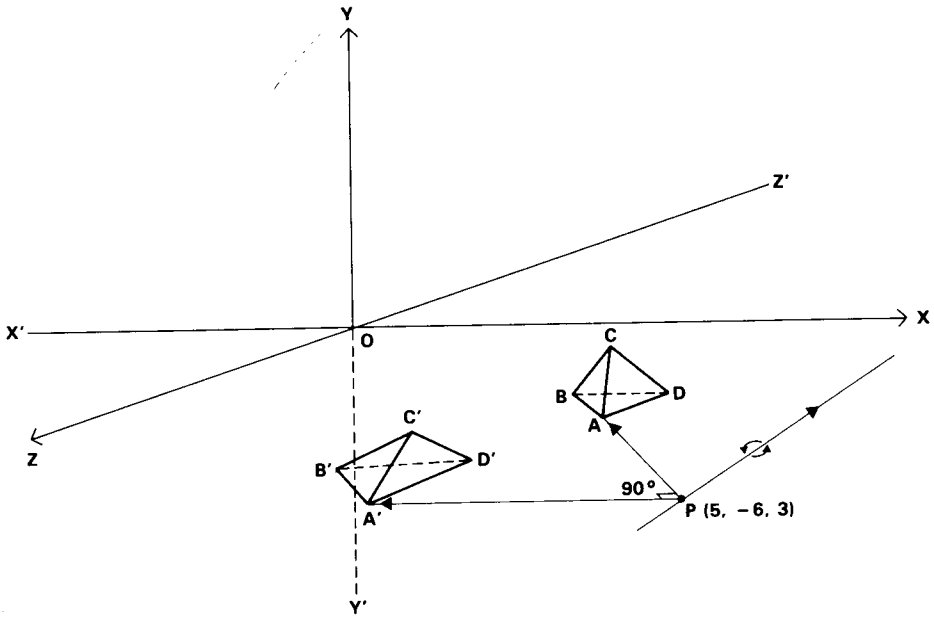


Figure 73. Resultant Matrix Transformation

This microprogram can also be written to calculate sums of products with all pipeline registers enabled so that the FPU can operate in its fastest mode. Because of timing relationships, the C register is used in some steps to hold the intermediate sum of products. Latency due to pipelining and chained data manipulation is 11 cycles for calculation of the first coordinate, and four cycles each for the other three coordinates.

After calculation of the first vertex, 16 cycles are required to calculate the four coordinates of each subsequent vertex. Table 54 presents the sequence of calculations for the first two coordinates, xA and yA.

Products in Table 54 are numbered according to the clock cycle in which the operands and instruction were loaded into the RA, RB, and I register, and execution of the instruction began. Sums indicated in Table 54 are listed below:

s1 = p1 + 0	s5 = p5 + p7	s9 = p10 + p12
s2 = p1 + p3	s6 = p6 + p8	xA = p1 + p2 + p3 + p4
s3 = p2 + p4	s7 = p9 + 0	yA = p5 + p6 + p7 + p8
s4 = p5 + 0	s8 = p9 + p11	

Table 54. Fully Pipelined Single-Precision Sum of Products (PIPES2-PIPES0 = 000)

CLOCK CYCLE	I BUS	DA BUS	DB BUS	I REG	RA REG	RB REG	MUL PIPE	ALU PIPE	P REG	S REG	C REG	Y BUS
0	Mul	x11	a11									
1	Mul	x21	a12	Mul	x11	a11						
2	Chn	x31	a13	Mul	x21	a12	p1					
3	Mul	x41	a14	Chn	x31	a13	p2		p1			
4	Chn	x12	a11	Mul	x41	a14	p3	s1	p2			
5	Chn	x22	a12	Chn	x12	a11	p4	†	p3	s1	p2	
6	Chn	x32	a13	Chn	x22	a12	p5	s2	p4	†	p2	
7	Chn	x42	a14	Chn	x32	a13	p6	s3	p5	s2	p2	
8	Chn	x13	a11	Chn	x42	a14	p7	s4	p6	s3	s2	
9	Chn	x23	a12	Chn	x13	a11	p8	xA	p7	s4	p6	
10	Chn	x33	a13	Chn	x23	a12	p9	s5	p8	xA	p6	xA
11	Chn	x43	a14	Chn	x33	a13	p10	s6	p9	s5	p6	
12	Chn	x14	a11	Chn	x43	a14	p11	s7	p10	s6	s5	
13	Chn	x24	a12	Chn	x14	a11	p12	yA	p11	s7	p10	
14	Chn	x34	a13	Chn	x24	a12	p13	s8	p12	yA	p10	yA
15	Chn	x44	a14	Chn	x34	a13	p14	s9	p13	s8	p10	

†Contents of this register are not valid during this cycle.

Chebyshev Routines for the SN74ACT8847 FPU

Introduction

Using the SN74ACT8847, very efficient routines can be developed for the implementation of transcendental functions. A high degree of accuracy can be achieved by taking advantage of the 'ACT8847's ability to perform calculations using double-precision floating point operands.

This application note describes how to use the 'ACT8847 to implement seven different transcendental functions. TIM (Texas Instruments Meta-Macro Assembler) assembly files have been written for all seven functions and these files are available upon request from Texas Instruments. The algorithm chosen to implement these functions is the Chebyshev expansion method [1]. Table 55 lists the functions that have been implemented, along with the number of cycles required, and time required to perform the calculations. Also listed in the table is the cycle count and time required to perform the same calculation using the Motorola MC68881 Floating Point Coprocessor and the Intel 80387 Numeric Processor Extension.

The Chebyshev expansion method was chosen rather than some of the more well known methods, such as the Taylor series and Newton-Raphson approximation, for a variety of reasons. The primary advantage of Chebyshev's method is that it provides a uniform convergence rate in the number of terms required to achieve the desired accuracy. Thus the range of the input value will have little effect on the accuracy of the result. Another advantage is that the number of terms required to calculate the

approximation is relatively small. This provides for faster execution. Also, Chebyshev's method can be applied to any function which is continuous and of bounded variation. Lastly, tables are available which contain the constants necessary to implement Chebyshev's method.

In order that this application note be useful to the largest audience, only those instructions and features common to all 'ACT8847 versions have been used to implement the routines.

Contact Texas Instruments VLSI Logic applications group at (214) 997-3970 for a copy of the seven TIM assembly files.

Table 55. Cycle Count and Execution Speed for the Seven Chebyshev Functions

FUNCTION	CYCLE COUNT†			EXECUTION SPEED‡ IN MICROSECONDS		
	'ACT8847	MC68881	80387	'ACT8847	MC68881	80387
Sine	51	416	122 to 771	1.53	25.0	7.32 to 46.3
Cosine	51	416	123 to 772	1.53	25.0	7.38 to 46.3
Tangent	84	498	191 to 497	2.52	29.9	11.5 to 29.8
ArcSine	68	606	Not Avail.	2.04	36.4	Not Avail.
ArcCosine	68	650	Not Avail.	2.04	39.0	Not Avail.
ArcTangent	104	428	314 to 487	3.12	25.7	18.8 to 29.2
Exponentiation	52	522	Not Avail.	1.56	31.3	Not Avail.

†For MC68881 cycle count refer to 'MC68881 Floating Point Coprocessor User's Manual', Document No. MC68881UM/AD, Page 6-13. For 80387 cycle count refer to '80387 Programmer's Reference Manual', Document No. 231917-001, Page E-36.
‡'ACT8847 cycle speed is 30 ns, 33 MHz
MC68881 cycle speed is 60 ns, 16.6 MHz
80387 cycle speed is 40 ns, 25 MHz

Overview of Chebyshev's Expansion Method

If f(x) is continuous and of bounded variation over the interval $-1 \leq x \leq 1$, then f(x) may be approximated by the following equation:

$$f(x) = 1/2a_0 + a_1T_1(x) + a_2T_2(x) + \ldots$$

$$= \sum_{r=0}^{\infty} a_rT_r(x)$$

Note that the range for x is between -1 and 1. For most functions, this restriction requires that the input, x, be range reduced before the calculation begins. Range reducing an argument means to scale the argument down to a certain range. In the case of Chebyshev approximations, the range is usually $-1 \leq x \leq 1$, or $0 \leq x \leq 1$.

In the equation for f(x) above, the constants represented by a_n are known as Chebyshev coefficients. The variables represented by T_r are known as Chebyshev polynomials and can be derived from the following relationship and values:

$$T_{r+1}(x) - 2xT_r(x) + T_{r-1}(x) = 0,$$
$$T_0(x) = 1,$$
$$T_1(x) = x$$

To illustrate Chebyshev's expansion method, the procedure to approximate function f(x) using the first seven polynomials is now covered. Let

$$\begin{aligned} f(x) = {} & 1/2a_0 + \\ & a_1T_1(x) + \\ & a_2T_2(x) + \\ & a_3T_3(x) + \\ & a_4T_4(x) + \\ & a_5T_5(x) + \\ & a_6T_6(x) \end{aligned}$$

Substituting in the expressions for the polynomials,

$$\begin{aligned} f(x) = {} & 1/2a_0 + \\ & a_1(x) + \\ & a_2(2x^2-1) + \\ & a_3(4x^3-3x) + \\ & a_4(8x^4-8x^2+1) + \\ & a_5(16x^5-20x^3+5x) + \\ & a_6(32x^6-48x^4+18x^2-1) \end{aligned}$$

Rearranging the expression, by grouping powers of x,

$$\begin{aligned} f(x) = {} & x^0(1/2a_0 - a_2 + a_4 - a_6) + \\ & x^1(a_1 - 3a_3 + 5a_5) + \\ & x^2(2a_2 - 8a_4 + 18a_6) + \\ & x^3(4a_3 - 20a_5) + \\ & x^4(8a_4 - 48a_6) + \\ & x^5(16a_5) + \\ & x^6(32a_6) \end{aligned}$$

Next make the following substitutions:

$$\begin{aligned}
\text{Let } c_0 &= 1/2a_0 - a_2 + a_4 - a_6 \\
c_1 &= a_1 - 3a_3 + 5a_5 \\
c_2 &= 2a_2 - 8a_4 + 18a_6 \\
c_3 &= 4a_3 - 20a_5 \\
c_4 &= 8a_4 - 48a_6 \\
c_5 &= 16a_5 \\
c_6 &= 32a_6
\end{aligned}$$

Substituting the c's into the last equation for f(x),

$$f(x) = c_0x^0 + c_1x^1 + c_2x^2 + c_3x^3 + c_4x^4 + c_5x^5 + c_6x^6$$

Applying Horner's Rule yields,

$$f(x) = (((((c_6x + c_5)x + c_4)x + c_3)x + c_2)x + c_1)x + c_0$$

In the remainder of the paper, the above equation will be referred to as C_{series}. Therefore,

$$C_{series\_f(x)} = (((((c_6x + c_5)x + c_4)x + c_3)x + c_2)x + c_1)x + c_0$$

The last step prior to approximating f(x) is to calculate the c's by substituting the values for the Chebyshev coefficients into the equations for c_0 through c_6.

Format for the Remainder of the Application Note

Each of the seven functions will be covered in a separate section. Each section will include the following information:

1. General steps required to perform the calculation including a description of any preprocessing and/or postprocessing
2. An algorithm for each of the above steps
3. What system intervention, if any, is required; this intervention may take the form of branching based on comparision status generated by the 'ACT8847, or storing and then later retrieving intermediate results
4. The number of 'ACT8847 cycles required to calculate f(x)
5. A listing of the c's
6. Pseudocode table showing how the calculation is accomplished. The pseudocode tables list the contents of all the relevent 'ACT8847 registers and buses for each instruction.
7. Microcode table listing the instructions

References

[1] C. W. Clenshaw, G. F. Miller, and M. Woodger, "Algorithms for Special Functions I," Numerische Mathematik, Vol 4, 1963, pages 403 through 419.

[2] C. W. Clenshaw, "Chebyshev Series for Mathematical Functions," Vol 5 of the Mathematical Tables of the National Physical Laboratory, Department of Scientific Industrial Research, England, 1960.

Cosine Routine Using Chebyshev's Method

All floating point inputs and outputs are double precision. The input is in radians.

Steps Required to Perform the Calculation

STEP 1 — Preprocessing; range-reduce the input, X, to a range of [−1,1]. Next square this range-reduced value, multiply it by 2.0, and finally subtract 1.0. X3 is the range-reduced input value, it must be stored externally. 'TRUNC' means to truncate.

```
X1 ← X*(2.0/pi)
X2 ← (4(TRUNC(0.25(X1 + 2.0)))) − X1 + 1.0
If X2 > 1.0
   Then X3 ← 2.0 − X2
   Else X3 ← X2
X4 ← 2.0*(X3*X3) − 1.0
```

STEP 2 — Core Calculation; X4 in Step 1 will be referred to as 'x' in the core calculation.

$$X5 \leftarrow C_{series\_cos}$$
$$\leftarrow (((((((c_8*x + c_7)*x + c_6)*x + c_5)*x + c_4)*x + c_3)*x + c_2)*x + c_1)*x + c_0$$

STEP 3 — Postprocessing; multiply the output of the core calculation times X3.

Cosine(X) ← X5*X3

Algorithms for the Three Steps

Step 1 perform the preprocessing:

```
T1  ←X*(2.0/pi)          2.0/pi entered as a constant
T2  ←T1 + 2.0
T3  ←0.25*T2 and         CREG ← T1, T3 and T4 result
T4  ←1.0 – CREG          from a chained instruction
T5  ←INT(T3)             round controls set to truncate
T6  ←4*T5                CREG ← T4
T7  ←DOUBLE(T6)          convert from integer to double
T8  ←T7 + CREG
CMP (1.0,T8)
If (1.0 > T8)            CREG ← T8
    Then T9 ← 2.0 – CREG T9 is X3 in Step 1, must
    Else T9 ← CREG       be stored externally
                         CREG → T9

T10 ←CREG*CREG
T11 ←T10 *2.0
T12 ←T11 – 1.0           T12 is X4 in Step 1, the
                         input to the core routine
```

Step 2 perform the core calculation:

```
T13 ←c8*CREG             CREG ← T12
T14 ←T13 + c7
T15 ←T14*CREG
T16 ←T15 + c6
T17 ←T16*CREG
T18 ←T17 + c5
T19 ←T18*CREG
T20 ←T19 + c4
T21 ←T20*CREG
T22 ←T21 + c3
T23 ←T22*CREG
T24 ←T23 + c2
T25 ←T24*CREG
T26 ←T25 + c1
T27 ←T26*CREG
T28 ←T27 + c0
```

Step 3 perform the postprocessing:

```
Cosine(X) ← T28*T9
```

Required System Intervention

As seen in the algorithm for Step 1, the 'ACT8847 performs a compare. The results of this compare determine which one of two calculations is to be performed. The system, in which the 'ACT8847 is a part, must make the decision as to which of the two calculations is to be performed. In addition, the system must store X3 and then later furnish X3 as an input to the 'ACT8847.

Number of 'ACT8847 Cycles Required to Calculate Cosine(x)

Calculation of Cosine(x) requires 46 cycles. In addition, it is assumed that five additional cycles are required due to the compare instruction, and resulting system intervention. Therefore, the total number of cycles to perform the Cosine(x) calculation is 51.

Listing of the Chebyshev Constants (c's)

The constants are represented in IEEE double-precision floating point format.

c_8 = 3D19D46B7D4C8F32
c_7 = BD962909C5C01ED6
c_6 = 3E0D53517735F927
c_5 = BE7CC930FD0ADA9D
c_4 = 3EE3E0AF61F7677F
c_3 = BF41E5FDEF25C403
c_2 = 3F92A9FB40C119ED
c_1 = BFD23B03366AA0C9
c_0 = 3FF4464BCC8CBA1F

Pseudocode Table for the Cosine(x) Calculation

Table 56. Pseudocode for Chebyshev Cosine Routine (PIPES2-0 = 010, RND1-0 =00)

CLK	DA BUS	DB BUS	RA REG	RB REG	CLK MODE	INSTR	MUL PIPE	ALU PIPE	P REG	C REG	S REG	Y BUS	COMMENT
1	X MSH	X LSH			0	RA2*RB2							X is the input
2	2DIVPI MSH	2DIVPI LSH	X	2DIVPI	0	RA2*RB2							2DIVPI is a constant representing 2.0/pi
3	1.0 MSH	1.0 LSH	X	2DIVPI	0	PR4+RB4	RA2*RB2						Preload RA with 1.0 for use in cycles 5 and 11
4	2.0 MSH	2.0 LSH	1.0	2.0	0	PR4+RB4			P1				
5	0.25 MSH	0.25 LSH	1.0	0.25	1	SR5*RB5 RA5−CR5				P1	S1		
6			1.0	0.25	0	DP2I(PR7)	SR5*RB5	RA5−CR5					Double precision → integer
7			1.0	0.25	0	DP2I(PR7)			P2		S2		Cycles 6,7 set RND1, 0 = 01
8		4	1.0	4	0	SR8*RB8				S2	S3		
9			1.0	4	1	I2DP(PR9)			P3				Integer → double-precision
10			1.0	4	1	CR10+SR10					S4		
11			1.0	4	1	COMPARE RA11,SR11					S5		If SR11 > RA11 then 13a If SR11 ≤ RA11 then 13b
12			1.0	4	0	NOP				S5			Wait for system response
13a	2.0 MSH	2.0 LSH	1.0	2.0	1	RB13−CR13							Execute 13a or 13b
13b			1.0	4	1	PAS(CR13)							Pass contents of CREG
14			1.0	2.0 or 4	1	CR14*CR14					S6	S6	S6 is either RB13-CR13 or CR13 from PASS CR13, and must be stored externally for use in cycle 43
15	2.0 MSH	2.0 LSH	1.0	2.0 or 4	0	RA16*PR16	CR14*CR14			S6		S6	Output S6 in cycles 14 and 15
16			2.0	2.0 or 4	0	RA16*PR16			P4				

Table 56. Pseudocode for Chebyshev Cosine Routine (PIPES2-0 = 010, RND1-0 = 00) (Continued)

CLK	DA BUS	DB BUS	RA REG	RB REG	CLK MODE	INSTR	MUL PIPE	ALU PIPE	P REG	C REG	S REG	Y BUS	COMMENT
17			2.0	2.0 or 4	0	PR18 + RB18	RA16 * PR16						
18	−1.0 MSH	−1.0 LSH	2.0	−1.0	0	PR18 + RB18			P5				
19	c_8 MSH	c_8 LSH	2.0	c_8	1	SR19 * RB19					S7		Start core calculation
20			2.0	c_8	0	PR21 + RB21	SR19 * RB19			S7			S7 is input to core calc.
21	c_7 MSH	c_7 LSH	2.0	c_7	0	PR21 + RB21			P6				
22			2.0	c_7	1	SR22 * CR22					S8		
23			2.0	c_7	0	PR24 + RB24	SR22 * CR22						
24	c_6 MSH	c_6 LSH	2.0	c_6	0	PR24 + RB24			P7				
25			2.0	c6	1	SR25 * CR25					S9		
26			2.0	c_6	0	PR27 + RB27	SR25 * CR25						
27	c_5 MSH	c_5 LSH	2.0	c_5	0	PR27 + RB27			P8				
28			2.0	c_5	1	SR28 * CR28					S10		
29			2.0	c_5	0	PR30 + RB30	SR28 * CR28						
30	c_4 MSH	c_4 LSH	2.0	c_4	0	PR30 + RB30			P9				
31			2.0	c_4	1	SR31 * CR31					S11		
32			2.0	c_4	0	PR33 + RB33	SR31 * CR31						
33	c_3 MSH	c_3 LSH	2.0	c_3	0	PR33 + RB33			P10				
34			2.0	c_3	1	SR34 * CR34					S12		
35			2.0	c_3	0	PR36 + RB36	SR34 * CR34						
36	c_2 MSH	c_2 LSH	2.0	c_2	0	PR36 + RB36			P11				

Table 56. Pseudocode for Chebyshev Cosine Routine (PIPES2-0 = 010, RND1-0 = 00) (Concluded)

CLK	DA BUS	DB BUS	RA REG	RB REG	CLK MODE	INSTR	MUL PIPE	ALU PIPE	P REG	C REG	S REG	Y BUS	COMMENT
37			2.0	c_2	1	SR37*CR37					S13		
38			2.0	c_2	0	PR39+RB39	SR37*CR37						
39	c_1 MSH	c_1 LSH	2.0	c_1	0	PR39+RB39			P12				
40			2.0	c_1	1	SR40*CR40					S14		
41			2.0	c_1	0	PR42+RB42	SR40*CR40						
42	c_0 MSH	c_0 LSH	2.0	c_0	0	PR42+RB42			P13				
43	S6 MSH	S6 LSH	2.0	S6	1	SR43*RB43	S15						Begin postprocessing
44			2.0	S6	0	DUMMY	SR43*RB43						Instruction is double-precision RA+RB, allows time for answer to propagate to the Y bus
45			2.0	S6	0	NOP			P14			P14	Output MSH of answer
46			2.0	S6	0	NOP			P14			P14	Output LSH of answer

Microcode Table for the Cosine(x) Calculation

All numbers are in hex. Any field with a length that is not a multiple of 4 is right justified and zero filled. For the microcode table, the value of X has been chosen to be 1/2 pi.

PA	DA	DB	PB	ENA	ENB	CLKC	PIPES	CLKMOD	CONFIG	SELOP	$\overline{RESET}$	$\overline{HALT}$	ENC	FLOWC	INSTR	RND	FAST	SRCC	BYTEP	SELST	TEST	SELY	$\overline{OEY}$	$\overline{OES}$	$\overline{OEC}$
F	3FF921FB	54442D18	F	0	0	_	2	0	3	FF	1	1	1	0	1C0	0	0	0	0	3	3	1	0	0	0
F	3FE45F30	6DC9C883	F	1	1	_	2	0	3	FF	1	1	1	0	1C0	0	0	0	0	3	3	1	0	0	0
F	3FF00000	00000000	F	0	0	_	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0
F	40000000	00000000	F	1	1	_	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0
F	3FD00000	00000000	F	0	1	↑	2	1	3	BD	1	1	0	0	581	0	0	1	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	_	2	0	3	FB	1	1	1	0	1A3	1	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	_	2	0	3	FB	1	1	1	0	1A3	1	0	0	0	3	3	1	0	0	0
F	00000000	00000004	F	0	1	↑	2	0	1	BF	1	1	0	0	240	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	_	2	1	3	FB	1	1	1	0	1A2	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	_	2	1	3	F6	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	_	2	1	3	FE	1	1	1	0	182	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	↑	2	0	3	FF	1	1	0	0	300	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	_	2	1	3	F7	1	1	1	0	1A0	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	_	2	1	3	5F	1	1	1	0	1C0	0	0	0	0	3	3	1	0	0	0
F	40000000	00000000	F	0	0	↑	2	0	3	EF	1	1	0	0	1C0	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	1	0	_	2	0	3	EF	1	1	1	0	1C0	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	_	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0
F	BFF00000	00000000	F	0	1	_	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0
F	3D19D46B	7D4C8F32	F	0	1	_	2	1	3	BF	1	1	1	0	1C0	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	↑	2	0	3	FB	1	1	0	0	180	0	0	0	0	3	3	1	0	0	0
F	BD962909	C5C01ED6	F	0	1	_	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0

Microcode Table for the Cosine(x) Calculation (Continued)

PA	DA	DB	PB	ENA	ENB	CLKC	PIPES	CLKMOD	CONFIG	SELOP	$\overline{\text{RESET}}$	$\overline{\text{HALT}}$	ENC	FLOWC	INSTR	RND	FAST	SRCC	BYTEP	STEST	TEST	SELY	$\overline{\text{OEY}}$	$\overline{\text{OES}}$	$\overline{\text{OEC}}$
F	00000000	00000000	F	0	0	_	2	1	3	9F	1	1	1	0	1C0	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	_	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0
F	3E0D5351	7735F927	F	0	1	_	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	_	2	1	3	9F	1	1	1	0	1C0	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	_	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0
F	BE7CC930	FD0ADA9D	F	0	1	_	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	_	2	1	3	9F	1	1	1	0	1C0	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	_	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0
F	3EE3E0AF	61F7677F	F	0	1	_	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	_	2	1	3	9F	1	1	1	0	1C0	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	_	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0
F	BF41E5FD	EF25C403	F	0	1	_	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	_	2	1	3	9F	1	1	1	0	1C0	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	_	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0
F	3F92A9FB	40C119ED	F	0	1	_	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	_	2	1	3	9F	1	1	1	0	1C0	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	_	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0
F	BFD23B03	366AA0C9	F	0	1	_	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	_	2	1	3	9F	1	1	1	0	1C0	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	_	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0
F	3FF4464B	CC8CBA1F	F	0	1	_	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	1	_	2	1	3	BF	1	1	1	0	1C0	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	_	2	0	3	FF	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	_	2	0	3	FF	1	1	1	0	300	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	_	2	0	3	FF	1	1	1	0	300	0	0	0	0	3	3	0	0	0	0

Sine Routine Using Chebyshev's Method

All floating point inputs and outputs are double precision. The input is in radians.

Steps Required to Perform the Calculation

STEP 1 – Preprocessing; range reduce the input, X, to a range of [– 1,1]. Next square this range-reduced value, multiply it by 2.0, and finally subtract 1.0. X3 is the range-reduced input value, it must be stored externally. 'TRUNC' means to truncate.

```
X1 ← X*(2.0/pi)
X2 ← X1 – (4(TRUNC(0.25(X1 + 1.0))))
If X2 > 1.0
   Then X3 ← 2.0 – X2
   Else X3 ← X2
X4 ← 2.0*(X3*X3) – 1.0
```

STEP 2 – Core calculation; X4 in Step 1 will be referred to as 'x' in the core calculation.

$$X5 \leftarrow C_{series\_sin}$$

$$\leftarrow (((((((c_8 * x + c_7) * x + c_6) * x + c_5) * x + c_4) * x + c_3) * x + c_2) * x + c_1) * x + c_0$$

STEP 3 – Postprocessing; multiply the output of the core calculation times X3.

```
Sine(X) ← X5*X3
```

Algorithms for the Three Steps

Step 1 perform the preprocessing:

```
T1  ←X*(2.0/pi)          2.0/pi entered as a constant
T2  ←T1 + 1.0
T3  ←0.25*T2             CREG ← T1
T4  ←INT(T3)             round controls set to truncate
T5  ←4*T4
T6  ←DOUBLE(T5)          convert from integer to double
T7  ←CREG – T6
CMP (1.0,T7)             compare 1.0 to T7
If (1.0 > T7)            CREG ← T7
    Then T8 ← 2.0 – CREG T8 is X3 in Step 1, must
    Else T8 ← CREG       be stored externally
                         CREG → T8
T9  ←CREG*CREG
T10 ←T9 *2.0
T11 ←T10 – 1.0           T11 is X4 in Step 1 above, the input to
                         the core routine
                         T11 = 'x' from Step 2 above
```

Step 2 perform the core calculation:

$T12 \leftarrow c8 * CREG$	$CREG \leftarrow T11$
$T13 \leftarrow T12 + c_7$	
$T14 \leftarrow T13 * CREG$	
$T15 \leftarrow T14 + c_6$	
$T16 \leftarrow T15 * CREG$	
$T17 \leftarrow T16 + c_5$	
$T18 \leftarrow T17 * CREG$	
$T19 \leftarrow T18 + c_4$	
$T20 \leftarrow T19 * CREG$	
$T21 \leftarrow T20 + c_3$	
$T22 \leftarrow T21 * CREG$	
$T23 \leftarrow T22 + c_2$	
$T24 \leftarrow T23 * CREG$	
$T25 \leftarrow T24 + c_1$	
$T26 \leftarrow T25 * CREG$	
$T27 \leftarrow T26 \leftarrow c_0$	

Step 3 perform the postprocessing:

$Sine(X) \leftarrow T27 * T8$

Required System Intervention

As seen in the algorithm for Step 1, the 'ACT8847 performs a compare. The results of this compare determine which one of two calculations is to be performed. The system, in which the 'ACT8847 is a part, must make the decision between which two calculations are to be performed. In addition, the system must store X3 and then later furnish X3 as an input to the 'ACT8847.

Number of 'ACT8847 Cycles Required to Calculate Sine(x)

Calculation of Sine(x) requires 46 cycles. In addition, it is assumed that five additional cycles are required due to the compare instruction and resulting system intervention. Therefore, the total number of cycles to perform the Sine(x) calculation is 51.

Listing of the Chebyshev Constants (c's)

The constants are represented in IEEE double-precision floating point format.

c_8 = 3D19D46B7D4C8F32
c_7 = BD962909C5C01ED6
c_6 = 3E0D53517735F927
c_5 = BE7CC930FD0ADA9D
c_4 = 3EE3E0AF61F7677F
c_3 = BF41E5FDEF25C403
c_2 = 3F92A9FB40C119ED
c_1 = BFD23B03366AA0C9
c_0 = 3FF4464BCC8CBA1F

Pseudocode Table for the Sine(x) Calculation

Table 57. Pseudocode for Chebyshev Sine Routine (PIPES2-0 = 010, RND1-0 = 00)

CLK	DA BUS	DB BUS	RA REG	RB REG	CLK MODE	INSTR	MUL PIPE	ALU PIPE	P REG	C REG	S REG	Y BUS	COMMENT
1	X MSH	X LSH			0	RA2*RB2							X is the input
2	2DIVPI MSH	2DIVPI LSH	X	2DIVPI	0	RA2*RB2							2DIVPI is a constant representing 2.0/pi
3			X	2DIVPI	0	PR4+RB4	RA2*RB2						
4	1.0 MSH	1.0 LSH	X	1.0	0	PR4+RB4			P1				
5	0.25 MSH	0.25 LSH	X	0.25	1	SR5*RB5				P1	S1		
6	1.0 MSH	1.0 LSH	X	0.25	0	DP2I(PR7)	SR5*RB5						Double precision → integer
7			1.0	0.25	0	DP2I(PR7)			P2				Cycles 6,7 set RND1,0 = 01
8		4	1.0	4	0	SR8*RB8					S2		
9			1.0	4	1	I2DP(PR9)			P3				Integer → double precision
10			1.0	4	1	CR10−SR10					S3		
11			1.0	4	1	COMPARE RA11,SR11					S4		If SR11 → RA11 then 13a If SR11 ≤ RA11 then 13b
12			1.0	4	0	NOP				S4			Wait for system response
13a	2.0 MSH	2.0 LSH	1.0	2.0	1	RB13−CR13							Execute 13a or 13b
13b			1.0	4	1	PAS(CR13)							Pass contents of CREG
14			1.0	2.0 or 4	1	CR14*CR14					S5	S5	S5 is either RB13−CR13 or CR13 from PASS CR13, and must be stored externally for use in cycle 43
15	2.0 MSH	2.0 LSH	1.0	2.0 or 4	0	RA16*PR16	CR14*CR14			S5		S5	Output S5 in cycles 14 and 15
16			2.0	2.0 or 4	0	RA16*PR16			P4				
17			2.0	2.0 or 4	0	PR18+RB18	RA16*PR16						
18	−1.0 MSH	−1.0 LSH	2.0	−1.0	0	PR18+RB18			P5				

Table 57. Pseudocode for Chebyshev Sine Routine (PIPES2-0 = 010, RND1-0 = 00) (Continued)

CLK	DA BUS	DB BUS	RA REG	RB REG	CLK MODE	INSTR	MUL PIPE	ALU PIPE	P REG	C REG	S REG	Y BUS	COMMENT
19	c_8 MSH	c_8 LSH	2.0	c_8	1	SR19•RB19					S6		Start core calculation
20			2.0	c_8	0	PR21 + RB21	SR19•RB19			S6			S7 is input to core calc.
21	c_7 MSH	c_7 LSH	2.0	c_7	0	PR21 + RB21			P6				
22			2.0	c_7	1	SR22•CR22					S7		
23			2.0	c_7	0	PR24 + RB24	SR22•CR22						
24	c_6 MSH	c_6 LSH	2.0	c_6	0	PR24 + RB24			P7				
25			2.0	c_6	1	SR25•CR25					S8		
26			2.0	c_6	0	PR27 + RB27	SR25•CR25						
27	c_5 MSH	c_5 LSH	2.0	c_5	0	PR27 + RB27			P8				
28			2.0	c_5	1	SR28•CR28					S9		
29			2.0	c_5	0	PR30 + RB30	SR28•CR28						
30	c_4 MSH	c_4 LSH	2.0	c_4	0	PR30 + RB30			P9				
31			2.0	c_4	1	SR31•CR31					S10		
32			2.0	c_4	0	PR33 + RB33	SR31•CR31						
33	c_3 MSH	c_3 LSH	2.0	c_3	0	PR33 + RB33			P10				
34			2.0	c3	1	SR34•CR34					S11		
35			2.0	c_3	0	PR36 + RB36	SR34•CR34						
36	c_2 MSH	c_2 LSH	2.0	c_2	0	PR36 + RB36			P11				
37			2.0	c_2	1	SR37•CR37					S12		
38			2.0	c_2	0	PR39 + RB39	SR37•CR37						
39	c_1 MSH	c_1 LSH	2.0	c_1	0	PR39 + RB39			P12				
40			2.0	c_1	1	SR40•CR40					S13		

Table 57. Pseudocode for Chebyshev Sine Routine (PIPES2-0 = 010, RND1-0 = 00) (Concluded)

CLK	DA BUS	DB BUS	RA REG	RB REG	CLK MODE	INSTR	MUL PIPE	ALU PIPE	P REG	C REG	S REG	Y BUS	COMMENT
41			2.0	c_1	0	PR42 + RB42	SR40 * CR40						
42	c_0 MSH	c_0 LSH	2.0	c_0	0	PR42 + RB42			P13				
43	S5 MSH	S5 LSH	2.0	S5	1	SR43 * RB43					S14		Begin postprocessing
44			2.0	S5	0	DUMMY	SR43 * RB43						Instruction is double-precision RA + RB, allows time for answer to propagate to the Y bus
45			2.0	S5	0	NOP			P14			P14	Output MSH of answer
46			2.0	S5	0	NOP			P14			P14	Output LSH of answer

Microcode Table for the Sine(x) Calculation

All numbers are in hex. Any field with a length that is not a multiple of 4 is right justified and zero filled. For the microcode table, the value of X has been chosen to be 1/2 pi.

PA	DA	DB	PB	ENA	ENB	CLKC	PIPES	CLKMOD	CONFIG	SELOP	$\overline{RESET}$	$\overline{HALT}$	ENC	FLOWC	INSTR	RND	FASTC	SRCC	BYTEP	STEST	TESTS	SELY	$\overline{OEY}$	$\overline{OES}$	$\overline{OEC}$
F	3FF921FB	54442D18	F	0	0	⎍	2	0	3	FF	1	1	1	0	1C0	0	0	0	0	3	3	1	0	0	0
F	3FE45F30	6DC9C883	F	1	1	⎍	2	0	3	FF	1	1	1	0	1C0	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	⎍	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0
F	3FF00000	00000000	F	0	1	⎍	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0
F	3FD00000	00000000	F	0	1	⎍↓	2	1	3	BF	1	1	0	0	1C0	0	0	1	0	3	3	1	0	0	0
F	3FF00000	00000000	F	0	0	⎍	2	0	3	FB	1	1	1	0	1A3	1	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	1	0	⎍	2	0	3	FB	1	1	1	0	1A3	1	0	0	0	3	3	1	0	0	0
F	00000000	00000004	F	0	1	⎍	2	0	1	BF	1	1	1	0	240	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	⎍	2	1	3	FB	1	1	1	0	1A2	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	⎍	2	1	3	F6	1	1	1	0	181	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	⎍	2	1	3	FE	1	1	1	0	182	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	⎍↓	2	0	3	FF	1	1	0	0	300	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	⎍	2	1	3	F7	1	1	1	0	1A0	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	⎍	2	1	3	5F	1	1	1	0	1C0	0	0	0	0	3	3	1	0	0	0
F	40000000	00000000	F	0	0	⎍↓	2	0	3	EF	1	1	0	0	1C0	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	1	0	⎍	2	0	3	EF	1	1	1	0	1C0	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	⎍	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0
F	BFF00000	00000000	F	0	1	⎍	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0
F	3D19D46B	7D4C8F32	F	0	1	⎍	2	1	3	BF	1	1	1	0	1C0	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	⎍↓	2	0	3	FB	1	1	0	0	180	0	0	0	0	3	3	1	0	0	0
F	BD962909	C5C01ED6	F	0	1	⎍	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0

Microcode Table for the Sine(x) Calculation (Continued)

PA	DA	DB	PB	ENA	ENB	CLKC	PIPES	CLKMOD	CONFIG	SELOP	$\overline{\text{RESET}}$	$\overline{\text{HALT}}$	ENC	FLOWC	INSTR	RND	FAST	SRCC	BYTEP	STEST	TEST	SELY	$\overline{\text{OEY}}$	$\overline{\text{OES}}$	$\overline{\text{OEC}}$
F	00000000	00000000	F	0	0	_	2	1	3	9F	1	1	1	0	1C0	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	_	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0
F	3E0D5351	7735F927	F	0	1	_	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	_	2	1	3	9F	1	1	1	0	1C0	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	_	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0
F	BE7CC930	FD0ADA9D	F	0	1	_	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	_	2	1	3	9F	1	1	1	0	1C0	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	_	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0
F	3EE3E0AF	61F7677F	F	0	1	_	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	_	2	1	3	9F	1	1	1	0	1C0	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	_	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0
F	BF41E5FD	EF25C403	F	0	1	_	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	_	2	1	3	9F	1	1	1	0	1C0	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	_	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0
F	3F92A9FB	40C119ED	F	0	1	_	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	_	2	1	3	9F	1	1	1	0	1C0	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	_	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0
F	BFD23B03	366AA0C9	F	0	1	_	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	_	2	1	3	9F	1	1	1	0	1C0	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	_	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0
F	3FF4464B	CC8CBA1F	F	0	1	_	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0
F	3FF00000	00000000	F	0	1	_	2	1	3	BF	1	1	1	0	1C0	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	_	2	0	3	FF	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	_	2	0	3	FF	1	1	1	0	300	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	_	2	0	3	FF	1	1	1	0	300	0	0	0	0	3	3	0	0	0	0

Tangent Routine Using Chebyshev's Method

All floating point inputs and outputs are double precision. The input is in radians.

Steps Required to Perform the Calculation

STEP 1 — Preprocessing; range reduce the input, X, to a range of [− 1,1]. Next square this range-reduced value, multiply it by 2.0, and finally subtract 1.0. X3 is the range-reduced input value, it must be stored externally. 'TRUNC' means to truncate. If X2 > 1.0, then in the postprocessing part of the routine, the answer is the reciprocal of X5*X3.

X1 ← X*(4.0/pi)
X2 ← X1 − (4(TRUNC(0.25(X1 + 1.0))))
If X2 > 1.0
  Then X3 ← 2.0 − X2
  Else X3 ← X2
X4 ← 2.0*(X3*X3) − 1.0

STEP 2 — Core Calculation; X4 in Step 1 will be referred to as 'x' in the core calculation.

$X5 \leftarrow C_{series\_tan}$

$\leftarrow ((((((((((((c_{14})*x + c_{13})*x + c_{12})*x + c_{11})*x + c_{10})*x + c_9)*x + c_8)*x + c_7)*x + c_6)*x + c_5)*x + c_4)*x + c_3)*x + c_2)*x + c_1)*x + c_0$

STEP 3 — Postprocessing; multiply the output of the core calculation times X3. If X2 > 1.0, then the reciprocal of X5*X3 is the answer, if X2 ≤ 1.0 then X5*X3 is the answer.

Tangent(X) ← X5*X3 (or reciprocal of X5*X3)

Algorithms for the Three Steps

Step 1 perform the preprocessing:

T1 ←X*(4.0/pi)	4.0/pi entered as a constant
T2 ←T1 + 1.0	
T3 ←0.25*T2	CREG ← T1
T4 ←INT(T3)	round controls set to truncate
T5 ←4*T4	
T6 ←DOUBLE(T5)	convert from integer to double
T7 ←CREG ← T6	
CMP (1.0,T7)	
If (1.0 > T7)	CREG ← T7
Then T8 ← 2.0 − CREG	T8 is X3 in Step 1, must
Else T8 ← CREG	be stored externally

```
T9  ←CREG*CREG                    CREG ← T8
T10 ←T9*2.0
T11 ←T10 − 1.0                    T11 is X4 in Step 1, the
                                  input to the core routine
```

Step 2 perform the core calculation:

```
T12 ←c14*CREG
T13 ←T12 + c13                    CREG ← T11
T14 ←T13*CREG
T15 ←T14 + c12
T16 ←T15*CREG
T17 ←T16 + c11
T18 ←T17*CREG
T19 ←T18 + c10
T20 ←T19*CREG
T21 ←T20 + c9
T22 ←T21*CREG
T23 ←T22 + c8
T24 ←T23*CREG
T25 ←T24 + c7
T26 ←T25*CREG
T27 ←T26 + c6
T28 ←T27*CREG
T29 ←T28 + c5
T30 ←T29*CREG
T31 ←T30 + c4
T32 ←T31*CREG
T33 ←T32 + c3
T34 ←T33*CREG
T35 ←T34 + c2
T36 ←T35*CREG
T37 ←T36 + c1
T38 ←T37*CREG
T39 ←T38 + c0
```

Step 3 perform the postprocessing:

```
T40 ←T39*T8
If X2 (in Step 1) > 1.0
   Then Tangent(X) ← 1.0/T40
   Else Tangent(X) ← T40
```

Required System Intervention

As seen in the algorithm for Step 1, the 'ACT8847 performs a compare. The results of this compare determine which one of two calculations is to be performed. The system, in which the 'ACT8847 is a part, must make the decision as to which of the two calculations is to be performed. In addition, the system must store X3 and then later furnish X3 as an input to the 'ACT8847. Finally, the system will have to determine if it is necessary to take the reciprocal of the final product (T40 in the Algorithm for Step 3) to yield the answer. If it is necessary to take the reciprocal, then the system will be required to direct the variable T40 from the 'ACT8847's output bus to the input buses. This is because operands for division instructions must be provided by the RA and RB registers; feedback is not an option.

Number of 'ACT8847 Cycles Required to Calculate Tangent(x)

Calculation of Tangent(x) requires 79 cycles. In addition, it is assumed that five additional cycles are required for system intervention due to the compare instruction. Therefore, the total number of cycles required to perform the Tangent(x) calculation is 84.

Listing of the Chebyshev Constants (c's)

The constants are represented in IEEE double-precision floating point format.

c_{14} = 3D747D842210CC35
c_{13} = 3DA1D66636043991
c_{12} = 3DCCD078F52B3A73
c_{11} = 3DF938F9CDDFF864
c_{10} = 3E2620430E99B5B7
c_{9} = 3E535C2C953CE515
c_{8} = 3E80F07AFC099D7F
c_{7} = 3EADA4D789EB45C4
c_{6} = 3ED9F03D4C51A771
c_{5} = 3F06B236DE4D014C
c_{4} = 3F33DBFB01B3F415
c_{3} = 3F6160DE701F3A53
c_{2} = 3F8E70A18736FC10
c_{1} = 3FBAEA2653199611
c_{0} = 3FEC14B2675B10BA

Psuedocode Table for the Tangent(x) Calculation

Table 58. Pseudocode for Chebyshev Tangent Routine (PIPES2-0 = 010, RND1-0 = 0)

CLK	DA BUS	DB BUS	RA REG	RB REG	CLK MODE	INSTR	MUL PIPE	ALU PIPE	P REG	C REG	S REG	Y BUS	COMMENT
1	X MSH	X LSH			0	RA2*RB2							X is the input
2	4DIVPI MSH	4DIVPI LSH	X	4DIV PI	0	RA2*RB2							4DIVPI is a constant representing 4.0/pi
3			X	4DIVPI	0	PR4+RB4	RA2*RB2						
4	1.0 MSH	1.0 LSH	X	1.0	0	PR4+RB4			P1				
5	0.25 MSH	0.25 LSH	X	0.25	1	SR5*RB5				P1	S1		
6	1.0 MSH	1.0 LSH	X	0.25	0	DP2I(PR7)	SR5*RB5						Double precision → integer
7			1.0	0.25	0	DP2I(PR7)			P2				Cycles 6,7 set RND1,0 = 01
8		4	1.0	4	0	SR8*RB8					S2		
9			1.0	4	1	I2DP(PR9)			P3				Integer → double precision
10			1.0	4	1	CR10-SR10					S3		
11			1.0	4	1	COMPARE RA11,SR11					S4		If SR11 > RA11 then 13a If SR11 ≤ RA11 then 13b
12			1.0	4	0	NOP				S4			Wait for system response
13a	2.0 MSH	2.0 LSH	1.0	2.0	1	RB13−CR13							Execute 13a or 13b
13b			1.0	4	1	PAS(CR13)							Pass contents of Creg
14			1.0	2.0 or 4	1	CR14*CR14					S5	S5	S5 is either RB13-CR13 or CR13 from PASS CR13, and must be stored externally for use in cycle 61
15	2.0 MSH	2.0 LSH	1.0	2.0 or 4	0	RA16*PR16	CR14*CR14			S5		S5	Output S5 in cycles 14 and 15
16			2.0	2.0 or 4	0	RA16*PR16			P4				
17			2.0	2.0 or 4	0	PR18+RB18	RA16*PR16						
18	−1.0 MSH	−1.0 LSH	2.0	−1.0	0	PR18+RB18			P5				

Table 58. Pseudocode for Chebyshev Tangent Routine (PIPES2-0 = 010, RND1-0 = 0) (Continued)

CLK	DA BUS	DB BUS	RA REG	RB REG	CLK MODE	INSTR	MUL PIPE	ALU PIPE	P REG	C REG	S REG	Y BUS	COMMENT
19	c_{14} MSH	c_{14} LSH	2.0	c_{14}	1	SR19•RB19					S6		Start core calculation
20			2.0	c_{14}	0	PR21 + RB21	SR19•RB19			S6			S7 is input to core calc.
21	c_{13} MSH	c_{13} LSH	2.0	c_{13}	0	PR21 + RB21			P6				
22			2.0	c_{13}	1	SR22•CR22					S7		
23			2.0	c_{13}	0	PR24 + RB24	SR22•CR22						
24	c_{12} MSH	c_{12} LSH	2.0	c_{12}	0	PR24 + RB24			P7				
25			2.0	c_{12}	1	SR25•CR25					S8		
26			2.0	c_{12}	0	PR27 + RB27	SR25•CR25						
27	c_{11} MSH	c_{11} LSH	2.0	c_{11}	0	PR27 + RB27			P8				
28			2.0	c_{11}	1	SR28•CR28					S9		
29			2.0	c_{11}	0	PR30 + RB30	SR28•CR28						
30	c_{10} MSH	c_{10} LSH	2.0	c_{10}	0	PR30 + RB30			P9				
31			2.0	c_{10}	1	SR31•CR31					S10		
32			2.0	c_{10}	0	PR33 + RB33	SR31•CR31						
33	c_9 MSH	c_9 LSH	2.0	c_9	0	PR33 + RB33			P10				
34			2.0	c_9	1	SR34•CR34					S11		
35			2.0	c_9	0	PR36 + RB36	SR34•CR34						
36	c_8 MSH	c_8 LSH	2.0	c_8	0	PR36 + RB36			P11				
37			2.0	c_8	1	SR37•CR37					S12		
38			2.0	c_8	0	PR39 + RB39	SR37•CR37						
39	c_7 MSH	c_7 LSH	2.0	c_7	0	PR39 + RB39			P12				
40			2.0	c_7	1	SR40•CR40					S13		
41			2.0	c_7	0	PR42 + RB42	SR40•CR40						
42	c_6 MSH	c_6 LSH	2.0	c_6	0	PR42 + RB42			P13				

Table 58. Pseudocode for Chebyshev Tangent Routine (PIPES2-0 = 010, RND1-0 = 0) (Concluded)

CLK	DA BUS	DB BUS	RA REG	RB REG	CLK MODE	INSTR	MUL PIPE	ALU PIPE	P REG	C REG	S REG	Y BUS	COMMENT
43			2.0	c_6	1	SR43•CR43					S14		
44			2.0	c_6	0	PR45 + RB45	SR43•CR43						
45	c_5 MSH	c_5 LSH	2.0	c_5	0	PR45 + RB45			P14				
46			2.0	c_5	1	SR46•CR46					S15		
47			2.0	c_5	0	PR48 + RB48	SR46•CR46						
48	c_4 MSH	c_4 LSH	2.0	c_4	0	PR48 + RB48			P15				
49			2.0	c_4	1	SR49•CR49					S16		
50			2.0	c_4	0	PR51 + RB51	SR49•CR49						
51	c_3 MSH	c_3 LSH	2.0	c_3	0	PR51 + RB51			P16				
52			2.0	c_3	1	SR52•CR52					S17		
53			2.0	c_3	0	PR54 + RB54	SR52•CR52						
54	c_2 MSH	c_2 LSH	2.0	c_2	0	PR54 + RB54			P17				
55			2.0	c_2	1	SR55•CR55					S18		
56			2.0	c_2	0	PR57 + RB57	SR55•CR55						
57	c_1 MSH	c_1 LSH	2.0	c_1	0	PR57 + RB57			P18				
58			2.0	c_1	1	SR58•CR58					S19		
59			2.0	c_1	0	PR60 + RB60	SR58•CR58						
60	c_0 MSH	c_0 LSH	2.0	c_0	0	PR60 + RB60			P19				
61	S5 MSH	S5 LSH	2.0	S5	1	SR61•RB61					S20		Begin postprocessing
62			2.0	S5	0	DUMMY	SR61•RB61						Instruction is RA + RB, used to allow time for result to propagate to Y bus

Table 58. Pseudocode for Chebyshev Tangent Routine (PIPES2-0 = 010, RND1-0 = 0) (Continued)

CLK	DA BUS	DB BUS	RA REG	RB REG	CLK MODE	INSTR	MUL PIPE	ALU PIPE	P REG	C REG	S REG	Y BUS	COMMENT
63			2.0	S5	0	NOP			P20			P20	Output MSH, if cycle 13b was executed then P20 is the answer; if cycle 13a was executed then the answer is 1.0/P20, which is calculated next
64	1.0 MSH	1.0 LSH	2.0	S5	0	DIV						P20	Output LSH
65	P20 MSH	P20 LSH	1.0	P20	0	DIV							Operands for Division must come from RA and RB, feedback is not an option
66			1.0	P20	0	NOP							Wait for Division result
67			1.0	P20	0	NOP							Wait for Division result
68			1.0	P20	0	NOP							Wait for Division result
69			1.0	P20	0	NOP							Wait for Division result
70			1.0	P20	0	NOP							Wait for Division result
71			1.0	P20	0	NOP							Wait for Division result
72			1.0	P20	0	NOP							Wait for Division result
73			1.0	P20	0	NOP							Wait for Division result
74			1.0	P20	0	NOP							Wait for Division result
75			1.0	P20	0	NOP							Wait for Division result
76			1.0	P20	0	NOP							Wait for Division result
77			1.0	P20	0	NOP							Wait for Division result
78			1.0	P20	0	NOP			P21			P21	Output MSH of answer
79			1.0	P20	0	NOP			P21			P21	Output LSH of answer

Microcode Table for the Tangent(x) Calculation

All numbers are in hex. Any field with a length that is not a multiple of 4 is right justified and zero filled. For the microcode table, the value of X has been chosen to be 1/3 pi.

PA	DA	DB	PB	ENA	ENB	CLKC	PIPES	CLKMOD	CONFIG	SELOP	$\overline{RESET}$	$\overline{HALT}$	ENC	FLOWC	INSTR	RND	FASTC	SRCC	BYTEP	SSTEST	TESTS	SSLY	$\overline{OEY}$	$\overline{OES}$	$\overline{OEC}$
F	3FF0C152	382D7365	F	0	0	_	2	0	3	FF	1	1	1	0	1C0	0	0	0	0	3	3	1	0	0	0
F	3FF45F30	6DC9C883	F	1	1	_	2	0	3	FF	1	1	1	0	1C0	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	_	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0
F	3FF00000	00000000	F	0	1	_	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0
F	3FD00000	00000000	F	0	1	⌐	2	1	3	BF	1	1	0	0	1C0	0	0	1	0	3	3	1	0	0	0
F	3FF00000	00000000	F	0	0	_	2	0	3	FB	1	1	1	0	1A3	1	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	1	0	_	2	0	3	FB	1	1	1	0	1A3	1	0	0	0	3	3	1	0	0	0
F	00000000	00000004	F	0	1	_	2	0	1	BF	1	1	1	0	240	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	_	2	1	3	FB	1	1	1	0	1A2	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	_	2	1	3	F6	1	1	1	0	181	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	_	2	1	3	FE	1	1	1	0	182	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	⌐	2	0	3	FF	1	1	0	0	300	0	0	0	0	3	3	1	0	0	0
F	40000000	00000000	F	0	1	_	2	1	3	F7	1	1	1	0	183	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	_	2	1	3	5F	1	1	1	0	1C0	0	0	0	0	3	3	1	0	0	0
F	40000000	00000000	F	0	0	⌐	2	0	3	EF	1	1	0	0	1C0	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	1	0	_	2	0	3	EF	1	1	1	0	1C0	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	_	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0
F	BFF00000	00000000	F	0	1	_	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0
F	3D747D84	2210CC35	F	0	1	_	2	1	3	BF	1	1	1	0	1C0	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	⌐	2	0	3	FB	1	1	0	0	180	0	0	0	0	3	3	1	0	0	0
F	3DA1D666	36043991	F	0	1	_	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0

Microcode Table for the Tangent(x) Calculation (Continued)

PA	DA	DB	PB	ENA	ENB	CLKC	PIPES	CLKMOD	CONFIG	SELOP	$\overline{\text{RESET}}$	$\overline{\text{HALT}}$	ENC	FLOWC	INSTR	RND	FASTC	SRCC	BYTEP	SESTST	TESTS	SELY	$\overline{\text{OEY}}$	$\overline{\text{OES}}$	$\overline{\text{OEC}}$
F	00000000	00000000	F	0	0	_	2	1	3	9F	1	1	1	0	1C0	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	_	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0
F	3DCCD078	F52B3A73	F	0	1	_	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	_	2	1	3	9F	1	1	1	0	1C0	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	_	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0
F	3DF938F9	CDDFF864	F	0	1	_	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	_	2	1	3	9F	1	1	1	0	1C0	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	_	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0
F	3E262043	0E99B5B7	F	0	1	_	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	_	2	1	3	9F	1	1	1	0	1C0	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	_	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0
F	3E535C2C	953CE515	F	0	1	_	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	_	2	1	3	9F	1	1	1	0	1C0	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	_	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0
F	3E80F07A	FC099D7F	F	0	1	_	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	_	2	1	3	9F	1	1	1	0	1C0	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	_	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0
F	3EADA4D7	89EB45C4	F	0	1	_	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	_	2	1	3	9F	1	1	1	0	1C0	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	_	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0
F	3ED9F03D	4C51A771	F	0	1	_	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	_	2	1	3	9F	1	1	1	0	1C0	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	_	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0
F	3F06B236	DE4D014C	F	0	1	_	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0

Microcode Table for the Tangent(x) Calculation (Continued)

PA	DA	DB	PB	ENA	ENB	CLKC	PIPES	CLKMOD	CONFIG	SELOP	$\overline{RESET}$	$\overline{HALT}$	ENC	FLOWC	INSTR	RND	FAST	SRCC	BYTEP	SELST	TESTY	SELY	$\overline{OEY}$	$\overline{OES}$	$\overline{OEC}$
F	00000000	00000000	F	0	0	_	2	1	3	9F	1	1	1	0	1C0	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	_	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0
F	3F33DBFB	01B3F415	F	0	1	_	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	_	2	1	3	9F	1	1	1	0	1C0	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	_	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0
F	3F6160DE	701F3A53	F	0	1	_	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	_	2	1	3	9F	1	1	1	0	1C0	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	_	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0
F	3F8E70A1	8736FC10	F	0	1	_	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	_	2	1	3	9F	1	1	1	0	1C0	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	_	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0
F	3FBAEA26	53199611	F	0	1	_	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	_	2	1	3	9F	1	1	1	0	1C0	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	_	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0
F	3FEC14B2	675B10BA	F	0	1	_	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0
F	3FE55555	55555555	F	0	1	_	2	1	3	BF	1	1	1	0	1C0	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	_	2	0	3	FF	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	_	2	0	3	FF	1	1	1	0	300	0	0	0	0	3	3	1	0	0	0
F	3FF00000	00000000	F	0	0	_	2	0	3	FF	1	1	1	0	1E0	0	0	0	0	3	3	0	0	0	0
F	3FE279A7	4590331D	F	1	1	_	2	0	3	FF	1	1	1	0	1E0	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	_	2	0	3	FF	1	1	1	0	300	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	_	2	0	3	FF	1	1	1	0	300	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	_	2	0	3	FF	1	1	1	0	300	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	_	2	0	3	FF	1	1	1	0	300	0	0	0	0	3	3	1	0	0	0

Microcode Table for the Tangent(x) Calculation (Concluded)

PA	DA	DB	PB	ENA	ENB	CLKC	PIPES	CLKMOD	CONFIG	SELOP	$\overline{\mathrm{R}}$ESET	$\overline{\mathrm{H}}$ALT	ENC	FLOWC	INSTR	RND	FAST	SRCC	BYTEP	SELST	TEST	SELY	$\overline{\mathrm{O}}$EY	$\overline{\mathrm{O}}$ES	$\overline{\mathrm{O}}$EC
F	00000000	00000000	F	0	0	_	2	0	3	FF	1	1	1	0	300	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	_	2	0	3	FF	1	1	1	0	300	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	_	2	0	3	FF	1	1	1	0	300	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	_	2	0	3	FF	1	1	1	0	300	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	_	2	0	3	FF	1	1	1	0	300	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	_	2	0	3	FF	1	1	1	0	300	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	_	2	0	3	FF	1	1	1	0	300	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	_	2	0	3	FF	1	1	1	0	300	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	_	2	0	3	FF	1	1	1	0	300	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	_	2	0	3	FF	1	1	1	0	300	0	0	0	0	3	3	0	0	0	0

ArcSine & ArcCosine Routine Using Chebyshev's Method

All floating point inputs and outputs are double precision. The output is in radians.

Steps Required to Perform the Calculation

STEP 1 — Preprocessing; range reduction is not needed, because an input, X, outside the range of [− 1,1] indicates an error. This routine requires that the X^2 be less than or equal to 1/2. The first operation to be performed is to square X, then multiply it by 4.0, and finally subtract 1.0.

$X1 \leftarrow X*X*4 - 1$

STEP 2 — Core Calculation; X1 in Step 1 will be referred to as 'x' in the core calculation.

$$X2 \leftarrow C_{series\_asin\&acos}$$

$$\leftarrow (((((((((((((((c_{18}*x + c_{17})*x + c_{16})*x + c_{15}*x + c_{14})*x + c_{13})*x + c_{12})*x + c_{11})*x + c_{10})*x + c_9)*x + c_8)*x + c_7)*x + c_6)*x + c_5)*x + c_4)*x + c_3)*x + c_2)*x + c_1)*x + c_0$$

STEP 3 — Postprocessing; multiply the output of the core calculation times SQRT(2.0), then multiply this product by X, the original input. This yields ArcSine(X). To calculate ArcCosine(X), the following identity is used:

$$\text{ArcCosine}(X) = \text{pi}/2 - \text{ArcSine}(X)$$

$X3 \leftarrow X2*\text{SQRT}(2.0)$
$\text{ArcSine}(X) \leftarrow X3*X$
$\text{ArcCosine}(X) \leftarrow \text{pi}/2 - \text{ArcSine}(X)$

Algorithms for the Three Steps

Step 1 perform the preprocessing:

$T1 \leftarrow X*X$
$T2 \leftarrow 4.0*T1$
$T3 \leftarrow T2 - 1$ — T3 is X1 in Step 1, the input to the core routine

Step Two perform the core calculation:

T4 ← c_{18}*CREG
T5 ← T4 + c_{17} CREG ← T3
T6 ← T5*CREG
T7 ← T6 + c_{16}
T8 ← T7*CREG
T9 ← T8 + c_{15}
T10 ← T9*CREG
T11 ← T10 + c_{14}
T12 ← T11*CREG
T13 ← T12 + c_{13}
T14 ← T13*CREG
T15 ← T14 + c_{12}
T16 ← T15*CREG
T17 ← T16 + c_{11}
T18 ← T17*CREG
T19 ← T18 + c_{10}
T20 ← T19*CREG
T21 ← T20 + c_9
T22 ← T21*CREG
T23 ← T22 + c_8
T24 ← T23*CREG
T25 ← T24 + c_7
T26 ← T25*CREG
T27 ← T26 + c_6
T28 ← T27*CREG
T29 ← T28 + c_5
T30 ← T29*CREG
T31 ← T30 + c_4
T32 ← T31*CREG
T33 ← T32 + c_3
T34 ← T33*CREG
T35 ← T34 + c_2
T36 ← T35*CREG
T37 ← T36 + c_1
T38 ← T37*CREG
T39 ← T38 + c_0

Step 3 perform the postprocessing:

T40 ← X*T39
ArcSine(X) ← T40*SQRT(2.0) SQRT(2.0) entered as a constant
ArcCosine(X) ← pi/2 − ArcSine(X)

Required System Intervention

There is no system intervention required to calculate ArcSine(X) and ArcCosine(X).

Number of 'ACT8847 Cycles Required to Calculate ArcSine(x) and ArcCosine(x)

The total number of cycles required to perform the ArcSine(x) and ArcCosine(x) calculation is 68.

Listing of the Chebyshev Constants (c's)

The constants are represented in IEEE double-precision floating point format.

c_{18} = 3DA4A49F8CCD9E73
c_{17} = 3DC05DFE52AAD200
c_{16} = 3DCCF31E26F94C8D
c_{15} = 3DE86CDA3C8CAEB0
c_{14} = 3E0768D9F4E950EA
c_{13} = 3E2383A37598FC80
c_{12} = 3E403E4B2F65F0DE
c_{11} = 3E5BAFC8245ABDF8
c_{10} = 3E77E3333AFF1AB4
c_{9} = 3E94E3A4D4220C9C
c_{8} = 3EB296DD4C084ACB
c_{7} = 3ED0E913F5F9D496
c_{6} = 3EEFA74E896F8FA8
c_{5} = 3F0EC76B7832DBB6
c_{4} = 3F2F978698C8B2E4
c_{3} = 3F519B1087542073
c_{2} = 3F7696895FFC05A0
c_{1} = 3FA375CA61D2988C
c_{0} = 3FE7B20423D1D930

Pseudocode Table for the ArcSine(x) and ArcCosine(x) Calculation

Table 59. Pseudocode for Chebyshev ArcSine and ArcCosine Routine (PIPES2-0 = 010, RND1-0 = 00)

CLK	DA BUS	DB BUS	RA REG	RB REG	CLK MODE	INSTR	MUL PIPE	ALU PIPE	P REG	C REG	S REG	Y BUS	COMMENT
1	X MSH	X LSH			0	RA2*RB2							X is the input
2	X MSH	X LSH	X	X	0	RA2*RB2							
3	4.0 MSH	4.0 LSH	X	X	0	RA4*PR4	RA2*RB2						
4			4.0	X	0	RA4*PR4			P1				
5			4.0	X	0	PR6+RB6	RA4*PR4						
6	−1.0 MSH	−1.0 LSH	4.0	−1.0	0	PR6+RB6			P2				
7	c_{18} MSH	c_{18} LSH	4.0	c_{18}	1	SR7*RB7					S1		Start core calculation
8			4.0	c_{18}	0	PR9+RB9	SR7*RB7			S1			S1 is input to core calc.
9	c_{17} MSH	c_{17} LSH	4.0	c_{17}	0	PR9+RB9			P3				
10			4.0	c_{17}	1	SR10*CR10					S2		
11			4.0	c_{17}	0	PR12+RB12	SR10*CR10						
12	c_{16} MSH	c_{16} LSH	4.0	c_{16}	0	PR12+RB12			P4				
13			4.0	c_{16}	1	SR13*CR13					S3		
14			4.0	c_{16}	0	PR15+RB15	SR13*CR13						
15	c_{15} MSH	c_{15} LSH	4.0	c_{15}	0	PR15+RB15			P5				
16			4.0	c_{15}	1	SR16*CR16					S4		
17			4.0	c_{15}	0	PR18+RB18	SR16*CR16						
18	c_{14} MSH	c_{14} LSH	4.0	c_{14}	0	PR18+RB18			P6				
19			4.0	c_{14}	1	SR19*CR19					S5		
20			4.0	c_{14}	0	PR21+RB21	SR19*CR19						
21	c_{13} MSH	c_{13} LSH	4.0	c_{13}	0	PR21+RB21			P7				
22			4.0	c_{13}	1	SR22*CR22					S6		
23			4.0	c_{13}	0	PR24+RB24	SR22*CR22						

Table 59. Pseudocode for Chebyshev ArcSine and ArcCosine Routine (PIPES2-0 = 010, RND1-0 = 00) (Continued)

CLK	DA BUS	DB BUS	RA REG	RB REG	CLK MODE	INSTR	MUL PIPE	ALU PIPE	P REG	C REG	S REG	Y BUS	COMMENT
24	c_{12} MSH	c_{12} LSH	4.0	c_{12}	0	PR24 + RB24			P8				
25			4.0	c_{12}	1	SR25 * CR25					S7		
26			4.0	c_{12}	0	PR27 + RB27	SR25 * CR25						
27	c_{11} MSH	c_{11} LSH	4.0	c_{11}	0	PR27 + RB27			P9				
28			4.0	c_{11}	1	SR28 * CR28					S8		
29			4.0	c_{11}	0	PR30 + RB30	SR28 * CR28						
30	c_{10} MSH	c_{10} LSH	4.0	c_{10}	0	PR30 + RB30			P10				
31			4.0	c_{10}	1	SR31 * CR31					S9		
32			4.0	c_{10}	0	PR33 + RB33	SR31 * CR31						
33	c_9 MSH	c_9 LSH	4.0	c_9	0	PR33 + RB33			P11				
34			4.0	c_9	1	SR34 * CR34					S10		
35			4.0	c_9	0	PR36 + RB36	SR34 * CR34						
36	c_8 MSH	c_8 LSH	4.0	c_8	0	PR36 + RB36			P12				
37			4.0	c_8	1	SR37 * CR37					S11		
38			4.0	c_8	0	PR39 + RB39	SR37 * CR37						
39	c_7 MSH	c_7 LSH	4.0	c_7	0	PR39 + RB39			P13				
40			4.0	c_7	1	SR40 * CR40					S12		
41			4.0	c_7	0	PR42 + RB42	SR40 * CR40						
42	c_6 MSH	c_6 LSH	4.0	c_6	0	PR42 + RB42			P14				
43			4.0	c_6	1	SR43 * CR43					S13		
44			4.0	c_6	0	PR45 + RB45	SR43 * CR43						
45	c_5 MSH	c_5 LSH	4.0	c_5	0	PR45 + RB45			P15				
46			4.0	c_5	1	SR46 * CR46					S14		
47			4.0	c_5	0	PR48 + RB48	SR46 * CR46						
48	c_4 MSH	c_4 LSH	4.0	c_4	0	PR48 + RB48			P16				

Table 59. Pseudocode for Chebyshev ArcSine and ArcCosine Routine (PIPES2-0 = 010, RND1-0 = 00) (Concluded)

CLK	DA BUS	DB BUS	RA REG	RB REG	CLK MODE	INSTR	MUL PIPE	ALU PIPE	P REG	C REG	S REG	Y BUS	COMMENT
49			4.0	c_4	1	SR49*CR49					S15		
50			4.0	c_4	0	PR51+RB51	SR49*CR49						
51	c_3 MSH	c_3 LSH	4.0	c_3	0	PR51+RB51			P17				
52			4.0	c_3	1	SR52*CR52					S16		
53			4.0	c_3	0	PR54+RB54	SR52*CR52						
54	c_2 MSH	c_2 LSH	4.0	c_2	0	PR54+RB54			P18				
55			4.0	c_2	1	SR55*CR55					S17		
56			4.0	c_2	0	PR57+RB57	SR55*CR55						
57	c_1 MSH	c_1 LSH	4.0	c_1	0	PR57+RB57			P19				
58			4.0	c_1	1	SR58*CR58					S18		
59			4.0	c_1	0	PR60+RB60	SR58*CR58						
60	c_0 MSH	c_0 LSH	4.0	c_0	0	PR60+RB60			P20				
61	X MSH	X LSH	4.0	X	1	SR61*RB61					S19		Begin postprocessing
62	SQRT(2) MSH	SQRT(2) LSH	4.0	X	0	RA63*PR63	SR61*RB61						SQRT(2) is the real value of square root of 2.0
63			SQRT 2	X	0	RA63*PR63			P21				
64			SQRT 2	X	0	DUMMY	RA63*PR63						Instruction is double-precision RA+RB, prevents ArcCosine from over-writing ArcSine result
66	pi/2 MSH	pi/2 LSH	SQRT 2	pi/2	1	RB66-PR66			P22			P22	Output LSH of ArcSine
67			SQRT 2	pi/2	0	NOP					S20	S20	Output MSH of ArcCosine
68			SQRT 2	pi/2	0	NOP					S20	S20	Output LSH of ArcCosine

Microcode Table for the ArcSine(x) and ArcCosine(x) Calculation

All numbers are in hex. Any field with a length that is not a multiple of 4 is right justified and zero filled. For the microcode table, the value of X has been chosen to be 1/(SQRT(2.0)).

PA	DA	DB	PB	ENAB	ECLKC	PIPES	CLKMOD	CONFIG	SELOP	$\overline{RESET}$	$\overline{HALT}$	ENC	FLOWC	INSTR	RND	FASTC	SRCTC	BYTEP	SESTT	TESTT	SELSTY	$\overline{OEY}$	$\overline{OES}$	$\overline{OEC}$
F	3FE6A09E	667F3BCD	F	0	0	2	0	3	FF	1	1	1	0	1C0	0	0	0	0	3	3	1	0	0	0
F	3FE6A09E	667F3BCD	F	1	1	2	0	3	FF	1	1	1	0	1C0	0	0	0	0	3	3	1	0	0	0
F	40100000	00000000	F	0	0	2	0	3	EF	1	1	1	0	1C0	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	1	0	2	0	3	EF	1	1	1	0	1C0	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0
F	BFF00000	00000000	F	0	1	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0
F	3DA4A49F	8CCD9E73	F	0	1	2	1	3	BF	1	1	1	0	1C0	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	2	0	3	FB	1	1	0	0	180	0	0	0	0	3	3	1	0	0	0
F	3DC05DFE	52AAD200	F	0	1	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	2	1	3	9F	1	1	1	0	1C0	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0
F	3DCCF31E	26F94C8D	F	0	1	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	2	1	3	9F	1	1	1	0	1C0	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0
F	3DE86CDA	3C8CAEB0	F	0	1	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	2	1	3	9F	1	1	1	0	1C0	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0
F	3E0768D9	F4E950EA	F	0	1	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	2	1	3	9F	1	1	1	0	1C0	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0

Microcode Table for the ArcSine(x) and ArcCosine(x) Calculation (Continued)

PA	DA	DB	PB	ENA	ENB	CLKC	PIPES	CLKMOD	CONFIG	SELOP	$\overline{RESET}$	$\overline{HALT}$	ENC	FLOWC	INSTR	RND	FAST	SRCC	BYTEP	SELST	TEST	SELY	$\overline{OEY}$	$\overline{OES}$	$\overline{OEC}$
F	3E2383A3	7598FC80	F	0	1	_	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	_	2	1	3	9F	1	1	1	0	1C0	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	_	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0
F	3E403E4B	2F65F0DE	F	0	1	_	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	_	2	1	3	9F	1	1	1	0	1C0	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	_	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0
F	3E5BAFC8	245ABDF8	F	0	1	_	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	_	2	1	3	9F	1	1	1	0	1C0	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	_	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0
F	3E77E333	3AFF1AB4	F	0	1	_	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	_	2	1	3	9F	1	1	1	0	1C0	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	_	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0
F	3E94E3A4	D4220C9C	F	0	1	_	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	_	2	1	3	9F	1	1	1	0	1C0	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	_	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0
F	3EB296DD	4C084ACB	F	0	1	_	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	_	2	1	3	9F	1	1	1	0	1C0	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	_	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0
F	3ED0E913	F5F9D496	F	0	1	_	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	_	2	1	3	9F	1	1	1	0	1C0	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	_	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0
F	3EEFA74E	896F8FA8	F	0	1	_	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	_	2	1	3	9F	1	1	1	0	1C0	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	_	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0

Microcode Table for the ArcSine(x) and ArCosine(x) Calculation (Concluded)

PA	DA	DB	PB	ENA	ENB	CLKC	PIPES	CLKMOD	CONFIG	SELOP	$\overline{\text{RESET}}$	$\overline{\text{HALT}}$	ENC	FLOWC	INSTR	RND	FASTC	SRCC	BYTEP	SSTST	TESTS	SSELY	$\overline{\text{OEY}}$	$\overline{\text{OES}}$	$\overline{\text{OEC}}$
F	3F0EC76B	7832DBB6	F	0	1	__	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	__	2	1	3	9F	1	1	1	0	1C0	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	__	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0
F	3F2F9786	98C8B2E4	F	0	1	__	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	__	2	1	3	9F	1	1	1	0	1C0	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	__	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0
F	3F519B10	87542073	F	0	1	__	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	__	2	1	3	9F	1	1	1	0	1C0	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	__	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0
F	3F769689	5FFC05A0	F	0	1	__	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	__	2	1	3	9F	1	1	1	0	1C0	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	__	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0
F	3FA375CA	61D2988C	F	0	1	__	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	__	2	1	3	9F	1	1	1	0	1C0	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	__	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0
F	3FE7B204	23D1D930	F	0	1	__	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0
F	3FE6A09E	667F3BCD	F	0	1	__	2	1	3	BF	1	1	1	0	1C0	0	0	0	0	3	3	1	0	0	0
F	3FF6A09E	667F3BCD	F	0	0	__	2	0	3	EF	1	1	1	0	1C0	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	1	0	__	2	0	3	EF	1	1	1	0	1C0	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	__	2	0	3	FF	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	__	2	0	3	FF	1	1	1	0	300	0	0	0	0	3	3	1	0	0	0
F	3FF921FB	54442D18	F	0	1	__	2	1	3	FB	1	1	1	0	183	0	0	0	0	3	3	0	0	0	0
F	00000000	00000000	F	0	0	__	2	0	3	FF	1	1	1	0	300	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	__	2	0	3	FF	1	1	1	0	300	0	0	0	0	3	3	0	0	0	0

ArcTangent Routine Using Chebyshev's Method

All floating point inputs and outputs are double precision. The output is in radians.

Steps Required to Perform the Calculation

STEP 1 – Preprocessing; If the magnitude of the input, X, is greater than 1.0, then the reciprocal must be taken. If the magnitude of X is not greater than 1.0, then pass X. Let this number (either X or 1.0/X) be referred to as X1. Next multiply X1 times 2.0, then multiply this resulting number by X1. Finally, subtract 1.0 from this last product.

If $|X| > 1.0$
  Then $X1 \leftarrow 1.0/X$
  Else $X1 \leftarrow X$
$X2 \leftarrow X1*2.0*X1 - 1.0$

STEP 2 – Core Calculation; X2 in Step 1 will be referred to as 'x' in the core calculation.

$$X3 \leftarrow C_{series\_atan}$$

$$\leftarrow ((((((((((((((((((c_{19}*x + c_{18})*x + c_{17})*x + c_{16})*x + c_{15})*x + c_{14})*x + c_{13})*x + c_{12})*x + c_{11})*x + c_{10})*x + c_9)*x + c_8)*x + c_7)*x + c_6)*x + c_5)*x + c_4)*x + c_3)*x + c_2)*x + c_1)*x + c_0$$

STEP 3 – Postprocessing; multiply the output of the core calculation times X1. Let this number be referred to as X4. The next computation will yield the answer. If X was greater than 1.0, then subtract X4 from pi/2. If X was less than –1.0, then subtract X4 from –pi/2. If neither of the two conditions above are true, then X4 is the answer.

$X4 \leftarrow X3*X1$
If $X > 1.0$
  Then ArcTangent(X) $\leftarrow$ pi/2 – X4
  Else If $X < -1.0$
    Then ArcTangent(X) $\leftarrow$ –pi/2 – X4
    Else ArcTangent(X) $\leftarrow$ X4

Algorithms for the Three Steps

Step 1 perform the preprocessing:

```
If |X| > 1.0
  Then T1 ← 1.0/X          T1 is X1 in Step 1, must be stored
       T2 ← T1*2.0         externally
       T3 ← T2*CREG        CREG ← T1
       T4 ← T3 – 1.0
  Else T1 ← X
       T2 ← T1*2.0
       T3 ← T2*T1
       T4 ← T3 – 1.0
```

Step 2 perform the core calculation:

```
T5  ←c19*CREG
T6  ←T5 + c18              CREG ← T4
T7  ←T6*CREG
T8  ←T7 + c17
T9  ←T8*CREG
T10 ←T9 + c16
T11 ←T10*CREG
T12 ←T11 + c15
T13 ←T12*CREG
T14 ←T13 + c14
T15 ←T14*CREG
T16 ←T15 + c13
T17 ←T16*CREG
T18 ←T17 + c12
T19 ←T18*CREG
T20 ←T19 + c11
T21 ←T20*CREG
T22 ←T21 + c10
T23 ←T22*CREG
T24 ←T23 + c9
T25 ←T24*CREG
T26 ←T25 + c8
T27 ←T26*CREG
T28 ←T27 + c7
T29 ←T28*CREG
T30 ←T29 + c6
```

T31 ← T30 * CREG
T32 ← T31 + c_5
T33 ← T32 * CREG
T34 ← T33 + c_4
T35 ← T34 * CREG
T36 ← T35 + c_3
T37 ← T36 * CREG
T38 ← T37 + c_2
T39 ← T38 * CREG
T40 ← T39 + c_1
T41 ← T40 * CREG
T42 ← T41 + c_0

Step 3 perform the postprocessing:

```
T43 ← T42 * T1
If X > 1.0                          CREG ← T43
     Then ArcTangent(X) ← pi/2 − CREG
     Return
If X < −1.0
Then ArcTangent(X) ← −pi/2 − CREG
     Return
ArcTangent(X) ← CREG
```

Required System Intervention

As seen in the algorithm for Step 1, the 'ACT8847 performs a compare. The results of this compare determine what kind of preproccessing is to be performed. In Step 3, there are two more compare operations. The system must therefore perform additional decision making. In addition, the system must store T1, and later (in the postprocessing) provide this value to the 'ACT8847.

Number of 'ACT8847 Cycles Required to Calculate ArcTangent(x)

Calculation of ArcTangent(x) requires at most 89 cycles (including the divide instruction). In addition, it is assumed that 15 additional cycles are required due to the compare instructions, and resulting system intervention. Therefore, the total number of cycles to perform the ArcTangent(x) calculation is 104.

Listing of the Chebyshev Constants (c's)

The constants are represented in IEEE double-precision floating point format.

c_{19} = BDC4D6CC6308553F
c_{18} = 3DDFFD56FCFD2315
c_{17} = BDE880782D99D071
c_{16} = 3E0409670CB71218
c_{15} = BE237C8239249B77
c_{14} = 3E3F1358EC1D6AC0
c_{13} = BE587CD25F4AFBED
c_{12} = 3E73D2388B0B8A86
c_{11} = BE9028E921CA6A94
c_{10} = 3EAA814997A38D4E
c_{9} = BEC5EDAD9A21FE5F
c_{8} = 3EE256E57BA07FAE
c_{7} = BEFF171F48FDF707
c_{6} = 3F1ACFA9F95CA0DF
c_{5} = BF37A8464221D994
c_{4} = 3F558DF7A83283C9
c_{3} = BF749B3E2E433683
c_{2} = 3F955A300BFB8078
c_{1} = BFBA1494C19FADD4
c_{0} = 3FEBDA7A85BD40CB

Pseudocode Table for the ArcTangent(x) Calculation

Table 60. Pseudocode for Chebyshev ArcTangent Routine (PIPES2-0 = 010, RND1-0 = 00)

CLK	DA BUS	DB BUS	RA REG	RB REG	CLK MODE	INSTR	MUL PIPE	ALU PIPE	P REG	C REG	S REG	Y BUS	COMMENT
1	1.0 MSH	1.0 LSH	0			COMPARE RA2, |RB2|							X is the input Compare 1.0 and ABS(X)
2	X MSH	X LSH	1.0	X	0	RA2*RB2 RA2, |RB2|							If ABS(X) is greater than 1.0 perform 1.0/X, otherwise go to cycle 16b
3			1.0	X	0	NOP							Wait for system response
4			1.0	X	1	DIV							Divide: 1.0/X
5			1.0	X	0	NOP							Wait for Division result
6			1.0	X	0	NOP							Wait for Division result
7			1.0	X	0	NOP							Wait for Division result
8			1.0	X	0	NOP							Wait for Division result
9			1.0	X	0	NOP							Wait for Division result
10			1.0	X	0	NOP							Wait for Division result
11			1.0	X	0	NOP							Wait for Division result
12			1.0	X	0	NOP							Wait for Division result
13			1.0	X	0	NOP							Wait for Division result
14			1.0	X	0	NOP							Wait for Division result
15			1.0	X	0	NOP							Wait for Division result
16a	2.0 MSH	2.0 LSH	1.0	X	0	RA17*PR17			P1			P1	If the reciprocal of X was performed, then execute cycles 16a through 19a
17a			2.0	X	0	RA17*PR17						P1	
18a			2.0	X	0	CR19*PR19	RA17*PR17			P1			
19a			2.0	X	0	CR19*PR19			P2a				In cycles 16a and 17 a output P1 and store it for use in cycle 79
16b	2.0 MSH	2.0 LSH	1.0	X	0	RA17*RB17							If the reciprocal of X was

Table 60. Pseudocode for Chebyshev ArcTangent Routine (PIPES2-0 = 010, RND1-0 = 00) (Continued)

CLK	DA BUS	DB BUS	RA REG	RB REG	CLK MODE	INSTR	MUL PIPE	ALU PIPE	P REG	C REG	S REG	Y BUS	COMMENT
17b			2.0	X	0	RA17*RB17							not performed, then execute
18b	X MSH	X LSH	X	X	0	RA19*PR19	RA17*RB17						cycle 16b through 19b
19b			X	X	0	RA19*PR19			P2b				
20			2.0 or X	X	0	PR21 + RB21	CR19*PR19 or RA19*PR19						The RA register is not used again until cycle 81 so rather than indicating
21	−1.0 MSH	−1.0 LSH	2.0 or X	−1.0	0	PR21 + RB21			P3a or P3b				the contents ' 2.0 of RA as: or X '
22	c_{19} MSH	c_{19} LSH	2 or X	c_{19}	1	SR22*RB22					S1		use the term ' 2 or X' Start the core calculation
23			2 or X	c_{19}	0	PR24 + RB24	SR22*RB22			S1			
24	c_{18} MSH	c_{18} LSH	2 or X	c_{18}	0	PR24 + RB24			P4				
25			2 or X	c_{18}	1	SR25*CR25					S2		
26			2 or X	c_{18}	0	PR27 + RB27	SR25*CR25						
27	c_{17} MSH	c_{17} LSH	2 or X	c_{17}	0	PR27 + RB27			P5				
28			2 or X	c_{17}	1	SR28*CR28					S3		
29			2 or X	c_{17}	0	PR30 + RB30	SR28*CR28						
30	c_{16} MSH	c_{16} LSH	2 or X	c_{16}	0	PR30 + RB30			P6				
31			2 or X	c_{16}	1	SR31*CR31					S4		
32			2 or X	c_{16}	0	PR33 + RB33	SR31*CR31						
33	c_{15} MSH	c_{15} LSH	2 or X	c_{15}	0	PR33 + RB33			P7				
34			2 or X	c_{15}	1	SR34*CR34					S5		
35			2 or X	c_{15}	0	PR36 + RB36	SR34*CR34						
36	c_{14} MSH	c_{14} LSH	2 or X	c_{14}	0	PR36 + RB36			P8				

Table 60. Pseudocode for Chebyshev ArcTangent Routine (PIPES2-0 = 010, RND1-0 = 00) (Continued)

CLK	DA BUS	DB BUS	RA REG	RB REG	CLK MODE	INSTR	MUL PIPE	ALU PIPE	P REG	C REG	S REG	Y BUS	COMMENT
37			2 or X	c14	1	SR37*CR37					S6		
38			2 or X	c_{14}	0	PR39+RB39	SR37*CR37						
39	c_{13} MSH	c_{13} LSH	2 or X	c_{13}	0	PR39+RB39			P9				
40			2 or X	c_{13}	1	SR40*CR40					S7		
41			2 or X	c_{13}	0	PR42+RB42	SR40*CR40						
42	c_{12} MSH	c_{12} LSH	2 or X	c_{12}	0	PR42+RB42			P10				
43			2 or X	c_{12}	1	SR43*CR43					S8		
44			2 or X	c_{12}	0	PR45+RB45	SR43*CR43						
45	c_{11} MSH	c_{11} LSH	2 or X	c_{11}	0	PR45+RB45			P11				
46			2 or X	c_{11}	1	SR46*CR46					S9		
47			2 or X	c_{11}	0	PR48+RB48	SR46*CR46						
48	c_{10} MSH	c_{10} LSH	2 or X	c_{10}	0	PR48+RB48			P12				
49			2 or X	c_{10}	1	SR49*CR49					S10		
50			2 or X	c_{10}	0	PR51+RB51	SR49*CR49						
51	c_9 MSH	c_9 LSH	2 or X	c_9	0	PR51+RB51			P13				
52			2 or X	c_9	1	SR52*CR52					S11		
53			2 or X	c_9	0	PR54+RB54	SR52*CR52						
54	c_8 MSH	c_8 LSH	2 or X	c_8	0	PR54+RB54			P14				
55			2 or X	c_8	1	SR55*CR55					S12		
56			2 or X	c_8	0	PR57+RB57	SR55*CR55						
57	c_7 MSH	c_7 LSH	2 or X	c_7	0	PR57+RB57			P15				
58			2 or X	c_7	1	SR58*CR58					S13		
59			2 or X	c_7	0	PR60+RB60	SR58*CR58						
60	c_6 MSH	c_6 LSH	2 or X	c_6	0	PR60+RB60			P16				

Table 60. Pseudocode for Chebyshev ArcTangent Routine (PIPES2-0 = 010, RND1-0 = 00) (Continued)

CLK	DA BUS	DB BUS	RA REG	RB REG	CLK MODE	INSTR	MUL PIPE	ALU PIPE	P REG	C REG	S REG	Y BUS	COMMENT
61			2 or X	c_6	1	SR61*CR61					S14		
62			2 or X	c_6	0	PR63+RB63	SR61*CR61						
63	c_5 MSH	c_5 LSH	2 or X	c_5	0	PR63+RB63			P17				
64			2 or X	c_5	1	SR64*CR64					S15		
65			2 or X	c_5	0	PR66+RB66	SR64*CR64						
66	c_4 MSH	c_4 LSH	2 or X	c_4	0	PR66+RB66			P18				
67			2 or X	c_4	1	SR67*CR67					S16		
68			2 or X	c_4	0	PR69+RB69	SR67*CR67						
69	c_3 MSH	c_3 LSH	2 or X	c_3	0	PR69+RB69			P19				
70			2 or X	c_3	1	SR70*CR70					S17		
71			2 or X	c_3	0	PR72+RB72	SR70*CR70						
72	c_2 MSH	c_2 LSH	2 or X	c_2	0	PR72+RB72			P20				
73			2 or X	c_2	1	SR73*CR73					S18		
74			2 or X	c_2	0	PR75+RB75	SR73*CR73						
75	c_1 MSH	c_1 LSH	2 or X	c_1	0	PR75+RB75			P21				
76			2 or X	c_1	1	SR76*CR76					S19		
77			2 or X	c_1	0	PR78+RB78	SR76*CR76						
78	c_0 MSH	c_0 LSH	2 or X	c_0	0	PR78+RB78			P22				
79	T1 MSH	T1 LSH	2 or X	T1	1	SR79*RB79					S20		T1 is either P1 or is X depending on what action was called for at cycle 2 Begin the post processing
80	X MSH	X LSH	2 or X	T1	0	COMPARE X,1.0							If X > 1.0 then execute 83 through 86, otherwise skip to 83b. In either case execute 80 through 82

Table 60. Pseudocode for Chebyshev ArcTangent Routine (PIPES2-0 = 010, RND1-0 = 00) (Concluded)

CLK	DA BUS	DB BUS	RA REG	RB REG	CLK MODE	INSTR	MUL PIPE	ALU PIPE	P REG	C REG	S REG	Y BUS	COMMENT
81	1.0 MSH	1.0 LSH	X	1.0	0	COMPARE X,1.0			P23				
82			X	1.0	0	NOP				P23			Wait for system response
83			X	1.0	0	RB84 – CR84							Execute if X > 1.0
84	pi/2 MSH	pi/2 LSH	X	pi/2	0	RB84 – CR84							
85			X	pi/2	0	NOP					S21a	S21a	Output MSH of answer
86			X	pi/2	0	NOP					S21a	S21a	Output LSH of answer The calculation is done
83b	– 1.0 MSH	– 1.0 LSH	X	1.0	0	COMPARE – 1.0,X							Execute if X ≤ 1.0. If – 1.0 > X then execute 86b through 89b, otherwise skip to 86c. In either case execute 83b thru 85b
84b	X MSH	X LSH	– 1.0	X	0	COMPARE – 1.0,X				P23			
85b			– 1.0	X	0	NOP				P23			Wait for system response
86b			– 1.0	X	0	RB87 – CR87							Execute if – 1.0 > X
87b	– pi/2 MSH	– pi/2 LSH	– 1.0	– pi/2	0	RB87 – CR87							
88b			– 1.0	– pi/2	0	NOP					S21b	S21b	Output MSH of answer
89b			– 1.0	pi/2	0	NOP					S21b	S21b	Output LSH of answer. The calculation is done.
86c			– 1.0	X	1	PASS(CR86)							Execute if X is within the range [– 1,1], Pass CREG
87c			– 1.0	X	0	NOP					S21c	S21c	Output MSH of answer
88c			– 1.0	X	0	NOP					S21c	S21c	Output LSH of answer

Microcode Table for the ArcTangent(x) Calculation

All numbers are in hex. Any field with a length that is not a multiple of 4 is right justified and zero filled. For the microcode table, the value of X has been chosen to be SQRT(3.0).

PA	DA	DB	PB	ENA	ENB	CLKC	PIPES	CLKMODE	CONFIG	SELOP	$\overline{\text{R}}$ESET	$\overline{\text{H}}$ALT	ENC	FLOWC	INSTR	RND	FAST	SRCC	BYTEP	SELST	TEST	SELY	$\overline{\text{OEY}}$	$\overline{\text{OES}}$	$\overline{\text{OEC}}$
F	3FF00000	00000000	F	0	0	_	2	0	3	FF	1	1	1	0	18A	0	0	0	0	3	3	1	0	0	0
F	3FFBB67A	E8584CAB	F	1	1	_	2	0	3	FF	1	1	1	0	18A	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	_	2	0	3	FF	1	1	1	0	300	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	_	2	1	3	FF	1	1	1	0	1E0	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	_	2	0	3	FF	1	1	1	0	300	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	_	2	0	3	FF	1	1	1	0	300	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	_	2	0	3	FF	1	1	1	0	300	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	_	2	0	3	FF	1	1	1	0	300	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	_	2	0	3	FF	1	1	1	0	300	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	_	2	0	3	FF	1	1	1	0	300	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	_	2	0	3	FF	1	1	1	0	300	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	_	2	0	3	FF	1	1	1	0	300	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	_	2	0	3	FF	1	1	1	0	300	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	_	2	0	3	FF	1	1	1	0	300	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	_	2	0	3	FF	1	1	1	0	300	0	0	0	0	3	3	1	0	0	0
F	40000000	00000000	F	0	0	_	2	0	3	EF	1	1	1	0	1C0	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	1	0	_	2	0	3	EF	1	1	1	0	1C0	0	0	0	0	3	3	0	0	0	0
F	00000000	00000000	F	0	0	_/‾	2	0	3	6F	1	1	0	0	1C0	0	0	1	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	_	2	0	3	6F	1	1	1	0	1C0	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	_	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0
F	BFF00000	00000000	F	0	1	_	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0
F	BDC4D6CC	6308553F	F	0	1	_	2	1	3	BF	1	1	1	0	1C0	0	0	0	0	3	3	1	0	0	0

Microcode Table for the ArcTangent(x) Calculation (Continued)

PA	DA	DB	PB	ENA	ENB	CLKC	PIPES	CLKMOD	CONFIG	SELOP	$\overline{RESET}$	$\overline{HALT}$	ENC	FLOWC	INSTR	RND	FASCT	SARSC	BYTEP	STEST	TESTST	SELSY	$\overline{OEY}$	$\overline{OES}$	$\overline{OEC}$
F	00000000	00000000	F	0	0	⌐	2	0	3	FB	1	1	0	0	180	0	0	0	0	3	3	1	0	0	0
F	3DDFFD56	FCFD2315	F	0	1	_	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	_	2	1	3	9F	1	1	1	0	1C0	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	_	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0
F	BDE88078	2D99D071	F	0	1	_	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	_	2	1	3	9F	1	1	1	0	1C0	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	_	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0
F	3E040967	0CB71218	F	0	1	_	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	_	2	1	3	9F	1	1	1	0	1C0	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	_	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0
F	BE237C82	39249B77	F	0	1	_	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	_	2	1	3	9F	1	1	1	0	1C0	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	_	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0
F	3E3F1358	EC1D6AC0	F	0	1	_	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	_	2	1	3	9F	1	1	1	0	1C0	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	_	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0
F	BE587CD2	5F4AFBED	F	0	1	_	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	_	2	1	3	9F	1	1	1	0	1C0	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	_	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0
F	3E73D238	8B0B8A86	F	0	1	_	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	_	2	1	3	9F	1	1	1	0	1C0	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	_	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0
F	BE9028E9	21CA6A94	F	0	1	_	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0

Microcode Table for the ArcTangent(x) Calculation (Continued)

PA	DA	DB	PB	ENA	ENB	CLKC	PIPES	CLKMOD	CONFIG	SELOP	RESET	HALT	ENC	FLOWC	INSTR	RND	FAST	SRC	BYTEP	STEST	TEST	SEL	OEY	OES	OEC
F	00000000	00000000	F	0	0	__	2	1	3	9F	1	1	1	0	1C0	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	__	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0
F	3EAA8149	97A38D4E	F	0	1	__	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	__	2	1	3	9F	1	1	1	0	1C0	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	__	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0
F	BEC5EDAD	9A21FE5F	F	0	1	__	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	__	2	1	3	9F	1	1	1	0	1C0	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	__	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0
F	3EE256E5	7BA07FAE	F	0	1	__	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	__	2	1	3	9F	1	1	1	0	1C0	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	__	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0
F	BEFF171F	48FDF707	F	0	1	__	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	__	2	1	3	9F	1	1	1	0	1C0	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	__	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0
F	3F1ACFA9	F95CA0DF	F	0	1	__	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	__	2	1	3	9F	1	1	1	0	1C0	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	__	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0
F	BF37A846	4221D994	F	0	1	__	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	__	2	1	3	9F	1	1	1	0	1C0	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	__	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0
F	3F558DF7	A83283C9	F	0	1	__	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	__	2	1	3	9F	1	1	1	0	1C0	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	__	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0
F	BF749B3E	2E433683	F	0	1	__	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0

Microcode Table for the ArcTangent(x) Calculation (Concluded)

PA	DA	DB	PB	ENA	ENB	CLKC	PIPES	CLKMOD	CONFIG	SELOP	$\overline{\text{RESET}}$	$\overline{\text{HALT}}$	ENC	FLOWC	INSTR	RND	FASTC	SRCC	BYTEP	SELST	TESTS	SELY	$\overline{\text{OE}}$	$\overline{\text{OE}}$	$\overline{\text{OE}}$
F	00000000	00000000	F	0	0	⎍	2	1	3	9F	1	1	1	0	1C0	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	⎍	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0
F	3F955A30	0BFB8078	F	0	1	⎍	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	⎍	2	1	3	9F	1	1	1	0	1C0	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	⎍	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0
F	BFBA1494	C19FADD4	F	0	1	⎍	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	⎍	2	1	3	9F	1	1	1	0	1C0	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	⎍	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0
F	3FEBDA7A	85BD40CB	F	0	1	⎍	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0
F	3FE279A7	4590331C	F	0	1	⎍	2	1	3	BF	1	1	1	0	1C0	0	0	0	0	3	3	1	0	0	0
F	3FFBB67A	E8584CAB	F	0	0	⎍	2	0	3	FF	1	1	1	0	182	0	0	0	0	3	3	1	0	0	0
F	3FF00000	00000000	F	1	1	⎍	2	0	3	FF	1	1	1	0	182	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	⌐⎍	2	0	3	FF	1	1	0	0	300	0	0	1	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	⎍	2	0	3	F7	1	1	1	0	183	0	0	0	0	3	3	1	0	0	0
F	3FF921FB	54442D18	F	0	1	⎍	2	0	3	F7	1	1	1	0	183	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	⎍	2	0	3	FF	1	1	1	0	300	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	⎍	2	0	3	FF	1	1	1	0	300	0	0	0	0	3	3	0	0	0	0

Exponential Routine Using Chebyshev's Method

All floating point inputs and outputs are double precision.

Steps Required to Perform the Calculation

STEP 1 — Preprocessing; first multiply the input, X, by $\log_2 e$ (yielding X1). Next, convert this product to an integer, using truncate mode (yielding X2). Form the variable EX by adding 1024 to X2. EX is used in the postprocessing part of the routine. Subtract 1023 from EX to find the variable N (N is actually X2 incremented by 1). Convert N to a floating point number (yielding X3). Subtract X1 from X3, multiply this difference by 2.0, and then finally subtract 1.0. This last computation is the input to the core routine.

$X1 \leftarrow X * \log_2 e$
$X2 \leftarrow TRUNC(X1)$
$EX \leftarrow 1024 + X2$
$N \leftarrow EX - 1023$
$X3 \leftarrow DOUBLE(N)$
$X4 \leftarrow 2.0*(X3 - X1) - 1.0$

STEP 2 — Core Calculation; X4 in Step 1 will be referred to as 'x' in the core calculation.

$$X5 \leftarrow C_{series\_exp}$$
$$\leftarrow ((((((((((c_{11}*x + c_{10})*x + c_9)*x + c_8)*x + c_7)*x + c_6)*x + c_5)*x + c_4)*x + c_3)*x + c_2)*x + c_1)*x + c_0$$

STEP 3 — Postprocessing; multiply the output of the core calculation times 2^N. To generate 2^N, perform the following: shift left logical 20 positions (bits) the variable EX (which was calculated in Step 1). The resulting bit pattern will be the double precision floating point representation of 2^N. However, the 'ACT8847 will not at this point recognize the bit pattern as a floating point number. So this number must be output from the Y bus, and then input (declaring the input to be a double precision floating point number) on the input bus. Now the 'ACT8847 will process 2^N as a double float, and so the core output, X5, can be multiplied by 2^N to produce the final result. 'SLL' means to shift left logical.

$X6 \leftarrow EX$ SLL by 20 bits
$Y\ bus \leftarrow X6$
$DA\ bus \leftarrow Y\ bus$
$Exp(X) \leftarrow X5 * X6$

Algorithms for the Three Steps

Step 1 perform the preprocessing:

T1 ←X*$\log_2 e$	$\log_2 e$ entered as a constant
T2 ←INT(T1)	round controls set to truncate
T3 ←1024 + T2	T3 is EX in Step 1, must be stored externally, CREG ← T1
T4 ← T3 − 1023	
T5 ←1*T4	makes T4 available to A2 MUX
T6 ←DOUBLE(T5)	convert from integer to double
T7 ←T6 − CREG	
T8 ←2.0*T7	
T9 ←T8 − 1.0	T9 is X4 in Step 1, the input to the core routine

Step 2 perform the core calculation:

T10 ←c_{11}*CREG	
T11 ←T10 + c_{10}	CREG ← T9
T12 ←T11*CREG	
T13 ←T12 + c_9	
T14 ←T13*CREG	
T15 ←T14 + c_8	
T16 ←T15*CREG	
T17 ←T16 + c_7	
T18 ←T17*CREG	
T19 ←T18 + c_6	
T20 ←T19*CREG	
T21 ←T20 + c_5	
T22 ←T21*CREG	
T23 ←T22 + c_4	
T24 ←T23*CREG	
T25 ←T24 + c_3	
T26 ←T25*CREG	
T27 ←T26 + c_2	
T28 ←T27*CREG	
T29 ←T28 + c_1	
T30 ←T29*CREG	
T31 ←T30 + c_0	

Step 3 perform the postprocessing:

T32 ← T3 SLL by 20 bits	Shift T3 20 bits left
Y bus ← T32	Output and then Input T32 CREG ← T31
DA bus ← Y bus (= T32)	Two cycles required to input both halves of T32
Exp(X) ← T32*CREG	

Required System Intervention

The system is required to store the variable EX, and then later provide this variable. In addition, the system is required to route the variable T32 (in Step 3) from the Y bus to the DA bus.

Number of 'ACT8847 Cycles Required to Calculate Exp(x)

Calculation of Exp(x) requires 52 cycles. Since there are no decisions which the system is required to perform, the total number of cycle to perform the Exp(X) calculation is 52.

Listing of the Chebyshev Constants (c's)

The constants are represented in IEEE double-precision floating point format.

c_{11} = BD45A7FC05D3B501
c_{10} = 3D957BFD2DBF487C
c_9 = BDE351B821AC16D5
c_8 = 3E2F5B0E17440879
c_7 = BE769E51EE631E87
c_6 = 3EBC8D7530548DD5
c_5 = BEFEE4FD234A4926
c_4 = 3F3BDB696E8987AC
c_3 = BF741839EB88156E
c_2 = 3FA5BE298ADF0369
c_1 = BFCF5E46537AB906
c_0 = 3FE6A09E667F3BCC

Pseudocode Table for the Exp(x) Calculation

Table 61. Pseudocode for Chebyshev Exponential Routine (PIPES2-0 = 010, RND1-0)

CLK	DA BUS	DB BUS	RA REG	RB REG	CLK MODE	INSTR	MUL PIPE	ALU PIPE	P REG	C REG	S REG	Y BUS	COMMENT
1	X MSH	X LSH			0	RA2•RB2							X is the input
2	Log_2e MSH	Log_2e LSH	X	Log_2e	0	RA2•RB2							
3			X	Log_2e	0	DP2I(PR4)	RA2•RB2						Double-precision → integer
4			X	Log_2e	0	DP2I(PR4)			P1				
5	1024		1024	Log_2e	0	RA5+SR5				P1	S1		
6	−1023		−1023	Log_2e	0	RA6+SR6					S2	S2	Store S2, which is the variable EX, for use in cycle 46
7		1	−1023	1	0	SR7•RB7					S3		
8			−1023	1	1	I2DP(PR8)			P2				Integer → double-precision
9			−1023	1	1	SR9−CR9					S4		
10	2.0 MSH	2.0 LSH	−1023	2.0	1	SR10•RB10					S5		
11			−1023	2.0	0	PR12+RB12	SR10•RB10						
12	−1.0 MSH	−1.0 LSH	−1023	−1.0	0	PR12+RB12			P3				
13	c_{11} MSH	c_{11} LSH	−1023	c_{11}	1	SR13•RB13					S6		Start core calculation, S6 is the input to the core calculation
14			−1023	c_{11}	0	PR15+RB15	SR13•RB13			S6			
15	c_{10} MSH	c_{10} LSH	−1023	c_{10}	0	PR15+RB15			P4				
16			−1023	c_{10}	1	SR16•CR16					S7		
17			−1023	c_{10}	0	PR18+RB18	SR16•CR16						
18	c_9 MSH	c_9 LSH	−1023	c_9	0	PR18+RB18			P5				
19			−1023	c_9	1	SR19•CR19					S8		
20			−1023	c_9	0	PR21+RB21	SR19•CR19						

Table 61. Pseudocode for Chebyshev Exponential Routine (PIPES2-0 = 010, RND1-0) (Continued)

CLK	DA BUS	DB BUS	RA REG	RB REG	CLK MODE	INSTR	MUL PIPE	ALU PIPE	P REG	C REG	S REG	Y BUS	COMMENT
21	c_8 MSH	c_8 LSH	−1023	c_8	0	PR21 + RB21			P6				
22			−1023	c_8	1	SR22 * CR22					S9		
23			−1023	c_8	0	PR24 + RB24	SR22 * CR22						
24	c_7 MSH	c_7 LSH	−1023	c_7	0	PR24 + RB24			P7				
25			−1023	c_7	1	SR25 * CR25					S10		
26			−1023	c_7	0	PR27 + RB27	SR25 * CR25						
27	c_6 MSH	c_6 LSH	−1023	c_6	0	PR27 + RB27			P8				
28			−1023	c_6	1	SR28 * CR28					S11		
29			−1023	c_6	0	PR30 + RB30	SR28 * CR28						
30	c_5 MSH	c_5 LSH	−1023	c_5	0	PR30 + RB30			P9				
31			−1023	c_5	1	SR31 * CR31					S12		
32			−1023	c_5	0	PR33 + RB33	SR31 * CR31						
33	c_4 MSH	c_4 LSH	−1023	c_4	0	PR33 + RB33			P10				
34			−1023	c_4	1	SR34 * CR34					S13		
35			−1023	c_4	0	PR36 + RB36	SR34 * CR34						
36	c_3 MSH	c_3 LSH	−1023	c_3	0	PR36 + RB36			P11				
37			−1023	c_3	1	SR37 * CR37					S14		
38			−1023	c_3	0	PR39 + RB39	SR37 * CR37						
39	c_2 MSH	c_2 LSH	−1023	c_2	0	PR39 + RB39			P12				
40			−1023	c_2	1	SR40 * CR40					S15		
41			−1023	c_2	0	PR42 + RB42	SR40 * CR40						
42	c_1 MSH	c_1 LSH	−1023	c_1	0	PR42 + RB42			P13				
43			−1023	c_1	1	SR43 * CR43					S16		

Table 61. Pseudocode for Chebyshev Exponential Routine (PIPES2-0 = 010, RND1-0) (Concluded)

CLK	DA BUS	DB BUS	RA REG	RB REG	CLK MODE	INSTR	MUL PIPE	ALU PIPE	P REG	C REG	S REG	Y BUS	COMMENT
44			−1023	c_1	0	PR45 + RB45	SR43 • CR43						
45	c_0 MSH	c_0 LSH	−1023	c_0	0	PR45 + RB45			P14				
46	S2	20	S2	20	0	SLL RA46,RB46							Begin post processing. S2 is the variable EX, and was calculated in cycle 5. Shift left logical S2 20 bit positions
47			S2	20	0	NOP				S17	S18	S18	Allows time for S18 to be output from the Y bus and input to the DA bus
48	S18		S2	20	0	RA48 • CR48							
49	0		S18′	20	0	RA48 • CR48							RA holds S18′, which is the double precision floating point equivalent of 2^N, where N was calculated in cycle 6
50			S18′	20	0	DUMMY	RA48 • CR48						Instruction is RA + RB, used to allow time for result to propagate to Y bus
51			S18′	20	0	NOP			P15			P15	Output MSH of answer
52			S18′	20	0	NOP			P15			P14	Output LSH of answer

Microcode Table for the Exp(x) Calculation

All numbers are in hex. Any field with a length that is not a multiple of 4 is right justified and zero filled. For the microcode table, the value of X has been chosen to be 6.25.

PA	DA	DB	PB	ENA	ENB	CLKC	PIPES	CLKMOD	CONFIG	SELOP	$\overline{\text{RESET}}$	$\overline{\text{HALT}}$	ENC	FLOWC	INSTR	RND	FASTC	SRCTEP	BYTEST	STEST	TESTST	SSELY	$\overline{\text{OEY}}$	$\overline{\text{OES}}$	$\overline{\text{OEC}}$
F	40190000	00000000	F	0	0	_	2	0	3	FF	1	1	1	0	1C0	0	0	0	0	3	3	1	0	0	0
F	3FF71547	652B82FE	F	1	1	_	2	0	3	FF	1	1	1	0	1C0	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	_	2	0	3	FB	1	1	1	0	1A3	1	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	_	2	0	3	FB	1	1	1	0	1A3	1	0	0	0	3	3	1	0	0	0
F	00000400	00000000	F	1	0	⌐	2	0	1	FE	1	1	0	0	200	0	0	1	0	3	3	1	0	0	0
F	FFFFFC01	00000000	F	1	0	_	2	0	1	FE	1	1	1	0	200	0	0	0	0	3	3	1	0	0	0
F	00000000	00000001	F	0	1	_	2	0	1	BF	1	1	1	0	240	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	_	2	1	3	FB	1	1	1	0	1A2	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	_	2	1	3	F6	1	1	1	0	183	0	0	0	0	3	3	1	0	0	0
F	40000000	00000000	F	0	1	_	2	1	3	BF	1	1	1	0	1C0	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	_	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0
F	BFF00000	00000000	F	0	1	_	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0
F	BD45A7FC	05D3B501	F	0	1	_	2	1	3	BF	1	1	1	0	1C0	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	⌐	2	0	3	FB	1	1	0	0	180	0	0	0	0	3	3	1	0	0	0
F	3D957BFD	2DBF487C	F	0	1	_	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	_	2	1	3	9F	1	1	1	0	1C0	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	_	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0
F	BDE351B8	21AC16D5	F	0	1	_	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	_	2	1	3	9F	1	1	1	0	1C0	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	_	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0
F	3E2F5B0E	17440879	F	0	1	_	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0

Microcode Table for the Exp(x) Calculation (Continued)

PA	DA	DB	PB	ENA	ENB	CLKC	PIPES	CLKMOD	CONFIG	SELOP	$\overline{RESET}$	$\overline{HALT}$	ENC	FLOWC	INSTR	RND	FAST	SRCTC	BYTEP	STEST	TESTST	SSELY	$\overline{OEY}$	$\overline{OES}$	$\overline{OEC}$
F	00000000	00000000	F	0	0	__	2	1	3	9F	1	1	1	0	1C0	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	__	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0
F	BE769E51	EE631E87	F	0	1	__	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	__	2	1	3	9F	1	1	1	0	1C0	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	__	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0
F	3EBC8D75	30548DD5	F	0	1	__	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	__	2	1	3	9F	1	1	1	0	1C0	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	__	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0
F	BEFEE4FD	234A4926	F	0	1	__	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	__	2	1	3	9F	1	1	1	0	1C0	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	__	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0
F	3F3BDB69	6E8987AC	F	0	1	__	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	__	2	1	3	9F	1	1	1	0	1C0	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	__	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0
F	BF741839	EB88156E	F	0	1	__	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	__	2	1	3	9F	1	1	1	0	1C0	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	__	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0
F	3FA5BE29	8ADF0369	F	0	1	__	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	__	2	1	3	9F	1	1	1	0	1C0	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	__	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0
F	BFCF5E46	537AB906	F	0	1	__	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	__	2	1	3	9F	1	1	1	0	1C0	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	__	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0
F	3FE6A09E	667F3BCC	F	0	1	__	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0

Microcode Table for the Exp(x) Calculation (Concluded)

PA	DA	DB	PB	ENA	ENB	CLKC	PIPES	CLKMOD	CONFIG	SELOP	$\overline{\text{RES}}$ET	$\overline{\text{HA}}$LT	ENC	FLOWC	INSTR	RND	FASCT	SRCC	BYTEP	SESST	TEST	SSELY	$\overline{\text{OE}}$Y	$\overline{\text{OE}}$S	$\overline{\text{OE}}$C
F	00000409	00000014	F	1	1	_	2	0	1	FF	1	1	1	0	228	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	1	⌐	2	0	3	FF	1	1	0	0	300	0	0	0	0	3	3	1	0	0	0
F	40900000	00000000	F	0	0	_	2	0	2	DF	1	1	1	0	1C0	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	1	0	_	2	0	2	DF	1	1	1	0	1C0	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	_	2	0	3	FF	1	1	1	0	180	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	_	2	0	3	FF	1	1	1	0	300	0	0	0	0	3	3	1	0	0	0
F	00000000	00000000	F	0	0	_	2	0	3	FF	1	1	1	0	300	0	0	0	0	3	3	0	0	0	0

High-Speed Vector Math and 3-D Graphics

Introduction

Texas Instruments SN74ACT8837 and SN74ACT8847 floating point units (FPU) are designed to execute high-speed, high-accuracy mathematical computations. The devices are especially suited for matrix manipulations such as those used in graphics or digital signal processing. These FPUs multiply and add data elements by executing sequences of microprogrammed calculations to form new matrices. Each device may be configured for either single- or double-precision operation. Single-precision operation is assumed throughout this report.

The 'ACT8847 is a functional superset of the 'ACT8837 and operates at higher clock rates (up to 33 MHz) than the 16-MHz '8837. Unlike the 'ACT8837, the 'ACT8847 can perform integer and logical operations and has built-in, hardwired algorithms for division and square root operations.

This application report outlines the timing, data flow, and programming for several common data vector calculations and matrix transformations. Further, it illustrates some of the programming "tricks" resulting in fastest operation. Throughout, this document compares the timing schemes for programs in which all registers, including the ALU and multiplier internal pipeline registers, are enabled ("pipelined" mode) with those for equivalent programs in which the internal pipeline registers are disabled ("unpiped" mode). Equations are provided to help the programmer select the more efficient mode, and performance figures are included for both devices, with times given for 15-MHz and 30-MHz operations.

This report begins by covering simple vector arithmetic operations, which are categorized as "computational" or "compare" functions for convenience. This document then compares these operations as they are used in graphics applications to perform three-dimensional coordinate transformations, perspective viewing, and clipping.

SN74ACT8837 and SN74ACT8847 Floating Point Units

Both the 'ACT8837 and 'ACT8847 floating point units (FPU) combine a multiplier and an arithmetic-logic unit (ALU) in a single microprogrammable VLSI device. These devices are implemented in TI's advanced one-micron CMOS technology and are fully compatible with the IEEE standard for binary floating point arithmetic, STD 754-1985, for either single- or double-precision operation.

Instruction inputs can select independent ALU operation, independent multiplier operation, or simultaneous ALU/multiplier operation. Each FPU can handle three types of data input formats. The ALU accepts data operands in integer format or IEEE floating

point format. In the 'ACT8837, integers are converted to normalized floating point numbers with biased exponents prior to further processing. A third type of operand, denormalized numbers, can also be processed after the ALU has converted them to "wrapped" numbers, which are explained in detail in the *SN74ACT8800 Family Data Manual.* The 'ACT8837 multiplier operates only on normalized floating point numbers or wrapped numbers. The 'ACT8847 multiplier also operates on integer operands.

Data enters the 'ACT8837 or 'ACT8847 through two 32-bit data buses, DA and DB (see Figures 74 and 75), which can be configured to operate as a single 64-bit data bus for double-precision operations. Data can be latched in a 64-bit temporary register or loaded directly into the input registers, RA and RB, which pass data to the multiplier and ALU.

A clock-mode control allows the temporary register to be clocked on the rising or falling edge of the clock to support double-precision ALU operations at the same rate as single-precision operations. Using the temporary register, double-precision numbers on a single 32-bit input bus can be loaded in one clock cycle.

The input registers RA and RB are the first of three levels of internal data registers. Additionally, the ALU and multiplier each have an internal pipeline register and an output register. The ALU's output register is denoted by "S" (sum), and the multiplier's output register is denoted by "P" (product). Any or all of these internal registers may be bypassed.

A 64-bit constant register (C) with a separate clock is provided for temporary storage of a multiplier result, ALU result, or constant for feedback to the multiplier and ALU. An instruction register and a status register are also included.

Four multiplexers select the multiplier and ALU operands from the input, C, S, or P registers. Results are output on the 32-bit Y bus; a Y output multiplexer selects the most or least significant half of the result for output.

In addition to add, subtract, and multiply functions, the 'ACT8837 can be programmed to perform floating point division using a Newton-Raphson algorithm. Absolute value conversions, floating point-to-integer and integer-to-floating point conversions, and a compare instruction are also available.

The 'ACT8847 FPU is fully compatible with IEEE Standard 754-1985 for addition, subtraction, multiplication, division, square root, and comparison. The 'ACT8847 FPU also performs integer arithmetic, logical operations, and logical shifts. Additionally, absolute value conversions and floating point-to-integer and integer-to-floating point conversions are available.

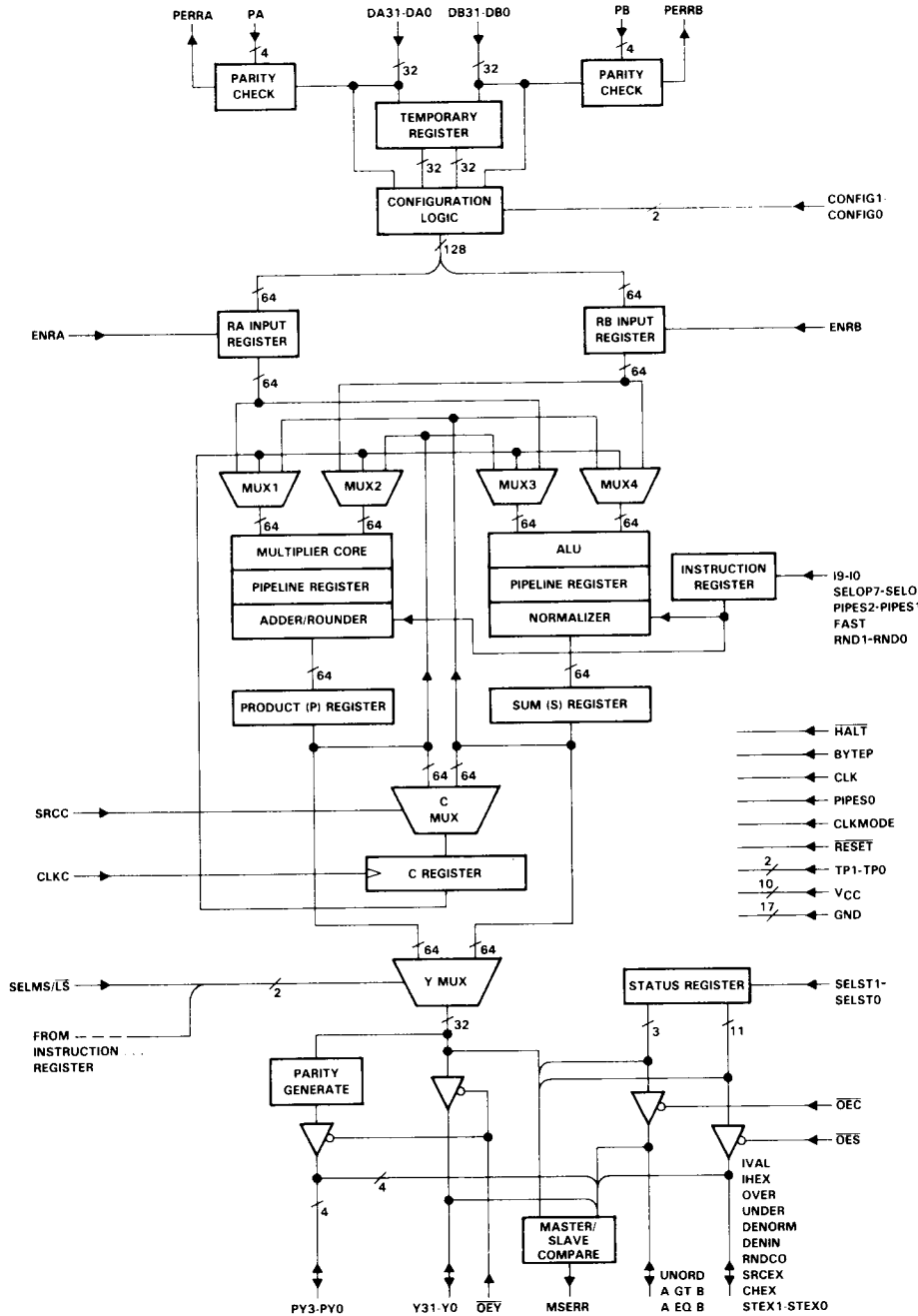


Figure 74. SN74ACT8837 Floating Point Unit

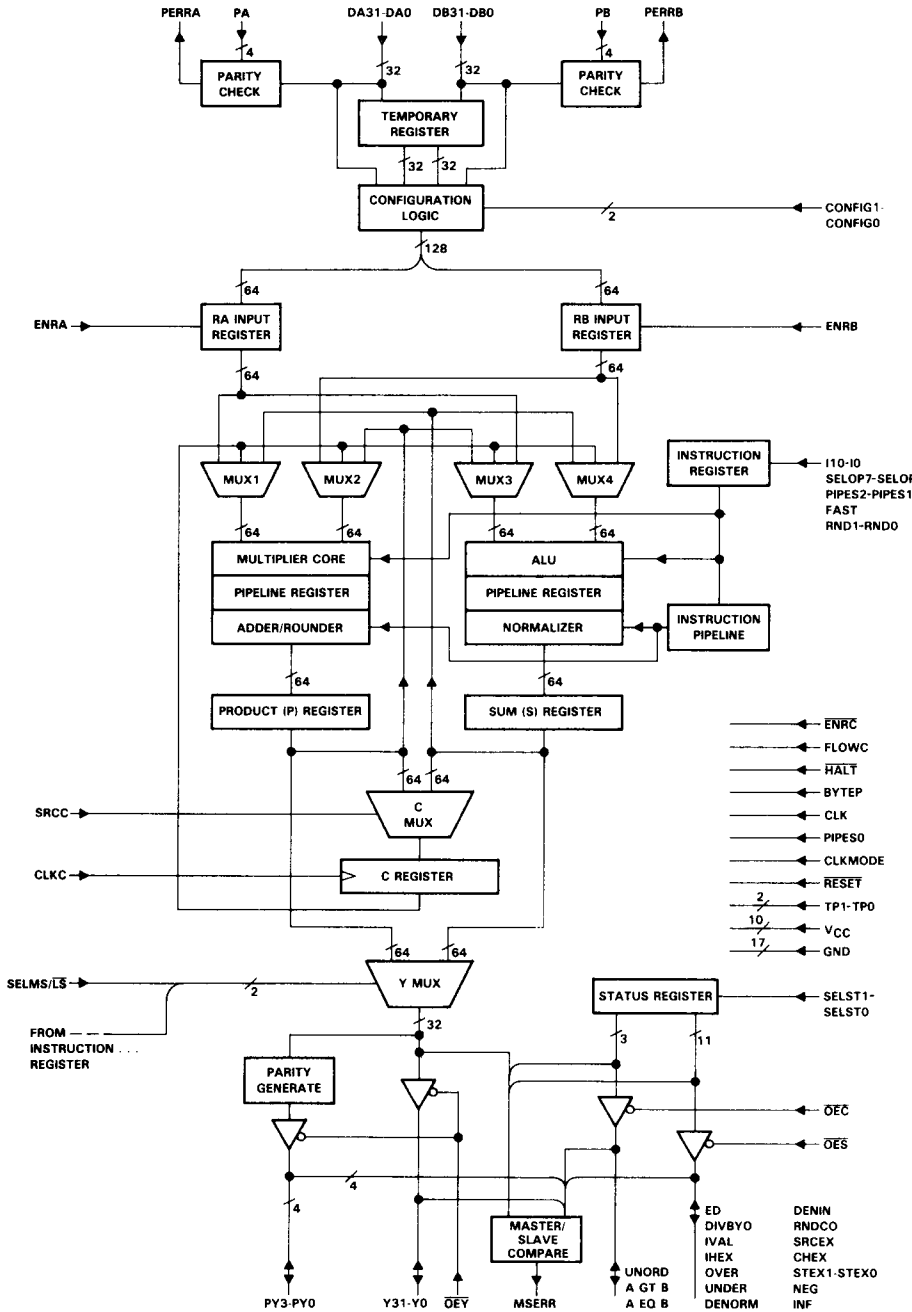


Figure 75. SN74ACT8847 Floating Point Unit

For both the 'ACT8837 and 'ACT8847, the ALU and multiplier can operate in parallel to perform sums of products and products of sums. Detailed information regarding the instruction inputs for the various 'ACT8837 and 'ACT8847 configurations and operations is given in the *SN74ACT8800 Family Data Manual*.

Mathematical Processing Applications

TI's SN74ACT8837 and SN74ACT8847 high-speed floating point units (FPU) are designed to perform high-accuracy, computationally-intensive mathematical operations. In particular, these FPUs can meet the computational demands of high-end graphics workstations and advanced signal processing. Both applications involve repetitive computations on arrays of data typically expressed as vector arithmetic operations.

For example, the calculation of the sum of products, or multiply-accumulate function, is frequently used in both signal and graphics processing. In general form, the sum of products equation is:

$$S = \sum_{i=1}^{n} k_i x_i, \text{ for coefficients } k_i \text{ and data } x_i.$$

This sum of products is the central function involved in multiplying matrices. Such matrices might represent a system of linear differential equations or the geometrical transformation of a graphic object. Specifically, an $n \times n$ matrix A multiplied by an $n \times m$ matrix B yields an $n \times m$ matrix C whose elements c_{ij} are given by:

$$c_{ij} = \sum_{k=1}^{n} a_{ik} \times b_{kj} \text{ for } i = 1,...,n \text{ and } j = 1,...,m.$$

The 'ACT8837 and 'ACT8847 are designed to handle efficiently this kind of parallel multiplication and addition.

Graphics Applications

The basic principle of graphics processing is that any object can be reduced to a combination of points, lines, and polygons and then defined as a collection of points in three-dimensional space. Because points, planes, transformation matrices and other common data structures are vectors, most of the computations involved in graphics processing are vector operations.

Computations for a 3-D graphics display are highly involved due to the complexity introduced by the z-axis. Viewing an object from a particular perspective involves transforming the object's world coordinates, or its coordinates in the model space, into viewing, or eyepoint, coordinates. A series of translations and rotations map the viewing system axes onto the world coordinate axes. Each individual point must be translated, rotated and, if necessary, scaled in a proper order. Once the coordinate transformation is complete, the coordinates are clipped to a viewing volume. Clipping algorithms employ arithmetic operations to determine whether an object, or part of an object, is inside or outside a pyramidal volume. Hidden surface routines may then be employed to delete surfaces that fall behind a "nearer" surface from the viewer's perspective.

Matrix arithmetic is required for scaling, rotating, translating, or shearing an object, as well as for the final process of projecting its visible parts to a two-dimensional frame buffer. Any sequence of these transformations can be represented as a single matrix formed by concatenating the matrices for the individual operations. The generalized 4×4 matrix for transforming a three-dimensional object is shown below, partitioned into four component matrices, each of which produces a specific effect on the image. The 3×3 matrix produces linear transformation in the form of scaling, shearing, and rotation. The 1×3 row matrix produces translation, while the 3×1 column matrix produces perspective transformation with multiple vanishing points. The final single-element 1×1 matrix produces overall scaling.

$$T = \left[\begin{array}{c|c} 3 \times 3 & 3 \times 1 \\ \hline 1 \times 3 & 1 \times 1 \end{array}\right]$$

Overall operation of the matrix T on the position vectors of a graphics object produces a combination of shearing, rotation, reflection, translation, perspective, and overall scaling.

Vector Arithmetic

Programs that require repetitive computations on multiple sets of operands lend themselves to vector-processing algorithms, in which the operands are viewed as succeeding elements of long "data vectors." The next two sections outline the programming for commonly-used vector operations. Most of these examples conclude with a comparison of program timing for pipelined (internal pipeline registers enabled) and unpiped (internal pipeline registers disabled) operation. For convenience, the operations are labeled "computational," which includes simple and compounded adds, multiplies, and divides, or "compare," which can be used to select maximum or minimum values from succeeding pairs of numbers or from a list.

Computational Operations on Data Vectors

This section covers the following vector operations: vector add, vector multiply, vector divide, sum of products (also called inner, scalar, or dot product), and product of sums. Since matrix multiplication is composed of a sequence of sum of products operations, these two functions are discussed in the same section. In some cases, a whole class of operations is covered under one heading. For example, the vector add operation includes sums and differences of A_i, B_i, $|A_i|$, and $|B_i|$ in all combinations.

Vector Add

The vector add operation adds corresponding components of data vectors to obtain the components of the output vector. Hence, for input vectors A and B and output vector Y, each with N components,

$$Y_i = A_i + B_i, \qquad 1 \leq i \leq N.$$

The 'ACT8837 and 'ACT8847 perform this calculation in unchained, independent ALU mode.

Table 62 shows the contents of the data registers at successive clock cycles for N = 6 with the FPU operating in pipelined mode. Since the data travels by way of the internal pipeline register, two cycles pass before the first sum appears in the S register. The contents of the internal pipeline register are not given in the flow.

Table 62. Data Flow for Pipelined Single-Precision Vector Add, N = 6

RA	A1	A2	A3	A4	A5	A6			
RB	B1	B2	B3	B4	B5	B6			
S			A1+B1	A2+B2	A3+B3	A4+B4	A5+B5	A6+B6	
P									
C									
Y			Y1	Y2	Y3	Y4	Y5	Y6	
CLK	1	2	3	4	5	6	7	8	9

Data transfers and operations for each clock cycle are summarized in the program listing in Table 63. Detailed information on the instruction inputs required to perform each operation is included in sections 5 and 7. Note that the selection of the output source (in this case, the S register), which is determined by the I6 instruction bit, is programmed along with the ALU or multiplier operation that generates the output.

Table 63. Program Listing for Pipelined Single-Precision Vector Add, N = 6

REGISTER TRANSFERS			ALU OPERATION	MULTIPLIER OPERATION
1.	LOAD RA, RB;	Y ← S	ADD(RA,RB)	
2.	LOAD RA, RB;	Y ← S	ADD(RA,RB)	
3.	LOAD RA, RB;	Y ← S	ADD(RA,RB)	
.				
.				
6.	LOAD RA, RB;	Y ← S	ADD(RA,RB)	

Timing and programming are similar for other independent ALU operations involving two operands, such as (A − B), (B − A), and compare (A,B). However, when the compare function is used, two status bits must be generated before numeric values can be output (see "Compare Operations on Data Vectors").

Because the vector add program closely parallels that for vector multiplication, pipelined and unpiped modes for both vector add and multiply are compared in the next section.

Vector Multiply

The vector multiply operation multiplies corresponding elements of data vectors to obtain the components of the output vector. Hence, for input vectors A and B and output vector Y, each with N components,

$$Y_i = A_i \times B_i, \quad 1 \leq i \leq N.$$

The 'ACT8837 and 'ACT8847 perform this calculation in unchained, independent multiplier mode.

Pipelined Mode

Table 64 shows the contents of the data registers at successive clock cycles for N = 6 with the FPU operating in pipelined mode. The product may be replaced by a variety of other independent multiplier operations, such as $-(A \times B)$, $A \times |B|$, $-(A \times |B|)$, $|A| \times |B|$, and $-(|A| \times |B|)$. Data transfers and operations for each clock cycle are summarized in the program listing in Table 65.

Table 64. Data Flow for Pipelined Single-Precision Vector Multiply, N = 6

RA	A1	A2	A3	A4	A5	A6			
RB	B1	B2	B3	B4	B5	B6			
S									
P			A1×B1	A2×B2	A3×B3	A4×B4	A5×B5	A6×B6	
C									
Y									
Y1			Y1	Y2	Y3	Y4	Y5	Y6	
CLK	1	2	3	4	5	6	7	8	9

Table 65. Program Listing for Pipelined Single-Precision Vector Multiply, N = 6

REGISTER TRANSFERS			ALU OPERATION	MULTIPLIER OPERATION
1.	LOAD RA, RB;	Y ← P		MULT(RA,RB)
2.	LOAD RA, RB;	Y ← P		MULT(RA,RB)
3.	LOAD RA, RB;	Y ← P		MULT(RA,RB)
.				
.				
6.	LOAD RA, RB;	Y ← P		MULT(RA,RB)

Unpiped Mode

Table 66 shows the contents of the data registers at successive clock cycles during a vector multiply operation for N = 6 with the FPU operating in unpiped mode. The vector add operation progresses similarly. Since there is no "single-clocked storage" in the internal pipeline register, each product or sum is performed in one cycle.

Table 66. Data Flow for Unpiped Single-Precision Vector Multiply, N = 6

RA	A1	A2	A3	A4	A5	A6			
RB	B1	B2	B3	B4	B5	B6			
S									
P		A1×B1	A2×B2	A3×B3	A4×B4	A5×B5	A6×B6		
C									
Y		Y1	Y2	Y3	Y4	Y5	Y6		
CLK	1	2	3	4	5	6	7	8	9

Comparison of Pipelined and Unpiped Modes

For both vector add and vector multiply operations carried out in pipelined mode, results are output to the Y bus on clocks 3, . . . , N + 2. In unpiped mode, results are output to the Y bus on clocks 2, . . . , N + 1, thereby saving a cycle. Unfortunately, it is necessary to operate at a lower clock rate in unpiped mode than in pipelined mode. The following equation can be used to determine which of the two modes provides the faster performance in a particular application. Pipelined operation is faster if:

$$(N + 2)/F_p < (N + 1)/F_u,$$

where F_p and F_u are the clock rates in pipelined and unpiped modes, respectively. As of publication, pipelined mode provides faster performance for input vectors with N > 2.

Sum of Products

The sum of products operation multiplies corresponding elements of data vectors and adds the resulting products. The operation is also referred to as the inner product, scalar product, or dot product of two vectors, since these are the names for the function as it is used in vector algebra. For input vectors A and B, each with N components, the sum of products operation yields a single output Y defined as follows:

$$Y = \sum_{i=1}^{N} (A_i \times B_i)$$

The 'ACT8837 and 'ACT8847 perform this calculation in chained mode so that concurrent operation of the ALU and multiplier is possible.

Pipelined Mode

Table 67 shows the contents of the data registers at successive clock cycles for N = 8 with the FPU operating in pipelined mode.

Table 67. Data Flow for Pipelined Single-Precision Sum of Products, N = 8

RA	A1	A2	A3	A4	A5	A6	A7	A8						
RB	B1	B2	B3	B4	B5	B6	B7	B8						
S					S1		S3	S4	S5	S6	S7	S8		S7+8
P			P1	P2	P3	P4	P5	P6	P7	P8				
C					P2	P2						S7		
Y														Y
CLK	1	2	3	4	5	6	7	8	9	10	11	12	13	14

Here, $P_i = A_i \times B_i$, $S_1 = P_1 + 0$, $S_3 = P_3 + S_1$, $S_4 = P_4 + P_2$, $S_6 = P_6 + S_4$, $S_7 = P_7 + S_5$, and $S_8 = P_8 + S_6$. The values of the sums could be more succinctly expressed as $S_i = P_i + S_{i-2}$ (with $S_0 = S_{-1} = 0$), except that $S_2 = P_2 + 0 = P_2$ does not actually appear in the data flow as a sum in the S register. Instead, the C register holds P_2 for two cycles.

This approach, although introducing a certain lack of symmetry into the programming, frees up the S register at a point allowing the efficient overlap of succeeding sum of products operations without any dead cycles. A new sum of products operation can begin at CLK 9, and the S register remains free to hold the first operation's result in CLK 14. Similary, by storing S7 in the C register in CLK 12, rather than multiplying it by one, the P register remains free to hold "P2" for the next pair of data vectors. By CLK 12, $S_7 = P_1 + P_3 + P_5 + P_7$ and $S_8 = P_2 + P_4 + P_6 + P_8$, so that $Y = S_7 + S_8$.

Data transfers and operations for each clock cycle are summarized in the program listing in Table 68.

Table 68. Program Listing for Pipelined Single-Precision Sum of Products, N = 8

REGISTER TRANSFERS			ALU OPERATION	MULTIPLIER OPERATION
1.	LOAD RA, RB			MULT(RA,RB)
2.	LOAD RA, RB			MULT(RA,RB)
3.	LOAD RA, RB		ADD(P,0)	MULT(RA,RB)
4.	LOAD RA, RB;	C← P		MULT(RA,RB)
5.	LOAD RA, RB		ADD(P,S)	MULT(RA,RB)
6.	LOAD RA, RB		ADD(P,C)	MULT(RA,RB)
7.	LOAD RA, RB		ADD(P,S)	MULT(RA,RB)
8.	LOAD RA, RB		ADD(P,S)	MULT(RA,RB)
9.			ADD(P,S)	
10.			ADD(P,S)	
11.		C←S		
12.		Y←S	ADD(S,C)	

The above algorithm imposes no delay between input vectors. The time required to carry out the sum of products operation on M pairs of input vectors in succession, each of length N, is $N \times M + 6$ cycles.

Unpiped Mode

In the unpiped version of the sum of products, the data flow is more straightforward. Again, chained mode is employed to allow the ALU and multiplier to operate concurrently. Table 69 shows the contents of the data registers at successive clock cycles for N = 8 with the FPU operating in unpiped mode. Here, $P_i = A_i \times B_i$, and $S_i = S_{(i-1)} + P_i$, with $S_0 = 0$.

Table 69. Data Flow for Unpiped Single-Precision Sum of Products, N = 8

RA	A1	A2	A3	A4	A5	A6	A7	A8						
RB	B1	B2	B3	B4	B5	B6	B7	B8						
S			S1	S2	S3	S4	S5	S6	S7	S8				
P		P1	P2	P3	P4	P5	P6	P7	P8					
C														
Y										Y				
CLK	1	2	3	4	5	6	7	8	9	10	11	12	13	14

A new problem can be presented at CLK 9 without any delay between the vectors. Therefore, the time required to compute the sums of products for M pairs of vectors, each of length N, is $N \times M + 2$ clock cycles.

Comparison of Pipelined and Unpiped Modes

The following equation can be used to determine which of the two modes provides the faster performance in a particular application. Pipelined operation is faster if:

$$(M \times N + 6)/F_p < (M \times N + 2)/F_u,$$

where F_p and F_u are the clock rates in pipelined and unpiped modes, respectively. Because the unpiped mode's longer clock cycle usually outweighs its savings in cycles, pipelined mode provides faster performance for input vectors with $N > 4$.

Product of Sums

The product of sums operation adds corresponding elements of data vectors and multiplies the resulting sums. For input vectors A and B, each with N components, the product of sums operation yields a single output Y defined as follows:

$$Y = \prod_{i=1}^{N} (A_i + B_i)$$

The product of differences can be computed by simply making the ALU operation (A – B) or (B – A). The 'ACT8837 and 'ACT8847 perform this calculation in chained mode so that concurrent operation of the ALU and multiplier is possible. The data flow and program listing for the product of sums are identical to those for the sum of products, except that the roles of add and multiply are reversed. The criteria used to decide between pipelined and unpiped modes are also identical to those previously given.

Vector Divide

The vector divide operation divides corresponding elements of data vectors to obtain the components of the output vector. Hence, for vectors A and B and output vector Y, each with N components,

$$Y_i = A_i / B_i, \quad 1 \leq i \leq N.$$

The 'ACT8837 and 'ACT8447 perform this calculation using the Newton-Raphson iterative method. This algorithm, which is described in detail in the *SN74ACT8800 Family Data Manual,* calculates the value of a quotient Y by approximating the reciprocal of the divisor B and then multiplying the dividend A by that approximation.

The following sections review the vector divide programs for the 'ACT8837 and the 'ACT8847. In the 'ACT8847, the divide algorithm is built-in.

For division using single-element inputs A and B, the value of the reciprocal of B, denoted by X, is determined iteratively using the following equation:

$$X_{i+1} = X_i (2 - B \times X_i)$$

The seed approximation, X_0, is assumed to be given. The iteration stops when X is determined to the desired level of precision. Assuming the presence of a seed ROM providing 4-bits accuracy, three iterations are necessary to correctly determine a single-precision result X. Given the seed for $1/B = X_0$, $X_{i+1} = X_i (2 - B \times X_i)$. A is eventually multiplied by the value X_3.

An 8-bit seed ROM is commonly employed and gives single-precision accuracy in only two iterations and double-precision accuracy in three iterations. Instructions for implementing an 8-bit seed ROM are included in the *SN74ACT8800 Family Data Manual*. This example assumes that a 4-bit seed is used to develop the program.

Pipelined Mode

The 'ACT8837 performs the vector divide in chained mode. Table 70 shows the data flow for pipelined operation. The value of $(2 - B \times X_i)$ is denoted as T_i. Note that the value X_3 does not appear, per se, in the table, but is expressed in terms of X_2 to save unnecessary calculations. The output Y is determined from the calculation of $(A \times X_2) \times T_2$ in cycle 17, which is equivalent to $A \times X_3$, since $X_3 = X_2 \times T_2$.

In order to keep X_i available for the final calculation of X_{i+1}, a few programming "tricks" are employed to keep the original value of each X_i within the chip while it is being altered in the calculation of $(2 - B \times X_i)$. First, X_i is stored in the S register by adding 0 to it. Then, when the S register is needed, X_i is moved to the P register by multiplying it by 1.

Table 70. Data Flow for 'ACT8837 Pipelined Single-Precision Vector Divide, N = 1

RA	X0						B			
RB	B									
S			X0		T0				X1	
P			B×X0		X0		X1		B×X1	
C										
Y										
CLK	1	2	3	4	5	6	7	8	9	10

RA			B							
RB					A					
S	T1				X2		T2			
P	X1		X2		B×X2		A×X2		Y	
C										
Y									Y	
CLK	11	12	13	14	15	16	17	18	19	20

Data transfers and operations are summarized in the program listing in Table 71. Because no operations begin on even-numbered cycles, only the odd-numbered clock cycles are shown.

Table 71. Program Listing for 'ACT8837 Pipelined Single-Precision Vector Divide, N = 1

	REGISTER TRANSFERS	ALU OPERATION	MULTIPLIER OPERATION
1.	LOAD RA, RB	ADD(RA,0)	MULT(RA,RB)
3.		ADD(2,−P)	MULT(S,1)
5.			MULT(S,P)
7.	LOAD RA	ADD(P,0)	MULT(RA,P)
9.		ADD(2,−P)	MULT(S,1)
11.			MULT(S,P)
13.	LOAD RA	ADD(P,0)	MULT(RA,P)
15.	LOAD RB	ADD(2,−P)	MULT(S,RB)
17.	Y ← P		MULT(S,P)

In steps 1, 7, and 13, 0 is added to X_i so that X_i appears two cycles later in the S register. In steps 3 and 9, the X_i value in the S register is multiplied by 1 so that it appears in the P register two cycles later. In step 15, X_i (from the S register) is multiplied by the dividend A just input to RB.

Because no operations begin on even cycles, two vector divide operations may be interleaved, calculating two quotients in 20 cycles. Table 72 shows the data flow for computing two quotients, Y_1 and Y_2, where Y_1 = A/B and Y_2 = C/D. The approximation for 1/B is denoted by W_i, and the approximation for 1/D is denoted by X_i. $T_i = (2 - B \times W_i)$, and $Q_i = (2 - D \times X_i)$.

Table 72. Data Flow for 'ACT8837 Pipelined Single-Precision Interleaved Vector Divide, N = 2

RA	W0	X0					B	D		
RB	B	D								
S			W0	X0	T0	Q0			W1	X1
P			B×W0	D×X0	W0	X0	W1	X1	B×W1	D×X1
C										
Y										
CLK	1	2	3	4	5	6	7	8	9	10

RA			B	D						
RB					A	C				
S	T1	Q1			W2	X2	T2	Q2		
P	W1	X1	W2	X2	B×W2	D×X2	A×W2	C×X2	Y1	Y2
C										
Y									Y1	Y2
CLK	11	12	13	14	15	16	17	18	19	20

The program listing for an interleaved vector divide is similar to that for a single divide operation, with functions listed in each odd line and duplicated in the next even line for the second operation.

As previously stated, the time needed to compute two single-precision divide operations starting with a 4-bit seed ROM is 20 clock cycles. Since a new pair of divides can start at CLK = 19, the time required to perform the vector divide operation on two N-dimensional vectors is given by the following equation:

$$\text{TIME} = [18 \times \text{CEILING}(N/2) + 2] \text{ cycles},$$

where the ceiling function rounds to the next highest integer for fractional values. With an 8-bit seed ROM, the time reduces to [12 × CEILING(N/2) + 2] cycles, which equals 2.5 million divides per second at 15 MHz.

Unpiped Mode

Table 73 shows the data flow for a vector divide in unpiped, chained mode.

Table 73. Data Flow for 'ACT8837 Unpiped Single-Precision Vector Divide, N = 1

RA	X0			B			B			
RB	B							A		
S		X0	T0		X1	T1		X2	T2	
P		B×X0	X0	X1	B×X1	X1	X2	B×X2	A×X2	Y
C										
Y										Y
CLK	1	2	3	4	5	6	7	8	9	10

This program uses the same methods as the pipelined version to keep X_i within the chip. The time needed to compute a vector divide of two N-element vectors is (9N + 1) cycles with a 4-bit seed ROM and (6N + 1) cycles with an 8-bit seed ROM.

Comparison of Pipelined and Unpiped Modes

Using a 4-bit seed ROM, pipelined mode is faster if:

$$[18 \times \text{CEILING}(N/2) + 2]/F_p < (9N + 1)/F_u,$$

where F_p and F_u are the clock rates in pipelined and unpiped modes. As of publication, pipelined mode provides faster performance for input vectors with N > 1.

A General Principle

The vector divide example illustrates a general programming principle that should be considered whenever a program begins a new instruction every other cycle. In cases where the C register is not used, it is simple to interleave another program, even one not performing the same function.

Interleaving programs is not as easy if the C register is used because the C register is the only nonpiped register. However, even using the C register, programs may often be interleaved by staggering one against the other so that their use of the C register does not overlap in time. Many of the programs so far discussed can be thought of as two such interleaved programs, with the C register being used to delay the first result until it can be combined with the second. (See, for example, the sum of products operation.)

SN74ACT8847 Vector Divide

Since the 'ACT8847 has a built-in algorithm for divide, the microprogram is more simple than that for the 'ACT8837. Table 74 shows the data flow for pipelined operation. Data transfers and operations are summarized in the program listing in Table 75.

Table 74. Data Flow for 'ACT8847 Pipelined Single-Precision Vector Divide

RA	A1						A2			
RB	B1						B2			
S										
P								A1/B1		
C										
Y								Y1		
CLK	1	2	3	4	5	6	7	8	9	10

Table 75. Program Listing for 'ACT8847 Pipelined Single-Precision Vector Divide

REGISTER TRANSFERS	ALU OPERATION	MULTIPLIER OPERATION
1. LOAD RA, RB; Y ← P		DIVIDE
.		.
.		.
.		.
7. LOAD RA, RB; Y ← P		DIVIDE
.		.
.		.
13. LOAD RA, RB; Y ← P		DIVIDE
.		.
.		.

Note that the microinstructions are presented on the steps indicated (1, 7, 13, . . .), with a six-cycle lapse before the next operands can be input to RA and RB. Performing a vector divide of two N-element single-precision vectors takes (6N + 2) cycles in pipelined mode. M such pairs of vectors would require [6(N × M) + 2] cycles in pipelined mode. In unpiped mode, the equation is 7(N × M).

Compare Operations on Data Vectors

In independent ALU mode (unchained), two operands may be compared for equality (A = B) and order (A > B). Additionally, the absolute values of either or both operands may be compared. The compare function uses two status bits, the AGTB and AEQB output signals. (When any operation other than a compare is performed, either by the ALU or the multiplier, the AEQB signal is used as a zero detect. Hence, numerical results cannot be output in the same cycle in which comparison status is output.)

For greatest efficiency, programs for compare operations should be written without requiring conditional branches in the sequencer. If branches can be avoided, the microcoding is simplified and the programs are immediately scalable to SIMD systems employing many 'ACT8837 or 'ACT8847 chips.

This section covers vector max/min and list max/min operations.

Vector MAX/MIN

The vector max/min operations compare corresponding elements of data vectors and select the maximum or minimum value to obtain the components of the output vector. Hence, for input vectors A and B and output vector Y, each with N components,

$$Y_i = \text{MAX/MIN}(A_i, B_i), \qquad 1 \leq i \leq N.$$

Pipelined Mode

Table 76 shows the suggested data flow for a pipelined vector MAX operation, where Y_i is set to the max of (A_i, B_i) for all i. Included are rows to indicate the setting of the chain mode instruction bit (I9 for the 'ACT8837, I10 for the 'ACT8847) and the status bit being sensed.

Table 76. Data Flow for Pipelined Single-Precision Vector MAX

CHAIN	N	Y	Y	Y	N	Y	Y	Y	N	Y
RA	A	A1		B1	A2	A2		B2	A3	A3
RB	B1				B2				B3	
S				A1		B1		A2		B2
P						A1				A2
C										
Y						Y1				Y2
STATUS			A>B				A>B			
CLK	1	2	3	4	5	6	7	8	9	10

A comparison starts at CLK = 1, 5, etc., when the chain-mode instruction bit is low. The result appears at CLK = 3, 7, etc., indicated by the AGTB and AEQB signals. AGTB is saved off-chip for use as instruction bit I6 (output source) at CLK 4, 8, etc. This value for I6 selects the output source, either the multiplier or the ALU result, at CLK 6, 10, etc. For example, if a comparison result is A > B, the AGTB signal goes high and is used to set I6 high. I6 then selects the multiplier result (A_i) to output. Similarly, if A ≤ B, AGTB and I6 are low, and the ALU result (B_i) is output. The circuitous route taken by A_i on the way to the P register is necessary because it is not possible to pass RA or RB through the multiplier in parallel with passing the other through the ALU.

The program is not particularly well-packed and produces the vector max of a pair of vectors of length N in (4N + 2) cycles. For M pairs of vectors of length N, the total time is (4MN + 2) cycles. The program can be improved by applying the interleaving principle previously discussed. The steps are rearranged so that a new operation begins every other cycle, thus allowing two compare programs to be interleaved. Table 77 shows the suggested data flow for a pipelined vector min/max operation, where Y_i = MAX/MIN(A_i, B_i) and Z_i = MAX/MIN(C_i, D_i).

Table 77. Data Flow for Pipelined Single-Precision Interleaved Vector MAX/MIN

CHAIN	N	N	Y	Y	Y	Y	N	N	Y	Y	Y	Y	N	N
RA	A1	C1	A1	C1	B1	D1	A2	C2	A2	C2	B2	D2		
RB	B1	D1					B2	D2						
S					A1	C1	B1	D1			A2	C2	B2	D2
P							A1	C1					A2	C2
C														
Y							Y1	Z1					Y2	Z2
STATUS			A>B	A>B					A>B	A>B				
CLK	1	2	3	4	5	6	7	8	9	10	11	12	13	14

Again, A_i (and C_i) reaches the P register by an indirect route. However, this tighter program performs M vector comparisons, two vector comparisons at a time, in [6 × N × CEILING(M/2) + 2] cycles. (As previously defined, the ceiling function rounds to the next highest integer for fractional values.) In this example, two separate vector

comparisons on two-dimensional vectors are performed, giving $6 \times 2 \times 1 + 2 = 14$ cycles. For M = 2 pairs of vectors, all of length N, the second program is as good as the first. For M > 2, the interleaved program performs increasingly better as M gets larger.

This second program requires more off-chip logic, since the status outputs at CLK 3 and 4 must be saved separately off-chip for use at CLK 5 and 6, respectively. This problem can easily be avoided by starting the calculations on the second pair of vectors two cycles later than shown (i.e., at CLK 4). The time necessary to perform the vector MAX operation on M pairs of N-dimensional vectors, two pairs concurrently, then increases to [6 × N × CEILING(M/2) + 4] cycles.

Data transfers and operations for the odd lines only are summarized in the program listing in Table 78. The complete program is obtained by repeating the equivalent of each odd-numbered line in the next even line for the second pair of vectors.

Table 78. Program Listing for Pipelined Single-Precision Interleaved Vector MAX/MIN

REGISTER TRANSFERS		ALU OPERATION	MULTIPLIER OPERATION
1.	LOAD RA, RB	COMPARE(RA,RB)	
3.	LOAD RA	ADD(RA,0)	
5.	LOAD RA; Y←P/S	ADD(RA,0)	MULT(S,1)

Unpiped Mode

Table 79 shows the data flow for an unpiped vector MAX operation.

Table 79. Data Flow for Unpiped Single-Precision Vector MAX

CHAIN	N	Y	Y	N	Y	Y	N	Y	Y
RA	A1	A1	B1	A2	A2	B2	A3	A3	B3
RB	B1			B2			B3		
S			A1	B1		A2	B2		A3
P				A1			A2		
C									
Y				Y1			Y2		
STATUS		A>B			A>B			A>B	
CLK	1	2	3	4	5	6	7	8	9

The status bit is saved off-chip at CLK = 2, 5, etc., and used at CLK = 3, 6, etc., as the I6 bit of the instruction. I6 selects either the multiplier or ALU result to output to the Y bus at CLK = 4, 7, etc.

The program computes the vector comparison of M pairs of vectors of length N in [3 × M × (N + 1)] cycles.

Comparison of Pipelined and Unpiped Operation

Pipelined operation is faster if:

$$[6 \times N \times \text{CEILING}(M/2) + 2]/F_p < (3 \times M \times N + 1)/F_u,$$

where F_p and F_u are the clock rates in pipelined and unpiped modes, respectively. As of publication, pipelined mode provides faster performance for $M > 1$.

List MAX/MIN

The list max/min operations select the maximum or minimum value, Z, of a list of N elements. Hence, for input vector A with N components and output Z,

$$Z = \text{MAX/MIN}(A_i), \qquad 1 \leq i \leq N.$$

List min/max is an essential operation in computer graphics because it is used to find the "extents" of a polygon or polyhedron. The extents are the maximum values of X, Y, and Z among the list of vertices for the object in question. Many forms of comparison are possible since the absolute value of either or both ALU operands may be employed. However, the example in this section assumes that the largest element of a list of N elements is desired.

Pipelined Mode

Table 80 shows the data flow for a pipelined list MAX operation,
where $M_1 = \text{MAX}(A_1, A_2)$; $M_i = \text{MAX}[M_{(i-1)}, A_{(i+1)}]$, $2 \leq i \leq N - 2$.

Table 80. Data Flow for Pipelined Single-Precision List MAX

CHAIN	Y	N	Y	Y	Y	N	Y	Y	Y	N	Y	Y	Y	Y	Y	Y
RA	A1	A1	A2			A3	A3			A4	A4					
RB		A2														
S			A1		A2				M1				M2			M3
P					A1				A3				A4			
C						M1	M1			M2	M2			M3		
Y																M3
STATUS				A>B				A>B				A>B				
CLK	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

As with vector comparison, the max/min of the absolute values is available, since the chip operates in independent ALU mode on the comparison steps. The comparison is between the RA register and the RB register in step 2 and between RA and C in steps 6, 10, etc. In these steps, the chip is switched into unchained, independent ALU mode. The status is saved off-chip and used to set the SRCC signal, which selects whether the P or S data goes into the C register in steps 5, 9, etc.

When the list max is in the C register, at CLK = 4N − 2, the C register contents must then be passed through one of the functional units to the output. The MAX/MIN of an N-element list therefore takes 4N cycles. M such vectors can be processed in [M(4N − 1) + 1] cycles.

Data transfers and operations for the list max operation are summarized in the program listing in Table 81. The program is carried out in pipelined mode, alternating between unchained and chained modes. The list max reaches the output in cycle 4N.

Table 81. Program Listing for Pipelined Single-Precision List MAX

	REGISTER TRANSFERS		ALU OPERATION	MULTIPLIER OPERATION
1.	LOAD RA		ADD(RA,0)	
2.	LOAD RA, RB		COMPARE(RA,RB)	
3.	LOAD RA		ADD(RA,0)	MULT(S,1)
4.				
5.		C ← P/S		
6.	LOAD RA		COMPARE(RA,C)	
7.	LOAD RA		ADD(C,0)	MULT(RA,1)
8.				
9.		C ← P/S		
REPEAT STEPS 6 THROUGH 9 UNTIL STEP 4N−2 IS REACHED, THEN:				
	4N − 2	Y ← S	ADD(C,0)	

Comparison of Pipelined and Unpiped Modes

The equivalent unpiped program takes [M(3N−1)+1] cycles. Pipelined mode is fastest if:

$$[M(4N - 1) + 1]/F_p < [M(3N - 1) + 1]/F_u,$$

where F_p and F_u are the clock rates in pipelined and unpiped modes, respectively. As of publication, pipelined mode provides faster performance for all M and N.

Graphics Applications

This section summarizes the concepts related to creating a three-dimensional image and examines a few of the matrix operations used in three-dimensional graphics processing. These operations include coordinate transformations and clipping operations. Additionally, this section illustrates some of the programming techniques used to perform these operations.

Creating a 3-D Image

Conceptually, translating 3-D images to 2-D display screens involves defining a view volume that limits the scope of the vista the viewer can see at one time. For simplicity, a standardized frame of reference, in which the viewer's eye is located at the origin of the coordinate system, is adopted in this example.

As illustrated in Figures 76a and 76b, the arbitrary world coordinates of the objects under scrutiny are transformed into normalized "viewing" or "eye" coordinates that reflect this frame of reference. Once the normalizing transformation is complete, the images within the view volume are projected onto a 2-D view plane, which is assumed to be located, like a projection screen, at a suitable relative distance from the viewer (see Figures 76c and 77).

A basic model for creating a 3-D view, illustrated in Figure 78a, transforms arbitrary world coordinates to normalized viewing coordinates and then "clips" the image to remove lines that do not fall within the normalized view volume. Clipping is followed by projecting the image to the 2-D projection plane (or "window"). The image is then mapped onto a canonical 2-D viewport display and from there onto the physical device.

To incorporate image transformations, another model must be adapted (see Figure 78b). After clipping, instead of projecting to the view plane, a perspective transformation is performed on the clipped viewing coordinates, transforming the view volume into a 3-D viewport, the "screen system" in which image transforms are performed. Then the image is projected to the 2-D viewport display and onto the physical device.

In both models, the clipping operation is performed on coordinates in the viewing system. This approach is referred to as "clipping in the eye system." In practice, clipping is often performed after transformation to the screen system. A trivial accept/reject test is performed on viewing coordinates, the image is transformed to the screen system, and then clipping is performed.

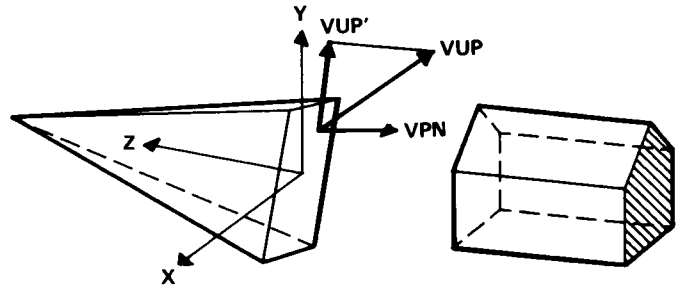


Figure 76a. In a sequence of transformations, the world coordinate positions for the house are transformed into the normalized viewing coordinate system (also called the eye system). For clarity, the house is pictured outside the view column. Also shown are the direction vectors VUP (view up), VPN (view normal), and VUP' (the projection of VUP parallel to VUN onto the view plane.

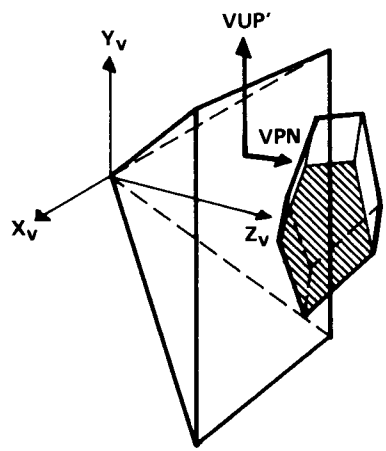


Figure 76b. After a series of translations, rotations, and shearing and scaling operations, the view volume becomes the canonical perspective projection view volume, which is a truncated pyramid with apex at the origin, and the house has been transformed from the world to the viewing coordinate system.

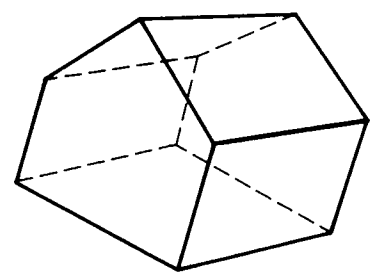


Figure 76c. This figure illustrates the projection of the house from the perspective of the viewer, with eye located at the origin of the coordinate system.

Figure 76. Creating a 3-D Image

J. D. Foley and A. Van Dam, *Fundamentals of Interactive Computer Graphics*, Addison-Wesley Publishing Company, Reading, MA, 1982, 291-293. Reprinted with permission.

The following sections illustrate programming techniques used in both of these approaches to normalizing, clipping, and transforming a 3-D image. The operations are grouped as "3-D Coordinate Transforms," "Clipping in the Eye System," and "Clipping in the Screen System."

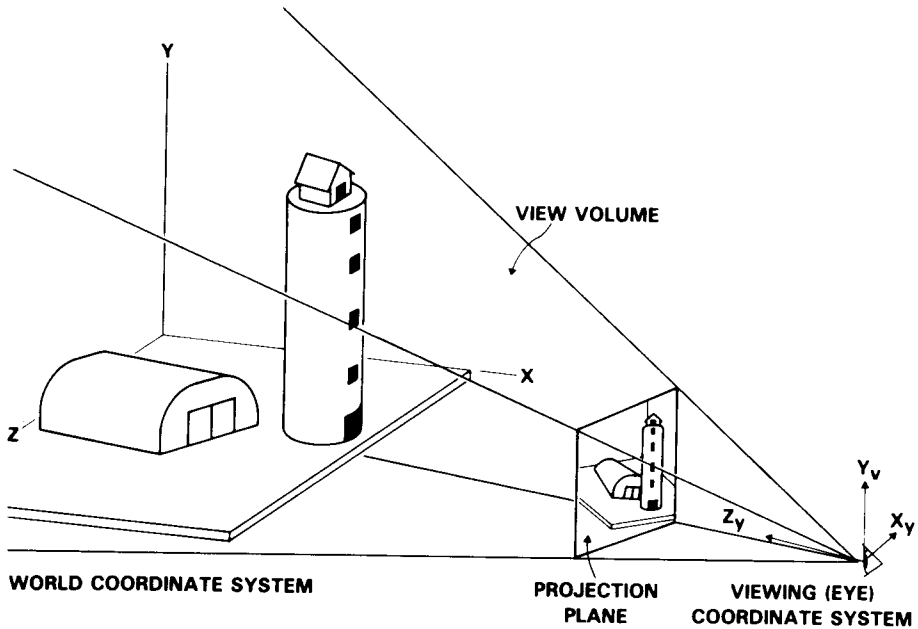


Figure 77. View Volume

Adapted with permission from a paper by Stephen R. Black entitled "Digital Processing of 3-D Data to Generate Interactive Real-Time Dynamic Pictures" from Volume 120 of the 1977 SPIE journal "Three Dimensional Imaging."

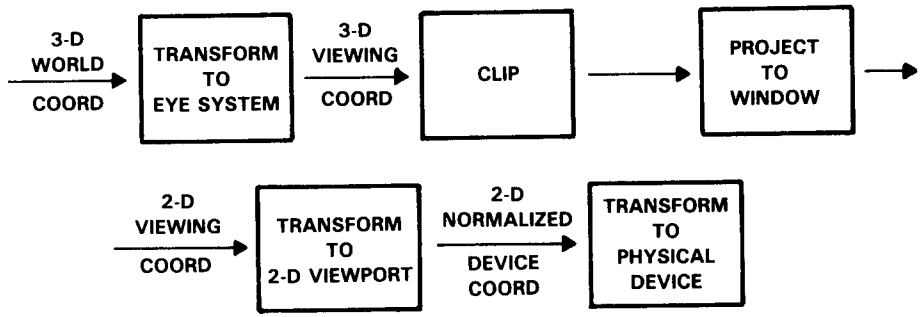


Figure 78a. Model of Procedure for Creating a 3-D Graphic

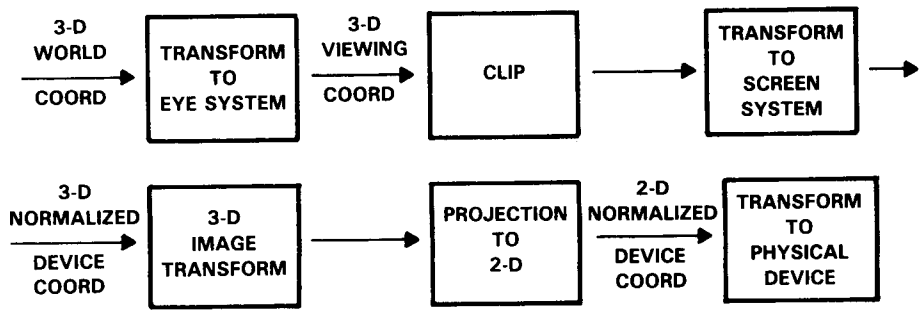


Figure 78b. Model for Creating and Transforming a 3-D Image

Three-Dimensional Coordinate Transforms

One of the computationally-intensive functions of a 3-D computer graphics system is that of transforming points within the object space, such as translating an object or rotating an object about an arbitrary axis. Equally complex is the transformation of points within the object space (or "world coordinate system") into points defined by a particular perspective and located within the viewing space (or "eye coordinate system"). This latter process, known as the viewing transformation, generates points in a left-handed cartesian system with the eye at the origin and the z-axis pointing in the direction of view. The arbitrary world-system view volume and the objects therein are translated, rotated, sheared, and scaled to match the predefined, canonical view volume of the eye system.

For a "realistic" image, the canonical view volume will be a truncated pyramid that mimics the cone of vision available to the human eye. Alternatively, the volume can be a unit cube. The series of operations that make up each transformation differ, but if homogeneous coordinates are used, either transformation can be expressed as a simple matrix multiply.

For each point (X, Y, Z) in the world system, a projection in homogeneous coordinates is denoted by (X_h, Y_h, Z_h, W_h) where,

$$(X_h, Y_h, Z_h, W_h) = (X \times W_h, Y \times W_h, Z \times W_h, W_h),$$

and W_h is simply a scale factor, typically unity when floating point numbers are used. (With fixed point values, nonunity values of W_h are used to maximize use of the numeric range.) To transform a point in homogeneous coordinates, it is post-multiplied by a 4 × 4 transform matrix:

$$[X_h', Y_h', Z_h', W_h'] = [X_h, Y_h, Z_h, W_h] \times \begin{bmatrix} A11 & A12 & A13 & A14 \\ A21 & A22 & A23 & A24 \\ A31 & A32 & A33 & A34 \\ A41 & A42 & A43 & A44 \end{bmatrix}$$

The transformed point can later be converted back to 3-space by dividing by W_h:

$$(X', Y', Z') = (X_h'/ W_h', Y_h'/ W_h', Z_h'/ W_h')$$

The transform matrix is constructed by multiplying together a sequence of matrices, each of which performs a simple task. The product of 4 or 5 elementary matrices may be used to perform some complex overall operation on a set of points representing an object or an entire scene. Once constructed, the transform matrix is used on each point of the object to be transformed.

This section describes two approaches to the viewing transformation--the general case and the specific yet typical case in which a reduced version of the transform matrix may be used. Performance times are given for 15-MHz and 30-MHz frequencies, which roughly correspond to the operating speeds of the '8837 and '8847, respectively.

Operation with General Transform Matrix

Table 82 shows part of the data flow for the pipelined and chained program for the product of the homogeneous point [X, Y, Z, W] and the 4 × 4 transform matrix A.

Table 82. Partial Data Flow for Product of [X, Y, Z, W] and General Transform Matrix

RA	X	Y	Z	W	X	Y	Z	W	X	Y
RB	A11	A21	A31	A41	A12	A22	A32	A42	A13	A23
S					S1(1)		S3(1)	S4(1)	S1(2)	T1
P			P1(1)	P2(1)	P3(1)	P4(1)	P1(2)	P2(2)	P3(2)	P4(2)
C					P2(1)	P2(1)		S3(1)	P2(2)	P2(2)
Y										X'
CLK	1	2	3	4	5	6	7	8	9	10

The technique is that already illustrated for the sum of products operation. The numbers in parentheses indicate which column of the transform matrix is involved in the operation. Here, $P1_{(i)} = X \times A_{1i}$, $P2_{(i)} = Y \times A_{2i}$, etc. $S1_{(i)} = P1_{(i)} + 0$, $S3_{(i)} = S1_{(i)} + P3_{(i)}$, $S4_{(i)} = P2_{(i)} + P4_{(i)}$, and $T_i = S3_{(i)} + S4_{(i)}$. T1 = X', T2 = Y', T3 = Z', T4 = W'. As in the sum of products illustration, in order to make the most efficient use of the S register, P2 is used directly instead of summing by 0 to form S2.

The time to transform N points in a system is 16N + 6 cycles. The system can transform approximately .94 million points per second at a clock rate of 15 MHz and 1.875 million points per second at a clock rate of 30 MHz.

Operation with the Reduced Transform Matrix and $W_h = 1$

Because viewing transformations are frequently carried out using a single-vanishing-point perspective, the 3 × 1 column that performs perspective transformations with multiple vanishing points is often not used. Additionally, with $W_h = 1$, the 1 × 1 scale factor is often equal to one. In these cases, the transform matrix takes the following form:

$$\begin{bmatrix} \cdots 0 \\ \cdots 0 \\ \cdots 0 \\ \cdots 1 \end{bmatrix}$$

With multiple vanishing points, and in other graphics operations such as clipping, 4 × 4 matrices are used with nonzero values in the fourth column. The transform matrix is termed "reduced" when its fourth column is the same as that previously shown. In such cases, the transform of each point requires only 9 multiplications and 9 additions.

Table 83 shows part of the data flow for the reduced matrix program.

Table 83. Partial Data Flow for Product of [X, Y, Z, W] and Reduced Transform Matrix

RA	X	Y	Z	×	X	Y	Z	×	X
RB	A11	A21	A31	A41	A12	A22	A32	A42	A13
S						S1(1)	S2(1)		T1
P			P1(1)	P2(1)	P3(1)		P1(2)	P2(2)	P3(2)
				P1(1)	P2(1)		S1(1)	P1(2)	P2(2)
Y									X'
CLK	1	2	3	4	5	6	7	8	9

Again, the numbers in parentheses refer to the column of the transform matrix involved in the operation. In this case, however, only the first three columns are used. Hence, for $1 \le i \le 3$, $P1_{(i)} = X \times A_{1i}$, $P2_{(i)} = Y \times A_{2i}$, etc. $S1_{(i)} = P1_{(i)} + A_{4i}$, $S2_{(i)} = P2_{(i)} + P3_{(i)}$, and $T_i = S1_{(i)} + S2_{(i)}$. T1 = X', T2 = Y', T3 = Z'. Note that W values are not calculated since they are all 1.

The time to transform N points in a system is (12N + 5) cycles. The system can transform 1.25 million points per second at 15 MHz and 2.5 million points per second at 30 MHz.

Three-Dimensional Clipping

Once an image is transformed into viewing coordinates, it must be clipped so that lines extending outside the view volume are removed. There are several approaches to clipping, some more efficient than others. This section surveys the most commonly used techniques and estimates the throughput of several single- and multi-processor arrangements.

First considered is the technique of fully clipping the line segments to fit within the viewing pyramid in the eye coordinate system. This technique is commonly referred to as "clipping before division."

Clipping in the screen system is considered second. This method eliminates lines that are obviously invisible in the eye system; the rest are clipped after projection to the screen.

Clipping in the Eye System

If an object is composed of straight line segments and a perspective view is to be taken, the viewing volume is a pyramid defined by the following plane equations:

$$X = K \times Z,\ X = -K \times Z,\ Y = K \times Z,\ Y = -K \times Z,$$

where K is a constant to be defined below. Thus, $-KZ < (X,Y) < KZ$. Two other clipping planes are usually employed at $Z = N$ and $Z = F$, where N and F are the near and far limits, respectively, of the view. This gives:

$$N < Z < F.$$

Looking in the direction of the z-axis (see Figure 79), the eye can imagine a screen located at a distance N from the eye. K is formed from the half-screen height divided by N. A specific line segment might intersect any or all of the six clipping planes. One common approach to this problem is to use six processors in a pipeline, each clipping the line to one plane.

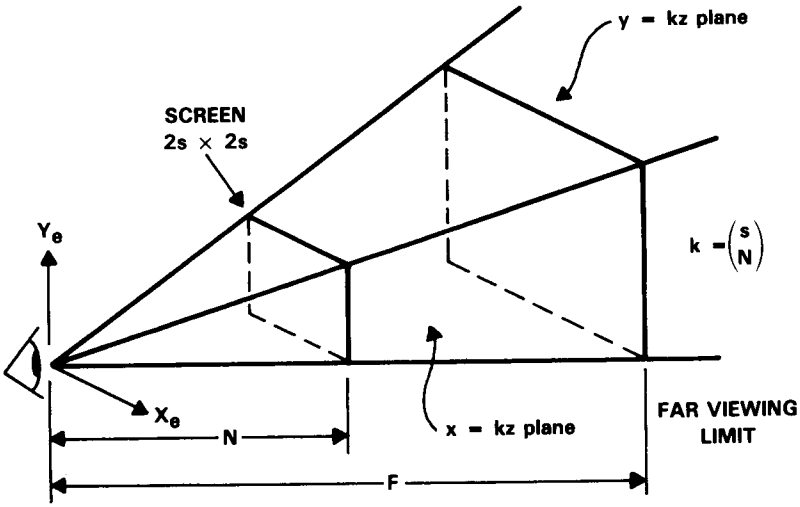


Figure 79. Viewing Pyramid Showing Six Clipping Planes

Consider the case of clipping the line defined by the points P1 = (X1, Y1, Z1) and P2 = (X2, Y2, Z2) against the Z = N plane. First computed are (Z1 − N) and (Z2 − N). If both are negative, the line is invisible, and a notation meaning an empty line is passed on. If both are positive, both ends of the line are on the visible side of the Z = N plane, and the line is passed on unclipped.

When one of these computed values is negative and the other positive, the line must be clipped and the new values for its endpoints passed down the rest of the pipeline. To do so, a parameter t that indicates what fraction of a segment Z1Z2, and therefore of P1P2 as a whole, lies on the P1 side of the Z = N plane, is computed as follows:

$$t = (Z1 - N)/(Z1 - Z2).$$

In general, the value of the parameter is derived as described in Newman and Sproull,[1] using the following equations of the line: X = X1 + (X2 − X1)u; Y = Y1 + (Y2 − Y1)u; Z = Z1 + (Z2 − Z1)u. These equations are each inserted into the corresponding plane equation. In the current example, N = Z1 + (Z2 − Z1)t.

Since N is between Z1 and Z2, t is always positive, and the signs of Z1 − N and Z2 − N are used to determine which end to clip. If Z1 − N is negative, the P1 end is clipped, using the value of t to determine the delta in X1 and Y1. The coordinates for the new endpoint of the shortened line segment are given by:

$$X1' = X1 + (X2 - X1) \times t,\ Y1' = Y1 + (Y2 - Y1) \times t,\ Z1' = N.$$

[1] Newman, W. M., and Sproull, R. F., *Principles of Interactive Computer Graphics*, McGraw-Hill, 1979.

Similarly for the case when the P2 end must be clipped:

$$X2' = X1 + (X2 - X1) \times t,\ Y2' = Y1 + (Y2 - Y1) \times t,\ Z2' = N.$$

An alternative to clipping to one plane at a time entails clipping to all six planes at once. Both approaches are examined in the following sections.

Clipping to One Plane at a Time

When a pipeline of six processors is used, each clipping the same line to one plane, each processor must wait for data from the previous processor and hold its solution until the next processor is ready to receive it. There is no reason to seek shortcuts through the computations by including branches in the program because there is little point in one of the processors completing its task earlier than the rest. This statement is true whether the six processors are driven from the same or from separate sequencers. Similarly, operating the pipeline asynchronously buys little time. Synchronous operation in the case of a clipping pipeline is likely to be almost as fast as, and much simpler and cheaper than, asynchronous operation.

Because shortcuts are not beneficial, the program can be written assuming the maximum amount of work will be required at each stage, whether the line requires clipping at that stage or not. If it is assumed that invisible lines are caught and eliminated as a separate, initial computation, branches from the clipping pipeline can be eliminated entirely. An alternative approach, in which branches would be beneficial, involves using two, three, or more 'ACT8837 or 'ACT8847 chips in parallel, rather than as a pipeline, each performing all six stages of clipping for individual lines. The program lends itself to this approach because the computations in each stage of the clipping pipeline are identical.

The method for clipping a line segment against the Z = N plane as one stage in a clipping pipeline, assuming invisible lines have been previously eliminated, will be illustrated. Two t values are computed – t_1 for clipping the P1 end of the line segment and t_2 for clipping the P2 end. If $Z1 < N$, $t_1 = (Z1 - N)/(Z1 - Z2)$; otherwise, $t_1 = 0$. If $Z2 < N$, $t_2 = (Z2 - N)/(Z1 - Z2)$; otherwise, $t_2 = 0$. The new endpoints for the line segment are computed as follows:

$$X1' = X1 + (X2 - X1) \times t_1,$$
$$Y1' = Y1 + (Y2 - Y1) \times t_1,$$
$$Z1' = Z1 + (Z2 - Z1) \times t_1$$

$$X2' = X2 - (X2 - X1) \times t_2,$$
$$Y2' = Y2 - (Y2 - Y1) \times t_2,$$
$$Z2' = Z2 - (Z2 - Z1) \times t_2.$$

Note that the denominator is the same in the equations for t_1 and t_2; it is this reciprocal computation that is expensive in time. However, in the 'ACT8837, it is also simple to interleave other computations with that of the reciprocal, and in the '8847, the built-in divide is very fast.

A simple trick is used to compute the t_i values in a streamlined fashion. $H_i = (Z_i - N)$ is first computed, followed by the sum $H_i' = H_i - |H_i|$. Note that if $(Zi - N)$ is negative, $H_i' = 2H_i = 2(Zi - N)$; otherwise, $H_i' = 0$. Hence, in a straightforward manner, a suitable numerator for t_i has been computed, regardless of the sign of $(Zi - N)$. This approach avoids resorting to an "if/then" decision to compute t_i.

To scale the denominator to the numerator, $D = 2(Z1 - Z2)$ is computed, and the Newton-Raphson algorithm in the '8837 or the built-in divide instruction in the '8847 is used to determine the values of 1/D, $t_1 = |H_1'/ D|$, and $t_2 = |H_2'/ D|$. New values of (X1, Y1, Z2) and (X2, Y2, Z2) are then computed using t_1 and t_2.

The data flow and program listing for the clipping against Z = N operation as performed on the 'ACT8837 are given in Tables 84 and 85. Here, $t_1 = |(H_i - |H_i|)/D|$. Also, $d = Z1 - Z2$, $H_1 = Z1 - N$, $H_1' = H_1 - |H_1|$, $H_2 = H_2 - |H_2|$, R_i = successive approximations for 1 / d, $T_i = (2 - d \times R_i)$, and $R_{(i+1)} = Ti \times R_i$.

Table 84. Data Flow for Clipping a Line Segment Against the Z = N Plane Using the SN74ACT8837

CHAIN	Y	Y	Y	Y	N	Y	N	Y	Y	Y	Y	Y	Y	Y		N	N
RA	Z1	Z1	Z2	R0				X2	Y2	d						H1'	H2'
RB	Z2	N	N	d				X1	Y1			O−S					
S			d	H1	H2	R0	H1'	T0	H2'	X2−X1	Y2−Y1	R1		T1			
P						d×R0		R0		R1		d×R1		O−S R1		1/D	
C					H1	H2	H2										1/D
Y			d				H1'		H2'	X2−X1	Y2−Y1						
STATUS																	
CLK	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

CHAIN	Y	Y	Y	Y	Y	Y	Y	Y	Y			
RA	(X2−X1)	(Y2−Y1)	(Z2−Z1)		(X2−X1)	(Y2−Y1)	(Z2−Z1)					
RB			X1	Y1	Z1		X1	Y1	Z1			
S				t2	X1'	Y1'	Z1'		X2'	Z2'	Z2'	
P	t1	t2	(X2−X1)	(Y2−Y1)	(Z2−Z1)		(X2−X1)	(Y2−Y1)	(Z2−Z1)			
			×t1	×t1	×t1		×t2	×t2	×t2			
C		t1	t1	t2	t2	t2						
Y					X1'	Y1'	Z1'		X2'	Y2'	Z2'	
STATUS												
CLK	18	19	20	21	22	23	24	25	26	27	28	

Table 85. Program Listing for Clipping a Line Segment Against the Z = N Plane Using the SN74ACT8837

	REGISTER TRANSFERS			ALU OPERATION	MULTIPLIER OPERATION
1.	LOAD RA, RB	Y←S		ADD (RA,−RB)	
2.	LOAD RA, RB			ADD (RA,−RB)	
3.	LOAD RA, RB			ADD (RA,−RB)	
4.	LOAD RA, RB;		C←S	ADD (RA,O)	MULT(RA,RB)
5.	LOAD RA, RB	Y←S	C←S	ADD (C,−|C|)	
6.				ADD (2,−P)	MULT(S,I)
7.		Y←S		ADD (C,−|C|)	
8.	LOAD RA, RB	Y←S		ADD (RA,−RB)	MULT(S,P)
9.		Y←S		ADD (RA,−RB)	
10.	LOAD RA			ADD (P,O)	MULT(RA,P)
11.					
12.	LOAD RB			ADD (2,−P)	MULT(S,RB)
13.					
14.					MULT(S,P)
15.					
16.	LOAD RA		C←P		MULT(|RA|,|P|)
17.	LOAD RA				MULT(|RA|,|C|)
18.	LOAD RA		C←P		MULT(RA,P)
19.	LOAD RA			ADD (P,O)	MULT(RA,C)
20.	LOAD RA, RB	Y←S		ADD (P,RB)	MULT(RA,C)
21.	LOAD RA, RB	Y←S	C←S	ADD (P,RB)	
22.	LOAD RA, RB	Y←S		ADD (P,RB)	MULT(RA,C)
23.	LOAD RA, RB				MULT(RA,C)
24.	LOAD RB	Y←S		ADD (P,RB)	MULT(RA,C)
25.	LOAD RB	Y←S		ADD (P,RB)	
26.		Y←S		ADD (P,RB)	
27.					
28.					

In pipelined mode, computing (Z1 − Z2) takes 2 cycles. This value is passed off-chip and used to get the first approximation to 0.5/(Z1 − Z2) from an 8-bit seed ROM. Iteration to correctly determine the value begins in the 4th cycle, with subsequent operations starting on even-numbered cycles. The computations of H1' and H2' are interleaved with the divide algorithm and are completed before it.

(X2 − X1), (Y2 − Y1), and (Z2 − Z1) are also computed during the divide. The values of t_1 and t_2 are ready in steps 18 and 19. New values of X1, X2, Y1, Y2, Z1, and Z2 are all computed and output by step 28. Each chip, therefore, clips against one clipping plane in 28 cycles. With a two-cycle overlap, the next line segment can be presented in cycle 26.

For the two X and two Y clipping planes, the calculations are slightly more complicated. For the X = KZ plane, the two parameters t_i are defined in terms of the values $W_1 = KZ_1$, $W_2 = KZ_2$ and $H_1 = W_1 - X_1$, $H_2 = W_2 - X_2$ as follows:

$$t_1 = |H_1'/2(H_1 - H_2)| \text{ and } t_2 = |H_2'/2(H_1 - H_2)|,$$

where, as before, $H_i' = H_i - |H_i|$. The equations for the new endpoints, (X1', Y1', Z1') and (X2', Y2', Z2'), are the same as before. It is still possible to compute the new endpoints in under 30 cycles. At 15 MHz, a six-chip '8837 system would clip 577,000 line segments per second.

In the '8847 a similar process is employed, but the built-in divide instruction is used beginning in step 7 and ending in step 15. t_1 and t_2 are calculated by step 18, and the entire operation completes in step 27, one cycle shorter than for the '8837. The data flow is shown in Table 86. A six-processor '8847 system operating at 30 MHz would clip 1.2 million line segments per second with a new operation beginning every 25 cycles.

Table 86. Data Flow for Clipping a Line Segment Against the Z = N Plane Using the SN74ACT8847

RA	Z1	Z1	Z2	X2				0.5	Y2	H1'	H2'	SAME AS FOR '8837		
RB	Z2	N	N	X1				d	Y1					
S			d	H1	H2	X2−X1	H1'	H2'			Y2−Y1			
P										1/D		t1	t2	STEPS 20 THRU 28
C					H1	H2					1/D		t1	
Y			d			X2−X1	H1'	H2'			Y2−Y1			
STATUS														
CLK	1	2	3	4	5	6	7	8	14	15	16	17	18	

Since the performance levels obtained from the six-chip systems described below are slower than the rate of endpoint transformation by a single-chip system, some further speed improvement is desirable. Hence, rather than going through the code for clipping to the X and Y planes, another approach is proposed.

Clipping to All Six Planes at a Time

The "window edge clipping method" derived in Newman and Sproull can be used to clip to all six planes at once. Recall that the viewing volume for a perspective view is a pyramid defined by the following plane equations:

$$X = K \times Z,\ X = -K \times Z,\ Y = K \times Z,\ Y = -K \times Z,\ Z = N,\ Z = F,$$

where K = S/N, as defined in a previous section. Given a segment with endpoints P1 = (X1, Y1, Z1) and P2 = (X2, Y2, Z2), to perform the entire clipping operation on all six planes at once, the following two six-tuples must be computed:

$$Q = (W1+X1, W1-X1, W1+Y1, W1-Y1, Z1-N, F-Z1) = (Q1, Q2,. . .),$$
$$R = (W2+X2, W2-X2, W2+Y2, W2-Y2, Z2-N, F-Z2) = (R1, R2,. . .),$$

where $W_1 = KZ_1$ and $W_2 = KZ_2$.

Consider the case where $X1 < -W1$. Then, $W1 + X1 < 0$; i.e., $Q1 < 0$. In general, a negative element of Q indicates that P1 is on the invisible side of one of the clipping planes, while a negative element of R indicates the same for P2. To clip the line, the six parameters t_i for clipping the P1 end and the six parameters s_i for clipping the P2 end are computed. Here, $t_i20=20Q_i/(Q_i - R_i)$ and $s_i = R_i/(R_i-Q_i)$. (Again, the equations of the line as described in Newman and Sproull are used).

For example, to find the value t_1 for clipping P1 to the $X = -W = -KZ$ plane, the following equation is used:

$$X1 + (X2 - X1)t_1 = -K[Z1 + (Z2 - Z1)t_1].$$

Solving for t_1,

$$t_1 = (X1 + W1)/[(X1 + W1)-(X2 + W2)] = Q1 / (Q1 - R1).$$

In general, $t_i = Q_i/(Q_i - R_i)$. Similarly, $s_i=R_i/(R_i-Q_i)$.

To actually carry out the computations of t_i and s_i, the trick discussed above is performed, and each element of Q and R is replaced with the difference of the element and its absolute value, to form Q' and R'. That is,

$Q_i' = 2 \times Q_i$ if $Q_i < 0$, and $Q_i' = 0$ otherwise.
$R_i' = 2 \times R_i$ if $R_i < 0$, and $R_i' = 0$ otherwise.

Next calculated is $t_i = Q_i'/[2(Q_i - R_i)]$ and $s_i = R_i'/[2(R_i-Q_i)]$, followed by $T1 = MAX(t_i)$ and $T2 = 1 - MAX(s_i)$. The P1 end is clipped using T1 and the P2 end is clipped using T2.

In an '8837 three-processor parallel system, in which each processor is given the task of computing two t_i and two s_i values, computing the Q_i' and R_i' values takes 14 cycles, with the values of $Q_i - R_i$ computed by step 13. The six divides, $0.5/(Q_i - R_i)$, are completed in step 30, assuming an 8-bit seed ROM is used. The max/min operations take place in parallel in two processors and complete at step 54 (24 + 30), and the new endpoints are ready by step 60 (6 + 54). The timing is the same using the '8847.

The data flow and program listing for computing t_1, t_2, s_1, and s_2 by one of the three '8837 processors is given in Tables 87 and 88.

Table 87. Data Flow for Computing t_1, t_2, s_1, and s_2 Using an SN74ACT8837

CHAIN	Y	Y	Y	Y	Y	N	N	N	Y	Y
RA	K	K							W2	Q1
RB	Z1	Z2	X1	X1	X2				X2	R1
S					Q1	Q2	R1	Q1'	Q2'	R1'
P			W1	W2						
C				W1	W2	Q1	Q2	R1		
Y								Q1'	Q2'	R1'
STATUS										
CLK	1	2	3	4	5	6	7	8	9	10
CHAIN	Y	N	Y	Y	Y	Y	Y	Y	Y	Y
RA	Q2		R0							
RB	R2		d1							
S	R2	Q1−R1	Q2−R2	R2'	R0		T0			
P					d×R0		R0		R1	
C		R2								
Y		Q1−R1	Q2−R2	R2'						
STATUS										
CLK	11	12	13	14	15	16	17	18	19	20
CHAIN	Y	Y	Y	Y	Y	Y	Y	Y	N	N
RA					Q1'	Q2'	R1'			
RB	O−S							R2'		
S	R1		T1					1/D2		
P	d×R1		O−SR1		1/D1	1/D2	t1	t2	S1	S2
C						1/D1	1/D1			
Y							t1	t2	S1	S2
STATUS										
CLK	21	22	23	24	25	26	27	28	29	30

NOTE: Cycles 13, 15, 17, 19, . . . ,25 compute 1/D1 = 0.5/d1;
Cycles 14, 16, 18, 20, . . . ,26 compute 1/D2 = 0.5/d2, di = Qi − Ri.

Table 88. Program Listing for Three-Processor Clip to Compute t_1, t_2, s_1, and s_2 Only

	REGISTER TRANSFERS			ALU OPERATION	MULTIPLIER OPERATION
1.	LOAD RA, RB				MULT (RA,RB)
2.	LOAD RA, RB				MULT (RA,RB)
3.	LOAD RB		C←P	ADD (P,RB)	
4.	LOAD RB		C←P	ADD (C, −RB)	
5.	LOAD RB		C←S	ADD (C,RB)	
6.		Y←S	C←S	ADD (C,−|C|)	
7.		Y←S	C←S	ADD (C,−|C|)	
8.		Y←S		ADD (C,−|C|)	
9.	LOAD RA, RB			ADD (RA,−RB)	
10.	LOAD RA, RB	Y←S		ADD (RA,−RB)	
11.	LOAD RB, RB	Y←S	C←S	ADD (RA,−RB)	
12.		Y←S		ADD (C,−|C|)	
	CODE FOR TWO DIVISIONS				
25.	LOAD RA	Y←S	C←P		MULT (RA,P)
26.	LOAD RA	Y←S		ADD (P,O)	MULT (RA,P)
27.		Y←S			MULT (RA,C)
28.		Y←S			MULT (S,RB)

This approach facilitates the transform of 288,000 line segments per second in a 3-chip '8837 system running at 15 MHz and 576,000 line segments in an '8847 system running at 30 MHz. If branches are permitted in the sequencer, a considerable speedup is available for situations in which a large proportion of line segments are either invisible, and may be eliminated, or are completely visible, and may be passed without clipping. A single-processor system takes no more than 32 cycles, sometimes as few as 10 cycles, to reject an invisible line, whereas it takes 91 cycles to process lines that need both ends clipped. Hence, in a situation where 50% of the line segments are invisible, the speed is in excess of 360,000 line segments per second at 20 MHz and 540,000 segments/second at 30 MHz. It is not uncommon for 80% of lines to be invisible, in which case the speed would increase to 584,000 line segments at 20 MHz and 877,000 line segments at 30 MHz.

To take advantage of this speedup, the only change in the sequence given above is that while computing Q and R, the logical AND and OR is formed for the signs of the corresponding pairs of values, Q_i and R_i. This is best performed off-chip if the '8837 is being used but may be done using independent ALU (unchained) mode in the '8837 or a logical operation in the '8847. For the '8837, with two operands Q_i and R_i, Table 89 shows the A > B status bit for an A > B comparison on $A = -Q_i \times |R_i|$ and $B = |Q_i| \times R_i$ for all signs of Q_i and R_i.

Table 89. A > B Comparison Function Table

Sign Q_I	Sign R_I	Sign A = $-Q_I \times \lvert R_I \rvert$	Sign B = $\lvert Q_I \rvert \times R_I$	A > B	A = B
−	−	+	−	T	F
−	+	+	+	F	T
+	−	−	−	F	T
+	+	−	+	F	F

The A > B status provides the needed AND function of the sign bits of Q_i and R_i. In computing these A > B values, if A > B is TRUE, the sequencer branches to code that rejects the line as invisible. A comparison A > B of A = ($Q_i \times |R_i|$) and B = ($|Q_i| \times R_i$) gives the logical AND of the complement of the sign bits. It is TRUE when both Q_i and R_i are positive. If all six values are TRUE, the sequencer can branch to code that passes the line segment unclipped.

For a three-processor parallel system, lockstep operation with a single sequencer is still possible since all three processors are working on the same line segment, and the branch options apply equally to them all. The estimated time for a three-processor system is 56 cycles; not much interleaving is possible.

Now that the operations have been reduced to a minimum, the remaining steps are necessarily sequential. Rejecting invisible or passing totally visible line segments without division, however, is still beneficial.

Clipping in the Screen System

In most graphics systems, full line clipping is not performed in the eye system. Instead, a trivial accept/reject test is performed, in which the line segments are simply tested against the six clipping planes. If a line has both ends on the invisible side of any one of the clipping planes, it is rejected. Lines surviving this test may still be outside the viewing pyramid. In any case, the lines are transformed to the screen coordinate system and then clipped against a cube defined by the simple plane equations $-1 < (X, Y, Z) < 1$. The next three sections describe this process.

Trivial Accept/Reject Test

In the eye system, the clipping planes are:

$$X = W,\ X = -W,\ Y = W,\ Y = -W,\ Z = N, \text{ and } Z = F,$$

where W = K × Z. After −W1 and −W2 are computed, a sequence of comparison operations are performed, summarized as follows:

with X1 in RB and −W1 in P,	P > RB (i.e., −W1 > X1)
with X1 in RA and −W1 in C,	RA > |C| (i.e., X1 > W1)
with Y1 in RB and −W1 in C,	C > RB
with Y1 in RA,	RA > |C| comparison
with Z1 in RB and N in RA,	RA > RB (i.e. N > Z1)
with Z1 in RA and F in RB,	RA > RB (i.e., Z1 > F).

These six operations are carried out in successive cycles and then repeated for (X2, Y2, Z2). The two six-tuples are saved off-chip and a bit-wise AND is carried out. If any one of the resulting six boolean values is TRUE, the line is rejected. This entire operation takes only 16 cycles, thereby providing a speed of 1,071,000 line segments per second at 15 MHz and 2,143,000 line segments per second at 30 MHz. The data flow for an accept/reject test is given in Table 90. Accept/reject testing of individual points takes only 8 cycles.

Table 90. Data Flow for Accept/Reject Testing

CHAIN	N	N	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	N	N
RA	K	K		X1		Y1	N	Z1	−W2	X2	−W2	Y1	N	Z2		
RB	Z1	Z2	X1		Y1		Z1	F	X2	−W2	Y1	−W2	Z2	F		
S																
P			−W1	−W2												
C			−W1	−W1	−W1	−W1										
Y			−W2													
STATUS				−W1 > X1	X1 > W1	−W1 > Y1	Y1 > W1	N > Z1	Z1 > F	−W2 > X2	X2 > W2	−W2 > Y2	Y2 > W2	N > Z2	Z2 > F	
CLK	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

Transformation to the Screen System

After the line segments have passed the trivial accept/reject test, they are transformed to the screen coordinate system. The following transformation is first applied to the Z coordinate in order to scale its clipping planes to Z' = −W, and Z' = W:

$$Z' = [-W \times (F + N)]/(F - N) + (2 \times W \times Z)/(F - N).$$

The value of 1/(F − N) is constant for all line segments and is therefore computed only once. In fact, two constants, a = 2K/(F − N) and b = −(F + N)/2, can be available so that Z' = Z × a × (b + Z). (Note that other transformations on Z can also be used.)

After the trivial accept/reject test, the following transformation to the screen system occurs:

$$X_S = X/W,\ Y_S = Y/W,\ Z_S = Z'/W.$$

The clipping planes then have these equations:

$$X_S = -1,\ X_S = 1,\ Y_S = -1,\ Y_S = 1,\ Z_S = -1,\ Z_S = 1.$$

Z1' and Z2' can be formed in 8 cycles. Only two reciprocals, 1/W1 and 1/ W2, need to be computed, and they can be interleaved and completed in 13 cycles in an '8837 if an 8-bit seed ROM is employed and in 12 cycles in an '8847. The line segment is transformed to the screen system in a further 6 cycles. The total is 26 cycles for the 'ACT8847 and 27 cycles for the 'ACT8837. A single-processor system would transform 600,000 line segments per second with a 15 MHz clock and 1.2 million line segments per second at 30 MHz.

Note that the above projection does not preserve planarity. See Newman and Sproull for perspective projections that do preserve planes.

The Clipping Operation

The final operation on line segments is to clip them to the cube:

$$X_S = 1,\ X_S = -1,\ Y_S = 1,\ Y_S = -1,\ Z_S = 1 \text{ and } Z_S = -1.$$

It is important to realize that the required resolution of X_S, Y_S and Z_S may only be 10 or 11 bits. Any divisions needed in an '8837 implementation at this stage could feasibly be done entirely by table look-up. It would certainly not be necessary to perform more than one iteration if an 8-bit seed ROM is employed. Two divisions can therefore be interleaved and completed in 7 cycles. However, three iterations are assumed in this example to give full single-precision accuracy.

Consider a three-processor pipeline, with each processor clipping against two parallel planes. The first will clip against the x planes $-1 < X < 1$. For clipping the P1 end of the line segment, $Q = (1 + X1, 1 - X1)$ is computed and Q' is formed, where $Q_i' = Q_i - |Q_i|$. I.e.,

$$Q_1' = 2(1 + X1), \text{ if } (1 + X1) < 0;\ Q_1' = 0 \text{ otherwise.}$$
$$Q_2' = 2(1 - X1), \text{ if } (1 - X1) < 0;\ Q_2' = 0 \text{ otherwise.}$$

At least one of Q_i' will be zero; the other will be negative. Hence, $MIN(Q_1', Q_2') = Q_1' + Q_2' = [(1 + X1) - |1 + X1|] + [(1 - X1) - |1 - X1|]$. Therefore, $MIN(Q_1', Q_2') = (1 - |X1|) - |1 - |X1||$. So, $t = |(m_1 - |m_1|) / 2d|$ and $s = |(m_2 - |m_2|) / 2d|$, where $m_i = 1 - |Xi|$, and $d = X1 - X2$. Note that only one reciprocal is required per processor.

A three-processor parallel system would have each processor work on one dimension, supplying its pair of max parameters to a "second stage." The second stage would receive (t_x, s_x), (t_y, s_y), (t_z, s_z) from the above system, compute max(t) = T and max(s) = S, and then clip the line as before:

$$X1' = X1 + (X2 - X1)T,$$
$$X2' = X2 - (X2 - X1)S.$$

The data flow and program listing for the program run by a processor working on the X dimension are given in Tables 91 and 92.

Table 91. Data Flow for the X Processor

CHAIN	Y	N	Y	Y	N	Y	N	Y	Y	Y	Y	Y	Y	Y
RA	X1	|	|	R0						d				
RB	X2	X1	X2	d								0.5		
S			d	m1	m2	R0	n1	T0	n2			R1		T1
P						d×R0		R0		R1		d×R1		0.5R1
C					m1	m2	m2							
Y			d				n1		n2					
STATUS														
CLK	1	2	3	4	5	6	7	8	9	10	11	12	13	14

CHAIN	Y	N	N	Y	Y									
RA		n1	n2											
RB														
S														
P		1/D		t	s									
C			1/D											
Y				t	s									
STATUS														
CLK	15	16	17	18	19	20	21	22	23	24	25	26	27	28

NOTE: $d = X1 - X2$; $n_i = m_i - |m_i|$

Table 92. Program Listing for the X Processor

	REGISTER TRANSFERS		ALU OPERATION	MULTIPLIER OPERATION
1.	LOAD RA, RB	Y←S	ADD (RA,−RB)	
2.	LOAD RA, RB		ADD (RA,−RB)	
3.	LOAD RA, RB		ADD (RA,−RB)	
4.	LOAD RA, RB		ADD (RA,O)	MULT (RA,RB)
5.		Y←S	ADD (C,−|C|)	
6.			ADD (2,−P)	MULT (S,1)
7.		Y←S	ADD (C,−|C|)	
8.				MULT (S,P)
9.				
10.	LOAD RA		ADD (P,O)	MULT (RA,P)
11.				
12.	LOAD RB		ADD (2,−P)	MULT (S,RB)
13.				
14.				MULT (S,P)
15.				
16.	LOAD RA	Y←P		MULT (RA,P)
17.	LOAD RA	Y←P		MULT (RA,P)
18.				
19.				

The three-processor parallel clipping system operates on a fixed loop of 17 instructions and can therefore clip 0.88 million line segments per second at 15 MHz and 1.76 million line segments per segment at 30 MHz. The second stage could not keep up with this rate without being implemented as several processors. A single processor can form the two max values in 23 cycles (a loop of 21 cycles) while two processors would take only 12 cycles (a loop of 10). The final clipping of the two endpoints takes about 11 cycles (a loop of 9 cycles).

To summarize, the fastest clipping system operates in the normalized screen coordinate system. It has six processors arranged in three stages – a three-processor parallel system with each processor working on each dimension; a two-processor system to form the two max values; and a single-processor third stage to clip the endpoints. The combined speed would be equal to that of the first stage, as previously described. A slightly slower four-processor system would use one processor for computing the two max values in the second stage.

Summary of Graphics Systems Performance

The previous section considered several approaches to the design of computer graphics systems based on the 'ACT8837 and the 'ACT8847. Table 93 summarizes the results. Table 94 shows the options available in combining the sub-systems listed in Table 93 into a design for a graphics system.

Table 93. Summary of Graphics Systems Performance

SUB-SYSTEM		SPEED AT 15 MHz	SPEED AT 30 MHz
a Transform, 4×4 matrix,	1 ACT88X7 cycle	0.94 M points/s	1.875 M points/s
b Transform, 3×3 matrix,	1 ACT88X7 cycle	1.25 M points/s	2.5 M points/s
c Eye clipping pipe,	6 ACT88X7 cycles	0.577 M lines/s	1.2 M lines/s
d Eye clipping	3 ACT88X7 cycles	0.288 M lines/s	0.576 M lines/s
e Eye Accept/Reject test	1 ACT88X7 cycle	1.071 M lines/s	2.143 M lines/s
f Screen clipping	5 ACT88X7 cycles	0.88 M lines/s	1.76 M lines/s
g Screen clipping	4 ACT88X7 cycles	0.71 M lines/s	1.42 M lines/s

Table 94. Available Options for Graphics System Designs

SYSTEM		SPEED AT 15 MHz	SPEED AT 30 MHz
I (a or b) + c,	7 ACT88X7 cycles	0.577 M lines/s	1.2 M lines/s
II (a or b) + d,	2 ACT88X7 cycles	0.288 M lines/s	0.576 M lines/s
III (a or b) + f,	6 ACT88X7 cycles	0.88 M lines/s	1.76 M lines/s
IV 2×(a or b) + c + g,	7 ACT88X7 cycles	2.5 M lines/s	3.75 M lines/s

In the fourth system, it is assumed that 2 processors are used for the transform of endpoints so as to balance the high clipping rate. It is also assumed that the accept/reject stage will eliminate more than 60% of the line segments so that the clipping system can keep up with the transform processors.